CONTINGENCY HANDLING IN MISSION PLANNING FOR MULTI-ROBOT TEAMS

by

Shaurya Shriyam

A Dissertation Presented to the
FACULTY OF THE USC GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(Mechanical Engineering)

December 2019

Dedication

This dissertation is dedicated

To my parents, Shreyash,

school teachers,

and college professors.

Acknowledgements

Doing a Ph.D. degree is an excellent academic journey, and I am very thankful to the University of Maryland, College Park, and the University of Southern California for providing me with an opportunity to experience it. The robotic labs I worked within both the universities had state-of-the-art robots and greatly stimulated my research progress. I would like to thank my advisor, Professor S. K. Gupta, for his constant motivation and terrific problem-solving skills, and guiding me through the challenges that I faced during my doctoral program. One faces the regular challenges during the completion of a doctoral program such as coming up with new ideas, producing results worthy of publication, and so on. My sincere appreciation goes to my supervisor for his excellent supervision during these five years.

I would like to thank my committee members, Professor Azad Madni, and Professor Yan Jin, for serving in my committee. They provided helpful advice to improve my dissertation. Specifically, their guidance was useful for me to improve the structure and writing style of this dissertation

I would like to acknowledge the financial support provided by the National Science Foundation (NSF) towards the completion of my research goals accumulating into this final dissertation.

Although Robotic Smart Assistant for Manufacturing project is not part of this dissertation, it was jointly done with Professor Krishna Kaipa, and it was a great experience to work with the Baxter robot.

I would like to thank all of my labmates with whom I interacted and worked during my research program. The discussions that I had with Brual, Ariyan, Pradeep, Shantanu, Rishi, Jason, Josh,

Sarah, Aniruddha, Nithyanand, Akshay, and Srudeep significantly helped improve my research work.

Finally, I would like to thank my family for their support across multiple continents, especially to my Mom who always had faith in me, and to my Dad who always gave me incredible words of encouragement, and to my Brother who took great interest in my research work and gave interesting feedback.

Table of Contents

List Of Tables

List Of Figures

Abstract

In order to be deployed in real-world applications, autonomous multi-robot systems need to be able to handle contingency events. Many multi-robot missions are time-critical, so using overly conservative methods to handle contingencies will not be acceptable in such applications. Robots themselves may be prone to unreliable task execution and may require human assistance for handling such contingencies, which must be properly planned and coordinated. This dissertation models the handling of contingency events in the form of execution of contingency tasks. When an unexpected event occurs that adversely impacts the mission workflow, then the multi-robot system, with or without the guidance of human operators, must perform contingency tasks to ensure the safety and completion of the mission. Contingency tasks can arise due to robot failure occurring due to battery failure or communication lapse. Contingency tasks may also occur explicitly as an additional task that becomes a bottleneck for nominal mission execution. For example, if an autonomous vehicle were to run out of fuel, the vehicle must be taken to a filling center to refuel it. Additional tasks that may be required for mission completion are referred to as contingency tasks. Contingency tasks may also arise due to uncertain operating conditions or bad information about operating conditions. The general approach to handling contingencies in robotics has been to design failure recovery protocols which kick in when a specific contingency occurs. However, such a reactive approach does not work well in time-critical missions.

This dissertation presents the computational foundations for proactively handling contingencies encountered during multi-robot missions. A formal model checking framework is developed to evaluate the adequacy of contingency resolution strategies that may be encountered during

multi-robot missions. In this dissertation, the developed framework was successfully applied to an assembly line setup and a ship escorting mission which used unmanned surface vehicles, and the performance of the strategies designed to handle different contingencies was evaluated. This dissertation also presents a proactive approach for incorporating contingency tasks in nominal mission planning. It was found through simulation experiments that the proactive approach outperformed the reactive and conservative approaches. Finally, the dissertation proposes a method for task decomposition required for completing collaborative surveillance tasks assigned to multiple robots under the conditions of uncertain information and varying robot availability. It was found that the developed methods for task decomposition were robust against these uncertainties and were able to successfully guide the robots to successful task completion while minimizing the time taken.

Chapter 1

Introduction

1.1 Motivation

In recent years, unmanned robots have been increasingly used for a variety of operations, such as inspection of agricultural fields, search and rescue, surveillance, and defense applications [2–5]. In many of the applications, deploying a team of robots instead of a single robot leads to automation of mundane and critical activities which leads to higher efficiency, increased productivity, safer working conditions, robust operations and reduced costs in the long-term. Multi-robot teams are also increasingly being used to provide support in complex industrial operations [6–12]. Most of these applications require the execution of spatially separated tasks. Multi-robot teams are well-suited to such applications because they can execute multiple such tasks in parallel. Multiple robots can divide each task into sub-tasks, which can then be independently accomplished by each robot or the robots can work as a team and assist each other with different sub-tasks. Researchers have extensively examined the area of collaborative robotics [13–19]. Even in manufacturing applications, active research is going on to incorporate multi-robot setups in factory floors to boost operational efficiency and bring down costs [20–28].

The motivation for modeling such multi-robot missions is due to the missions involving complex sequences of survey-like tasks involving unmanned surface and ground vehicles. Teams (or

coalitions) of multiple robots are being extensively employed for automation projects and complex missions in domains such as factory cells, outdoor survey/inspection missions, and search-and-rescue missions.

Multi-robot missions require a set of objectives to be fulfilled. This in turn translates into a sequence of tasks that must be executed by each robot. Each robot plans its trajectories to execute these tasks. Such nominal mission planning techniques have been investigated in detail and generally yield good results. However, in more realistic scenarios where nominal mission plans have to be modified due to changes in the work environment, traditional methods of nominal mission planning do not perform very well.

Some unexpected event may occur causing bottleneck which requires an additional task to be executed to safely resume the mission. It may also happen that robots may fail to perform a task that they were assigned to finish and hence must again attempt to execute the task. Sudden changes in the environmental information associated with the mission scenario may require improving the current mission execution plan to ensure better performance. Mission execution must also be robust to varying availability of robots assigned to a mission. Such bad and unexpected events are referred to as contingencies. This dissertation is focused mainly on handling contingency situations for multi-robot systems effectively. The developed methods should be able to handle contingencies in real-time so that multi-robot teams can be deployed for challenging missions. The developed methods should proactively account for potential contingencies to ensure that plans can efficiently deal with contingencies.

There are several civilian applications where proactive handling of contingencies for multi-robot autonomous systems can significantly bring down mission costs while boosting the guarantee of safety for mission infrastructure as well as humans involved, thereby leading to the more rapid deployment of autonomous systems in diverse areas. Active research is taking place in making the robots better prepared to handle unfavourable conditions that arise during mission execution [29–33]. There are several multi-robot applications [34] which may potentially benefit

from better and proactive handling of contingencies. A few representative applications are enlisted below:

1. **Infrastructure security:** Many critical facilities require protection and constant patrolling. However, hiring enough human guards is just not economically viable in the long-term [35]. Neither are robotic technologies advanced enough to completely hand over the guarding and patrolling jobs to autonomous systems. Human operators embedded in multi-robot systems provide the best solution. However, there are possibilities for several unexpected adverse conditions that may interrupt the security missions of the robots [36]. Improved methods of handling contingencies prove to be very beneficial.

2. **Hazardous Waste Cleanup:** Sending humans for nuclear waste cleanup puts the lives of the workers at risk due to radioactive exposure and is highly unethical, if not inhumane. In the Fukushima Daiichi nuclear disaster, elderly pensioners volunteered to participate in the cleanup of the power station, thus saving young men from receiving a fatal dose of radiation. Given the prevalence of nuclear plants throughout the world and the trend getting stronger and stronger with more countries enthusiastically signing up for the program, a concerted and serious effort must be directed to automating nuclear plant operations. This may require handling situations such as unexpected communication failures and unexpected failure of robot's batteries [37].

3. **Urban search and rescue:** In dangerous disaster scenarios, there is extreme time pressure on the rescue workers and dogs to locate injured civilians and rescue them. If robots can replace humans and dogs, it will reduce the risk to workers' lives. But there is a risk of gas leaks and explosions happening unexpectedly or debris falling from a construction. There is significant research effort underway to achieve robotic search and rescue [38].

4. **Management and transportation of large containers:** In ports, airports, and railway yards, there is a frequent requirement of transporting heavy containers from one location to

another. Since these places are unstructured environments, there is a lot of potential for unexpected events to interrupt or disrupt the mission. If a team of robots is to be employed to conduct all container-related tasks, then safety features in the form of contingency handling is a must [39, 40].

5. **Space exploration:** NASA has laid down exciting visions for space exploration in the near future. One of its objectives is to increase the use of robots for construction, repair, and maintenance purposes in space stations and spaceships. Robots may be used additionally to provide assistance or companionship to the space crew. However, this is quite challenging because these robots have to operate in a sensitive environment. Any chemical leak, electronics failure, computer failure, or collision with big space debris could be fatal for the space crew and cause a loss of several billion dollars. However, research progress is continually being made in this sphere [41].

However, incorporating contingencies in multi-robot mission planning suffers from several challenges:

1. There is the modeling challenge to decide what information to abstract out or hide from explicitly considering in the mission model so that the effects of contingencies can be analyzed without making the model computationally very expensive.

2. If failures occur during multi-robot missions, then they may result in human injury or damage to expensive equipment. Thus, the contingency planning methods must also seek to guarantee safe mission completion.

3. It is challenging to use a model checking framework to assess the adequacy of contingency resolution strategies for multi-robot mission because it is computationally more time-taking than simulation testing. However, to seek robust guarantees in a rigorously proved mathematical framework, one should make effective use of the model checking toolkits developed by the formal methods community.

4. It is difficult to allocate tasks to robots under uncertainty when disrupting events such as contingencies are expected. On top of that, wishing to minimize the expected mission completion time in a computationally efficient manner requires developing heuristics that guide the solution search properly.

5. It must be taken into account how robots distribute a task assigned to them in an efficient manner. Use of computationally heavy methods may provide the optimal decomposition of a task among multiple robots but would be so time-taking that they cannot be deployed online. So a planning approach must be able to handle adverse effects like the uncertainty of task execution and completion, or varying robot availability, in a faster way even if it implies that only a near-optimal distribution of tasks among robots can be obtained.

1.2 Goal and Scope

The research presented in this dissertation is concerned with developing proactive approaches to handling contingencies for multi-robot teams that have been assigned to a complex mission. The research work presented in this dissertation focuses on contingencies that can be anticipated but their occurrence cannot be predicted in advance. The methods described in the rest of the work are designed for robot team sizes ranging from two to twenty robots.

The main research issues this work addresses are as follows:

1. *Modelling and verification of contingency resolution strategies for multi-robot missions:*
   An approach for evaluating contingency resolution strategies using temporal logic is presented. A framework is presented for nominal mission modeling, designing contingency resolution strategies and evaluating their effectiveness for the mission. The approach in this dissertation focuses on leveraging the use of model checkers to the domain of multi-robot missions to assess the adequacy of contingency resolution strategies to minimize the adverse

effects of contingencies on the mission execution. The missions are allowed to have deterministic as well as probabilistic transitions.

The use of formal methods and model checking software packages allows the development of autonomous systems with guarantees against faulty behaviors and inconsistencies. Such autonomous systems may be deployed on industrial scales and would make processes more efficient and streamlined. For this purpose, two completely different case studies are chosen and the application of model checking software is shown to provide guarantees of safety and success for complex multi-agent missions. A nominal mission model is proposed and contingency resolution strategies are developed in both scenarios. Then the results obtained via model checkers are analyzed. It is not possible to generate such logical results as well as expected values of critical mission variables so easily in any other way than modeling a mission as a formal transition system and encoding it in a model checker. This is because nominal mission specifications, contingency models, and contingency resolution strategies may need to be changed repeatedly to decide what works best. A methodology that allows flexibility in efficiently evaluating these possibilities in a mathematically rigorous way is bound to outperform conventional methods like running heavy simulations in memory-intensive software tools. These methods can be exploited to quickly evaluate contingency resolution strategies and screen them for their performance. Once some good ways have been identified to handle contingencies for a mission, then one can resort to simulation techniques to study the strategies in finer details. Thus, there will be no need for eliminating simulation testing paradigm.

The two case studies are as follows. The first one considers the escorting of a ship in a port where multiple contingencies may occur concurrently and analyze the proposed contingency resolution strategies. The second one considers a manufacturing scenario where multiple assembly stations collaborate to create a product. In this case, assembly operations may

fail, and human intervention is needed to complete the product. Several different strategies are investigated and their effectiveness is assessed based on mission characteristics.

This modeling framework is extended to a production line set up across multiple robotic assembly cells by integrating probabilistic modeling and temporal logic verification techniques. A production line is a complex system comprising of multiple assembly cells which in turn is composed of multiple assembly stations. The performance of the proposed operations plan is evaluated by decomposing the whole system into smaller subsystems which are analyzed separately. This may help in designing more optimized systems. An assembly cell is modeled in PRISM software, which is a commonly used probabilistic model checking tool. Contingencies may arise in each cell during the production operations and are resolved by humans or the assembly robots themselves. Multiple cells operating cooperatively to finish the production of parts are then simulated. Formal analysis is performed for each cell, which is a subsystem of the production line and then the aggregate behavior of the resulting production line is studied. Based on the feedback from PRISM analysis of individual cell behavior, it can be determined how to change cell settings to achieve the best production performance.

2. *Incorporation of Contingency Tasks in Nominal Task Allocation:*

   Complex logistics support missions require execution of spatially separated information gathering and situational awareness tasks. Mobile robot teams can play an important role in the automated execution of these tasks to reduce mission completion time. Planning strategies for such missions must take into account the formation of effective coalitions among available robots and assignment of tasks to robots to minimize the expected mission completion time. The occurrence of unexpected situations that adversely interfere with the execution of the mission may require the execution of contingency tasks so that the originally planned tasks may proceed with minimal disruption.

The work reported in this dissertation will be useful for deploying multi-robot teams to support complex logistics missions spread over a large area where the robots must be prepared to handle contingencies that can adversely impact the mission. The proposed proactive approach can be used to handle contingencies in information gathering, surveillance, guarding, and situational awareness tasks to support safe and secure transportation of important assets through sensitive areas. A port operation is used as an illustrative example where unmanned surface and aerial vehicles can be useful in ensuring the safety and security of the ports. This is a computationally challenging problem. This work proposes heuristics algorithms to solve the task allocation problem among many different agents efficiently. The approach presented in this work integrates the available information regarding mission and contingencies along with the resource constraints to plan the mission execution.

Initially reported *potential* contingency tasks may not always affect mission tasks owing to the uncertainty in the mission environment. When potential contingency tasks are reported, the planner updates its existing plan to minimize the expected mission completion time based on the probability of these contingency tasks impacting the mission, their impact on the mission and other task characteristics. Various heuristic-based strategies are described to compute task-allocations for robots for mission execution. Simulation experiments are performed to compare them and the computational performance of the best performing strategy is analyzed. It is shown that the proactive approach to contingency task management outperforms both the conservative and reactive approaches.

3. *Decomposition of Collaborative Surveillance Tasks:*

   This dissertation introduces an approach for decomposing exploration tasks among multiple Unmanned Surface Vehicles (USVs) in congested regions. To ensure the effective distribution of the workload, the algorithm has to consider the effects of the environmental constraints on the USVs. The performance of a USV is influenced by the surface currents, the risk of

collision with the civilian traffic, and varying depths due to tides, and weather. The team of USVs needs to explore a certain region of the harbor, and an algorithm must be developed to decompose the region of interest into multiple sub-regions. The algorithm overlays a 2D grid upon a given map to convert it to an occupancy grid and then proceeds to partition the region of interest among the multiple USVs assigned to explore the region. During partitioning, the rate at which each USV can travel varies with the applicable speed limits at the location. The objective is to minimize the time taken for the last USV to finish exploring the assigned area. The particle swarm optimization method is used to compute the optimal region partitions. The method is verified by running simulations in different test environments. The performance of the developed method is also analyzed in environments where speed restrictions are not known in advance.

1.3 Overview

The outline of this dissertation can be briefly summed up using the block diagram in Figure 1.1. To incorporate contingencies in nominal mission planning, three stages are considered:

1. The given mission model, usually provided in English, must be translated into a formal model. The modeling must also include the contingency model describing the contingencies that may occur and the nature of their occurrence. Using this information, contingency resolution strategies are designed manually. The objective is to select the most effective contingency resolution strategies by verifying their desired properties. For this purpose, model checking software packages are used. The results of *model checking* may also be used to modify the strategies to improve their expected performance.

2. A nominal mission planner is also developed in parallel that takes in the nominal mission model and generates a *task precedence graph* comprising of the order in which different tasks must be performed by the available robots. But the execution of different tasks must be

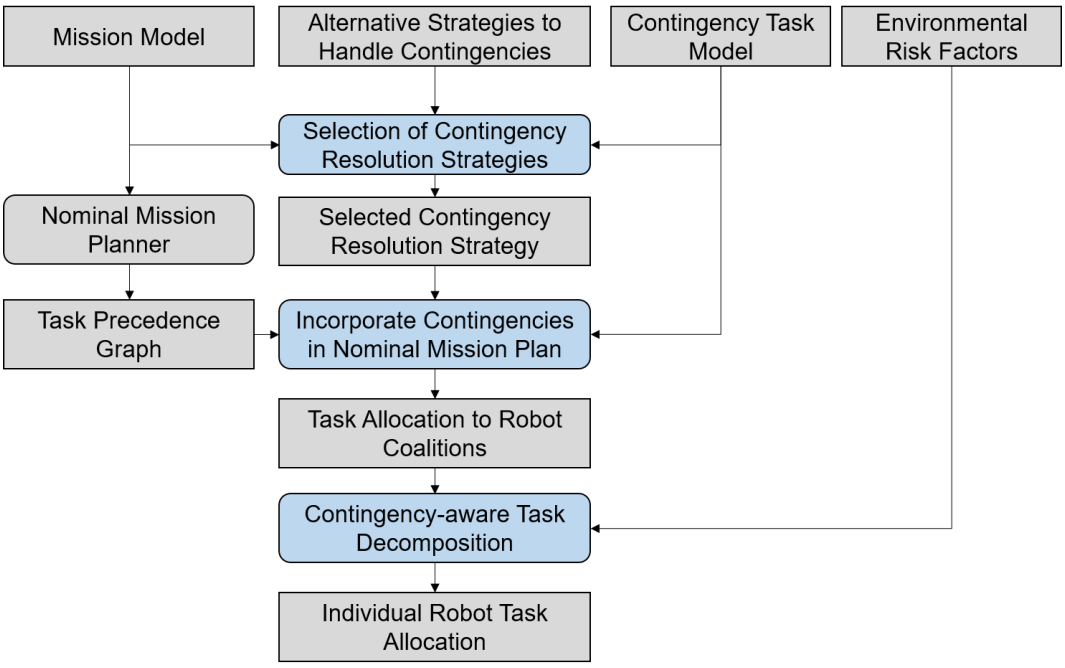


Figure 1.1: This is an outline of the research work investigated in the dissertation.

allocated to the agents and scheduled in such a way so that the expected mission completion time is minimized. Concurrently, the contingency information must be taken into account and proactively integrated into the task network. The resulting *task allocation* then specifies which task must be executed in what order by which robots.

3. When multi-robot tasks are scheduled for execution, then multiple robots are also assigned to the task. These robots must coordinate to finish the task optimally and efficiently. Contingencies may arise due to imperfect information, or bad information about the operating conditions of the task, or robots becoming suddenly unavailable for task execution due to being diverted to another task of higher priority. This entire information must be incorporated to decompose the task among several robots for efficient execution without interference and bottlenecks. Such a *task decomposition* results in task schedules for individual robots. These *individual robot task allocations* are easier to implement and can be monitored by onboard computers.

Chapter 2

Literature Review

This chapter provides a literature review of methods used for multi-robot mission planning that is used in this dissertation for incorporating contingencies. The methods discussed here are those from the domains of task allocation, scheduling, temporal logic based model verification, and task decomposition.

2.1 Mission Planning Using Model Checking

2.1.1 Multi-robot Mission Modeling for Model Checking

Model checking essentially helps in verifying the robustness of missions against contingency scenarios. Other researchers have investigated model checking for formulating constraints for automated multi-agent systems and then checking for any inconsistency [42, 43]. Most model checking work in robotics generally focuses on properties involving temporal logic because these are the most commonly queried and desirable properties for ascertaining the robustness and correctness of the system in the robotics community. Temporal logic properties are widely expressed using computation tree logic (CTL), linear temporal logic (LTL), or a combination of both. For stochastic processes, system specifications make use of probabilistic temporal logic like Probabilistic Computation Tree Logic (PCTL).

Bolton *et al.* [44] survey the usage of formal model verification methods in improving human-automation interaction (HAI). Such autonomous systems may suffer failures due to being brittle, and hence, unexpected scenarios may overwhelm the human operator. The cause of such failures lies mostly in the design of the automation software. Identifying shortcomings in the design of such automation systems and their interfaces for human operators allows automation to become more robust.

A major problem is associated with mode confusion where human fails to keep track of the mode in which the device or the automation system is performing. Over more extended periods of times, it leads to human getting confused and adopting random guessing strategy or some other incoherent strategy to keep guiding/controlling the system. This may lead to the surprise of the human operator eventually and cause disruption in the work output. It is therefore essential to take into account the human mental model by explicitly or implicitly including it in the transition system representation of the human-automation interface. The paper lays out the different kinds of formal properties generally verified by other researchers.

Temporal logic-based model verification methods offer several benefits when deployed for planning at a higher level of abstraction, viz. the task and mission planning layer of autonomous multi-robot systems. Pant *et al.* [45] use Signal Temporal Logic (STL) for specifying missions involving multiple drones that must take off and land from designated spots, must monitor the environment, reach a location while avoiding obstacles, perform periodic surveillance and data collection, collect sensor data from equipment fitted across a smart city. A simulation was carried out using up to 16 drones and actual experiments using two drones to check out the applicability of their method. They use unconstrained optimal control techniques to compute trajectories that are dynamically feasible and can be tracked efficiently using low-level off-the-shelf position and attitude controllers. They formulated their optimization problem using CasADi symbolic framework designed for automatic differentiation and optimal control and used the HSL_MA27 FORTRAN routine with IPOPT for solving the nonlinear program. They compared their method

with another STL-based trajectory generator called BluSTL, which took more time to solve the problem.

Liu *et al.* [46] encode mission specification for multi-agent systems with wireless communication constraints using Signal Temporal Logic (STL) and spatial-temporal logic (SpaTeL) formulas. Spatial-Temporal Logic (SpaTeL) is a unification of Signal Temporal Logic (STL) and Tree Spatial Superposition Logic (TSSL). SpaTeL is capable of describing high-level spatial patterns that change over time. The robot environment is a 2D grid area which is represented using a quadtree structure. A MILP optimization program is used to develop a distributed model predictive control (MPC) based strategy to satisfy the mission constraints. In their second work [47], they require multi-agent systems with inter-agent communication to satisfy motion specification in the form of signal temporal logic formulas. It is the task of a local motion controller to ensure that its associated agent satisfies certain user-defined local specifications while avoiding collisions with other agents and obstacles. It is also important to maintain high communication quality among the agents. A Gaussian fading model is used to model wireless communication signals. The signal temporal logic specifications and wireless communication conditions are expressed as mixed integer-linear (MIL) constraints in terms of agents' state variables and communication signal status. Many simulations carried out using MATLAB, and AMPL/Gurobi show that their strategy improves the communication quality without obstacle collision.

Nikou *et al.* [48] perform controller synthesis for multi-agent path planning where Metric Interval Temporal Logic specifications are given. Each agent has a dynamic model capturing the coupling effects as well as control inputs. A five-step controller synthesis algorithm is presented, which is tested out in simulation for time-bounded constraints of traveling to a region. Zhou *et al.* [49] present an optimization-based method for planning paths of a mobile robot for which bounded temporal constraints are specified using metric temporal logic (MTL). The metric temporal logic formulas are translated into mixed integer linear constraints and then solved using YALMIP-CPLEX software. They provided simulation examples that included static as well as moving

obstacles based on quadrotors and cars. High-level strategies provided by non-experts are encoded as preferences using the Planning Domain Definition Language (PDDL) 3.0 language by Kim *et al.* [50] and used by automated planners.

MacKenzie *et al.* [51] use behavior-based modeling of multi-agent intelligent systems using societal agent theory. Firstly, the primitive robot behaviors are abstracted into individual agents, and then assemblages are created out of them. Assemblages are groups of fundamental behaviors and coordination mechanisms among these agents to ensure coherent behavior.

The Configuration Description Language (CDL) is developed to encode these agent behaviors and makes use of their recursive nature. It also allows the same agents to be used in different missions using different robots. The transition from one state to another is modeled using a finite state automaton to select valid and feasible states that may be accessed by robots.

Lomuscio and Raimondi [52] consider deterministic, non-deterministic, and other classes of interpreted systems which are formal models to define multi-agent systems and interpret the semantics of Alternating-time Temporal Logic (ATL) operators in their context. The first class of agents may guess moves, the second class consists of deterministic agents, and the third class of agents chooses to perform specific actions consistently.

Finally, they present a model checking algorithm based on Ordered Binary Decision Diagrams (OBDDs) which users can apply to their problems using the MCMAS (Model Checker for Multi-Agent Systems) software. They used MCMAS to solve games such as Nim, a simple card game, and a scenario involving the failure of the coyote to catch the roadrunner inspired from the popular cartoon, *The Road Runner Show*.

Sahin *et al.* [53] developed a temporal logic called counting linear temporal logic plus (cLTL+) to model the collective behavior of multi-agent systems when heterogeneous dynamics is allowed. They formulate an optimization problem and use Gurobi to compute the trajectories for different robots. Then they studied asynchronous trajectories and found their method to work for bounded asynchrony with appropriate modifications.

DeCastro *et al.* [54] combine Reactive Mission Planning using Linear Temporal Logic, and Local Motion Planning using convex optimization while enforcing the dynamic constraints of the robot and guaranteeing collision avoidance in 2D and 3D workspaces. The reactive mission planner synthesizes a correct-by-construction controller based on the high-level linear temporal logic specifications of motion sequencing.

Then physical experiments with two centrally-controlled Nao robots and a dynamic obstacle (KUKA's youBot) were carried out to demonstrate deadlock resolution. Specifically, one of the Nao robots reversed its direction due to an impending deadlock with youBot.

Lindemann *et al.* [55] allow users to define a performance function in terms of a desired temporal behavior which the multi-robot system should satisfy based on a continuous state feedback control law. The Signal Temporal Logic (STL) based formalism is used to prescribe constraints for the multi-robot system. The developed control law is robust to disturbances and satisfies the signal temporal logic specifications within user-defined robustness.

They tackle specifications which require certain conditions to be met within certain time intervals or sequential specifications, meaning that a sequence of conditions will be satisfied in prescribed time intervals. A non-linear multi-robot system exhibiting single integrator dynamics while using a consensus protocol to stay close to each other is used to demonstrate tasks like formation control, sequencing, and dispersion.

Ke *et al.* [56] model the behaviors of a ground control station, unmanned ground vehicles, micro aerial vehicles, and high-level unmanned aerial vehicles using Kripke structures. Then they verify the mission model's validity using model checkers like SPIN. They also integrate the SPIN model checker to assist in motion planning by generating counterexamples.

Model checking techniques are very well-suited for state space search problems and by modeling path planning problems as state space search problems, they are effectively able to utilize linear temporal logic based model checking software packages to not only verify missions but also generate motion plans.

2.1.2 Model Checking Tools for Verification and Evaluation

There are several model checking software available online, e.g., NuSMV, SPIN, PRISM that automatically verify user properties for a given system model. Generally, system models have to be coded in a programming language specific to the model checking being used. For SPIN model checker, one would have to code the model in Promela language. For NuSMV model checker, one would code in the SMV specification language. PRISM model checker also has its programming language. Many such software frameworks have been used for robotic planning. Fraser *et al.* [57] provide an extensive survey of model checkers, their comparisons and reasons for their effectiveness.

Fainekos *et al.* [58] used NuSMV for path generation and MATLAB for controller synthesis to plan robot motion based on LTL constraints. Lahijanian *et al.* [59] designed the Robotic InDoor Environment (RIDE) simulator to move an iRobot Create mobile platform autonomously through corridors and reconfigurable intersections using PCTL-based MDP framework. Finucane *et al.* [60] designed the LTLMoP (Linear Temporal Logic MissiOn Planning) toolkit for designing, testing, and implementing hybrid controllers generated automatically from task specifications written in Structured English or Temporal Logic. Konur *et al.* [61] used PRISM model-checker for verification of global robotic swarm behavior. UPPAAL model checker is an integrated tool environment for modeling, simulation, and verification of real-time systems modeled as a collection of non-deterministic processes and has been used for various robotic applications also [62–64].

Karaman and Frazzoli *et al.* [65] solved the LTL based multi-vehicle routing problem by converting the LTL constraints into Mixed-Integer Linear Programming (MILP) problems. LTL formulas may also be converted to an automaton whose product is then taken with robot automaton and then automata graph search methods are constructed to plan robot motion [66, 67].

Shoukry *et al.* [68] solve a multi-robot path planning problem by computing collision-free trajectory controllers. They use a satisfiability modulo convex (SMC) programming approach

to decompose the problem into smaller sub-problems via Boolean constraints. They compared their method with sampling-based motion planners and outperformed them significantly for single robot reach-and-avoid specifications. They even simulated multi-robot scenarios where multiple targets had to be visited infinitely often. Kamali *et al.* [69] perform formal verification of safety properties for automotive platooning using the Agent Java Pathfinder (AJPF) model-checker. Machin *et al.* [70] developed a safety monitoring framework based on the NuSMV model checker for autonomous systems.

Liu *et al.* [71] subject an unmanned ground vehicle to traffic rules in the form of linear temporal logic specifications. Environmental conditions are used to trigger the behavior execution of the unmanned ground vehicle. The unmanned ground vehicle is modeled as a reactive system with its behaviors encoded as a transition system, and the linear temporal logic mission specifications are used to compute the resulting automata. Simulations were carried out in Python-based, open-source Linear Temporal Logic MissiOn Planning (LTLMoP) to plan the motion of the unmanned ground vehicle for reaching its goal location while reacting to the environmental factors appropriately.

Mehta *et al.* [72] design a robot compiler system which requires a user to input the functional specification of desired robotic behavior in Structured English. This requires some technical skill on the part of the user, but this is overall a very low requirement. That is then mapped into a linear temporal logic formula which is used to perform controller synthesis.

They also provide a library of robot building blocks from which the system filters certain blocks for each proposition. The system also allows for faulty linear temporal logic specifications, which it corrects based on behavioral conflicts. Two robots, one line-following, and another wall-following were made by using this system, and these robots accomplished their tasks while being assisted with sensors.

Choi *et al.* [73] present Kripke structures for three missions and verify their critical properties using the model checker called MCMAS (Model Checker for Multi-Agent Systems). An unmanned

ground vehicle is supposed to reach a goal location and is assisted by an unmanned aerial vehicle. Such a mission is made more complicated by adding multiple unmanned aerial vehicles and unmanned ground vehicles.

There is also a ground control station that initiates the mission and guides the robots until the mission ends using images transmitted to it. The ground control station takes safety precautions like instructing the unmanned aerial vehicles to land once the communication signal has been lost.

Humphrey [74] model the escorting of a VIP vehicle through a road network using multiple unmanned aerial vehicles. A human operator lays down the procedure through which the unmanned aerial vehicles explore the roads for safety and guide the VIP vehicle towards its target. This model of mission execution is coded up in Promela language.

The conditions that should hold for the given mission execution are specified using the language of linear temporal logic. SPIN model checker processes the linear temporal logic specifications and the Promela model to verify whether the properties hold or not. If the properties do not hold, then a counterexample is produced, which may be used to debug the mission execution plan further.

Rothwell *et al.* [75] present a model checking software framework called Verifiable Task Assignment and Scheduling Controller (VTASC) which takes in mission specifications in natural language as input and converts it to temporal logic formula like linear temporal logic (LTL), computation tree logic (CTL) and CTL's extension, real-time computation tree logic (RTCTL). Then a model checking software called NuSMV is used to check whether the user-defined properties are satisfied during mission execution plans as defined by the operators.

A graphical user interface (GUI) is used where users create the temporal logic specifications in the English language. The temporal logic translator must process the final English specification and input it to the NuSMV if and only if it passes grammatical checks. They compared VTSAC against two other such software tools: Specification Pattern Instantiation and Derivation EnviRonment (SPIDER) and Linear Temporal Logic for MissiOn Planning (LTLMoP).

Alonso-Mora *et al.* [76] propose an integrated mission and motion planning framework where the mission planning is done offline, and the motion plan is computed at runtime. The mission planner takes as input the linear temporal logic specifications and generates the automaton representing the mission. The localized motion planner uses a convex optimization based approach. They also describe the deadlock resolution strategy to resolve deadlocks by modifying the specification, if needed, to generate a feasible plan that may then be executed.

Bank *et al.* [77] proposed an autonomy framework for smart manufacturing systems using linear temporal logic and its corresponding model checkers like NuSMV. However, NuSMV solvers may be very time-consuming because they exhaustively search the state space. To solve this problem, they considered a toy problem where blocks have to be appropriately rearranged on a 2D grid. Then they reduced the exploration space with a divide-and-conquer approach. Rather than solving for the entire grid area at once, smaller areas were considered one by one and then solved using NuSMV. This led to very high speed-ups as NuSMV is very fast in solving small problem sets. Their simulations showed that they were able to achieve linear computational time performance.

Schillinger *et al.* [67] present a linear temporal logic based planning approach for on-demand multi-robot missions called simultaneous task allocation and planning (STAP) which constructs a team model for multi-robot missions to decompose the mission into more straightforward tasks and allocates them to different robots. Given a linear temporal logic mission specification for the mission, they generate an optimal task decomposition for the mission regardless of whether the explicit tasks are provided as input or not. The resource constraints like limited battery power are also taken into consideration for each robot. Several missions arising in office scenarios are then simulated in the Robot Operating System (ROS) to validate and analyze the performance of their algorithm against the classical approach of planning for task costs first. In their second work [78], they consider multi-robot mission scenarios in indoor office spaces. They simulated two robots in an office space which have to carry drinks from one location to another while avoiding

certain areas when carrying drinks. They also ran physical experiments with two robots. The mission constraints were encoded into a linear temporal logic formula. FlexBE software is used to generate complex robot behaviors using GUI facilities. It is also well-integrated with ROS. Spot is used to translate the linear temporal logic formula to an automaton. Then planning is done to minimize the maximal team cost incurred to finish the mission by using a label-setting approach for multi-objective planning problem.

Maly *et al.* [79] present a multi-layer planning framework for a mobile robot along with replanning capabilities when obstacles are encountered. The high-level layer takes as input a linear temporal logic specification for the robot to satisfy. The robot is a second-order car with hybrid dynamics, and the linear temporal logic specification specifies how a particular office area must be explored. A product automaton is formed out of the robot dynamics and linear temporal logic specifications. The high-level layer generates plans for which continuous robot trajectories are computed using a low-level planner. Graph-based distances to an accepting state in the automaton are used to satisfy the specification as closely as possible.

Saha *et al.* [80] study multi-robot mission settings whose specifications are given using linear temporal logic based safety conditions concerning obstacle avoidance and collision avoidance. The problem is first converted to a form understood by a Satisfiability Modulo Theories (SMT) based problem-solver called Z3 which then generates a trajectory for the robot by solving the system of constraints denoted by the linear temporal logic safety properties. Then simulation experiments for a group of quadrotors amid static obstacles were carried out successfully.

Guo *et al.* [81] combine motion planning and action planning to tackle tasks involving visiting regions sequentially and performing actions at various locations. The task specifications are given in the formalism of linear temporal logic. The robot automaton is constructed using an abstracted robot mobility model and its action map. A product automaton is then formed with the Büchi automaton representing the linear temporal logic specifications. Then a graph-search algorithm is used to compute the optimal task sequence using which the hybrid controllers are synthesized.

Li *et al.* [82] have developed a model checking framework called PAT: Process Analysis Toolkit. They used this and other model checkers like Spin, NuSMV to solve some classical planning problems like bridge crossing and sliding game problems. They found their model checker even to outperform state-of-the-art planners designed for these problems. PAT is highly efficient, ready to be used out of the box and modular. It was demonstrated to automate the municipal transportation management system by simplifying fare collection and notifying users about important information customized to their history.

2.1.3 Handling Stochastic Events

Contingencies arise in robotic systems because robotic behaviors as well as environment responses are probabilistic [83] and multi-robot systems may generally be modeled as Markov Decision Processes (MDPs) [84–87]. Cizelj *et al.* [88] present an approach to control a vehicle in a hostile environment while considering static obstacles as well as moving adversaries against whom the vehicle must protect itself from collisions. The vehicle mission model, as well as the effect of adversaries on the vehicle, is modeled as a Markov decision process. The mission objective is to start from a certain region, reach another region to pick up an item and drop it off at another designated region. They adopt the Probabilistic Computation Tree Logic (PCTL) control synthesis approach to plan the mission for the vehicle and use the off-the-shelf PCTL model-checking tool PRISM for this purpose.

Ding *et al.* [89] present probabilistically guaranteed optimal control policies over infinite time horizons for a Markov decision process with linear temporal logic constraints. The policy seeks to minimize the average cost per cycle (ACPC) starting from the initial state. Rigorous conditions for policy optimality are theoretically proved, and a dynamic programming algorithm is developed. This is demonstrated in MATLAB with and without linear temporal logic constraints for a robot navigating an environment and performing persistent tasks such as environmental monitoring and data gathering.

Faruq *et al.* [90] develop simultaneous task allocation and planning for multi-robot systems which are operating in uncertain environments and prone to failures. The robotic system is modeled as a Markov decision process. They propose a sequential model where each robot is considered independently one by one. Switch transitions are added to combine the individual robot models into a team Markov decision process.

The goal is to find a policy that maximizes the probability of satisfying prescribed co-safe linear temporal logic based task specifications without violating safety specifications also. They also develop the feature of allowing a task to be reallocated to another robot when its assigned robot fails. For this purpose, they synchronize the robot actions within the range of policies computed by their sequential model. They study the effect of increasing the number of failure points on computational time and reallocation frequency.

Feng *et al.* [91] consider a road network surveillance problem involving an unmanned aerial vehicle and a human operator. Human operator assists the unmanned aerial vehicle to visit newer way-points based on the drone's existing state and also instructs the unmanned aerial vehicle whether to continue loitering over a way-point to improve upon the sensory information that has already been gathered.

The human operator's performance is modeled in terms of his/her proficiency, workload, and fatigue. They model this mission as a Markov decision process and implement it in the PRISM model checker. They study the effect of model parameters on the minimization of expected mission completion time. Then they model the same mission as a stochastic two-player game and study the trade-offs between expected mission completion time and expected number of restricted operating zones visited by the unmanned aerial vehicle.

Kloetzer and Mahulea [92] consider a partitioned environment in which one or more robots must survey regions of interest which appear and disappear probabilistically. The automaton building and graph search approaches are used to find robot trajectories that maximize the probability of satisfying the task of surveying regions of interest. This task is specified as a linear

temporal logic formula. They also seek to minimize the traveled distance. All robot collisions are ignored during planning to avoid extra computational costs. Such collisions are resolved during trajectory following stage.

Chen *et al.* [93] use automata learning technique in uncertain, dynamic, and partially known environments to optimize surveillance missions. An indoor environment with doors is considered. Based on environmental data, the robot must learn how to incorporate the doors in its motion planning. Based on the learned door models, optimal control strategies are designed for the surveillance mission, which is modeled as a temporal logic game. The control policy is proven to be very close to the truly optimal one when unknown environmental information is fully learned.

2.1.4 Contingency Management

Failure recovery through contingency management [94–96] is a crucial component of multi-robot systems that lead to their autonomy in real-sense because unexpected and uncertain contingencies are bound to arise and may result in failure of robotic operations.

Christensen *et al.* [97] developed a method to detect failures of robotic sensors and actuators by observing changes in the sensory data that is recorded in real-time. For this purpose, faults are caused by software programming in the system, and their effects on sensory data are learned through neural networks.

McGuire *et al.* [98] used contextual multi-arm bandit algorithms to help the robots learn how to select the best assistants to recover from failures while accounting for uncertain/dynamic operating conditions and assistant capabilities. In many works on multi-robot systems, failures arising due to communication failures, lack of situational awareness, sensor failures, and localization failures are studied [99–101].

Tang *et al.* [102] incorporated the advancement in prognostics and health management systems to develop a proactive and automated contingency management system that takes into account

prognosis information for mission replanning and failure recovery. They demonstrated this framework on a Pioneer 3-AT ground robot by using NASA's Hybrid Diagnostic Engine (HyDE) as the fault analysis engine and particle filtering-based prognosis algorithms [103].

2.2 Multi-robot Task Allocation

There is an abundance of multi-robot task allocation (MRTA) in the robotics literature [104–107]. Classical approaches have relied on mixed integer linear programming (MILP) techniques [108–111].

Zhang *et al.* [112] study the problem of automation in an assembly line with robots and human workers. They solve a MILP problem to compute the initial agent placement. The required tasks are then assigned and scheduled on these agents using hill-climbing and Tercio algorithms, respectively.

Scheduling of activities on transportation and routing networks using constraint programming (CP) approach has been reported heavily [113,114]. In general, task scheduling problems in multi-robot scenarios can also be formulated as a constraint satisfaction problem making it amenable to the methods of constraint programming.

Booth *et al.* [115] show that CP outperforms MILP for task allocation and scheduling problems relevant to retirement homes. Novas and Henning [116] use CP to automate manufacturing, traveling, and storage activities in flexible manufacturing systems (FMSs) while accounting for tools, machines, and robot constraints.

Evolutionary techniques have also been effectively utilized in solving MRTA and scheduling problems. Godinho Filho *et al.* [117] have surveyed the increasing application of genetic algorithms in scheduling activities in FMSs while [118] contains an application of particle swarm optimization algorithm to FMS scheduling problem.

Mousavi *et al.* [119] have employed a hybrid method combining simulated annealing and tabu search to solve supply chain scheduling formulated as a two-stage MILP problem.

Multi-robot scheduling problems share several similarities to multi-processor scheduling problems and hence processor scheduling algorithms are heavily applied [1,120]. Application of processor scheduling algorithms mainly involves the use of different heuristics to schedule robot coalitions on tasks greedily [121, 122].

Mission scenarios involving mobile robots must take into account the temporospatial nature of tasks which also have inter-task precedence constraints [123, 124]. Task allocation and scheduling algorithms also need to take into account the uncertainty associated with task execution durations and robot traveling times due to the robots' and tasks' characteristics and their interactions with the environment [125, 126].

Peters and Bertuccelli [127] developed a MILP framework to compute robust schedules for collaborative human-UAV missions. Gombolay *et al.* [128] present Tercio algorithm which solves a simplified MILP problem to generate agent-task allocations from which a near-optimal sequence satisfying temporal and spatial constraints is computed using dynamic priority heuristics.

Owing to inter-robot communication failures and imperfect situational awareness across all the robots engaged in the mission, decentralized approaches have become a popular method of handling these challenges. These mainly consist of auction/market-based methods or their variants. Liu and Shell [129] developed a market strategy for MRTA problems where prices are strategically controlled to guide the purchasing behavior of the bidders and achieved global optimality. Wicke *et al.* [130] proposed a specific MRTA algorithm inspired by bounty-hunting practices robust to loss of agents and other noises while accounting for non-exclusivity of task assignments. Otte *et al.* [131] proposed a generalized Prim allocation auction algorithm to study MRTA problem with lossy communication channels.

Another successful multi-assignment approach of combining auction and consensus methods results in the consensus-based bundle algorithm (CBBA) [132]. Zhao *et al.* [133] developed a

distributed algorithm to allocate tasks among heterogeneous unmanned vehicles for search and rescue missions by maintaining significance values of tasks across all robots which performs much better than the CBBA. Das *et al.* [134] developed a distributed, market-based MRTA algorithm called Consensus Based Parallel Auction and Execution (CBPAE) for dynamic allocation of tasks in health-care facilities where the next task is not assigned to robots till they have finished their current task. It was shown that CBPAE was much quicker than CBBA.

When the numbers of robots are more than the number of feasible tasks and multiple robots are allowed to work on a task simultaneously, a crucial factor in the MRTA problem is to form effective teams (or coalitions) among the robots and assign them to appropriate tasks [135–138]. This problem becomes even more complicated when heterogeneous robots are considered. Guerrero and Oliver [139] model the physical interference caused when employing a team of robots to finish a task to compute the task execution time and use auctions to determine the effective coalition sizes. Gunn and Anderson [140] study dynamic team formation when a robot gets lost or separated from the team in urban search and rescue missions. Cases with replacement robots and failures of leader robots are also analyzed.

Hybrid methods try to make use of different parts of these primary methods to improve computational performance and solution quality. Lim *et al.* [141] designed four hybrid evolutionary techniques by putting together differential evolution, artificial bee colony, genetic and particle swarm optimization algorithms in different combinations. They were applied to optimize layouts for multi-robot cellular manufacturing systems. Lu *et al.* [142] proposed a hybrid multi-objective backtracking search algorithm to solve permutation flow-shop scheduling problem by considering both make-span and energy consumption.

Heuristic schedulers are generally combined with MILP models to converge to solutions quickly [143,144]. A unified framework for combining MILP and CP models achieves faster convergence to high-quality schedules [145,146]. Evolutionary techniques can be incorporated into the scheduling procedure to obtain better results [147, 148].

In processor scheduling literature, problems dealing with precedence-constrained have been addressed, but those methods have not been explored at length in the robotics community. One of the work in robotics used in this dissertation for benchmarking purpose is [1], which assumes that the set of admissible coalitions is already available and does not consider precedence constraints. But one could also enhance features of the mission model, such that it allows fractional task completion, task interruption, and accounts for uncertainty.

Ou *et al.* [149] present an online task allocation method for multi-gantry work cell in which a machine requires the gantry system to supply it with a part from its upstream buffer and send the processed part to downstream buffer. The goal is to maximize throughput, so the authors define a metric called permanent production loss to track the system's performance. Machines must not be allowed to be idle, and production systems must be robust to disruption events. This is accounted for by the concept of opportunity windows.

Flushing *et al.* [111] solve a multi-robot task allocation scenario where a mobile ad hoc network (MANET) needs to be set up and maintained. This requires computing data routing policies and data transmission schedules. They design a mixed-integer linear program (MILP) that allows a trade-off between task completion and communication performance. However, the mixed-integer linear program solutions only consider the total delivery ratio for the transmitted data, and therefore, an extra LP-based refinement mixed-integer linear program stage is also added to include another important metric, viz. transmission delays.

Whitbrook *et al.* [150] developed a distributed performance impact (PI) algorithm to allocate time-critical tasks among multiple heterogeneous robots. This paper suggests a useful extension to the PI algorithm to speed it up and solve more complex problems. Benchmarking it against the consensus-based bundle algorithm and baseline PI algorithm shows that it outperforms both. The baseline PI algorithm runs the risk of not exploring the search space fully. To tackle this issue, the task selection procedure is chosen to be an appropriate combination of the PI selection procedure and either the ϵ-greedy action selection method or the Boltzmann Softmax action

selection method. Simulations were carried out for search and rescue missions which comprise of vehicles, helicopters and unmanned aerial vehicles that must reach the victims to provide the resources they are carrying. The number of tasks is double the number of these vehicles. Average time to finish the rescue mission was reduced by as up to 8% using these modifications.

Nam and Shell *et al.* [105] model resource contention among multiple robots during task allocation. There is interference happening when a team of robots shares the same resource to perform a task. The authors model the performance degradation in task execution when multiple robots contend for the same resource and physically interfere with each other. These robots may perform the same tasks in different manners. Based on how the robots end up deciding to execute each task, a convex or linear penalization function is added to the objective function to be minimized. This problem is solved in two phases. In the first phase of optimization, the integral constraints of the linear programming problem are relaxed, and fractional solutions are found using the interior point method. In the second phase, these fractional solutions are then rounded to integral solutions using the multi-choice Hungarian algorithm and multi-choice Murthy's ranking algorithm. Physical experiments, as well as simulations, were carried out where globally optimal solutions are found quickly.

Shiroma and Campos [151] tackle the problem of task allocation in robots capable of doing multiple tasks simultaneously and ensuring coordination among the formed teams. They call this algorithm CoMutaR, which stands for coalition formation based on multi-tasking robots. They use schema theory and represent tasks as a set of motor schemas. This allows them to decompose the tasks into smaller robot-independent tasks such that the information collected incrementally by the system may be used by all the robots freely to perform their remaining tasks. Resources cannot be freely shared among robots. Coalitions among robots are formed using a single-round auction algorithm. Simulation experiments are carried out using player/stage/gazebo platform in ROS. Loosely coupled tasks like area surveillance and transportation are considered as well as tightly coupled tasks like box pushing.

Turpin *et al.* [152] solve the problem of concurrently assigning goal locations to robots and then plan collision-free trajectories for each robot. It considers an obstacle-free environment, and collisions among robots are prevented using online avoidance algorithms. The authors first introduce a centralized algorithm which is tested out on a team of quadrotors. The minimum clearance between quadrotors was shown to be safe. The assignment was done to minimize the square of the minimum distance traveled using the Hungarian algorithm. Then collision-free trajectories were computed using the calculus of variations. Also, a decentralized version of the algorithm was designed to perform local replanning based on inter-robot communications, which resulted in sub-optimal but safe trajectories.

Nunes and Gini [153] design an auction algorithm to allocate tasks with time windows among cooperative robots. The time windows specify the time range during which the tasks must be executed, and the tasks are allowed to overlap. Each robot models its temporal constraints using a simple temporal network. This is the variation they introduced in the sequential single-item auction algorithm and call it the temporal-sequential single-item auction algorithm. Their algorithm minimizes mainly the mission makespan in conjunction with the total distance traveled. Their algorithm outperforms a greedy algorithm and the consensus-based bundle auction during simulation in the JADE multi-agent platform.

Maoudj *et al.* [154] solve the problem of allocating and scheduling tasks for robots such that the total mission completion time is minimized. The allocation of tasks among robots is done using a negotiation based approach similar to the contract net protocol. Thereafter, priority-based scheduling rules that respect task constraints are used to minimize the time spent being idle by robots. They demonstrated their methods using a pick-transport-place operation involving two manipulators and two mobile manipulators.

Kanakia *et al.* [155] model robotic firefighting and ant foraging tasks as global games where each agent's action depends on others as well as an underlying signal. Also, there is imperfect information and uncertainty associated with this scenario. Here, the authors show the application

of continuous threshold functions to allocate tasks among robots which leads to efficient coordination among the swarm members and show how this leads to Bayesian Nash equilibria without communication among the agents. Additionally, it is shown that this method works even if the agents do not have a behavioral model of each other.

Chopra and Egerstedt [156] consider several spatially distributed service requests with time instants when the tasks must be executed are also specified. Each such request requires a set of skills. Only the robots which have at least one skill in common with the skill set required are allowed to travel to the task location and execute it.

This problem is formulated and solved as a combinatorial optimization problem. Its solution is validated using MATLAB simulation of a robotic music wall where different robots traverse over a wall to play a song, and each robot can play different instruments.

Deng *et al.* [157] solve task allocation and path planning problem for multiple autonomous underwater vehicles (AUVs). They use a Mobile Ad Hoc Network (MANET) simulator for testing purpose. They adopt a location aided auction algorithm to assign tasks to robots and benchmark it against generic auctioning. Some autonomous underwater vehicles are good at searching vast areas of water while others are good at identifying the nature of the target.

For planning trajectories of multiple autonomous underwater vehicles, they adopt a multi-objective feedback-based dynamic system that generates an action determination grid map to guide the autonomous underwater vehicle motions. This map encodes the combined benefit at each cell due to multiple objective functions, and it is used in conjunction with a vehicle distribution grid map that stores the location of other vehicles. They simulate two different acoustic noise conditions for testing their model. One models autonomous underwater vehicles in shallow waters over a sandy bottom in calm seas while the other simulates the autonomous underwater vehicles operating in shallow water reef in rough seas. They achieve this by varying the power spectral density of the background noise and the Nakagami fading coefficient.

Mosteo *et al.* [158] consider a multi-robot routing problem with limited communication ability. The robots cannot break the network connectivity among robots via radio waves while executing their allocated tasks. Whenever a communication link appears to be breaking, a motion-control mechanism based on spring action is activated to ensure that radio communication is not lost. Four task allocation algorithms are proposed that optimize over the total distance traveled, mission makespan, or the average task completion time. They use greedy and auction algorithms, and another one based on the traveling salesman problem. Then they also code an algorithm that allocates tasks by imitating the clock sweeping motion in polar coordinates.

Kaddouh *et al.* [159] present a mission involving multiple unmanned aerial vehicles where vehicle teams are dynamically created and reconfigured to solve different parts of the mission. There are constraints associated with the different locations where the unmanned aerial vehicles are supposed to perform the tasks, and these tasks comprise time windows, refueling, or transportation services. Multiple requests are allowed to be combined into one task to reduce costs. The resource allocation problem is formulated as a mixed integer linear program (MILP). Their solutions outperformed those produced by greedy and bundle methods for effectively allocating resources and unmanned aerial vehicles for the tasks and reducing take-off and landing costs. There was a trade-off between faster mission completion and cheaper mission completion, which they try to optimize.

Banfi *et al.* [160] deal with a scenario involving a set of mobile robots with limited-range omnidirectional sensors tasked with monitoring other mobile targets. The robots are modeled to have Bayesian beliefs regarding their targets and the environment which they exploit to monitor all the targets more fairly. A mixed integer linear program is formulated to maximize the amount of average monitoring done for the targets as well as to minimize the standard deviation of the detection rate of the targets. The latter optimization is done to ensure fairness in the sense that lower uncertainty means the targets were monitored equally. The mixed integer linear program is used to monitor for a specific interval of time after which robot beliefs are updated, and the mixed

integer linear program is then solved again. The simulation experiments varied replanning times and communication ranges to study their effects. Both centralized and distributed architectures were also implemented. The distributed architecture proved to be better overall due to its low computational cost. Real robot experiments were shown to report a deviation from the simulations due to collision avoidance routines and robots getting stuck in congested and narrow channels.

2.3 Decomposition of Exploration Tasks

A lot of work has been done to efficiently explore unknown regions based on different constraints using multiple robots. Also, in the general context of multi-robot tasks, there is a heavy emphasis on decomposing the main tasks effectively among the robots. Task decomposition thus arises naturally into a host of robotics applications, including industrial manufacturing.

Tham and Prager [161] use task decomposition methods to develop modular approaches to learning manipulator tasks through their sub-tasks using reinforcement learning. Since the state space and action space are both vast, they used a special type of neural network called a cerebellar model arithmetic computer (CMAC) based on the mammalian cerebellum.

Wu *et al.* [162] studied the specific example of mold manufacturing operation as representative of multi-site product manufacturing. The operation is decomposed into tasks that can be managed individually within a network of manufacturing firms. A graph algorithm is presented based on a bidding process to assign the tasks to different contenders.

Gindy and Ratchev [163] represented process capabilities of a facility using resource elements. Conventionally, sequences in which components visited different machines in a facility was used for this purpose. Rather than that, authors propose that machining operations be used to classify different components and then based on these classified component groups, machines are chosen. These resource elements are used to decompose a manufacturing facility into machining cells which

can operate autonomously. It avoids machine repetition, thus avoiding bottlenecks and is more compact.

Whiteson *et al.* [164] study the soccer drill called *keepaway* which is a complex multi-agent task. In this drill, the players assigned as keepers aim to learn good performances for the tasks. But since the task is so complex, the idea of decomposing the task based on decision trees is used. Then the learning can be done over the simpler subtasks and learned models could be combined to master the original complex task. The authors use three learning methods with varying degree of human assistance so that somehow humans are not constraining the search space of good solutions. They show that when the player assigned as a taker in the *keepaway* game is allowed to run after the ball with very high speeds, only the decomposition-enabled techniques can learn the winning strategies.

Stone and Veloso [165] design a distributed autonomous system for robotic soccer games allowing for periodic team synchronization. Tasks are decomposed into flexible roles. Agents are allowed to create formations. Each agent could change its roles within formations, thus allowing flexible teamwork. It was shown that the flexible teams outperformed rigid teams. Formation switching was also allowed, and it helped to maximize the number of goals scored.

Su and Zheng [166] worked on coordinating the motion of a legged mobile platform and a robot arm mounted on top of it, which is a challenging problem. They developed three different task decomposition approaches for reaching a given goal end-effector pose. The task of orienting the robot is decomposed to orienting the two robots separately in a defined sequence, thus making the problem easier to solve. The first two approaches do motion using both the platform and the arm. The third approach exclusively uses the platform. As expected, the approach that uses the arm more is better because arm manipulation consumes less energy than platform motion.

Tan *et al.* [167] facilitated a cable assembly operation using human-robot collaboration. The hierarchical task analysis technique is used to decompose the entire assembly operations into tasks assigned to only the human worker or both robots and human worker. When a task is assigned

to both of them, then the sub-tasks are partitioned between them and sequenced properly. This reduces the mental workload of the human worker and increases worker efficiency by reducing faults during assembly and reducing the time taken for assembly.

Huckaby and Christensen [168] use a taxonomy based on the decomposition of complex airplane assembly tasks into primitive robot skills. But this is easily generalizable to other manufacturing tasks. Perception tasks which will be done by vision systems are also incorporated in the taxonomy.

The problem of decomposing exploration tasks among multiple USVs is closely linked to the problem of area partitioning. Hert *et al.* [169] define boundary points that should lie in the partitioned polygons. Bast *et al.* [170] compute partitions of equal areas by merging and dividing convex pieces obtained originally via convex decomposition of the polygon such as in [171].

Bullo *et al.* [172] derive spatially distributed algorithms to compute a convex and equitable partition of a convex polygon based on power diagram with additional features involving the use of Voronoi diagrams. These works furnish mathematically robust solutions to the area partitioning problem.

There is an abundance of literature that focuses on the area partitioning problem in physical robot settings.

Chand and Carnegie [173] developed a hierarchy to utilize the available robots maximally. The robots are divided into three classes: manager robots have high processing and communication abilities. The planner robots have medium range communication abilities but low sensing and actuation abilities. The explorer robots have low communication abilities but high sensing and actuation abilities. Each robot gets scores for the task it performs, and when these task scores are used as feedback, it leads to significant improvement in some cases.

Ziparo *et al.* [174] consider scenarios where large teams of robots are coordinated for exploration problem using RFID tags. The RFID information helps to avoid collisions during local region exploration. A global monitoring process based on sequential planning is used to start

explorations at different regions to increase the explored area. This outperforms task assignment techniques because it avoids conflicts.

Visser and Slamet [175] study frontier-based exploration in office-like space. The USARSim framework is used to simulate the office space, obstacles, and robots in rescue scenarios and frontiers are assigned to robots with maximum utility based on information gain and probability of maintaining communication links. Planning is done by estimating the probability of communication success significantly longer into the future motion. And it is shown that when robots are allowed to share their maps, they avoid traveling the explored corridors doubly.

Matignon *et al.* [176] attempt to solve CAROTTE challenge where an unknown environment has to be explored and objects known *a priori* to the robots have to be identified in the area. For some of these objects, robots may also have extra tasks associated with them. This problem is formulated as a decentralized Markov Decision Process (MDP), but since it is challenging to solve, the authors instead consider using a set of individual MDPs. A distributed value function is used to take the interaction among robots into account and maximize the area coverage of the team with minimum interactions.

Yuan *et al.* [177] perform cooperative multi-robot exploration by expanding individual frontiers and selecting next target destinations to maximize information gain while minimizing travel cost and satisfying communication-limit constraints. They benchmarked their technique against the approach that automatically selects the nearest frontier cell. Their approach finishes faster and explores more area.

Low *et al.* [178] use robotic simulator webots to simulate a team of 4 robots getting assigned a region where mineral data must be collected, which amounts to exploring a wide area which has only a few mineral hotspots. An adaptive sampling based exploration algorithm is designed, which leads to lower energy consumption, lower mission time, and higher hotspot sampling. Adaptive samplers use estimators, which can be highly biased by data sampled at hotspots. So they use Rao-Blackewellized estimators to handle the bias leading to higher chances of uncovering less likely

hotspots. Low *et al.* [179] solve the problem of performing both wide area coverage and hotspot sampling together. They use non-myopic and adaptive strategy to choose the next exploration locations based on data already observed. The new locations are sampled for exploration to minimize spatial mapping uncertainty. Their approach outperforms greedy and non-adaptive strategies.

Ahmadi and Stone [180] deployed a team of robots to continuously monitor an area based on the changing importance of different locations. Simulation experiments that were carried out show how area gets partitioned with respect to conventional static partitioning algorithms. Their method also accommodates the addition or deletion of robots. They consider robots at different speeds also. Due to the changing importance of different locations, they are visited with non-uniform frequencies.

Zhao *et al.* [181] proposed a multi-robot exploration approach that uses a market-based approach for assigning frontiers to robots. Potential fields are used to coordinate the robots using virtual forces exerted on them. To avoid local minima, the potential field is not always kept active.

Basilico and Amigoni [182] consider an unexplored region with unknown targets inside this region for which the paper shows the efficiency of using Multi-Criteria Decision-Making techniques. The multiple criteria considered for frontier selection are information gain, the probability of communication link working, and traveling cost. To combine their scores into one global utility function, they make use of Choquet integrals computed using Kappalab R package. Their algorithm was benchmarked against other algorithms and shown to perform better, especially when diverse decision choices were available with sharply varying rewards.

Solanas and Garcia [183] partition the unexplored area periodically using a K-means clustering algorithm and then assign the partitions to robots closest to their centroids. They have benchmarked this method against Burgard's multi-robot exploration work. This algorithm focuses on spreading out robots to cover the area and performs better than Burgard's method. But it is not significantly more efficient than Burgard's method.

Wurm *et al.* [184] use Voronoi graphs to partition indoor environments into segments. Then the cost of sending each robot to the frontier targets of each segment is computed, and the Hungarian algorithm is used to assign robots to their optimal segments. Simulation experiments show that it is faster than frontier-based coordination. Real experiments using two identical Pioneer II robots in office-like environments were done to show that the environment was explored completely and without any interference between robots.

Pei *et al.* [185] consider robots exploring an environment such that connectivity is maintained and bandwidth is available to transmit video/audio feed from any explored region at any time back to the central command. They apply the set cover problem, Steiner tree problem, and the linear bottleneck assignment problem to solve such constrained exploration tasks. They compare their methods against two communication-limited exploration techniques. With respect to exploration time, they outperform the other two overall. Their method also guarantees better connectivity.

Simmons *et al.* [186] work on frontier-based exploration via distributed bidding among robots. But the system is not fully distributed. Higher efficiency can be achieved using a central planner that synthesizes the local data accrued by all robots. The frontiers are expanded based on information gain. A team of two Pioneer AT robots and an Urbie robot was used to test their algorithm's robustness and was found to reduce exploration time and increase mapping accuracy.

Doniec *et al.* [187] formulated frontier-based exploration problem by multiple robots as a constraints satisfaction problem and solved it by using a distributed and asynchronous algorithm. Constraints are placed so that communication links do not break and robots explore the unknown area more by not overlapping their sensing regions. Simulations were carried out in NetLogo to observe the algorithmic performance with multiple robots while allowing for obstacles and different communication and sensor ranges.

Sheng *et al.* [188] augmented a distributed bidding algorithm with two measures to accommodate the limited-range communication for efficient, coordinated frontier-based exploration by multiple robots. These features handle the limited communication ranges of the robots. The

robots explore areas such that it leads to maximum information gain in minimum time. A nearness measure is introduced so that the robots tend to stay close together within appropriate limits. An important consideration when two robots meet is how do they exchange as little of their knowledge as possible to synchronize their maps.

Vazquez and Malcolm [189] develop a behavior-based approach to explore an unknown area with communication limits. The behaviors include achieving communication connections, maintaining them, or avoiding collisions. This method is benchmarked against Yamauchi's seminal multi-robot exploration paper. Simulations were carried out for different environment sizes, and different communication limits and their method was found to be faster than Yamauchi's method.

Elmaliach *et al.* [190] study the area patrolling problem in closed areas where constraints are laid down on the frequency with which the target locations are visited, and an algorithm based on finding Hamiltonian cycle is introduced. Then they also introduce deadline constraints for handling contingency events and design algorithms to recover from such scenarios quickly.

Howard *et al.* [191] deployed mobile sensing robots in a stationary configuration to maximize the area covered. The authors use potential field-based methods to achieve the static sensor network configuration in the environment. Simulation experiments study the time to deploy a network as well as the node spacing that is generated. Velocity and acceleration limits are added to the potential field dynamics model to prevent oscillations during convergence. It does not, however, prevent the system from reaching static equilibrium.

Batalin and Sukhatme [192] seek to achieve good coverage of an unknown environment. To achieve this, they disperse the robots so that their coverage regions do not overlap. The informative approach is to form local coalitions among robots within each other's range. Then the information is exchanged, and directions of motions decided that spread out the sensor coverage to the maximum. Their molecular approach, on the other hand, assigns a direction to the robots

away from others without any local information exchange or deliberation. Both approaches outperform the basic approach based only on robots maximizing their covered area. However, they find the molecular approach to perform better than the informative one but only very slightly.

Marjovi *et al.* [193] consider a search and rescue mission in a big warehouse that caught fire and is filled with smoke. Khepera III robots with innovative smoke sensors are deployed in the warehouse to detect the fire sources while exploring the entire floor space. They employ frontier-based exploration technique such that the time taken to finish exploring the warehouse is minimized. The frontier is assigned to the robots based on their distances. They benchmark their method against the optimal solution to cover the region if it was known entirely.

Senthilkumar and Bharadwaj [194] study the multi-robot exploration of an unknown environment using an online ant-inspired robot dynamics to completely cover the area based on the spanning tree coverage method. They also handle partially occupied cells and allow for narrow open spaces to be shared among multiple robots. Their method is benchmarked against their previous algorithm in which spanning trees are not allowed to intersect in more open spaces so that collisions are avoided. Their new algorithm proves to be faster.

Zlot *et al.* [195] present their work on frontier-based exploration trying to maximize information gain. Their method is robust to communication failures and can handle dynamic addition or deletion of robots. The frontier is expanded based on greedy, random, or quadtree-based strategy. Physical experiments were carried out with a team of ActivMedia PioneerII-DX robots. They compare different strategies based on area covered per unit distance traveled and found exploration efficiency to improve by a factor of 3.4 for a four-robot team.

De Hoog *et al.* [196] consider robots that can be in two roles: explorers or relays. The explorers cover the complete area and return periodically to rendezvous points to pass on the data to relays. The relays then take the information to a central command center. They compare their algorithmic performance with frontier-based methods. In terms of area covered, non-limited

frontier methods perform better. In terms of responsiveness to command centers, communication-limited frontier methods perform better than role-based. But the role-based method provides a trade-off between complete area coverage and maintaining tight communication links between the robots and the command center.

De Hoog *et al.* [197] work with dynamic environments where paths can become suddenly blocked, and communication links may break suddenly. The rendezvous points are selected for explorer and relay robots by first using Hilditch's Algorithm for Skeletonization to compute a small set of points in the robot's chosen frontier and then selecting the one with best communication range and traversal costs. The frontier methods were faster than the rendezvous methods, but their command center performance was poorer. The authors' advanced rendezvous point selection method was shown to outperform the simple method in office-like, open, and cluttered environments.

Ouyang *et al.* [198] design a decentralized multi-robot active sensing algorithm and apply it to real-world scenarios. The task for multiple robots is to gather information at different locations to predict the unknown, non-stationary phenomenon and estimate the spatial correlation structure of the phenomenon. The phenomena studied here include the plankton density in oceanic regions and traffic speed data on urban road networks. These phenomena are modeled using a Dirichlet process mixture of Gaussian processes.

Hassan *et al.* [199] use Voronoi partitioning while optimizing multiple objectives and verifying the resulting multi-robot cleaning using torque heatmaps. Jager *et al.* [200] develop a dynamic, decentralized area partitioning method while accounting for limited inter-robot communication.

Carlsson [201] solves a stochastic vehicle routing problem by dividing a given region using probabilistic analysis of traveling salesman problem (TSP) tours such that all vehicles have a balanced workload. Higuera *et al.* [202] use subgradient search methods to divide an area among multiple robots based on their speed profiles. Agarwal *et al.* [203] compute rectangle partitions

of holed rectilinear regions and distribute them among multiple unmanned reconnaissance aerial vehicles (URAVs) based on their heterogeneous capabilities.

Yamauchi *et al.* [204] established the use of frontiers to guide maximal information gain based exploration. In this work, the idea was to detect the boundaries between explored and unexplored regions as frontier cells and greedily explore the closest frontier regions. Sheng *et al.* [205] consider limited-range communication and develop a distributed bidding algorithm where every robot bids for its claim to frontier cells and tries to remain close to each other to minimize communication volume. Burgard *et al.* [206] use multiple robots to explore an environment in minimum time by assigning utility values to frontier cells available to robots such that the robots explore varied locations. Ma *et al.* [207] use a genetic algorithm (GA) to compute the optimal assignment of the frontier cells to multiple robots based on the utility of these cells calculated using Burgard's technique.

Wang *et al.* [208] develop a frontier-based grid exploration approach where they use the PSO method to prevent multiple robots heading towards the same cell. Zhou *et al.* [209] use the PSO method to guide the robots to explore remote frontiers, thus minimizing the exploration time. Jung *et al.* [210] develop a coverage method based on dynamics of autonomous underwater vehicles (AUVs) and sea current disturbances. While each AUV in a team uses the developed method to cover regions, they avoid colliding with each other leading to entire area coverage.

Chapter 3

Modelling and Verification of Contingency Resolution Strategies for Multi-robot Missions Using Temporal Logic

In this chapter, a framework to model multi-robot missions using formal methods is presented and desired properties for such systems are verified.

3.1 Introduction

Multi-robot missions may encounter unexpected situations that can halt or alter mission tasks. For example, consider the case of a large ship that has to be escorted by robotic boats and drones. Each robot has been assigned its nominal task schedules. However, the ship's path may experience a blockage; robots' batteries may die; communication links may stop working; a civilian vessel may be spotted in the protected region around the ship. These situations will interrupt the regularly scheduled tasks.

Due to the presence of such contingencies, contingency resolution strategies must be designed to contain and minimize the adverse effects of contingencies without drastically affecting the primary mission objective such as minimizing the expected time to produce parts or reach a location. In the civilian vessel escorting example, one strategy could be to first direct a drone to assess the situation and, if the intrusion is confirmed to jeopardize the mission significantly, then

a robotic boat must approach the intruder and warn it to move away. However, there may be numerous such strategies having different impacts on the mission. Some may lower operational costs while others lower mission makespan. Appropriate methods are needed to select the most promising strategies.

Since contingencies occur unexpectedly or even rarely, one cannot test the resolution strategies as rigorously as one can test the nominal mission execution strategies. One could, in principle, run multiple high-fidelity simulations to decide how effective these strategies are and which ones are better. The problem with running such simulations is that they are not easily scalable for larger problem sizes because they try to encompass several minute details and run on professional simulation software packages which are resource-intensive and time-consuming. Also, manually coding different possible mission executions and verifying them through computer simulations is very time-consuming and maybe unrealistic.

This chapter focuses on leveraging the use of model checkers to the domain of multi-robot missions to assess the adequacy of contingency resolution strategies to minimize the adverse effects of contingencies on the mission execution. In robotics research, the use of formal methods has become quite popular to analyze mission execution protocols and contingency resolution strategies. Formal methods provide a complete framework for handling dynamic and uncertain systems in a mathematically rigorous, and logically consistent manner. However, such model checkers were traditionally designed for other applications such as embedded systems and computer hardware design. So when modeling multi-robot systems, generally some simplifications have to be made. The challenge is that one must not oversimplify the mission model so much so that running high-fidelity simulations would have proved to be the better choice. However, it must be ensured that the mission model adheres to the semantics implemented by the model checkers.

This chapter presents a framework for the nominal modeling of the mission using formal methods, the design of contingency resolution strategies for model checking and evaluation of these strategies with respect to desired mission objectives.

3.2 Problem Statement

The objective is to develop a framework to apply model checking tools to incorporate contingency models in nominal mission models and evaluate the effectiveness of contingency resolution strategies. Given a mission model and alternative strategies to handle contingencies, one must evaluate the performance of these strategies using a model checking framework. In the next section, the various components required for designing such a framework are described and then this framework is applied to two case studies in the next two chapters.

3.3 Overview of Approach

3.3.1 Nominal Mission Modeling for Deterministic Transitions

Mission modeling and verification methods require a nominal mission description as input. One must generate a formal transition model for the system based on such a description. Model checking refers to the automatic and exhaustive verification of whether a given specification is satisfied by the system. Therefore, both the mission model and specifications are needed for model

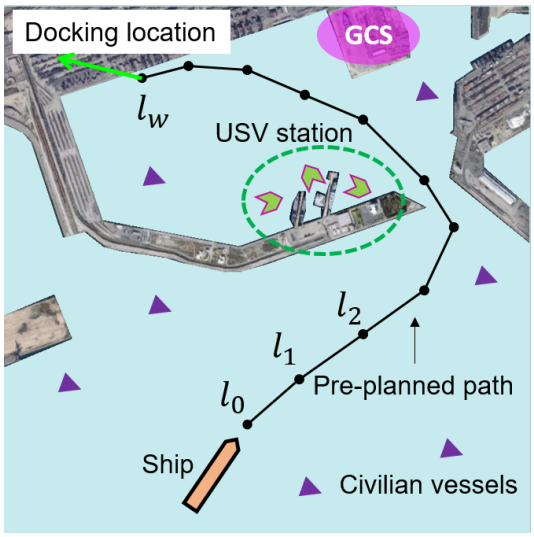


Figure 3.1: Characterization of multi-USV escorting mission scenario (Not drawn to scale)

checking. A mission example representative of multi-robot surveillance and patrolling missions is used for illustration purpose. Suppose there is a high-value target that has to be guided by a team of mobile robots through waypoints to reach the desired location ultimately. An example is shown in Figure 3.1, where a ship has just entered a port and must be escorted by a team of autonomous unmanned surface vessels to its designated location. However, the modeling presented here is independent of this specific mission and applies generally to multi-robot patrolling missions involving any mobile robot.

Based on such a mission description, the nominal tasks comprising the mission are identified. Most missions consist of spatially distributed tasks that may be performed concurrently by multiple agents. Agents may be human workers or robots. One example is the task of escorting the high-value target from one waypoint to the next performed by a team of mobile robots. The pre-conditions and post-conditions are identified for each task. If the pre-conditions for a specific task are satisfied then an agent, available and well-suited for carrying out the task, executes it which results in a change of the mission scenario as described by the task's post-conditions.

Then a mission execution plan is chosen that is well-suited for the mission under consideration. It is a sequence in which the available tasks must be executed to meet the mission objective while adhering to pre-conditions and post-conditions of all tasks. For example, the mobile robots will meet up with the high-value target at a safe location, and then the convoy will proceed through each successive waypoint until it reaches its final destination.

Missions are generally of two types, deterministic and probabilistic. If a mission only allows an agent to transition from a current state to a unique new state when certain conditions are met at the current state, then the mission is modeled as a deterministic process. If there is at least one agent state where one or more probabilistic transition rules are applicable, then the mission is modeled as a probabilistic process.

To perform nominal modeling and verification for deterministic missions, proceed as follows:-

- Identify the set of discrete states (let $v_{a_i}^j$ denote the j-th variable for the i-th agent a_i) available for each agent along with their domains and initial values. For example, the location of the high-value target may be a mission variable whose domain is given by the designated waypoints. The initial state for each such variable must also be characterized. For example, mobile robots begin their operations initially from some designated resting station.

- Identify the environment variables (let v_e^j denote the j-th environment variable) that keep track of relevant factors in mission scenario: For example, some variables may keep track of whether the high-value target is encountering a dangerous situation. It may be assumed that these contingency variables are maintained and updated by a ground control station (GCS). The GCS receives sensory data that helps it to detect such unexpected contingencies and communicate their occurrence to mobile robots.

- Encode transition rules (let t^j denote the j-th transition rule) which specify constraints for agents to transition from one state to another in terms of tasks' pre-conditions and post-conditions which in turn are specified in terms of agents' states and environment variables. For example, a transition rule specifying the motion from one waypoint to another represents a task for the mobile robot to execute. Such transition rules may be encoded with ease in model checkers like NuSMV as: *if pre-conditions, then post-conditions.*

Transition systems are adopted for modeling the missions. These are computer science tools used to describe transition rules for discrete-state systems. Transition systems are represented using Kripke structures. According to Kripke semantics, a Kripke frame is represented by $\langle S, R \rangle$ where S refers to the states that may be taken by the system. For missions, states are of two types, one representing the states which the agents assigned to the mission may reach, $v_{a_i}^j$, and another representing the states that environment may reach, v_e^j. $R \subset S \times S$ represents the accessibility relations. The accessibility relations indicate which states can system access or transition to from

a given state. R comprises of transition relations t^j which indicate which states a system may access or transition to from a given state.

Let AP denote the set of atomic propositions relevant for the system under study. When atomic propositions that can be verified for the system are provided, the Kripke model for the system gets defined. Kripke model is denoted by $\langle S, R, L\rangle$ where $L : S \times 2^{AP} \rightarrow \{\top, \bot\}$ is a labelling function that returns true or false for a given state and atomic proposition(s) over it. If multiple atomic propositions are queried, this function can only return true if all atomic propositions are individually satisfied by the queried state. Finally if the initial states of the system denoted by $I = [v^j_{a_i}(0), v^j_{\dot{e}}(0)]$ are identified, the Kripke structure is given by $\langle S, I, R, L\rangle$.

3.3.2 Nominal Mission Modeling for Probabilistic Transitions

Transition rules between states of a system are not always deterministic. For stochastic systems, probabilistic transition rules are specified [211]. Stochastic systems with discrete states are generally modeled as Markov Decision Processes (MDPs). An MDP is a discrete-time stochastic framework denoted by a 4-tuple $\mathcal{M} = \langle S, I, A_s, P_a\rangle$. Here S represents a finite set of states the system may transition to. I denotes the initial states the system begins from. A_s lays down the finite set of actions that are applicable at state $s \in S$. $P_a(s, s')$ represents the probability that the system will transition to state s' in the next step if it is currently in state s and adopts action a.

A finite path through an MDP is a sequence of alternating states and actions that may be denoted as follows:

$$\rho = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots \tag{3.1}$$

PRISM allows one to define reward structures that help in computing expected total rewards for the system. One can either define state-based rewards $r : S \rightarrow \mathbb{Q}$ or transition-based rewards $r : S \times A \rightarrow \mathbb{Q}$. Then the total reward for path ρ consisting of N state transitions is given by

$rew(r, \rho) \stackrel{\text{def}}{=} \sum_{i=0}^{N} r(s_i, a_i)$. The expected total reward is computed over all finite paths that meet some user-defined conditions.

Discrete-time Markov chains (DTMCs) are also feasible modeling tools in several cases. MDPs allow for states in which multiple transition rules are applicable and resolve such conflicts based on the principles of nondeterminism. However, DTMCs differ from MDPs in that they allow only one probabilistic transition rule for every possible state of the system.

NuSMV allows some simple handling of probabilistic behaviors, but it is limited in scope. The first case study discussed in next chapter is mostly deterministic, but it includes a few simple probabilistic transition rules. For example, if an agent may execute two tasks from its state x, one with 2/3 probability where it moves to state y and another with 1/3 probability where it moves to state z. It would appear in NuSMV as follows.

```
next(agent_state) :=
    case
        agent_state = x : {y, y, z};
    esac;
```

NuSMV excepts the user to pick each mission variable in a sequence. Then for all possible cases that the variable may encounter, one must specify the corresponding change in its state. NuSMV also requires the user to repeat the variable names to match their probabilities. Coding such transition rules in NuSMV may be inconvenient for complex stochastic systems. The probabilistic model checkers are more convenient and effective. PRISM is a popular tool for this purpose. Unlike the exhaustive enumeration of NuSMV, PRISM requires the user to encode different conditions that the user considers relevant for the mission and then specify how all the mission variables will change if those conditions hold. For example, if an agent may execute two tasks from its state x, one with 75% probability where it moves to state y and another with 25% probability where it moves to state z. It would appear in PRISM as follows.

```
[] agent_state = x -> 0.75 : y + 0.25 : z;
```

Real-world systems are often simulated by appropriately synchronizing these transition rules in PRISM. The extensive use of this tool for probabilistic model checking is demonstrated with the help of the second case study.

3.3.3 Modeling of Contingency Resolution Strategies

These are the steps to incorporate contingency information in mission execution. *Firstly, design the contingency resolution strategies which are invoked when a contingency is reported during mission execution.* However, these strategies do not merely assign an agent to execute a specific action to resolve a contingency. These strategies may require multiple actions to be performed in a sequence. So for different cases, different sets of actions may have to be invoked. Below, different examples of contingencies are given that may be encountered during multi-robot missions such as those described earlier.

- Suppose robots are required to maintain contact with GCS at all time. This may be necessary for robotic operations so that human remote operators may take control of the robots if they break down. Since the ground station must always maintain a communication link with the robots, a variable called "comm_link[i]" is introduced. This Boolean variable denotes whether the communication link between GCS and i-th mobile robot is working or not. Then a strategy is laid down for resolving communication failures. If the communication link breaks down, then the robot is supposed to go back to its station for repair purpose and avoid any unsafe situation in the open waters. When the link between the station and the robot is restored, it rejoins the rest of the convoy.

- It is common to find missions interrupted because suddenly some event took place which halts the mission. The only way to resume the mission is for the agents to address the event first. For example, when the high-value target is moving through a crowded region

and is being assisted by multiple mobile robots, it is possible for another vehicle to come dangerously close to the protected asset. To prevent collisions, the following strategy is designed. If another vehicle is reported in the protected zone, then a robot is instructed to first go to the region to confirm whether the vehicle is still there. If the vehicle's presence is confirmed, then the robot must approach and warn the intruder to leave that region and return to resume its mission only after the intruder has acknowledged the message.

- In busy seaports, a civilian vessel may come dangerously close to a ship. Following the previous example, one can consider a slight modification to the contingency resolution strategy. Suppose each unmanned surface vehicle (USV) had an unmanned aerial vehicle (UAV) mounted on top of it. The UAV is used to confirm the presence of the vessel because it may roam easily in the port region compared to the USV. If UAV confirms the civilian vessel, then the USV will approach and warn the intruder.

- The path ahead of the convoy may be blocked. Static obstruction, say debris may be found on the path which when reported, a robot must go to that location to confirm the obstruction and its nature. Upon confirmation, it must attempt clearing the path, and until the path is cleared, the mission is paused. Only when the path is cleared, the mission resumes. One can assume that if there is large-sized debris that the robot will be unable to clear, then it will message the central mission controller to send human-operated tugboats to clear the debris. Again, for the ship example, one may use the UAVs for contingency confirmation and the USVs for contingency resolution.

Secondly, encode the contingency resolution strategies using appropriate transition rules. Model checking is crucial in detecting whether humans have introduced any inconsistencies when coding such transition rules. For example, if the communication link for i-th robot breaks, then it should head towards the base station coded as `!comm_link[i]: base_station`.

If multiple contingencies coincide, then an essential aspect of contingency planning is to decide how the respective strategies should synchronize among themselves. Synchronizing contingency actions in different ways may significantly impact mission performance. This is clearly demonstrated in the second case study.

3.3.4 Model Checking and Verification

3.3.4.1 Deterministic Transition Systems

The system properties to be verified by the user are encoded with the help of temporal logic. After the mission model and contingency resolution strategies are coded in the programming language of the model checking software and subsequently compiled, the user assesses those system properties using available model checking tools. For deterministic systems, one must verify whether the system eventually reaches a desirable state. There may be multiple such desirable states.

For deterministic transitions, one may use LTL formulas to specify goal states. If states of a system are considered as nodes then the transition rules of the system define a graph-like structure. CTL and LTL operators help in reasoning in this domain. While CTL operators explicitly help in reasoning about branching paths, LTL operators help in reasoning about temporal nature of system execution in a single path. CTL operator **A** is used to reason about all paths that a system might take whereas operator **E** tries to reason whether there exists a path reaching some desired state. The LTL operators assume a set of atomic propositions A_P are defined. LTL builds up on the logical connectives (or : $\vee$, and : $\wedge$, negation : $\neg$) by proposing new temporal modal operators: **X** (neXt), **U** (Until), **F** (Finally), **G** (Globally). If $p_1, p_2 \in A_P$ are atomic propositions then $\mathbf{X}p_1$ indicates that p_1 holds true in the next time step, p_1 **U** p_2 indicates that p_1 is true at least until p_2 becomes true, $\mathbf{F}p_1$ indicates that eventually p_1 holds true and $\mathbf{G}p_1$ indicates that p_1 always holds true.

One may also verify whether risky or inconsistent behaviors have been injected into mission execution due to human error. For example, in vehicle routing problems, one generally wishes to verify whether the vehicle reaches a desirable state and stays there. For this, the following type of LTL specification may be used: **FG** ϕ, where ϕ is a logical proposition about the mission. ϕ may be composed of conjunctions and disjunctions of atomic propositions. The LTL formula states that the logical condition ϕ will *eventually* be *always* satisfied. The **G** operator is used to ask the model checker to verify whether the vehicle achieves equilibrium at the desired location once it reaches that location safely or it has a tendency to wander off to another location due to the way mission specifications were encoded.

Before verifying such properties, one must explicitly ask the model checker to discard specific paths of model execution. This is done by making a fairness assumption which restricts NuSMV only to consider those mission executions which are generally expected in real-world. When verifying the correctness of a given mission protocol, one wants to discard trivial counterexamples, say those produced due to livelock situations. This is because the contingencies may put the system in a never-ending loop. For example, suppose the convoy is interrupted by a contingency event, and the mobile robots are assigned to resolve it. Once the contingency has been resolved, and the convoy is ready to proceed with the mission again, another contingency event occurs. This may keep happening again and again. This is a livelock situation.

3.3.4.2 Probabilistic Transition Systems

Deterministic missions require exhaustive enumeration of states possibly reached by the system to verify LTL properties. But this may not be computationally possible due to physical memory limits. So it may be computationally more efficient to model missions as probabilistic processes. Many missions happen to be probabilistic. But then one is only able to provide probabilistic guarantees by computing the probability with which a system will reach a specific state in a particular duration or expected time elapsed before system encounters a goal state. For probabilistic

systems, probabilistic operator **P** is added to the previous LTL operators, and this temporal logic is referred to as Probabilistic Computation Tree Logic (PCTL). These PCTL formulas are most commonly used to query about probabilistic reachability conditions which may refer to querying the probability that the system eventually satisfies a set of atomic propositions in a conjunctive or disjunctive way or even a mixture of the two. PCTL verification results produced by PRISM are used to evaluate how the different contingency resolution strategies perform and how mission performance varies by changing mission settings.

For probabilistic model checking, numerical methods like value iteration are used. Numerical techniques create symbolic representations using binary decision diagrams (BDDs) and multiterminal BDDs (MTBDDs) and operate iteratively in conjunction with a sequential stopping rule. Systems exhibiting nondeterminism are modeled as MDPs, and one can only query the minimum or maximum values of probabilities and expected rewards. One is generally interested in querying about probabilistic reachability conditions such as: **P**min=? [$F^{\leq t}$ "cond"], asking what is the minimum probability that within first t state transitions the system satisfies the logical condition "cond" which may be a union or intersection of atomic propositions.

If a reward structure ${}^{T}\mathbf{R}$ assigns real-world time duration during system's state transitions then PCTL expression like ${}^{T}\mathbf{R}$min=? [**F** "cond"] queries the expected time in which system satisfies the specified condition. PRISM returns a finite output for this query if and only if **P***max*$\geq$ 1 [**F** "cond"] which means that the condition "cond" is eventually satisfied by the system *almost surely.* However, **P**max$\geq$ 1 [**F** "cond"] maybe true even though a stronger condition like **AF** "cond" fails. As of PRISM 4.4, one can only query the actual value of **P***max* because bound checking is not allowed. The bound checking expression was only used for clarity. MDPs are only solved using numerical methods which place a significant restriction on the problem sizes that may be investigated.

Missions are modeled as DTMCs so as to investigate problems of larger sizes because DTMCs are computationally more efficient than MDPs. In addition to numerical methods, even statistical

methods may be applied to DTMCs for performing model checking. They involve Monte Carlo simulation and sampling for hypothesis testing and are much faster. Statistical methods can handle larger problem sizes as compared to the numerical methods.

3.4 Task Network Generation

Temporal logic-based modeling is also used to generate task precedence graphs for multi-robot missions. For example, missions involving multiple mobile robots usually comprise precedence-constrained survey-like or travel-to-location tasks [212]. Often it is assumed that the mission supervisor provides such precedence graphs. However, temporal logic may be used to extract such precedence graphs from mission specifications itself. These task-precedence graphs are then provided as input to task allocation and scheduling algorithm.

Two types of LTL constraints generally suffice for all cases. Firstly, a system may have Sequencing with Avoidance and without Repetition (SAR) constraints. Suppose task τ_2 cannot be started until τ_1 has been finished and both tasks have to be done only once throughout the mission. The LTL expression for this SAR constraint is:

$$\mathbf{F}\,(\tau_1 \wedge \mathbf{F}\tau_2) \wedge \mathbf{G}(\tau_1 \implies \mathbf{XG}\,\neg\tau_1) \wedge \mathbf{G}(\tau_2 \implies \mathbf{XG}\,\neg\tau_2) \tag{3.2}$$

Each such SAR constraint creates a directed edge originating at a node representing task τ_1 and ending in node representing task τ_2.

Secondly, the system may have a simple reachability constraint like $\mathbf{F}\tau_1$, thus adding a node to the task network. Using these two types of LTL constraints that may be extracted from the mission description, one can construct task precedence graphs which serve as an input to the previously developed task allocation method. The resulting task network will be a directed acyclic graph with the possibility of free nodes. So it may not be a connected graph even in the undirected sense.

3.5 Summary

This chapter presents a framework that can model nominal missions with deterministic and probabilistic transitions. An approach is also provided for modeling the contingency resolution strategies with specific examples of contingencies. Existing model checkers are used to evaluate these strategies.

Chapter 4

Multi-USV Case Study

In this chapter, a case study involving a mission in port environments is presented to escort a ship to its docking location using multiple unmanned surface vehicles.

4.1 Introduction

NASA's Mission Control Centre is an excellent example of how complex missions are being centralized and automated. Today, the crew on board the international space missions mostly performs research activities while the spaceflight and other safety conditions are maintained through a ground station via remote control. This concept of automating complex missions involving robots and AI software through a base station supervised by humans is a major focus of current automation goals. It is desirable for humans to be able to supervise and control the entire mission through robots while sitting in a base station far away from the mission activities. However, automation technologies are still lacking in some areas so that humans may also be needed to collaborate with robots.

A mission involving multiple unmanned surface vehicles (USVs) in a busy port is analyzed where the USVs must escort a ship to its designated docking location. The autonomous vessels used during this mission are assumed to be under the supervision of a base station just like with space missions so that in case of a catastrophic failure or mission disruption, human supervisors

can take over. Three types of unexpected contingency events are introduced that the system must be robust against. Mission execution protocols are designed taking into account the desired mission objectives. But these protocols must be checked to be logically consistent and fail-proof, so their correctness is verified using NuSMV model checker [213].

4.2 Problem Statement

Given an English description of the ship escorting mission, the framework presented in the last chapter must be able to encode it in a model checker and evaluate the performance of the contingency resolution strategies. One must first model a ship escorting mission with the help of USVs mounted with UAVs. These robotic boats and drones must operate autonomously to assist the ship once it has entered the port until it has docked successfully. The different contingencies or accident scenarios that can occur must be modeled and strategies must be designed to use the robots to contain these situations without damaging the convoy mission. The framework developed in the previous chapter is used to evaluate the strategies to resolve different types of contingencies.

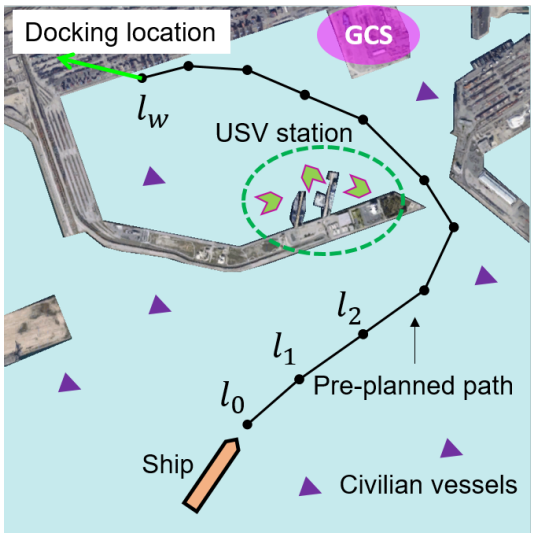


Figure 4.1: Characterization of multi-USV escorting mission scenario (Not drawn to scale)

4.3 Mission Modeling

In this section, a multi-USV mission scenario is analyzed as shown in Figure 4.1 where a ship which has just entered a port must be escorted by a team of USVs to its docking location and assisted till it has been properly docked. The USVs are initially located in the USV station and must finally return to it after mission completion.

Though the USVs are autonomous, they are supervised from a ground control station labeled in Figure 4.1 as GCS. The GCS receives sensory data that helps it to detect contingencies. The GCS is also required to maintain contact with USVs at all time. This is necessary for robotic operations for various reasons, the most important being the ability for human remote operators to take control of the USVs if the USVs fail.

The Kripke structure for each of the transition system involved in this mission is described one by one along with its corresponding NuSMV codes. For this purpose, the USV states must first be identified. Each USV can be located at the docking station (l_s) or one of the way-points generated for the ship motion ($l_0, l_1, \cdots, l_w$).

The straight line segment traversed by the ship from one way-point to another, say l_{i-1} to l_i is denoted by p_i. It is assumed that p_i is the path generated by ship to move linearly from location l_{i-1} to location l_i. The region around path p_i must be clear of other vessels.

In busy ports, commercial or civilian vessels may come dangerously close to a ship and thus to prevent collisions it is important to keep the area around a ship's path clear. The region around path-segment p_i is denoted by r_i. When a civilian vessel intrudes into the protected region of the ship, these situations are referred to as contingencies of type A. Thus, if a contingency of type A arises in region r_i, then a UAV is instructed to survey the region r_i to confirm whether the civilian vessel is still there. If the UAV confirms that the civilian vessel is still there, then its corresponding USV will approach the intruder's vessel and warn it to leave. The USV must keep tracking the moving intruder vessel till the vessel has cleared the region. In NuSMV, this

contingency state of the USV is modeled by assigning it the task of monitoring entire region r_i. Thus, $u_i = r_i$ denotes the contingency state of type A.

Contingencies of type B refer to the blockage of the path in front of the ship, thus making it dangerous, or at least not feasible, to go ahead. Now if contingency of type B occurs, then some static obstruction, say debris, have been found in the path of the ship which prevents the ship from traversing that segment, so a UAV is commanded to survey the path-segment to find if there is a static obstruction. If there is an obstruction, then the associated USV of the UAV will be directed to attempt clearing the path, and until the path is cleared, the mission is paused. Only when the path is cleared, the mission can resume again. It is assumed that if there is large-sized debris, then the USV will be unable to clear. So it will message the central mission controller to send human-operated tugboats to clear the debris. This interaction is not explicitly modeled, and it is assumed that the obstruction is eventually cleared due to the intervention by the USV, and only then the mission resumes.

So the additional states that can be accessed by the USVs are $p_1, \cdots, p_w$ and $r_1, \cdots, r_w$. In NuSMV, this is coded as follows:

```
u1_state : {ls,l0,l1,l2,p1,p2,r1,r2};
```

The states that ship s_0 may take is given by the computed way-points: $l_0, l_1, \cdots, l_w$. The location l_0 is assumed to be the starting location where the pilot assigned to the ship begins guiding it to enter the busy port. The central mission controller, which is generally a ground station on the port premises, computes the way-points needed to be traversed by the ship to reach its docking location, and it is denoted by l_w. These way-points are computed as soon as the ship enters the port at location l_0. Once the ship has reached its docking location, then it must perform complex and slow maneuvers to dock itself properly. During this docking action after reaching the way-point l_w, the ship is assigned to the next state, which will be the execution of the task of getting "docked" properly. In NuSMV, this reads as follows:

```
u1_state : {ls,l0,l1,l2,docked};
```

The initial state of all the USVs, as well as the UAVs, is the port station where they rest and are periodically serviced for maintenance purposes. The initial state of the ship is the first waypoint.

The occurrence of contingencies during the mission is simulated by two variables for each segment. The variable "contingency_type_A[i]" holds non-negative integer values representing the number of other vessels intruding into ship's space r_i. For simulation, an upper limit must be placed on the number of vessels that can appear in this region. Varying this limit significantly affects the simulation results. For each intruder's vessel, only one USV is assigned to handle it. Similarly the Boolean contingency variable "contingency_type_B[i]" denotes whether the path-segment p_i is obstructed or not. This flag may be raised even if the segment is deemed to have low depth or high current. If path-segment is obstructed, then at least two USVs are assigned to handle it. Again such constraints on the number of USVs are needed to handle the contingency tasks. They are user-defined and can be changed appropriately. If ρ^I represents the number of robots that must be assigned for a contingency task of type A, then $\rho^I = 1$. Also, assign $\rho^{II} = 2$, which has a similar meaning. In this work, ρ^I is fixed but other values may be allowed for ρ^{II}. In the former case, it is assumed that the USV's job is to warn intruder vessels by turning on a beacon alarm and alerting the vessel driver to leave the area. This can be efficiently done by a single USV. In the latter case, a static obstruction may have to be cleared. For this purpose, multiple USVs are employed in a coordinated effort to push away the debris. One can choose $\rho^{II} = 1$ if larger debris will not be encountered, or if human assistance will be provided when large obstructions are noticed. Similarly, one can have higher values of ρ^{II} to be safe, but employing USVs is expensive, so there must be a trade-off.

In the case of real-world operations, these variables are maintained and updated by the central ground station, which regularly communicates this information to the USVs. Though the central

ground station must listen to the environmental sensory data to update these values, in th present simulation, their values are randomly sampled from a probability distribution which is assumed to mirror the sensory data recorded over time previously. Therefore, the initial values for the contingency variables are generated by sampling randomly from a probability distribution.

Since the ground station must always maintain a communication link with the USVs, another state variable called "comm_link[i]" is introduced. This is a Boolean variable maintained for USV u_i, which denotes whether the communication link between the ground control station and the USV is working or not. Initially, this variable is set to *true* for all USVs because initially all USVs are at their station and assumed to be properly functioning before the mission commences.

For the above mission scenario, if inputs are given in the form of the number of USVs assigned to the escorting mission, the number of way-points that must be traversed to complete the mission, and traffic/environment modeling for the region around the ship's pre-planned path, then Matlab is used to generate a SMV file for the problem scenario. For example, if there are three USVs, three way-points, and only one civilian vessel can intrude upon the region around the ship's pre-planned path, then the generated SMV file's initial code looks as follows

```
MODULE main

    VAR

        s0_state: {l0,l1,l2,docked};

        u1_state: {ls,l0,l1,l2,p1,p2,r1,r2};

        u2_state: {ls,l0,l1,l2,p1,p2,r1,r2};

        u3_state: {ls,l0,l1,l2,p1,p2,r1,r2};

        a1_state: {ls,l0,l1,l2,p1,p2,r1,r2};

        a2_state: {ls,l0,l1,l2,p1,p2,r1,r2};

        a3_state: {ls,l0,l1,l2,p1,p2,r1,r2};

        comm_link: array 1..3 of boolean;

        contingency_type_A: array 1..2 of 0..1;
```

```
    contingency_type_B: array 1..2 of boolean;
DEFINE
    num_USVs_for_2nd_contingency:=2;
ASSIGN
    init(s0_state):=l0;
    init(u1_state):=ls;
    init(u2_state):=ls;
    init(u3_state):=ls;
    init(a1_state):=ls;
    init(a2_state):=ls;
    init(a3_state):=ls;
    init(comm_link[1]):=TRUE;
    init(comm_link[2]):=TRUE;
    init(comm_link[3]):=TRUE;
```

The transition relationships are described for these states as follows. The ship s_0 which will be assisted in docking can travel from way-point l_{i-1} to l_i only when certain conditions are met. All USVs along with their UAVs should have approached the ship's current location. The communication links of all these USVs with the central mission control station should be in proper working condition. Also, contingency tasks of type A and B should not be affecting the mission. These entire constraints are coded as follows:

```
u1_state=l0 & u2_state=l0 & u3_state=l0 & a1_state=l0 & a2_state=l0 &
a3_state=l0 & comm_link[1] & comm_link[2] & comm_link[3] & s0_state=l0 &
contingency_type_A[1]=0 & !contingency_type_B[1] : l1;
```

Also if the ship is at its last way-point, then it can only start getting docked if all the USVs along with their UAVs are around the ship to assist it while docking and their communication

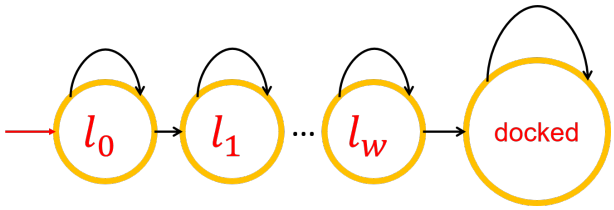


Figure 4.2: Kripke modeling for the ship

links are working properly. If none of these conditions are met, then the ship remains at its current location.

```
u1_state=l2 & u2_state=l2 & u3_state=l2 & a1_state=l2 & a2_state=l2 &
a3_state=l2 & comm_link[1] & comm_link[2] & comm_link[3] & s0_state=l2 : docked;
```

Figure 4.2 shows the Kripke graph for the ship based on the above transition rules.

Next, the transition rules for the USVs are described. Figure 4.3 shows the partial Kripke graph for a USV based on the following transition rules.

A port station is a location inside the port where USVs rest when they are idle and not participating in any mission. If they suffered a communication failure, then they must report to the port station to get the communication equipment repaired or wait safely till the communication link is back on. If the USV is at the port station, the communication link is working, and the ship is assigned to go to any of the pre-planned way-points, then the USV must also approach the same way-point. This is coded up as follows:

```
u1_state=ls & comm_link[1] &
(next(s0_state)=l0 | next(s0_state)=l1 | next(s0_state)=l2): next(s0_state);
```

This transition rule helps take care of both the cases that a USV may find itself in when located at the station. This can be readily verified. It is assumed that the mission begins when the ship has been guided by the pilot to the port location l_0.

So if the USV is at the station before the starting of mission execution, then the ship will remain stuck at port location l_0 until the USV also arrives at l_0; thus `next(s0_state)=l0` holds.

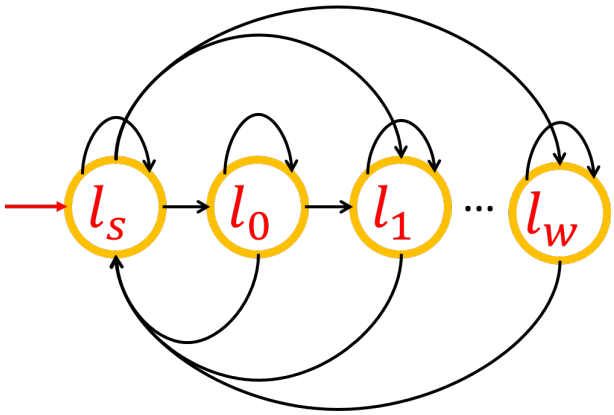


Figure 4.3: Kripke modeling for a USV in the absence of contingencies of type A and B

Then the USV heads towards the location `next(s0_state)` which is clearly l_0. It is assumed that a low-level controller implemented for the USVs will guide them to the ship's location l_0 such that all USVs are in proper formation around the ship and well-separated waiting for a new task to be assigned to them.

Another case may also occur so that the USV suffers a communication failure prompting it to return its base station l_s. In this case, it is desirable for the USV to remain at l_s till the communication link has been restored. Once restored, then the USV must meet the ship at the way-point which the ship is headed towards.

If the ship along with all USVs and their corresponding UAVs are at a way-point l_{i-1} and none of the three contingencies are impacting the mission, then the USV heads towards the next way-point l_i. For $i = 1$,

```
u1_state=l0 & u2_state=l0 & u3_state=l0 & a1_state=l0 & a2_state=l0 &
a3_state=l0 & s0_state=l0 & contingency_type_A[1]=0 &
!contingency_type_B[1] & comm_link[1]: l1;
```

If there are civilian vessels inside the region r_i but sufficient number of USVs have not been sent to warn off the vessels, and the UAV a_1 corresponding to the USV u_1 is already deployed in the region to warn off the vessel, then the USV must go to the region r_i to warn off the intruder

civilian vessel. In all such USV operations, the communication link between the USV and the ground station must be working properly for any operation to proceed.

```
u1_state=l0 & contingency_type_A[1]>count(u3_state=r1,u2_state=r1) &
a1_state=r1 & comm_link[1]: r1;
```

If there are not ρ^{II} USVs surveying and clearing the debris in the path-segment p_i when path blockage has been reported, then such contingencies of type B are also handled similarly, as shown below.

```
u1_state=l0 & toint(contingency_type_B[1])*num_USVs_for_2nd_contingency
> count(u3_state=p1,u2_state=p1) & a1_state=p1 & comm_link[1]: p1;
```

If the USV is in region r_i because it was sent to warn off intruder civilian vessels and the ship is at way-point l_{i-1} waiting for the contingency to be resolved, then once the vessels have acknowledged the warning signals of the USVs and moved away, the USV must go back to the ship's location. Similar logic holds for contingencies of type B.

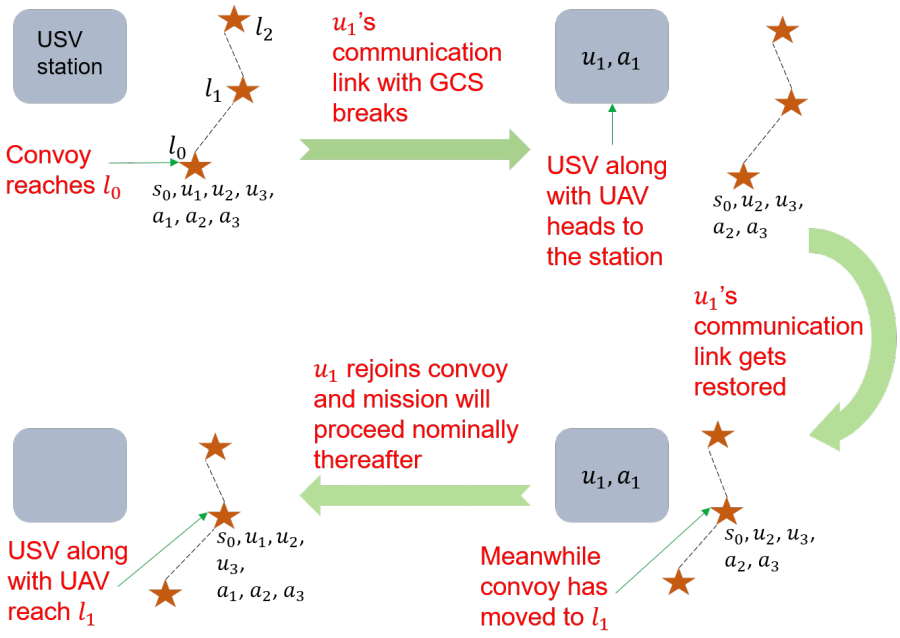


Figure 4.4: Strategy for handling communication failure

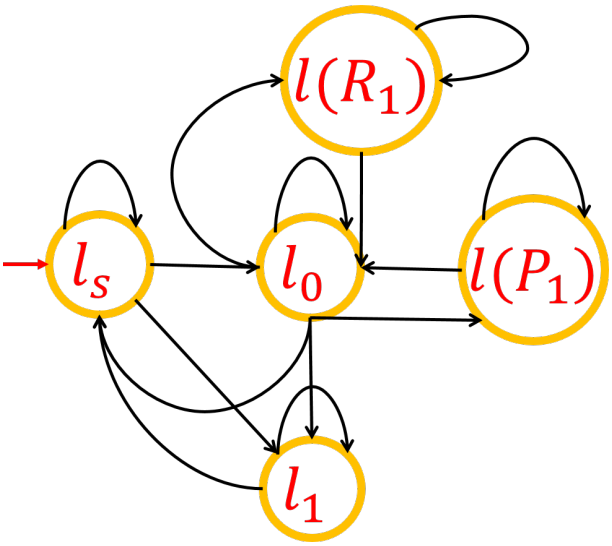


Figure 4.5: Kripke modeling for a USV with three way-points only

```
u1_state=r1 & s0_state=l0 & contingency_type_A[1]=0 & comm_link[1]: l0;

u1_state=p1 & s0_state=l0 & !contingency_type_B[1] & comm_link[1]: l0;
```

Finally, when the USVs have successfully helped the ship in docking to its assigned location, then they head back to their base station l_s.

```
u1_state=l2 & u2_state=l2 & u3_state=l2 & a1_state=l2 & a2_state=l2

& a3_state=l2 & s0_state=docked & comm_link[1]: ls;
```

If communication signal fails then the USV is headed towards the base station.

```
!comm_link[1] : ls;
```

Figure 4.5 shows the full Kripke graph for a USV based on the above transition rules.

Next, transition rules are given for the UAVs mounted atop the USVs. Suppose the USV u_i and UAV a_i are at location l_{j-1}, then if there are more intruder civilian vessels to warn off than the number of UAVs currently assigned to verify and detect their presence, then the UAV a_i must go to the region r_j to decide whether to trigger USV response. This is desired because it is easier to manoeuvre a UAV in a busy port than a USV, but UAVs will not be able to actually resolve the contingency. For that purpose, USVs must eventually be used. For $i = 1$ and $j = 1$,

```
u1_state=l0 & a1_state=l0 & contingency_type_A[1]>count(a3_state=r1,a2_state=r1)
& comm_link[1]: r1;
```

Contingencies of type B are similarly handled as well.

```
u1_state=l0 & a1_state=l0 & toint(contingency_type_B[1])*
num_USVs_for_2nd_contingency > count(a3_state=p1,a2_state=p1) & comm_link[1]: p1;
```

If the UAV is already in the region r_j and there are less UAVs than the number of intruder civilian vessels estimated to be present by the central ground station, then the UAV must wait for more UAVs to enter the region before heading to survey the location where the intruder civilian vessels are believed to be present.

```
a1_state=r1 & contingency_type_A[1]>count(a3_state=r1,a2_state=r1) &
comm_link[1]: r1;
```

Similar transition rule follows for contingencies of type B.

```
a1_state=p1 & toint(contingency_type_B[1])*num_USVs_for_2nd_contingency >
count(a3_state=p1,a2_state=p1) & comm_link[1]: p1;
```

When such cases are encountered by the vessels for which no transition rules have been encoded, then it is expected that the ship and the USVs continue performing the task they are doing or remain idle at the assigned task's terminal state. However, UAVs are supposed to be mounted on top of USVs, so they are expected to track USV's next moves in the above cases. These rules are shown below for the ship, USVs, and UAVs respectively.

```
TRUE : s0_state;
TRUE : u1_state;
TRUE : next(u1_state);
```

Below, the transition rules for the contingency variables are provided. In practice, these are maintained and updated by the ground control station, so these rules simulate the ground control station.

If as many USVs as the intruder civilian vessels are in a region surveying and warning them off, then all contingencies of type A are designated the status of being resolved. This is expressed in NuSMV as follows.

```
count(u1_state=r1,u2_state=r1,u3_state=r1)>=contingency_type_A[1]: 0;
```

Each path-segment has an associated range $[i_{min}, i_{max}]$ of intruder civilian vessel that may enter the region around the ship's path. It is assumed that $i_{min} = 0$ so that for every segment, there is a possibility of there being no civilian intruder vessel. At location l_0, the ship has just entered the port area, so it is expected that this area will be less crowded and hence have lower values of i_{max}.

As the ship's motion proceeds, it will encounter higher traffic regions. So the middle segments have a higher i_{max}. And during the end of ship's motion, it will be near the docking regions which will have lower civilian vessel population and hence lower values of i_{max}.

If UAVs arrive in the region where intruder civilian vessels were reported, then either the contingencies will get resolved or not. Contingency resolution here only means that when the UAV surveys the region for the intruder civilian vessel, then that vessel may have moved away or it was not there in the first place and the sensor data was wrong or misinterpreted by ground station.

Let the probability with which this can happen for every civilian vessel be p_{res}. It is assumed that this is 50%. So if i_u civilian vessels were reported in the region, then after UAVs have reached the region, the number of civilian vessels still remaining in the region may be $0, \cdots, i_u$. The probability of i'_u civilian vessels ($0 \leq i'_u \leq i_u$) still intruding the region is clearly $\binom{i_u}{i'_u} p_{res}^{i_u}$. So if $i_{max} = 3$ and let $i_u = 2$, then the transition rule would be:

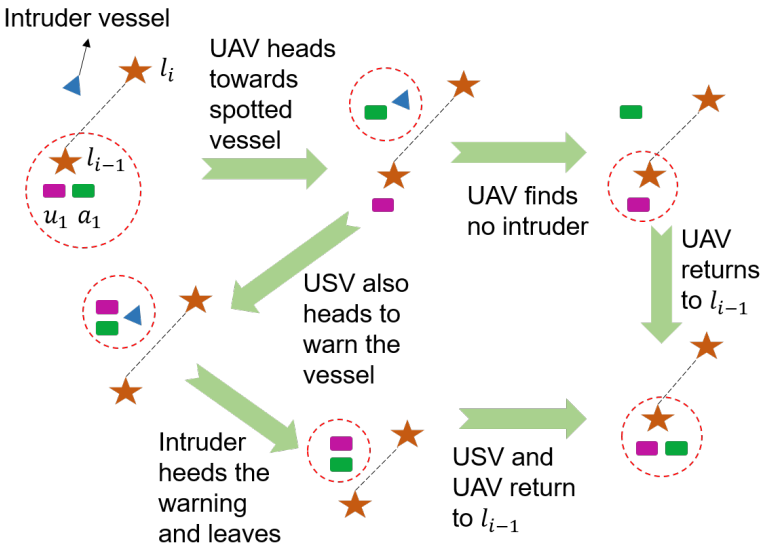


Figure 4.6: Resolution strategy for contingencies of type A

```
count(a1_state=r1,a2_state=r1,a3_state=r1)
>=contingency_type_A[1] & contingency_type_A[1]=2: {0,1,1,2};
```

Above, the following holds: `contingency_type_A[1]`$= i_u$. This is in relation to the path-segment r_1 where two civilian vessels have been reported. Thus, a similar rule must be coded for all possible values of i_u.

The number of civilian vessels that may be reported is considered to be uniformly random. So if a segment can have a maximum of say $i_{max} = 3$ vessels, then a final transition rule may be added as follows:

```
TRUE : 0..3;
```

The transition rules for contingencies of type B are handled in the same way. For example,

```
next(contingency_type_B[1]) := case
    count(u1_state=p1,u2_state=p1,u3_state=p1)>=
    num_USVs_for_2nd_contingency: FALSE;
    TRUE : {TRUE,FALSE};
```

```
esac;
```

No separate rule is needed for UAVs because this contingency occurs singly, and is similar to the case of contingency of type A if it is assumed that $i_{max} = i_u = 1$. It gets handled automatically by the final rule: `TRUE : {TRUE,FALSE};`

Also, one could let the communication link be randomly maintained by the NuSMV solver or associate a probability of p_{comm}. Say, the probability of the communication link going down is 1/3, then the transition rules for the communication link will be as follows:

```
next(comm_link) := case
    TRUE : {TRUE,TRUE,FALSE};
esac;
```

The Kripke structure for the ship, USVs, UAVs, and the ground control station (which is being simulated through the contingency and communication link variables) is incomplete without explicitly mentioning the labeling functions.

For the vessels, this is obvious because the labeling functions are the assertions regarding whether a vessel is at some location or doing some task. For example, in the case of the ship, `s0_state=l0` is a labeling function which asks whether the ship has entered the port region or not. These labeling functions are crucial for triggering state transitions. For example, there is no point in commanding the USVs to head out of their station until the ship has entered the port region.

Next, the labeling functions for the ground control station are clarified. The ground control station is indirectly being modeled through the modeling of contingencies and communication links.

Thus, whether there are enough USVs in a region to resolve a contingency or whether the communication link is properly working are examples of labeling functions used to trigger and control the base station's mission progress.

4.4 Model Checking and Verification

As explained in Section 3.3.4.1, it must be verified whether the ship docks successfully and all robots return to their station in the presence of all three types of contingencies. So LTL specification to be verified is as follows:

$$\phi_1 \coloneqq \mathbf{FG}\,(s_0 = \text{docked}) \wedge_{i=1}^{n} \mathbf{FG}\,(u_i = l_s) \wedge_{i=1}^{n} \mathbf{FG}\,(a_i = l_s) \tag{4.1}$$

The above formula requires the ship to be docked and both the USVs and their corresponding UAVs rested back at their base station. A fairness condition is also added such that the ship has safely reached the docking location at least once, and this is given by $\mathbf{F}\,(s_0 = \text{docked})$. This is done to prevent livelocks due to contingencies.

Some practical upper limits could also be put on the number of times each contingency could occur because NuSMV allows bounded past operators. There may also be sophisticated ways to handle such livelocks in the model itself.

One simple method is given for handling the same. This fairness condition may prevent the detection of bugs in the nominal mission modeling itself because the fairness condition may force NuSMV to ignore them.

So, the correctness of the nominal mission model must be verified in the absence of both contingencies and the fairness condition to detect bugs that may have been introduced by human coders. For example, a transition rule is coded for each USV as follows:

```
u1_state=Ls & comm_link[1] & (next(s0_state)=L0 | next(s0_state)=L1 |
next(s0_state)=L2) : next(s0_state);
```

This rule ensures that the USVs will initially join the ship at l_0 or if a USV has returned to its station due to communication failure then it rejoins the ship once its link is working again. So, the USV must track the ship's next state.

However, the ship's next state must be a waypoint. This is enforced by the disjunction of three atomic propositions, as shown above. This is done because if the ship's next state is *docked*, meaning the ship is at its last waypoint and trying to dock itself, then this rule will prompt the USVs to take the *docked* state.

But *docked* is not a state that the USVs can take. Therefore, if the human coder had forgotten to encode this last condition, then NuSMV will throw up an error. Thus, NuSMV allows bug correction.

However, if the USV is sent to location l_w in order to assist the ship while it is docking to its designated location, one could code it up as follows:

```
u1_state=Ls & comm_link[1] & next(s0_state)=docked : L2;
```

Here, we simply note that $w = 2$ because we consider only three way-points in the current example. Ii is also important to verify that the USV's communication links are working properly before it is commanded to join the other USVs at the docking location.

For the contingency-incorporated model, it is first verified that the mission model does not have deadlocks by checking for transition relation totality using the NuSMV command `check_fsm`.

It is confirmed that no deadlocks exist so Equation 4.1 is verified using the NuSMV command `check_ltlspec`. NuSMV confirms that the property holds. A model trace is generated to verify this by checking the correctness of $\neg\phi$. This returns a counterexample in the form of a mission trace.

The mission trace returned by NuSMV is trivial as it only shows the convoy moving through each way-point without the occurrence of any contingency event until the vessels and drones are rested in their desired positions.

A simple kind of inconsistency that also gets detected is as follows. Suppose $\rho^{II} = 3$ and only three USVs are available for the mission and a civilian vessel enters the ship's protected space, then no USV is available for clearing the ship's path if it gets blocked.

Also, the ship must visit the way-points in a sequence. Thus there are many SAR constraints implicitly encoded in the mission model which may be verified explicitly in NuSMV. The previous LTL expression needs to be modified as follows:

$$\begin{aligned}\phi_2 \coloneqq & \mathbf{FG}\,(s_0 = \text{docked}) \wedge_{i=1}^{n} \mathbf{FG}\,(u_i = l_s) \\ & \wedge_{i=1}^{n} \mathbf{FG}\,(a_i = l_s) \wedge \mathbf{F}((s_0 = l_0) \wedge \mathbf{F}((s_0 = l_1) \wedge \mathbf{F} \cdots)) \\ & \wedge_{i=0}^{w-1} (s_0 \neq l_{i+1}\,\mathbf{U}\, s_0 = l_i) \\ & \wedge_{i=0}^{w-1} \mathbf{G}(s_0 = l_{i+1} \implies \mathbf{XG}\, s_0 \neq l_i)\end{aligned} \tag{4.2}$$

This also passes the verification test. More interesting results are generated by starting the system from different initial states and designing more complex LTL formulas. Let USVs u_1 and u_2 along with their UAVs initially be at way-point l_2 and USV u_3 is at l_0. Also, assume that the communication link for USV u_2 is not working properly.

Other information remains the same as previously. Then NuSMV produces a counterexample when verifying $\neg\phi_1$ which is interpreted as follows: USV u_2 returns to the base and waits for the communication link to work again. In the next time-step, its communication link is working and it goes to l_0 while USV u_1 returns to the base. Then USV u_1 joins u_2 and u_3 at l_0. The mission proceeds nominally thereafter.

Another case is obtained by adding a contingency condition to the previous initial conditions that there is a civilian vessel intruding into region R_2. The LTL specification $\neg\phi_3$ is given below:

$$\phi_3 \coloneqq \phi_1 \wedge ((\mathscr{C}^{II} = 1)\,\mathbf{U}\, \vee_{i=1}^{n} (u_i = R_2)) \tag{4.3}$$

$\mathscr{C}^{II}$ is a shorthand for the variable "contingency_type_A[2]". NuSMV counterexample is as follows: The convoy proceeds up to l_1 as usual. Then blockage for path P_2 is reported while a civilian vessel is still in R_2. So, a_1 goes to $l(R_2)$ and a_2, a_3 go to $l(P_2)$. The contingencies are still present

Table 4.1: Runtime statistics collected for different problem sizes

Number of USVs	Number of way-points	Wall-clock time	RAM
3	3	7.38 sec	30.182 MB
3	4	2.54 min	81.639 MB
3	5	5.09 min	214.021 MB
4	3	34.01 min	350.021 MB
4	4	6.04 hr	3.187 GB
4	5	16.52 hr	8.090 GB
5	3	30.70 hr	8.885 GB

so the corresponding USVs head towards $l(R_2)$ and $l(P_2)$. Then the contingencies are resolved and the vessels and drones return to l_1 after which mission proceeds nominally.

Bounded model checking was also performed using either of the two Boolean satisfiability problem (SAT) solvers: Zchaff and MiniSat.

In both cases, a counterexample was generated at the eighth level as follows: The convoy proceeds up to l_1 as usual, and then a_1 goes to $l(R_2)$ because an intruder is sighted. Then communication links for the USVs u_2 and u_3 fail simultaneously. So they return to the base while u_1 reaches $l(R_2)$ to warn the civilian vessel. Then all contingencies are resolved. Finally, the convoy reassembles at l_1 and mission proceeds nominally.

NuSMV also reports run-time statistics. For three USVs and three way-points, the reported user time elapsed is 7.4 seconds, and the system time is 0.015 seconds. Thus, the total CPU execution time consumed is 7.415 sec. The wall-clock time for this process is 7.38 seconds. The RAM allocated for this process is $30,906$ KB ~ 30.182 MB.

The operating system is Ubuntu MATE 18.04.1 and workstation is Dell Precision Tower 3620 including eight Intel Xeon E3-1245V5 CPUs having 3.5GHz speed (but this process seems to be single-threaded) and a total of 32GB RAM. MATLAB was used to generate SMV files for different numbers of USVs and way-points.

From Table 4.1, it is observed that the running time and allocated memory rise exponentially with an increase in problem size and USVs impact run-time statistics much more drastically than way-points.

4.5 Summary

A formal model of a ship escorting mission in a busy port is presented. Since running high-fidelity simulation experiments is computationally prohibitive, it is demonstrated how verification techniques like model checking can be used to model and verify the performance of such missions. The correctness of contingency resolution strategies is verified using linear temporal logic formulas in NuSMV software.

Chapter 5

Multi-robot Assembly Case Study

In this chapter [1], a case study involving assembly cells is presented and it is described how the robots and humans interact to produce assemblies efficiently.

5.1 Introduction

In this chapter, a manufacturing assembly line is analyzed which must carry out complex assembly tasks. The modeling approach presented in this chapter is inspired from the following robotic assembly example. Figure 5.1 shows a three-manipulator assembly setup that has been assigned to assemble a mock-up of a satellite [215]. The parts required for the assembly process may arrive at the assembly station through a conveyor system from where they are picked up by a mobile manipulator (Figure 5.2) [216–219]. These parts are then assembled to build the satellite mock-up whose assembled CAD view is shown in Figure 5.3. Figure 5.4 shows the exploded view of the CAD model, and the YouTube video [215] shows the steps for assembling this model from scratch. For large-scale, real operations, each manipulator would be assigned its station and consider multiple such cells working cooperatively to produce complex end-products. Thus, stations would be producing sub-assemblies that would be processed in each cell. Moreover, the finished products from each cell will feed into each other to accomplish end-product completion.

[1] The work in this chapter is derived from the work published in [214]

It is generally challenging to automate assembly tasks of high complexity fully. For this purpose, human workers must collaborate with robots. Thus, human involvement is modeled, not in a far-off base station, but at the mission site itself along with the robots. The entire assembly line may still be supervised and monitored from a base station located away from the actual operation site. This mission is modeled and analyzed in the PRISM model checker [220].

5.2 Problem Statement

Given an English description of an assembly line process, it must be encoded formally in a model checker and then used to evaluate different contingency resolution strategies for preventing assembly line stoppage or failure. The assembly cells comprising of assembly stations must be modeled formally so that they can be encoded in a model checker. Human workers and robots must be incorporated in this formal model to automate assembly production. The effectiveness of humans to assist robots in recovering from faulty assemblies must be analyzed via different contingency resolution strategies. An assembly line can be set up and then the divide-and-conquer approach can be used to analyze it in smaller units. Those results can then be aggregated into a whole because an entire assembly line cannot be efficiently modeled in model checking tools.

5.3 Overview of Approach

High-level operations planning comprises of operation modeling, verification of the nominal operations plan, and assessment of different contingency resolution strategies. Initially, a nominal operation description is given using which the nominal tasks comprising the operation are identified. Agents assigned to the tasks may be human workers or robots. The pre-conditions and post-conditions are identified for each task. The set of discrete states that can be assigned to each agent along with their domains and initial values is also identified. Other environment variables can also be identified to track the operation progress. *Transition rules* are encoded that specify

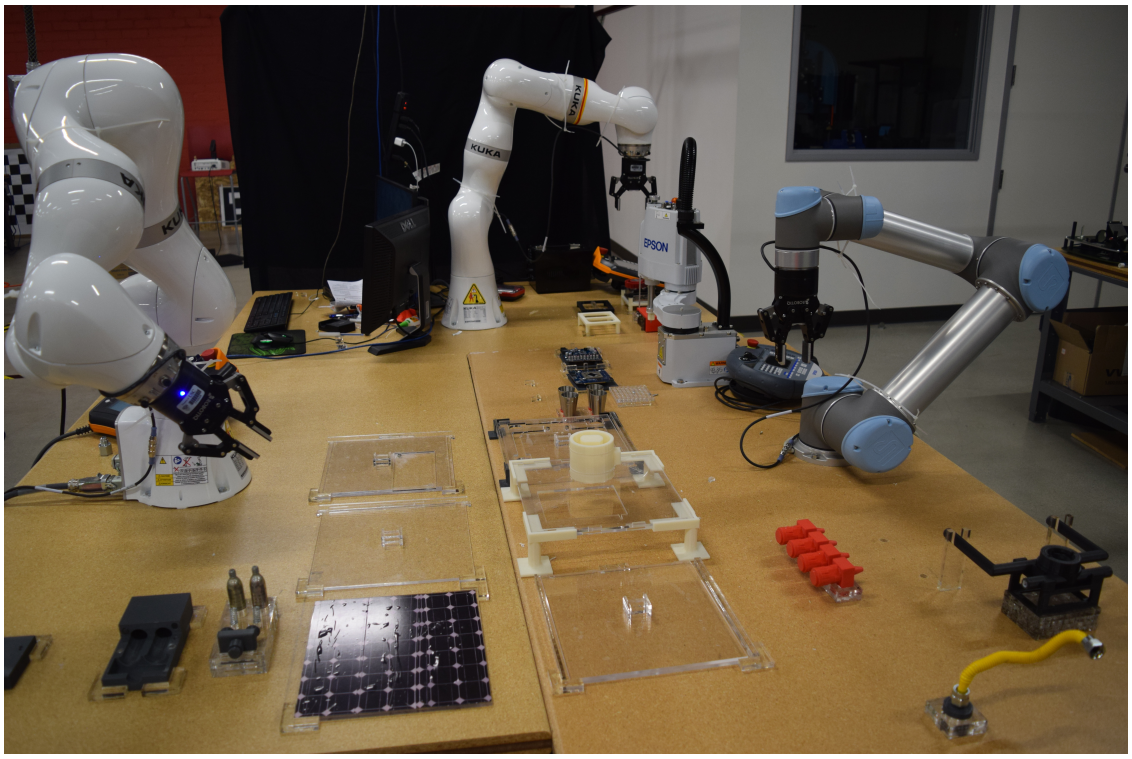


Figure 5.1: Three manipulators assigned for cooperative assembly operation

Figure 5.2: Mobile manipulator

constraints for agents to transition from one state to another in terms of tasks' pre-conditions and post-conditions. A nominal operations execution plan provides a sequence in which the available

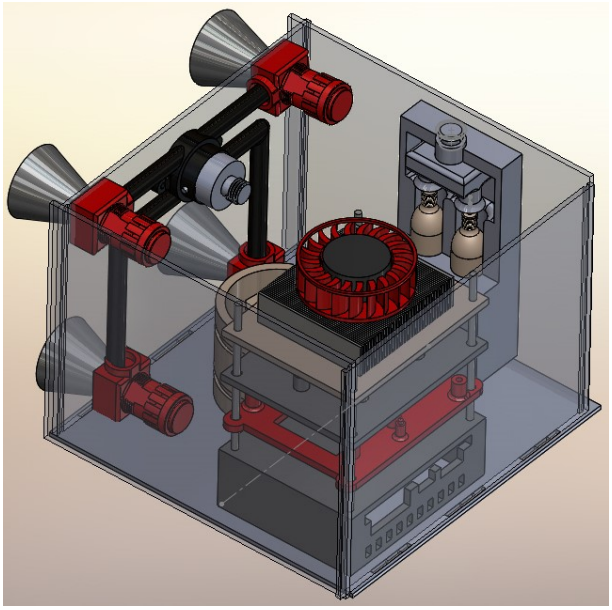


Figure 5.3: CAD model for the assembled satellite mock-up

tasks must be executed to meet the operation objective while adhering to pre-conditions and post-conditions of all tasks.

It is assumed that the transition rules between states of a system are probabilistic. Markov decision processes are adopted for modeling these operations. When there is only one probabilistic transition rule for every possible state of the system, then the process is modeled as a discrete-time Markov chain (DTMC) in PRISM which is commonly used for probabilistic model checking. Model checking for DTMCs is done via numerical or statistical methods in PRISM. Statistical methods allow handling of larger problem sizes.

The contingency resolution strategies are designed that get invoked when a contingency is reported during operation execution. These strategies may require multiple actions to be performed in a sequence which is simulated by appropriately synchronizing specific transition rules in PRISM. For different cases, different sets of actions may have to be invoked. For probabilistic systems, the formulated specifications are verified using Probabilistic Computation Tree Logic

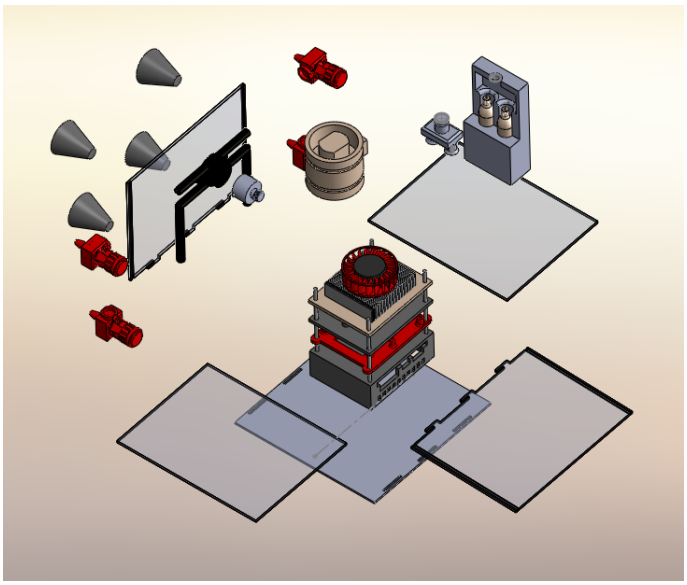


Figure 5.4: Exploded view of the CAD model for the satellite's mock-up

(PCTL). Probabilistic reachability conditions may be queried, which means querying the probability that the system eventually satisfies a set of atomic propositions. If a reward structure is defined, the expected value of different variables may also be computed.

The approach to analyzing multiple assembly cells working cooperatively is first to analyze specific groups of assembly stations (referred to as a cell) using formal methods and then perform a Monte Carlo analysis of the entire system working together. This is because large systems cannot be entirely modeled in model checkers due to computational memory requirements. However, if the more extensive system is decomposed into smaller subsystems, then it is possible to encode them in a model checker and analyze them efficiently. Thereafter, performing an aggregate analysis for the whole system using Monte Carlo simulations would not be so memory-intensive because model checking results could be used. It may be possible to analyze the entire system by only using Monte Carlo simulations without performing any decomposition and subsequent model checking, but that appraoch would fail to leverage the rich field of temporal logic model verification. Since the probabilistic model checking community has made significant progress in developing model

checkers to solve Markov models under temporal logic constraints, it has become a popular tool for use in robotics research as part of the broader application of formal methods to robotics.

5.4 Modeling of Assembly Cell

5.4.1 Nominal Operation Description

A simple assembly cell consisting of N assembly stations $\mathcal{S}_1, \mathcal{S}_2, \cdots, \mathcal{S}_N$ is described below. A visual schematic for $N = 3$ is presented in Figure 5.5. The variables used in the Figure are explained in Section 5.4.2. This model allows human intervention at each station when the desired production process fails. Each station is also provided with one dexterous mobile manipulator which performs all assembly-related tasks. The robot designated at station $\mathcal{S}_i$ is denoted by $\mathcal{M}_i$. A conveyor system is modeled to bring parts from the storage area. The parts are then picked from the conveyor belt and transported to the designated assembly station using a simple pick-and-place mobile robot. Each station $\mathcal{S}_i$ is provided with such a robot denoted by P_i. This conveyor belt system is referred to as $\mathcal{B}_1$.

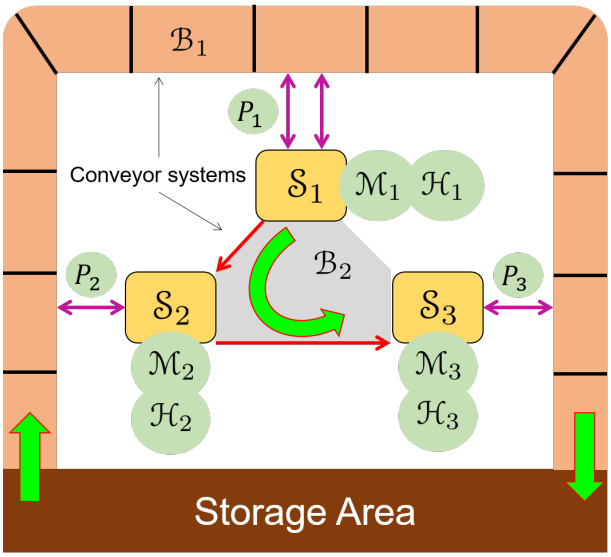


Figure 5.5: Schematic representation of multi-robot assembly cell

There is another conveyor belt system $\mathcal{B}_2$ operating across the stations. This is because it is assumed that serial production has been set up across the stations. The job of this conveyor system is merely to transfer parts from the preceding station to the next station for further processing. So, once the assembly robots have finished their tasks and are ready to transfer the part to the next station, they place the assembled unit on the conveyor system which then transfers the parts to the next station. Thus, the assembly process starts at the first station, and the fully assembled part is produced at the last station. Then another pick-and-place robot is designated to pick the part and transfer it to the desired location. Thus, an assembly cell consisting of N stations deploys $2N + 1$ robots. One could alternatively also model the cell to deploy fewer robots by combining the responsibilities of various robots into a single robot's tasks.

5.4.2 State Variables

The system model naturally translates into a set of variables. As shown in Figure 5.5, the operation begins with two sets of parts being delivered at the first assembly station $\mathcal{S}_1$. The number of first set of parts delivered to station $\mathcal{S}_1$ is denoted by $n(1,1)$ and the number of second set of parts by $n(1,2)$. The convention used is to use the first index to denote the assembly station and the second one to denote a variable index. Then the assemblies produced at station $\mathcal{S}_1$ is denoted by $n(1,3)$. For all other assembly stations $\mathcal{S}_i$ such that $i > 1$, one set of parts arrive via the conveyor system $\mathcal{B}_1$ and a set of partially completed assemblies arrive from their preceding stations $\mathcal{S}_{i-1}$. The number of parts delivered by robot P_i is denoted by $n(i,1)$ and the number of parts arriving from previous stations is denoted by $n(i,2)$. Thus, a new set of parts is delivered to stations, and these are fitted properly in the assemblies arriving from the preceding stations. Such parts will be called *intermediate units* (denoted by $n(i,1)$) and the partial assemblies will be referred to as *sub-assemblies* (denoted by $n(i,2)$). The total assemblies produced by each station at any time is tracked using the variable $n(i,3)$. For a given cell, one is generally interested in the total number of final assemblies produced by the last station, $n(N,3)$.

All variables naturally have a finite range associated with it due to space constraints. For simulation purposes, even smaller ranges may be imposed due to solver constraints. All the variables declared above are non-negative integers. The upper bounds for these variables are also impacted based on whether numerical or statistical model checking methods are used. If the numerical method is used, then the model size must be kept small. Therefore, high upper bounds cannot be placed for these variables. On the other hand, statistical methods provide approximate results but accommodate larger model sizes, thereby allowing for comparatively higher upper bounds.

All variables are initially expected to be zero. However, manufacturing units tend to keep multiple intermediate units at the station before the conveyor operation, and assembly process begins. This buffering of parts is done to avoid any bottlenecks in further operations due to a shortage of intermediate units required to undergo assembly. This concept is referred to as *initial part buffer*. Thus, all the variables $n(i,1)$ and $n(1,2)$ may be greater than zero, say N^I. All other variables are zero initially.

For example, below, the range of a variable $n(2,3)$ and its initial state are declared. Thus, at any instant, there must not be more than five assemblies produced by station $\mathcal{S}_2$. The variable $n(2,3)$ is shortened to `n23` in PRISM. This convention is followed throughout this chapter.

```
const int max_assemblies = 5 ;
n23 : [0..max_assemblies] init 0 ;
```

Next, the modeling of assembly robots $\mathcal{M}_i$ is described. The description of robot $\mathcal{M}_1$ differs from that of the rest because its model is a simplified version of that of others. So, first the model for robot $\mathcal{M}_i$ is described where $N \geq i \geq 2$. Figure 5.6 provides a visualization aid for this model. The sub-assemblies first arrive from station $\mathcal{S}_{i-1}$ to station $\mathcal{S}_i$.

Here, the sub-assemblies first undergo a pre-detection process by robot $\mathcal{M}_i$ to ensure whether the arriving sub-assemblies are as expected and ready for further processing. If pre-detection

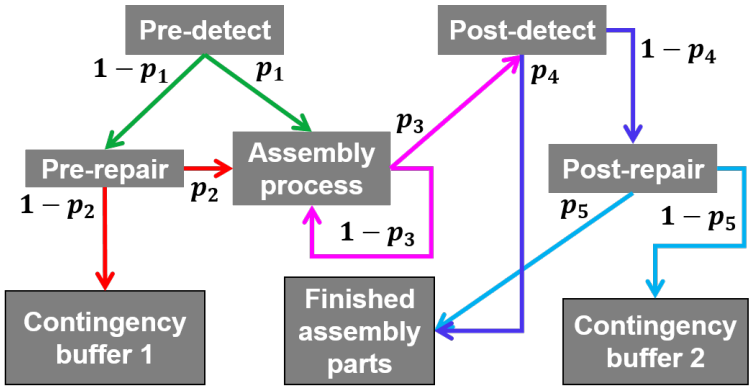


Figure 5.6: Probabilistic transition model for assembly robot

succeeds with probability p_1, then the sub-assembly is ready for the assembly process. Otherwise, the sub-assembly must undergo a pre-repair process. The number of parts queued for pre-repair is denoted by $x(i,1)$ whereas the number of parts queued up for assembly process is denoted by $x(i,2)$.

Each station $\mathcal{S}_i$ has two storage location referred to as *contingency buffers*: $\mathscr{C}(i,1)$ and $\mathscr{C}(i,2)$. Here, parts which fail during repairing operations are sent for repair by a human helper. Each station $\mathcal{S}_i$ is assigned its own human helper $\mathcal{H}_i$. So if the pre-repair process succeeds with probability p_2, then the sub-assembly is queued up for the assembly process. However, if during the pre-repair process, robot $\mathcal{M}_i$ fails to resolve the problems in the sub-assemblies, then those sub-assemblies are queued up in contingency buffer $\mathscr{C}(i,1)$. The number of parts queued up in contingency buffer $\mathscr{C}(i,1)$ is denoted by $c(i,1)$.

If the assembly process succeeds with probability p_3, then the assembled part is queued up for a post-detection operation where the robot again confirms using image processing algorithms whether the desired result has been achieved. Otherwise, if the assembly process fails (this is likely because some assembly procedures may be very complex to carry out and may test robot capabilities to its utmost limit), then the robot merely queues up the part for attempting its assembly later again. The number of parts queued up for post-detection is denoted by $x(i,3)$.

If post-detection succeeds with probability p_4, then the sub-assembly is ready for being transferred to the next station, $\mathcal{S}_{i+1}$. Otherwise, the sub-assembly must undergo a post-repair process. The number of parts queued for post-repair is denoted by $x(i,4)$.

If the post-repair process succeeds with probability p_5, then the sub-assembly is ready for being transferred to the next station, $\mathcal{S}_{i+1}$. However, if during the post-repair process, robot $\mathcal{M}_i$ fails to resolve the problems in the sub-assemblies, then those sub-assemblies are queued up in contingency buffer $\mathscr{C}(i,2)$. The number of parts queued up in contingency buffer $\mathscr{C}(i,2)$ is denoted by $c(i,2)$.

There is a human helper assigned to station $\mathcal{S}_i$ denoted by $\mathcal{H}_i$ who is responsible for both the contingency buffers and attempts to resolve each sub-assembly sent to it. If human succeeds with probability p_6 each time he/she attempts contingency resolution at contingency buffer $\mathscr{C}(i,1)$, then these resolved sub-assemblies are queued up for assembly by robot $\mathcal{M}_i$. For sub-assemblies held at $\mathscr{C}(i,2)$, the human may succeed with the same probability, and resolved sub-assemblies are declared as finished sub-assemblies for that station and ready for transfer via conveyor belt system $\mathcal{B}_2$ to the next assembly station, $\mathcal{S}_{i+1}$.

Model for robot $\mathcal{M}_1$ is a simplified version of the above as it does not have to perform the pre-detection and pre-repair operations. This is because the two sets of parts are coming from the storage area and not from a station within the cell. So it is assumed that parts arriving from the storage area do not need to be verified. They will already be verified by the conveyor belt transport system $\mathcal{B}_1$ but automated verification by conveyor system is not modeled here. Consequently, there is also no contingency buffer $\mathscr{C}(1,1)$. Robot $\mathcal{M}_1$ directly performs its assembly operation followed by post-detection and post-repair operations.

Based on the above modeling, one can check that the PRISM model contains $9N-2$ variables and $N+1$ modules. All the intermediate variables $x(i,\cdot)$ referred to above have size constraints as they represent the process of storing parts. The lower bound for each variable is zero while the upper bound is user-defined. It is assumed that the upper bound on all variables is N_{max} except

for $n(N, 3)$ whose upper bound is denoted by N^*. The rationale for doing so is to be able to compute the expected production time for assembling as many parts as computationally feasible. So N^* can be made large despite keeping N_{max} smaller.

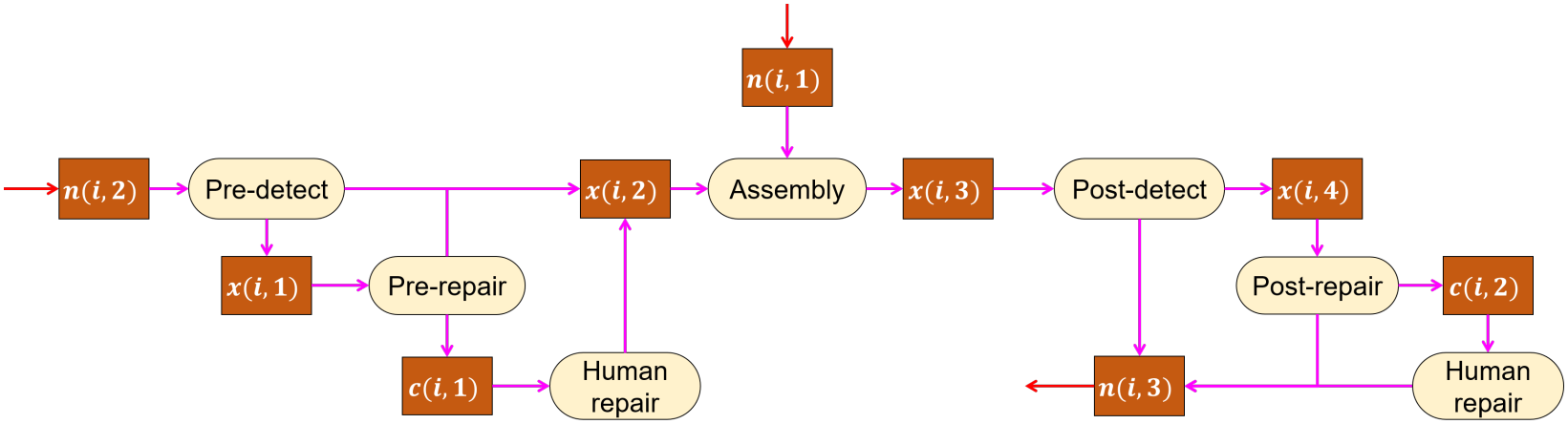


Figure 5.7: Schematic representation of assembly processes in station $\mathcal{S}_i$ for $i > 1$ for strategy s^1. The rectangles represent storage containers, and the rounded rectangles represent processes involved during assembly.

5.4.3 Transition Rules

The above description of the assembly model naturally results in probabilistic transition rules which can be coded in PRISM. The pre-detection process of robot $\mathcal{M}_2$ in station $\mathcal{S}_2$ is expressed as the following transition rule in PRISM.

```
const double p1 = 0.8;

const int max_units = 5 ;

[tick2] x21<max_units & x22<max_units & n22>0 -> (1-p1):(x21'=x21+1) &

(n22'=n22-1) + p1:(x22'=x22+1)&(n22'=n22-1);
```

According to the above rule, when a sub-assembly transferred from preceding station is pre-detected and succeeds it will increase the variable $x(2,2)$ by one and decrease $n(2,2)$ by one. Otherwise, if pre-detection flags the sub-assembly as having errors, then $x(2,1)$ is increased by one and $n(2,2)$ decreased by one. This rule must only be invoked if the variable bound constraints allow to do so. `tick2` is a label for this transition rule. Such labels are used for synchronizing one rule with other rules. This will be elaborated in the next section. Similar rules as above exist for pre-repair, assembly, post-detect, and post-repair processes.

Human helper $\mathcal{H}_2$ must operate both the contingency buffers and keep clearing the faulty sub-assemblies arriving in them. Consequently, the following rules hold for contingency buffer $\mathscr{C}(2,1)$:

```
const double p6 = 0.95;

const int max_units = 5 ;

[tick7] x22<max_units & c21>0 -> p6 : (x22'=x22+1)&(c21'=c21-1);

[tick7] !(x22<max_units & c21>0) -> true;
```

The first rule states that if there is a faulty sub-assembly in the contingency buffer, and there is space for at least one more sub-assembly to be queued for assembly, then human helper $\mathcal{H}_2$ attempts resolving the sub-assembly and may succeed with probability 95%. The second rule

only ensures that if the size constraints do not allow contingency resolution, there is a rule that explicitly states this with both rules sharing the same label. Accounting for all possibilities explicitly using transition rules with the same labels is required for synchronizing multiple rules properly. This will be elaborated in the next section.

5.4.4 Nominal Operation Strategy

Next, the nominal protocol is presented through which assembly operations proceed. The entire assembly process described in the previous section keeps repeating in cycles. Each cycle comprises of eight steps involving specific operations like pre-detection and post-repair. Each station is implemented as a separate PRISM module. In the first step of each cycle, both the conveyor systems are implemented concurrently.

The conveyor system $\mathcal{B}_1$ is supposed to bring parts from the storage and stop for the robots P_i to pick the parts and take them to their respective stations. For efficiency of the operations, it is assumed that the conveyor system would have been automated such that the belt stops such that the parts desired by robot P_i are optimally located. For each station, it must be checked that the constraint $n(i,1) < N_{max}$ for delivering intermediate units. Additionally, for the first station, which also receives a second set of intermediate units, one must also check $n(1,2) < N_{max}$. Therefore, $N+1$ constraints must be checked for the first conveyor system operation.

The conveyor system $\mathcal{B}_2$ transfers finished sub-assemblies from one station to the next. For N stations, $N-1$ such transfers can take place if size constraints allow so. For each station $\mathcal{S}_i$ excluding the first, one must only check two size constraints: $n(i-1,3) > 0$ and $n(i,2) < N_{max}$. The two size constraints can be combined into one using the conjunction operator. Thus, $N-1$ constraints must be checked for the second conveyor system operation.

All these constraints are independent of each other. So the total of $2N$ constraints lead to 2^{2N} cases, and transition rules are encoded for each of these cases for each station. Thus, there will be $N * 2^{2N}$ transition rules relating to the first step, and hence, they must all be synchronized

together. For this purpose, an additional clock module is used which cycles through eight steps. According to the nominal strategy, each cycle comprises of eight steps. Hence the clock has eight steps. A label `tick1` is created for its first step and label all $N * 2^{2N}$ rules also as `tick1`. Another simplifying assumption is made that both the conveyor systems are working deterministically. If different operations in these conveyor systems were probabilistic and had different probability values, then they could not be synchronized. This is a reasonable assumption because conveyor systems are very efficient. Most problems generally arise in the robotic aspect of the operations.

In the steps second through sixth, the assembly robot performs its pre-detection, pre-repair, assembly, post-detection, post-repair operations consecutively. Then in the seventh and eight steps, the human helper seeks to resolve faulty sub-assemblies in contingency buffers $\mathscr{C}(i, 1)$ and $\mathscr{C}(i, 2)$ respectively. No human activity is allowed inside the station while the robot is performing any of its tasks because of safety purpose.

The goal is to compute the expected production time for N^* parts. Therefore, a reward structure labeled as `time_duration` must be defined. Since conveyor operations are fast, and robot P_i may take some time to transfer intermediate units from the conveyor belt to the stations, it is assumed that it takes around 10 seconds. Time is recorded in minutes. So the time taken in the first step is rounded to 0.2 minute. Since computer vision algorithms have become very fast, significant time spent during the detection operations is while manipulating the camera to capture images from different angles or scan point clouds. So detection routines are also assumed to take 0.2 minute. The repair and assembly processes take two minutes. The human helper takes five minutes.

5.4.5 Contingency Resolution Strategies

The strategy described in the previous section is referred to as strategy s^1. A nondeterministic version of this protocol referred to as strategy A is also designed, which will be implemented as an MDP in PRISM. The first step remains the same. In the second step, the assembly robots

perform one of their detection actions taking a maximum of 0.2 minute. In the third step, the assembly robots perform one of their assembly/repair actions, taking a maximum of two minutes. In the fourth step, human helper repairs parts in one of the contingency buffers requiring five minutes. Four more strategies are provided to resolve the contingencies which are variations of strategy s^1.

Instead of keeping a human helper at each station, there could be only one human assigned to the entire cell. This is referred to as strategy s^2. Each cycle of strategy s^2 is implemented as a concatenation of N cycles of strategy s^1. Each of the N cycles are referred to as sub-cycles. In each sub-cycle except for the last one, there are eight steps. In the last sub-cycle, there are only seven steps. Thus, there are a total of $8N - 1$ steps in each cycle of strategy s^2. The first six steps of each sub-cycle are same as strategy s^1.

In the seventh step of each sub-cycle, the human seeks to resolve faulty sub-assemblies in contingency buffer $\mathscr{C}(i, 1)$, and in the eighth step of each sub-cycle, the human seeks to resolve faulty sub-assemblies in contingency buffer $\mathscr{C}(i, 2)$. Since the first station does not have contingency buffer $\mathscr{C}(1, 1)$, the contingencies for contingency buffer $\mathscr{C}(2, 1)$ are resolved in the seventh step.

Thus, for i-th sub-cycle such that $i < N$, the human helper resolves faulty sub-assemblies at contingency buffer $\mathscr{C}(i+1, 1)$ in the $7i$-th step, and at contingency buffer $\mathscr{C}(i, 2)$ in the $8i$-th step. For the last cycle, faulty sub-assemblies at contingency buffer $\mathscr{C}(N, 2)$ are resolved in its seventh step, and there is no eighth step, hence $8N - 1$ total steps.

Strategy s^3 is similar to s^1 but with a slightly different protocol of resolving contingencies. Suppose, strategy s^1 had only seven steps. The human helper at each station resolves faulty sub-assemblies in the first buffer in the seventh step for odd cycles, and in the second buffer in the seventh step for even cycles.

Thus, a cycle for strategy s^3 is implemented as a concatenation of two cycles of strategy s^1 each curtailed to only seven steps. Thus, human $\mathcal{H}_i$ such that $i > 1$ resolves faulty sub-assemblies at contingency buffer $\mathscr{C}(i, 1)$ in its seventh step, and at contingency buffer $\mathscr{C}(i, 2)$ in its fourteenth

step. Human $\mathcal{H}_1$ resolves faulty sub-assemblies at contingency buffer $\mathscr{C}(i,1)$ in both the seventh and fourteenth step.

Contingencies of a different nature may also arise during assembly processes such that the human analyses the faulty sub-assembly and figures out that if he/she makes a minor fix and sends the part back to the assembly robot for another pass through the sub-assembly, then the robot is highly likely to succeed.

This procedure also saves the human's time because rather than fully correcting the error with the sub-assemblies, he/she is only making a minor fix and using the robot for the rest of the work. For example, a human decides that if a washer is slightly tweaked, then the assembly robot will succeed with the rest of the assembly.

Also, even if the human worker feels that the faulty sub-assembly might take significant time to repair, he/she decides to send the part back to the automated assembly line hoping the robot to repair it successfully. Since this model of the human worker is aimed at saving time, it is assumed that he/she takes only two minutes for the contingency resolution operation.

So it is assumed that the human worker succeeds in resolving contingencies with $p_{61} = 80\%$ probability, fails and puts the part back in the contingency buffer with $p_{62} = 5\%$ probability, and sends the part for repair by the robot with $p_{63} = 15\%$ probability. So, the contingency resolution rule mentioned in Section 5.4.3 gets transformed as follows.

```
[tick7] x21<max_units & x22<max_units & c21>0 -> p61 : (x22'=x22+1)&(c21'=c21-1)
+ p62 : true + p63 : (x21'=x21+1)&(c21'=c21-1) ;
```

The probability values may be varied by the user for different settings. This helps us simulate the human worker who will actively use the assembly robot multiple times to speed up his/her own repairing task. Applying this model of the human worker on strategy s^2 produces strategy s^4, and on strategy s^1 produces strategy s^5.

Table 5.1: Effect of initial part buffer on production time

Initial Part Buffer	Strategy 1 Production Time (in min)	Strategy A Production Time (in min)
0	45.17	43.10
1	45.07	42.90
2	44.93	42.70
3	44.75	42.50
4	44.55	42.30

Table 5.2: Effect of contingency resolution probability on production time

Contingency Resolution Probability	Strategy 1 Production Time (in min)	Strategy A Production Time (in min)
0.50	48.67	46.58
0.60	47.52	45.43
0.70	46.64	44.56
0.80	45.96	43.88
0.85	45.67	43.59
0.90	45.41	43.33
0.95	45.17	43.10
1.00	44.96	42.89

5.5 Model Checking and Verification

5.5.1 Run-time Analysis

The computational performance of different contingency resolution strategies under different operation settings is studied to gain better insights into the complexity of the process. The operating system is Ubuntu MATE 18.04.1 and workstation is Dell Precision Tower 3620 including eight Intel Xeon E3-1245V5 CPUs having 3.5GHz speed and a total of 32GB RAM. For a 5-station cell running strategy s^1, with $N_{max} = 2$ and $N^* = 3$, probabilities of 90% for all operations, and zero initial buffer, it took 82.09 seconds to compute the reachable states using breadth-first search in 529 iterations. The resulting DTMC had about 116 quadrillion states and 483 quadrillion transitions. The underlying transition matrix consisted of 0.5 million nodes. The expected time for the cell to produce three parts is then computed using PRISM's hybrid computation engine with the following expression given below:

Table 5.3: Computational performance of s^1 with varying parameters

# stations	N_{max}	N^*	Time for model building (in sec)	Time for model checking (in sec)
2	2	3	0.63	0.05
2	2	2	0.57	0.04
3	2	2	7.77	0.26
3	2	3	7.43	0.38
3	1	3	0.16	0.41
3	1	5	0.25	0.94
3	3	3	59.30	0.42
4	2	3	38.86	4.66
4	1	3	0.54	4.13
4	3	3	367.11	3.05
5	1	2	1.09	30.13
5	1	3	1.19	46.40
5	2	2	99.65	37.10

```
R{"time_duration"}=? [ F n53=3 ]
```

where `time_duration` is the label given to the reward structure defined in Section 5.4.4. The sampling method is used because the numerical method cannot handle such a large problem size. The expected production time is computed to be 31.81 minutes based on 95% confidence level and confidence interval width of 30 seconds. It took 828 sampling iterations and 59.24 seconds. The average path length explored was 66. If instead, there was a 2-station cell, then the DTMC could be solved numerically in 6.18 seconds with 88 Jacobi iterations requiring 64.6MB RAM for the expected production time of 14.19 minutes.

Table 5.3 records the computational time for strategy s^1 for the same cell setting as above by varying the three parameters that affect the problem size: N, N_{max}, N^*. Tables 5.4 and 5.5 do the same for strategies s^2 and s^3 respectively. As expected, strategy s^1 can accommodate larger problems sizes more efficiently than strategy s^2 which in turn can accommodate larger problem sizes more efficiently than strategy s^3. Changing N_{max} primarily increases the time for model building whereas N^* primarily increases the time for model checking. Strategy s^4 and s^5 have similar computational performance as s^2 and s^1 respectively.

5.5.2 Assembly Cell Analysis

The results for a single assembly station are analyzed first. All variables are given an upper bound of four, and the expected time to produce ten assemblies is to be computed. And the initial part buffer is zero for each station. An assembly cell is perfect if all success probabilities are one. The expected production time for a perfect cell is the lowest bound achievable for the given transition system. These values are 26 and 26.12 minutes and strategies s^1 and A respectively. It is assumed that a set of realistic values are used for the probabilities. It is assumed that the human worker succeeds with a probability of 95% while all robot's actions succeed with 80% probability.

Table 5.4: Computational performance of s^2 with varying parameters

# stations	N_{max}	N^*	Time for model building (in sec)	Time for model checking (in sec)
2	2	3	1.50	0.07
2	1	3	0.05	0.05
2	3	3	9.04	0.05
2	2	4	1.78	0.09
3	1	2	0.46	0.37
3	2	2	47.51	0.50
3	2	3	47.04	0.78
3	2	5	67.54	2.19

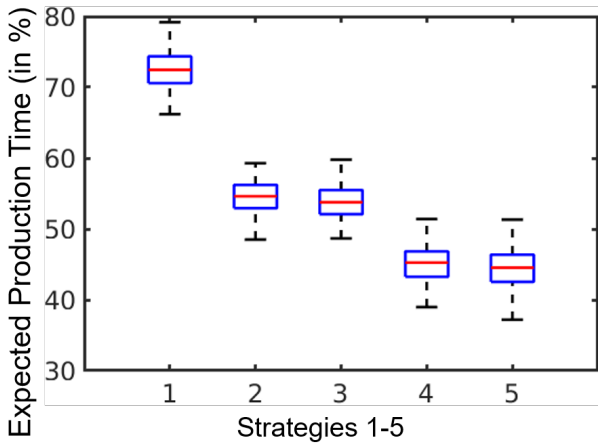


Figure 5.8: Effect of different strategies on expected production time

Table 5.5: Computational performance of s^3 with varying parameters

# stations	N_{max}	N^*	Time for model building (in sec)	Time for model checking (in sec)
2	2	3	1.45	0.08
2	3	3	7.43	0.07
2	3	4	8.87	0.08
2	4	5	34.89	0.13
3	2	3	18.25	0.55
3	3	4	219.01	0.90
4	2	3	133.89	6.34
5	1	3	2.91	119.00

Table 5.6: Effect of human characteristics on production time for strategy s^1

Contingency Resolution Probability	Contingency Resolution Duration	Expected Production Time (in min)
0.99	5	45.01
	7	46.72
	10	49.31
0.80	5	45.96
	7	48.05
	10	51.18

Table 5.7: Effect of human characteristics on production time for strategy A

Contingency Resolution Probability	Contingency Resolution Duration	Expected Production Time (in min)
0.99	5	42.93
	7	44.50
	10	46.82
0.80	5	43.88
	7	45.79
	10	48.61

Table 5.1 shows that increasing the initial part buffer from zero to four decreases expected production time by more than 30 seconds for both strategies s^1 and A which is very significant for factory throughput in the long run. The nondeterministic station comprising of cycles of four steps shows a decrease of 0.2 minutes per row because the addition of one item in the initial buffer helps skip one conveyor operation. This effect is not so vividly noticeable in the DTMC version.

Increasing the contingency buffer size does not have any notable effect. But increasing human resolution probability from 50% to 100% reduces time significantly for both strategies (Table 5.2).

Probabilistic model checking may also be used as a recruitment tool. Suppose a fast worker could finish resolving contingencies in five minutes, and an expert worker could succeed with a probability of 99%. Let a slow worker take ten minutes and a less skilled worker succeed with probability 80%. Then, the expected production times for all four combinations of the worker is recorded in Table 5.6 for strategy s^1 and in Table 5.7 for strategy A. For both strategies, a slow worker is a poor choice. Even if the slow worker does not take twice as much time, but only seven minutes even then a slow worker is not preferable. Thus, at 80% success rate for robot's actions, there does not appear to be any trade-off between being slow and being less expert in this mission scenario.

But these observations would vary for different robot's success probabilities. But this tool may be applied similarly to study those cases as well as more complicated assembly cell models where there are more human factors involved. Since the problem size above is small, numerical model checking methods are used in PRISM to generate the above results. Hence, multiple solution runs need not be executed because the numerical methods converge to the same number in every run.

The problem with analyzing multi-station assembly cells as MDPs is state space explosion which makes model building, as well as model checking, very memory-intensive and time-consuming, if not impossible. However, the DTMC protocol of strategy s^1 may be extended to multi-station assembly cells. A 2-station cell can be simply obtained by ignoring operations of station $\mathcal{A}_3$. For such a cell, suppose all variables are given an upper bound of two, and the expected time to produce three assemblies is to be computed. And the initial part buffer is one for each station. The expected production time for a perfect cell is 10.2 minutes.

Now let human succeed with 95% probability, robot assembly with 90% and other operations with 80%. The expected production time of strategy s^1 is compared with four other strategies in Figure 5.8 and the time is shown as the percentage increase with respect to that of the perfect

cell. The simulation is run 100 times to report statistics at 95% confidence level, and width of 0.5 minute. On an average, strategy s^5 performs the best but only marginally better than strategy s^4. Strategy s^2 shows significant improvement over strategy s^1 on account of using only one human to balance the task of two humans efficiently. However, when the two humans organize themselves better as in strategy s^3, then they outperform strategy s^2 by a small margin.

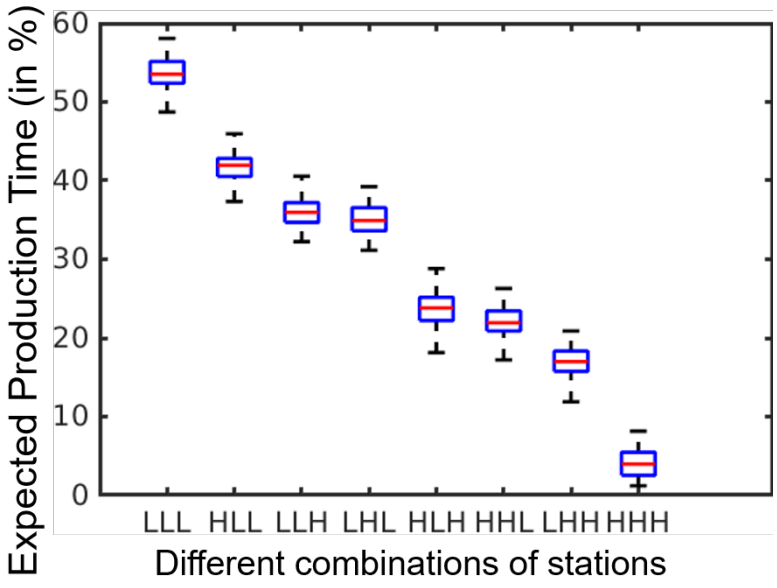
These strategies are compared only for 2-station cells because model checking for strategies s^2, s^3, s^4 becomes very time-consuming for 3-station cells. This is because these strategies required coding around 22 steps per cycle for three stations, and the modeling of the conveyor system triggers scaling issues. The conveyor system must take into account 2^6 possibilities arising due to storage-station interactions which for the three assembly stations leads to $3 * 2^6$ transition rules.

PRISM run-time statistics illustrate these issues clearly. Model construction for two stations with strategy 2 took 1.9 seconds, having around 4.3 million states and 6.5 million transitions. Statistical model checking took 0.14 seconds. But model construction for three stations with strategy s^2 took 21.4 seconds, and verification took 2.1 seconds.

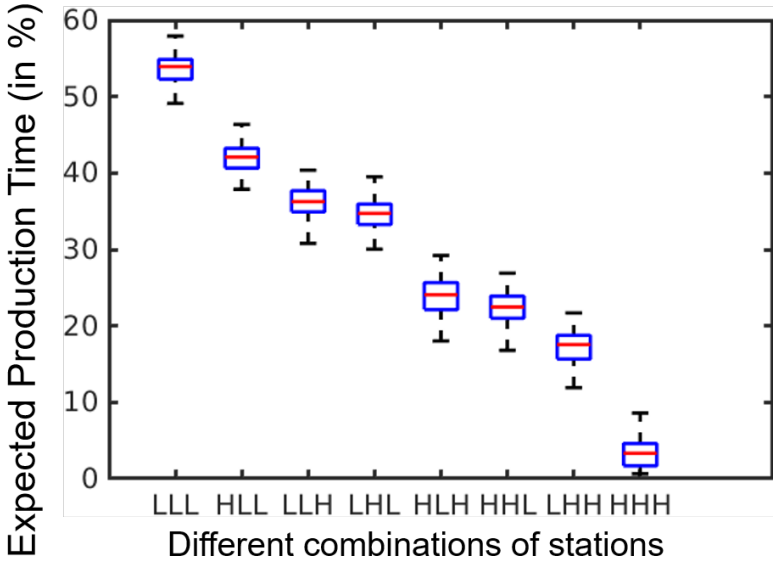
The 3-station assembly cell operating with strategy s^1 is considered next, and same variable upper bounds and initial part buffer are used as before. If all success probabilities within a station are at 90%, it is referred to as a *low* station. *High* stations have success probabilities throughout at 99%. The conveyor system is assumed to be operating perfectly. A perfect cell would take 12.8 minutes.

Figure 5.9a, shows the expected production time as the percentage increase with respect to that of the perfect cell for all 2^3 combinations of high and low stations in the cell. For example, LHL on x-axis denotes that the first and third stations are low, and the second one is high. The first station being low is not as critical as the next two stations being low.

A similar trend emerges for strategy s^5 (letting human helper 3-way split to be the same as $80 - 15 - 5\%$) in Figure 5.9b. At these high probabilities for the 3-station assembly cell, both strategies start performing similarly.



(a) Strategy s^1



(b) Strategy s^5

Figure 5.9: Effect of different combinations of stations on expected production time.

5.5.3 Aggregate Analysis of Large-scale Assembly Operations

This section focuses on decomposing large-scale assembly operations into smaller and relatively independent assembly cells, and then analyzing each cell using the probabilistic model checking

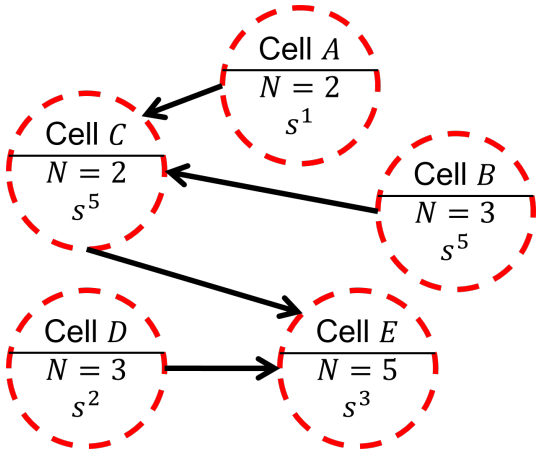


Figure 5.10: Scenario for multiple assembly cells interacting with each other

tool, PRISM. Finally, the results of model checking are aggregated at the highest assembly operation level to gain better insights into the system. A production line is set up across multiple assembly cells, as shown in Figure 5.10. The figure shows five cells labeled as A, B, C, D, E. It is assumed that the probability values representing operational efficiency are the same for each station within a cell. The specifications for each cell are given below:-

Cell A comprises of two stations and executes strategy s^1. $N_{max} = 3,\ N^* = 4,\ N^I = 2,\ p_1 = 90\%,\ p_2 = 90\%,\ p_3 = 90\%,\ p_4 = 90\%,\ p_5 = 90\%,\ p_6 = 95\%$.

Cell B comprises of three stations and executes strategy s^5. $N_{max} = 2,\ N^* = 5,\ N^I = 0,\ p_1 = 80\%,\ p_2 = 85\%,\ p_3 = 90\%,\ p_4 = 95\%,\ p_5 = 80\%,\ p_{61} = 80\%,\ p_{62} = 5\%,\ p_{63} = 15\%$.

Cell C comprises of two stations and executes strategy s^5. $N_{max} = 3,\ N^* = 3,\ N^I = 3,\ p_1 = 95\%,\ p_2 = 90\%,\ p_3 = 90\%,\ p_4 = 90\%,\ p_5 = 95\%,\ p_{61} = 80\%,\ p_{62} = 5\%,\ p_{63} = 15\%$.

Cell D comprises of three stations and executes strategy s^2. $N_{max} = 2,\ N^* = 3,\ N^I = 0,\ p_1 = 85\%,\ p_2 = 80\%,\ p_3 = 90\%,\ p_4 = 80\%,\ p_5 = 85\%,\ p_6 = 90\%$.

Cell E comprises of five stations and executes strategy s^3. $N_{max} = 3,\ N^* = 4,\ N^I = 1,\ p_1 = 85\%,\ p_2 = 80\%,\ p_3 = 90\%,\ p_4 = 87\%,\ p_5 = 86\%,\ p_6 = 92\%$.

The parts produced at cell A and cell B feed into cell C. Also, parts produced at cell C and cell D feed into cell E. So discrepancy between the operations at cell A and cell B may create

bottleneck situations for cell C, and similarly, cells C and D may create bottleneck situations for cell E. For example, cell A may produce parts at the rate of 3 minutes per part, and cell B at the rate of 5 minutes per part. So, cell C must wait for 2 minutes to let the part from cell B arrive assuming that all cells start their operations at the same time because cell C needs parts from both the cells to proceed.

PRISM places substantial constraints on the possible problem sizes that can be solved. Therefore, it is not computationally feasible to try to model the entire production line in PRISM. Each cell can be individually analyzed according to the modeling approach using PRISM software described in Section 5.4. The PCTL specifications may be used to compute the expected production time for N^* parts at each cell, say T^*. Thus, the time taken to produce a single part by each cell may be estimated to be $T_1 = T^*/N^*$. One could instead have queried PRISM to compute the expected time to produce one part which would be faster, but a better estimate can be found by averaging over the time to produce multiple parts because this is representative of a longer run of assembly operations. Using T_1 data for each cell, one may reason about the potential bottlenecks in a production line similar to Figure 5.10. The cell settings may be modified to minimize such bottlenecks.

Rather than the user writing a new PRISM script for each different cell settings which would be very time-consuming and prone to errors, an automated way is used to generate PRISM scripts in MATLAB for the specific parameter values of a cell. Thus, it becomes feasible to consider different possible cell settings and choose the ones which provide the best trade-off.

PRISM codes are generated for cells A, B, C, D, E automatically in MATLAB based on the input cell parameters. One can then query the expected time for the cells to produce N^* parts from which the expected time to produce one part can be approximated. The expected time T_1 for each cell to produce one part is given below.

Cell A requires 4.475 minutes, and cell B takes 5.894 minutes per part. These are expected values from statistical model checking in PRISM over 100 runs. Cell B is employing its optimal

strategy. However, cell A employs two humans and strategy s^3. So another strategy could be found where fewer humans are employed even if this results in worsening of cell A's production time as long as it does not overshoot 5.894 minutes. This is because cell C would not be able to proceed unless parts arrive from both cells A and B. If cell A switches to strategy s^2, then it employs only one human and brings down the operational costs significantly and takes 4.482 minutes per part.

Similarly, cells C and D must feed parts to cell E, but cell C takes 4.406 minutes and cell D takes 8.115 minutes per part. Cell C is employing its optimal strategy and requires two humans. However, even if one switches to strategies requiring fewer humans, the production time does not increase. In such cases, there may be a user preference to achieve decreased production times as it also leads to lower operational costs. However, the bottleneck issue remains unresolved because there is still going to be a significant gap between the time taken by cells C and D to produce their assemblies. So cell D's production time may be brought down without bringing any more humans. Cell D is employing strategy s^2 and requires only one human but if one switches to strategy s^4 it still requires one human and it brings down the production time to 7.673 minutes per part. The probability values associated with the human worker for cell D's strategy s^4 is set to $p_{61} = 80\%, p_{62} = 5\%, p_{63} = 15\%$. The few seconds saved as a result of this adjustment and the significant decrease in the operational costs due to reducing human workers translates to huge profits for the manufacturing operations in the long run. This analysis is fully automated in MATLAB, and the user does not need to interact with the model checker directly. Thus, the user only needs to specify his/her competing interests. For example, it was specified that the human costs as well as reduce bottlenecks must be brought down as much as possible. Simply optimizing the performance for each cell individually would not suffice. Here searching was restricted to strategies. The probability of human resolution for cell D was also chosen. This implies choosing a human worker with that specific skill level to work in the cell. However, one could search over other cell parameters like the initial part buffer, storage limits, and also

the other probability values, if that is allowed for the operation. That is again straightforward multi-parameter optimization problem and can be similarly encoded in MATLAB.

Cell E produces parts at the rate of 9.116 minutes per part. Based on these production rates, one may perform queue-theoretic analysis to study the generated throughput. However, that is beyond the scope of the present work.

Thus, with this approach, it is demonstrated with an example that analysis for fifteen assembly stations can be analyzed, whereas using PRISM, it is difficult to perform analysis for more than five stations efficiently. Similarly, large-scale assembly operations could be analyzed by decomposing it into smaller cells and reasoning about cell settings.

5.6 Summary

A simplified model of production lines and assembly cells is presented that mimics the real-world factory layout. Then this model is encoded in PRISM and PRISM's analytical tools are used to study the effect of different mission settings on the mission performance. The performance of different contingency resolution strategies is compared, thereby indicating the usage of model checking to plan for contingencies. The subsystems are analyzed individually, and their results are aggregated to evaluate the performance of the proposed operations plan for the entire system, thereby helping to design more optimized systems. It is demonstrated how the cell settings could be modified based on feedback from the model verification results to optimize production performance.

Chapter 6

Incorporation of Contingency Tasks in Nominal Task Allocation

In this chapter [1], a proactive approach for incorporating potential contingency tasks in task-schedules of multiple robots assigned to a complex mission is presented.

6.1 Introduction

The deployment of robot coalitions requires optimal task assignment. Such problems are generally formulated as multi-robot task allocation (MRTA) problem [222,223]. In this chapter, the problem scenarios considered involve single-task robots, multi-robot tasks, and a time-extended assignment problem (ST-MR-TA). However, it is solved using an iterative application of MRTA's instantaneous assignment formulation (ST-MR-IA). Since this can be formulated as a set partitioning problem which is strongly $NP-$hard, one naturally chooses to apply heuristics-based computationally efficient approximation algorithms. Heuristics approaches generally identify feasible tasks based on precedence constraints, enumerate feasible coalitions based on resource constraints, and then apply greedy heuristics for scheduling coalitions to tasks. The heuristics are designed to

[1] The work in this chapter is derived from the work published in [212, 221]

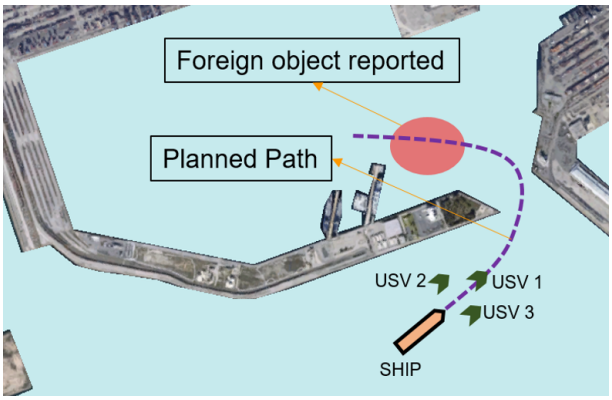


Figure 6.1: Example involving USVs where a contingency task impacts the mission

minimize the expected mission completion time or the maximum ending time of all the tasks. This is often referred to as makespan (or bottleneck) optimization.

In this chapter, there are additional features associated with the execution of mission tasks. A mission task can be interrupted, and its assigned robots can be reassigned to another mission task. A task may cause a bottleneck by keeping its assigned robots idle, which leads to poor resource utilization. This feature of interrupting tasks is incorporated to achieve lower mission completion times. Next, task execution may start even when a fraction of the team assigned to the task has arrived at the task location. This is because some robots in the team may take a larger duration to arrive at the task location. If robots arriving early remain idle while waiting for the other robots to arrive, the assigned coalition is not being properly utilized in finishing the task. When the minimum number of robots required by the task for execution have arrived at the task location, they start working and coordinating with other robots after they arrive to complete the task.

Cost-effectively deploying multi-robot teams requires mission planning to be automated. This involves the identification of tasks required for completing the mission while accounting for inter-task dependencies and then allocating robots to these *mission* tasks. The planned execution of mission tasks may get interrupted by the occurrence of unexpected events. The resumption of nominal mission workflow may require the execution of additional tasks. These new, additional

tasks are referred to as *contingency* tasks. Throughout the chapter, it is assumed that these unexpected events adversely affect the mission workflow because they require additional work for resolution and create unnecessary diversions. But they are not catastrophic because they do not cause mission failures and can be resolved through the execution of contingency tasks. Multi-robot teams should efficiently handle such contingency tasks to ensure the safety and security of their corresponding logistics operations. Below, three concrete examples to clarify the notion of contingency tasks are presented:

- ⋆ Figure 6.1 shows an example of a mission where multiple USVs must escort a ship to its designated berth location in a congested port. Routine mission tasks include: surveying the area and leading the ship to the dock. Suppose a monitoring system flags a foreign object that has been detected near the planned path of the ship, but this may be a case of false detection with a probability of 10%. For example, a small civilian vessel has appeared suddenly, and if the ship continues proceeding, it may generate waves that may be catastrophic for the occupants of the civilian vessel. This prompts the execution of a contingency task with 90% probability requiring the closest USV to go near the object/vessel and ensure that it does not collide with the ship by issuing warning signals or contacting port authorities for intervention. The escorting task can proceed only after this contingency task is completed.

- ⋆ Consider an automobile experiencing heavy traffic congestion so that it may run out of gas before ending at its desired goal location if the congestion persists for a long time. If that happens, the vehicle must reach a nearby filling station and refuel. The refueling operation is a contingency task. If the probability of traffic congestion being resolved is 90%, then the probability of refueling task impacting the mission is 10%.

- ⋆ Returning back to the example of the escorting ship. Suppose that debris has been spotted in or around a narrow channel. Due to water current, the debris may flow into the nearby channel with 50% probability and render the channel nonoperational for navigation purposes.

This gives rise to a contingency task with 50% probability requiring a nearby USV to go to the channel and flag the location of debris for immediate removal by port authorities. The ship and the escorting USVs need not arrive close to the channel for debris clearing to proceed. Only the assigned USV approaches the debris and coordinates with the central mission controller to clear the blockage for the convoy to proceed while the convoy itself may be stalled at a safer distance.

One may point out that these contingency tasks may be handled at the trajectory planning level if and when the need arises. Most literature dealing with contingency resolution and failure recovery is devoted to developing effective online strategies to the specific failures that may arise during their multi-robot missions. Then a distributed framework is implemented that results in the application of these specific failure strategies based on the just-in-time principle during the real-time mission execution. Most task allocation work does not consider contingency factors during the allocation and scheduling phases. Contingencies are handled on-the-fly. Thus, most works in contingency management in robotic task allocation paradigms do not involve centralized approaches.

Most of the related papers focus on devising specific methods for different failure modes, applying these to the robotic systems as and when required and analyzing the success rate. However, there are benefits in considering a centralized framework to handle these contingency tasks while accounting for their uncertainty. It is assumed that the tasks required to resolve failures and unexpected situations during mission execution are known in advance. The contingency tasks considered here are caused due to the nature of tasks that are being executed rather than being associated with the robots accomplishing these tasks. This means that if robots are unable to finish a task, then it is not due to shortcomings on the part of robots or robot failures. It happens only because the task cannot be finished, and other tasks (contingency tasks) must be finished first to finish the original tasks. The crux of such centralized operations is to allow

human cooperation and intervention because centralized mission control architectures comprise of a centralized mission controller (CMC) where mission execution is monitored by human operators who can interrupt the mission execution and remotely control the robots whenever such a need arises.

It is undesirable to only implement a low-level (trajectory planning level) reactive contingency management module because it renders the role of CMC useless in contributing to preparedness towards probable events of mission failures (contingencies). When human operators are in the loop, the system needs to be designed such that there are strong prior beliefs regarding which mission tasks will most likely be interrupted, and these can be updated during mission execution. Humans have a better chance of contributing to contingency preparedness with the aid of such systems. As seen in the multi-USV ship escorting example, one can strongly predict which contingencies may occur and how to prepare the robot teams for them. As more and more activities in ports are automated, it would become easier to predict situations encountered during mission execution. Therefore, it becomes more efficient to address the issue of contingency tasks at the mission planning level itself. This may be augmented with the implementation of the low-level reactive contingency management module for better success guarantees.

To incorporate a contingency task in a mission plan, one only requires to add new tasks to the mission plan. The central ground station monitoring situational awareness for the mission detects these *potential* contingency tasks where the word *potential* signifies the associated uncertainty and a non-zero probability of the mission getting completed without interruption by these contingency tasks. Such potential contingency tasks may be handled using these two simple approaches. A naive approach to incorporating the contingency task in the mission plan is to update the mission plan by inserting all of these new tasks. However, there is an uncertainty associated with these contingency tasks impacting the mission. So there is a non-zero probability associated with some of these *potential* contingency tasks to not impact the mission execution at all. This must be taken into account while making decisions regarding the contingency tasks. As outlined above,

a conservative approach would incorporate all the reported potential contingency tasks without taking into any consideration the probabilities of the contingency tasks to impact the mission adversely. A reactive approach is based on the real-time detection of a contingency task depending on the robot's current situational awareness. For example, when a task is ready for execution by the assigned coalition, the mission planner checks whether its associated contingency task needs execution. Thus, this approach only resorts to the execution of those contingency tasks which are triggered just before the execution of the associated mission task. Both of these approaches are seen to produce suboptimal results with respect to the minimization of the expected time of mission completion. This is because both approaches have obvious shortcomings. The conservative approach may execute mission tasks that have no impact on the mission workflow whereas the reactive approach may lead to robots of a coalition waiting in the idle mode while a single robot is finishing the contingency task. It would have been better to handle the contingency task earlier.

Based on the above discussion, a proactive approach is presented in this chapter to incorporate the potential contingency tasks. It tries to address the contingency tasks in the initial stage of mission execution based on their probabilities. There is a strong assumption on the nature of contingency tasks. The characteristics of the contingency tasks should not vary with time or be dependent on the percentage of mission completion. It is only then possible to address these contingency tasks independent of the progress in mission workflow.

In this work, several heuristics are applied to allocate tasks to the robots for the contingency-aware multi-robot task allocation (MRTA) problem. These heuristics are improvements based upon the simple greedy method. A multi-heuristic scheduling framework has been developed which can choose the best heuristic for the mission scenario based on the task characteristics. The main goal is to account for contingency tasks effectively. Based on the above reasons, it is clear that one must adopt a proactive approach to manage contingency tasks. It is also evident that the proactive approach requires information coordination among the robots regarding the required tasks. This cannot be achieved using distributed and decentralized techniques. The

proactive approach is deliberative and can be implemented only through a centralized architecture. But the proactive approach is bound to be computationally expensive as will become evident in Section 6.5. For five contingency tasks, an exhaustive method must run the mission planning module $4^5 \sim 1000$ times. For a nominal scenario of 50 tasks and six robots, the running time is 0.1 seconds in Matlab. Thus for solving the contingency scenario consisting of five contingency tasks, total time spent during computation is approximately 100 seconds. Assuming a ten-fold speed-up when deploying the technology in a low-level language, one gets a total running time of 10 seconds. Running times of such orders are completely agreeable for a centralized mission planning approach. But a heuristic is also developed to prune the decision search space of the contingency tasks, thus significantly bringing down planning run-time further. This allows the handling of a larger number of contingency tasks.

This chapter is organized as follows. Section 6.2 presents a motivating example of a multi-USV mission scenario to describe the nature of the problem solved in this chapter and its applicability. This is derived from the case study of Chapter 4. Section 6.3 provides the problem formulation. Section 6.4 describes the heuristics-based approach to solve the proposed MRTA problem while accounting for uncertainty. Section 6.5 outlines the approach of incorporating the contingency tasks in the nominal mission plan while minimizing the expected mission ending time. Results based on both the task allocation as well as contingency handling methodologies are then presented in Section 6.6.

6.2 Background

Following is a motivating example that helps in setting up the problem scenario for this chapter. Suppose a team of USVs in a port region $(u_1, u_2, \cdots, u_n)$ has been assigned to the task of assisting an arriving ship to dock safely to its designated berth location. There may be multiple such sea vessels like cruise or container ships proceeding towards their berth. Busy ports are also

accompanied by the high traffic of civilian/recreational vessels. The free region where the vessels are allowed to travel is known. Since the present chapter considers planning only at the mission level, one need not concern with any port-related restrictions on vessels speeds like minimum-wake zone. These are taken care of at the trajectory planning level. It is assumed that the entire environment is accessible to the central mission planner. The job of this centralized mission planner is to generate tasks that need to be executed and assign it to the USVs.

6.2.1 Environment Modeling

The mission planner keeps track of the relevant environmental factors by maintaining a list of environmental variables which it keeps updating based on sensory information feedback. Even today, ground stations guiding sensitive military missions or NASA control center controlling space missions perform similarly by gathering sensory data from numerous sensors fitted across the mission domain. These environmental variables are either continuous or Boolean. USV characteristics like the current vessel position and task characteristics like an assigned location where a task will be executed are modeled using continuous variables. These may be discretized due to constraints from computational resources. On the other hand, the Boolean variables are used to keep track of qualitative factors that are imperative for mission completion. They can only hold two values: true ($\top$) or false ($\bot$). A complete enumeration of such variables for the example mission is as follows:

- ℓ_1: has ship arrived in port
- ℓ_2: has docking location been assigned to ship
- ℓ_3: has ship path been computed
- ℓ_4: have survey areas been computed
- ℓ_5: have USVs been assigned to survey areas

- ℓ_6: have survey areas been partitioned among USVs
- ℓ_7: have coverage way-points been computed
- ℓ_8: have USVs surveyed regions of interest
- ℓ_9: have USVs approached the ship
- ℓ_{10}: have USVs escorted ship to the docking position
- ℓ_{11}: is ship at the docking position
- ℓ_{12}: has collision risk been detected

6.2.2 Resource Modeling

Tasks require resources for their execution. Motion-based tasks are executed using USVs. Computational tasks are executed using the computing power available at the central mission planner location as well as the individual computers mounted on each USV. In general, it is assumed that the resources are homogeneous with respect to their tasks.

Resources place additional constraints on task execution. In addition to a task's precondition for execution to be satisfied, it requires the assigned resources to be available at the beginning of task execution. Otherwise, the task remains halted until the required resources are freed up, leading to idle time during mission execution. Therefore it is important to utilize resources in such a way that the mission makespan is minimized.

However, the tasks do not tend to grab as much free resource as is available at the beginning of their execution because this may result in inefficiency. For example, even though three USVs may be available for surveying a region of interest, it may turn out to be faster to use only two USVs which are much closer to the region and hence can complete the area coverage in less time.

6.2.3 Task Modeling

Tasks considered here are atomic or elemental. Once assigned to the agent(s), they can be finished without requiring any further decomposition. As already mentioned, tasks can be of two types. Motion-based are related to trajectory tracking or motion control of USVs. Computational tasks can only be done using the computing resources and result in the computation of data necessary for mission completion or even initiation of mission execution.

These tasks may also be classified based on whether they are planned for before mission execution (offline) or during mission execution (real-time). For example, if USVs suddenly come across a civilian vessel obstructing their paths or the ship's path, then the USV closest to the concerned civilian vessel must come near it and issue a warning to clear the path. This is an example of a contingency task which cannot be planned beforehand. The central mission planner plans only for the offline tasks initially and then keeps updating this plan by introducing contingency tasks in the task network as and when required.

6.2.3.1 Mission Tasks

To be able to compute the task precedence network from the mission requirements, one must model the mission tasks $\tau \in \mathcal{T}$ as $5-$tuple $\tau = (\mathcal{I}, \mathcal{P}, \mathcal{R}, \mathcal{E}, \mathcal{T})$ where:

i. $\mathcal{I}$ denotes the information required as input for the robots to be able to execute the task. In the case of motion-based tasks, for example, one requires the USV positions.

ii. $\mathcal{P}$ represents the preconditions that must be satisfied before the task can be executed using robots or computing agents.

iii. $\mathcal{R}$ refers to the resource demand by the tasks to be finished.

iv. $\mathcal{E}$ records the effects of task execution relevant to mission progress. Motion-based tasks may change the continuous variables representing environment states (external effects) as well

Table 6.1: Computational tasks

Tasks	Inputs	Preconditions	Effects
monitor ship arrival in port (τ_1)	• Ship real-time position • Port region where ship must arrive	• $\ell_1 = \bot$	• ℓ_1 updated
assign docking position (τ_2)	• Ship docking position • Ship position	• $\ell_1 = \top$ • $\ell_2 = \bot$	• ship assigned docking position • $\ell_2 = \top$
compute ship path (τ_3)	• Ship docking position • Ship position	• $\ell_1 = \top$ • $\ell_2 = \top$ • $\ell_3 = \bot$	• ship path computed • $\ell_3 = \top$
compute areas to explore (τ_4)	• collision-free path	• $\ell_1 = \top$ • $\ell_3 = \top$ • $\ell_4 = \bot$	• areas of interest computed • $\ell_4 = \top$
assign USVs to areas (τ_5)	• areas to be explored • available USVs	• $\ell_1 = \top$ • $\ell_4 = \top$ • $\ell_5 = \bot$	• Assignment of USVs to areas • $\ell_5 = \top$
partition areas among USVs (τ_6)	• areas to be explored • available USVs	• $\ell_1 = \top$ • $\ell_4 = \top$ • $\ell_5 = \top$ • $\ell_6 = \bot$	• Partitioning of areas of interest among USVs • $\ell_6 = \top$
compute coverage way-points for survey areas (τ_7)	• areas to be explored • available USVs	• $\ell_1 = \top$ • $\ell_4 = \top$ • $\ell_5 = \top$ • $\ell_6 = \top$ • $\ell_7 = \bot$	• coverage way-points computed • $\ell_7 = \top$

as update boolean variables (internal effects). On the other hand, computational tasks may update boolean states or issue instructions to the available agents.

v. $\mathfrak{T}$ denotes the logical conditions which must be fulfilled in order to declare the successful termination of the task. This allows the handling of the temporal nature of tasks. When tasks being executed get finished, all the appropriate environment states must be updated, and this is achieved through the verification of these logical termination conditions. It allows the mission planner to keep track of the goal and perform feedback-based planning.

The above task elements have been defined in Tables 6.1 and 6.2 for all the computational and motion-based tasks which are considered in the illustrative mission scenario. The column for the termination conditions for all computational tasks has been omitted because these refer to the completion of the corresponding computations. So it is trivial to specify the same every time.

The resources required by the tasks have also not been specified because it is assumed that the entire pool of agents that is free and relevant to the concerned task may be utilized.

Once the task network has been computed, coalition formation and scheduling of tasks remain. To solve this MRTA problem, it suffices to model the mission tasks $\tau \in \mathcal{T}$ more succinctly as $4-$tuple $\tau = (\mathbf{r}_{in},\ \mathbf{r}_{out},\ \rho \in [\rho_{min}, \rho_{max}],\ t^e_{min})$. The operator $\mathbf{r}(\cdot)$ refer to the spatial position operator throughout this chapter.

It is assumed that once the robots are assigned to task τ, they must arrive at the location $\mathbf{r}_{in}(\tau)$ and when they have finished the task they end up at location $\mathbf{r}_{out}(\tau)$. In reality, it may be the case that different robots start their task execution from different positions depending on various other factors, and may then consequently end up being in different locations at the end of the task.

However, such complexities may also be handled more efficiently at the lower level planning once the mission has been planned and confirmed for execution. The number of robots ρ assigned to the task τ must lie in the range $[\rho_{min}, \rho_{max}]$. Such resource constraints are included to model tasks which cannot be started until a minimum number of robots have arrived at the task location $\mathbf{r}_{in}(\tau)$.

There must also be an upper bound on the coalition size assigned to the task because multi-robot teams require higher overhead charges like communication robustness, accurate localization, and sufficient battery levels. When ρ robots work on task τ, they take time t^e whose calculation is discussed in Section 6.4.1. The time t^e_{min} taken by ρ_{min} robots to finish the task is known beforehand and used to compute t^e. To account for uncertainty, it is modeled as a Gaussian distributed variable $t^e_{min} \sim \mathcal{N}(\mu,\, \sigma^2)$.

The mission proceeds by identifying feasible tasks, executing these identified tasks, and then recomputing the next set of feasible tasks. This keeps repeating until the mission goals have been successfully achieved. Figure 6.2 shows a task precedence graph of six mission tasks. This directed acyclic graph may be interpreted as follows.

Table 6.2: Motion tasks

Tasks	Inputs	Preconditions	Effects	Termination
survey region of interest (τ_8)	• coverage trajectories • USV positions	• $\ell_1 = \top$ • $\ell_7 = \top$ • $\ell_8 = \bot$	• region surveyed by USV • $\ell_8 = \top$	• when survey completed
approach ship (τ_9)	• Ship position • USV positions	• $\ell_1 = \top$ • $\ell_2 = \top$ • $\ell_8 = \top$ • $\ell_9 = \bot$	• ship approached by USV • $\ell_9 = \top$	• when approach completed
escort ship (τ_{10})	• Polygon representing docking region • Ship position • USV positions	• $\ell_1 = \top$ • $\ell_2 = \top$ • $\ell_8 = \top$ • $\ell_9 = \top$ • $\ell_{10} = \bot$	• ship escorted by USV • $\ell_{10} = \top$	• when ship enters docking region
station-keeping (τ_{11})	• Ship docking position • Ship position • USV positions	• $\ell_2 = \top$ • $\ell_{10} = \top$ • $\ell_{11} = \bot$	• ship docking monitored by USV • $\ell_{11} = \top$	• when ship docked
issue warning (τ_{12})	• Ship docking position • Ship position • USV positions	• $\ell_2 = \top$ • $\ell_{10} = \top$ • $\ell_{11} = \bot$ • $\ell_{12} = \top$	• danger to ship cleared • $\ell_{11} = \top$	• when warning issued and acknowledged

Initially only tasks τ_1 and τ_2 are feasible. After they are completed, three more tasks become feasible, viz. τ_3, τ_4 and τ_5. Once task τ_1 is finished, task τ_4 may be executed. Similarly, once task τ_2 is finished, task τ_5 may be executed. Only task τ_3 requires that both task τ_1 and τ_2 are completed. This implies that a sophisticated scheduling algorithm must take such constraints into account when trying to minimize the total duration for which the robots are idle. Finally, task τ_6

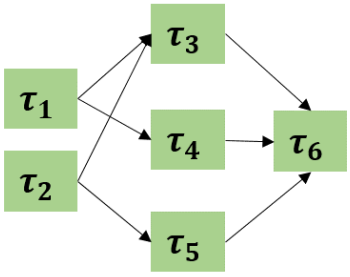


Figure 6.2: Example of task precedence graph

may only be begun when all three tasks τ_3, τ_4 and τ_5 are completed. Thus, mission proceeds in stages of multiple tasks.

6.2.3.2 Contingency Tasks

It is assumed that all contingency tasks with non-zero probability of impacting the mission are known as a set $\mathfrak{T}_c$. One can model each contingency task $\tau_c \in \mathfrak{T}_c$ as 5−tuple $\tau_c = (\mathbf{r},\ \rho \in [\rho_{min}, \rho_{max}], t^e_{c,min}, P^c, \tau_{in}, \delta)$. The first three elements have similar meaning as for mission tasks but some specifics will be elaborated below.

It is assumed that before mission planning, one checks based on the environmental conditions which contingency tasks may potentially impact the mission execution adversely. All contingency tasks with zero probability are ruled out. For the remaining potential contingency tasks, one could maintain a prior probability distribution $P^c(\tau_c)$ as a function of time and project this probability distribution into time as far as may be needed for mission completion. During mission execution, some new information may become available that updates this into a new posterior distribution and then replanning is done again just as at the beginning of mission execution but with newly updated information.

Contingency tasks are additional tasks inserted in the nominal task network from where it also derives its precedence constraints. The impact of a contingency task is analyzed through its effect on the task network and expected mission completion time t_M. A contingency task transforms the task precedence graph P_m by deleting specific edges and nodes and replacing them with new ones. This is shown in Figure 6.3 where the blue and green arrows are the directed edges from the preceding tasks of task τ_{in} and the orange arrows represent the directed edges to the next-in-line tasks.

Figure 6.3 shows that the contingency task τ_c splits the interrupted mission task τ_{in} into two fractions. In general, these fractions can be denoted by ξ_1 and ξ_2. The ξ_1 fraction of task τ_{in} remains unaffected by task τ_c because its effects appear to hamper only the remaining task

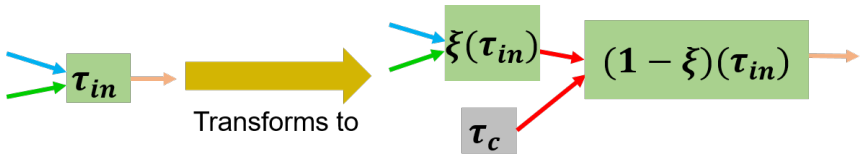


Figure 6.3: General rule for graph refinement to incorporate a contingency task

execution. When a robot team finishes the first ξ_1 fraction of task τ_{in}, one would naturally assume that along with the workload of contingency task τ_c, only $1-\xi_1$ task-fraction of task τ_{in} remains to be done. However, in the generalized modeling of contingency tasks, one should include the cases where task completion requirements of the original task τ_{in} have been changed so that it is now required to finish task-fraction ξ_2 of task τ_{in} which may be different than $1-\xi_1$. For simplicity, it is assumed that $\xi_1 = 1-\xi_2 = \xi$ throughout this chapter. It is this case that has been depicted in the figure also. So it is assumed that the original workload of task τ_{in} remains unchanged and task τ_{in} has only been interrupted by a task τ_c which is then simply an extra task to be inserted in the mission. The general case can be easily handled by extending the framework described here.

So in this kind of graph refinement, the uninterrupted task-fraction ξ_1 is finished simultaneously with the contingency task. After both work-loads have been finished, the remaining task-fraction of task τ_{in} is finished. Figure 6.4 shows another type of graph refinement. Here, the uninterrupted task-fraction ξ_1 must be finished first and only then the contingency task may proceed. This case is useful in modeling scenarios where the contingency task was not incorporated at the beginning of the mission but is encountered during mission execution and hence

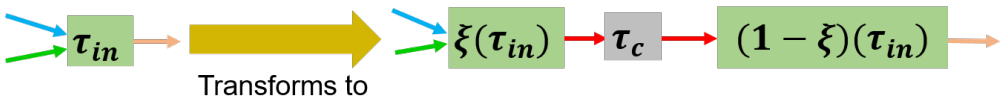


Figure 6.4: Rule for graph refinement when the contingency task was not properly incorporated in the nominal mission plan and encountered during mission execution

requires execution. In such cases, it is natural that the uninterrupted task-fraction is completed as expected, and then the agents reorganize to address the contingency task.

There is one important difference that sets the contingency tasks apart from the mission tasks. Contingency tasks must be finished before the interrupted task can be completed. On the other hand, the mission tasks constrain other mission tasks from starting unless they have been completed.

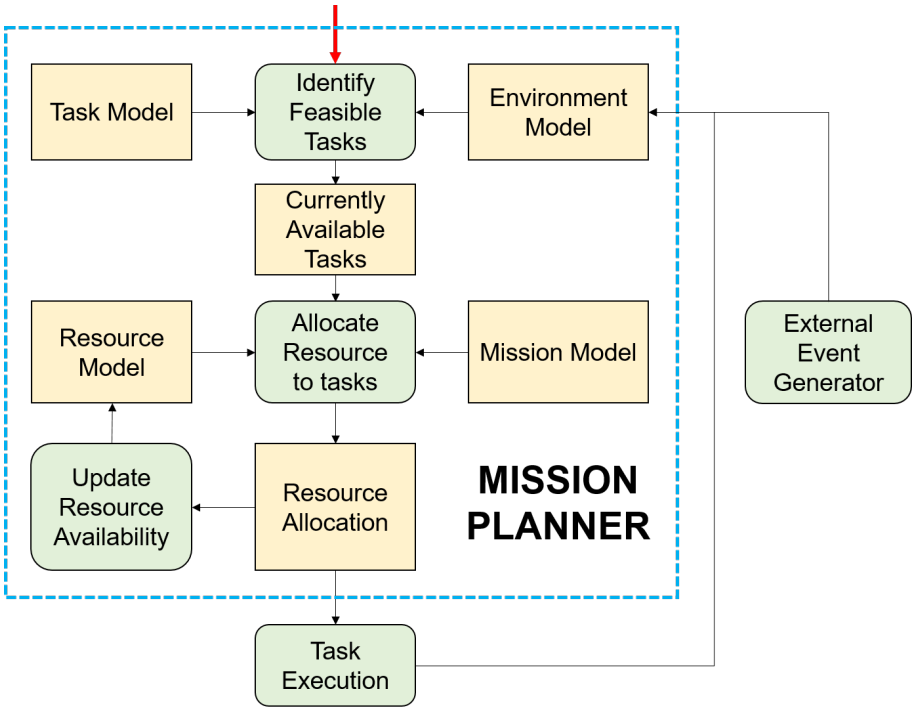


Figure 6.5: Overview of mission planning

6.2.4 Nominal Mission Planning

The initial mission specification generally includes an initial world state and the desired world state. Information regarding tasks and environment is assumed to be known beforehand. At the mission planning level, one starts by computing the set of all the feasible tasks. The nominal mission planner then allocates the right resources, instructs the robots to execute the feasible tasks, and records the new world state. If the new world state is the goal state, then the mission

terminates successfully. Otherwise, the current set of feasible tasks is again computed, and this loop must keep on repeating until the goal state is achieved. This scheme is shown in Figure 6.5. The external event generator is the simulator module used to simulate the occurrence of contingency tasks in real life.

If tasks are visualized as nodes, this scheme naturally results in a directed acyclic graph which encodes the task precedence constraints. Unlike search nodes in classical planning scenarios, the tasks referred to here have time durations and resource constraints associated with their execution. In the present example, the task network computed using this procedure is serial (a linear chain of the tasks described in Tables 6.1 and 6.2) because each new task becomes feasible only when its preceding task is finished. However, task networks may be more complex as will be encountered in subsequent studies. After computing the precedence network, the general mission workflow is known and the robots can be assigned to their tasks. These task assignments are communicated to the USVs who plan their trajectories with respect to the International Regulations for Preventing Collisions at Sea (COLREGs) [224].

6.3 Problem Statement

The exact problem that is solved with regards to the mission scenarios elucidated above is stated below, along with its inputs and outputs. The entire framework comprises of solving two problems sequentially. First, one must compute the coalition-task assignments and nominal schedules for mission execution (Section 6.4). The inputs and outputs for the same may be summarized as follows.

Given,

i. $R = [r_1, r_2, \cdots, r_n]$, a set of n robots available for the entire mission

ii. $\mathcal{T} = [\tau_1, \cdots, \tau_m]$, a set of mission tasks

iii. $P_m = (\mathcal{T}, E_m)$, a directed acyclic graph denoting the task precedence graph. The set $E_m = [(\tau_i, \tau_j), \cdots]$ consists of ordered pairs of tasks encoding the inter-dependence of the mission tasks.

iv. Coalition inefficiency α

Compute nominal mission execution plan comprising of task schedules for each robot so as to,

i. Minimize the expected mission completion time while accounting for:

- Traveling uncertainty
- Task execution uncertainty

ii. Satisfy all task precedence constraints

Coalition inefficiency factor α will be explained in detail in Section 6.4.1. Expression 6.1 denotes an example task schedule for robot r_i which is explained as follows. Robot r_i travels to task τ_a and then executes it [this is denoted by $\text{Trv}(\tau_a) \rightarrow \text{Exc}(\tau_a)$]. The robot then waits for task τ_b to become feasible and this action is denoted by [$\text{Wait}(\mathcal{P}(\tau_b))$]. When task τ_b becomes feasible for execution, robot r_i travels to the task's location denoted by [$\text{Trv}(\tau_b)$], waits for ρ_{min} robots to arrive because they are required by the task for execution. This waiting action is denoted by [$\text{Wait}(\rho_{min}(\tau_b))$]. Then the robot begins the task execution which is denoted by [$\text{Exc}(\tau_b)$]. This is how all the robots' schedules are expected to behave like.

$$\begin{aligned} r_i : \text{Trv}(\tau_a) \rightarrow \text{Exc}(\tau_a) \rightarrow \text{Wait}(\mathcal{P}(\tau_b)) \rightarrow \\ \text{Trv}(\tau_b) \rightarrow \text{Wait}(\rho_{min}(\tau_b)) \rightarrow \text{Exc}(\tau_b) \end{aligned} \tag{6.1}$$

After computing a nominal mission execution plan, one then incorporates the contingency information (Section 6.5).

Given,

i. Nominal mission completion plan $\mathcal{M}_n$

ii. $\mathcal{T}_c = [\tau_{c1}, \cdots, \tau_{cm_c}]$, a set of contingency tasks

iii. Coalition inefficiency α

iv. Fraction of contingency task completed for penalty type 2, κ (Algorithm 5)

v. Uncertainty associated with robot when traveling: σ_a (Section 6.4.4)

vi. Uncertainty associated with task execution τ_j: σ_{ej} (Section 6.4.4)

Compute,

i. Task schedule for each robot that does the following:

(a) Minimize the expected mission completion time accounting for:

- Traveling uncertainty
- Task execution uncertainty
- Known probabilities for contingency tasks impacting the mission

(b) Satisfy all task precedence constraints

ii. Replanning trigger conditions for those contingency tasks which have been deferred for now

Expression 6.2 provides an example task schedule for robot r_i when contingency tasks are incorporated. According to the given schedule, robot r_i first travels to the contingency task location denoted by [Trv(τ_a^{ctg})]. It then finishes the contingency task denoted by [Exc(τ_a^{ctg})], and travels to the mission task location denoted by [Trv(τ_a)]. Upon arrival, it waits for ρ_{min} robots to arrive at the task location denoted by [Wait($\rho_{min}(\tau_a)$)], and then begins task execution denoted by [Exc(τ_a)].

$$\begin{aligned} r_i : \text{Trv}(\tau_a^{ctg}) \rightarrow \text{Exc}(\tau_a^{ctg}) \rightarrow \\ \text{Trv}(\tau_a) \rightarrow \text{Wait}(\rho_{min}(\tau_a)) \rightarrow \text{Exc}(\tau_a) \end{aligned} \tag{6.2}$$

6.4 Task Schedule Optimization

In this section, strategies are described to form robot coalitions and assign them to the mission tasks. It is assumed that a robot performing its tasks is not affected by the environmental changes caused by other robots. To decide which coalitions to assign to which tasks, it is important to be able to score coalition-task pairs. For this purpose, a fractional task completion model [212] is used to compute the time required to finish the task by the assigned coalition.

6.4.1 Task Execution by Partial Teams

When a higher number of robots are available, then the task takes lesser time to finish. It is assumed that an inverse relationship exists between the task execution time and the coalition size. However, in real-life, a bigger coalition size leads to additional overhead costs like communication channels, which pushes the duration of task execution upwards. To include this effect, a coalition inefficiency factor α is introduced whose value is assumed to be 30% when the coalition size is 10, and linearly interpolated for other coalition sizes. Thus, if ρ robots have been assigned to task τ and it can be assumed that the robots are already at the task location so that traveling actions are not required, then the time taken for finishing the task is given by:

$$t^e(\rho) = \frac{t^e_{min}}{\rho}\rho_{min}(1+\alpha) \tag{6.3}$$

Assume a coalition $C \subset R$ is assigned to task τ. If the coalition violates the task's resource constraints, then the pair is given a score of infinity, essentially discarding it from future considerations. Otherwise, the arrival times for the robots of the coalition to reach the task location are computed, and the robots are sorted in the increasing order of their arrival times. This reordered robot list looks as follows:

$$C^s = [r_{\pi(1)}, r_{\pi(2)}, \cdots] \tag{6.4}$$

where

$$\pi : \mathbb{N} \cap [1, |C|\,] \rightarrow \mathbb{N} \cap [1, |R|] \tag{6.5}$$

Task execution does not begin until the robot labeled as $r_{\pi(\rho_{min})}$ arrives. Let the amount of unfinished workload associated with task τ be Δ, initialized to be 1, because the task has not yet been started. When the task is completed, then $\Delta(\tau)$ will be assigned zero value. Otherwise, it varies between 0 and 1.

The task execution time is computed based on the mean values $\mu(\cdot)$. When robot $r_{\pi(\rho_{min})}$ arrives then the team of assembled robots begins task execution. They may complete the task on their own, if no additional robots were to arrive, in time δt_e given by

$$\delta t_e = \mu(t^e_{min})(1 + \alpha(\rho_{min}))\Delta \tag{6.6}$$

Suppose the robot $r_{\pi(\rho_{min}+1)}$ arrives in time δt_a. If $\delta t_e > \delta t_a$, then the task has only been finished partially by the time the $(\rho_{min} + 1)^{th}$ robot arrives. To update the amount of workload remaining at this instant, one can assume a direct relationship between the amount of task finished and execution duration (Equation 6.7).

$$\Delta \leftarrow \Delta - \frac{\delta t_a}{\mu(t^e_{min})(1 + \alpha(\rho_{min}))} \tag{6.7}$$

The arriving robot $r_{\pi(\rho_{min}+1)}$ teams up with the previously arrived robots to finish the remaining task and this reasoning is repeated until the task has been finished.

However, if $\delta t_e < \delta t_a$ then the task is finished before the next robot arrives. However, the other robots assigned to the task are now rendered useless. To penalize such assignments, the score given is the time taken by the last robot to arrive at the task location.

Thus, every other robot remains idle for different duration of time, and this contributes to increasing the chances of discarding this coalition-task assignment.

According to the above model, there are only two types of idle behaviors possible for the robots. The first kind of scenario occurs when the first $(\rho_{min} - 1)$ robots wait for robot $r_{\pi(\rho_{min})}$ to arrive. This happens in all cases whether the task finished before or after all the robots arrived.

The second kind of scenario occurs if the task finished before some robots have arrived at the task's location. In such cases, all robots other than the last arriving robot have to wait additionally till the time $\delta t_a(r_{\pi(|C|)})$.

Resource constraints may significantly impact the scheduling process because these place constraints on the number of robots that must be available for the mission execution. Inequalities 6.8 and 6.9 are also required for efficient mission execution. Otherwise, either the resource constraints must be revised, or more robots must be added. If Inequality 6.8 is not satisfied, then mission aborts unsuccessfully.

Next, the rationale for these inequalities is provided. Observe that the task networks decompose mission workflow into stages. Suppose for stage S, the set of feasible tasks is denoted by $\mathcal{T}_f$. Then every stage requires at least the sum of the minimum number of robots required for each of its feasible task. The maximum over this minimum is computed over all stages. The stage that requires the most number of robots to be feasible for execution must require less than the number of robots available, $|R|$, hence Inequality 6.8.

Similarly, every stage will require the maximum number of robots. Any more robots than that are just not needed because otherwise, they will remain idle during the execution of tasks in that stage. The second inequality is not strictly required, but its role becomes clearer during the discussion of benchmarking results presented in Section 6.6.1. One does not want to unnecessarily have a very large number of robots engaged in the mission only because it is feasible to do so, and the second inequality helps to enforce that.

$$\max_{S} \sum_{\tau \in \mathcal{T}_f} \rho_{min}(\tau) \leq |R| \tag{6.8}$$

$$\min_{S} \sum_{\tau \in \mathcal{T}_f} \rho_{max}(\tau) \geq |R| \tag{6.9}$$

For every stage, if $\sum_{\tau \in \mathcal{T}_f} \rho_{max}(\tau) < |R|$ then mission can still proceed but some robots will have to be idle. One simple way to address situations of such nature more efficiently may be the following. Suppose the mean position averaged over the locations of all tasks to which the robots have been currently assigned for a particular stage is $\bar{\mathbf{r}}(\mathcal{T}_f)$.

The robots which are further away from the position $\bar{\mathbf{r}}(\mathcal{T}_f)$ are discarded and remain unassigned to any task. While the assigned robots are finishing their tasks, the unassigned robots advance to position $\bar{\mathbf{r}}(\mathcal{T}_f)$. This ensures that robots are not dispersed significantly over time and remain relatively close to each other.

However, the task resource constraints can also be revised to avoid such a situation. If robots are not being utilized, then the number of robots assigned to the mission can be decreased. Such a situation also indicates that one may increase ρ_{max} values for tasks chosen based on user-defined criteria.

6.4.2 Handling Divisible Tasks

This section models an additional feature associated with task execution that allows tasks to be interrupted during their execution. In some real-world mission, such features are either specified by the user or strictly prohibited based on the nature of robots, tasks, and environment. It is useful to study the effect of task divisions.

This is not exactly the same as task preemption studied in processor scheduling literature. The mission tasks considered in this chapter may be thought of, to some extent, as surveying tasks using unmanned robots. Such tasks can be interrupted during their execution, meaning the

robots are reassigned to other tasks. This feature is referred to as the *divisibility of tasks* because workload of the tasks are divided on the Gantt chart (Section 6.6.1) when they are redistributed among the robots.

Algorithm 1 Compute Task Completion fraction (CTCF)

Input: $\tau, C_\tau, \alpha, T_1$
Output: Δ

Task remaining, $\Delta \leftarrow \Delta(\tau)$
Robots at $\mathbf{r}(\tau)$, $\Gamma \leftarrow \emptyset$
Time elapsed, $t \leftarrow 0$
$\mathcal{R} \leftarrow$ sorted C_τ in the order in which they will arrive at $\mathbf{r}(\tau)$
$t_a(\rho_{min}) \leftarrow$ time taken by ρ_{min} robots to arrive at $\mathbf{r}(\tau)$
remove first ρ_{min} robots from $\mathcal{R}$
if $T_1 < t_a(\rho_{min})$ **then**
  update robot positions
  $\Delta \leftarrow 1$
else
  while $t < T_1$ **do**
    if $|\Gamma| == |C_\tau|$ **then**
      $\mathbf{r}(r_i) \leftarrow \mathbf{r}(\tau)\ \forall\ r_i \in C_\tau$
      update Δ
      $t \leftarrow T_1$
    else if $|\Gamma| < |C_\tau|$ **then**
      $T_2 \leftarrow \text{FTCM}(\tau, \Gamma, \alpha)$
      $T_3 \leftarrow t_a(\mathcal{R}.\text{front}\,(\,))$
      if $T_3 == T_1 - t$ **then**
        $\delta t \leftarrow T_3$
        $|\Gamma| \leftarrow |\Gamma| + 1$
        $\Gamma \leftarrow \Gamma \cup (\mathcal{R}.\text{pop}\,(\,))$
        $t \leftarrow t + \delta t$
      else
        $\delta t \leftarrow T_1 - t$
        $t \leftarrow T_1$
      end if
      if $min(T_1, T_2, T_3) == T_2$ **then**
        $\Delta \leftarrow 0$
      else
        update Δ
      end if
      update robot positions based on work performed during duration δt
    end if
  end while
end if

Unlike processor tasks, the tasks are not being suspended to be resumed at a later point in time. However, robot states are updated and possible reassignments are considered to see if faster

mission completion can be achieved. Task execution may still get suspended if all robots assigned to a task are reassigned to another task and new robots are assigned to this task because the new robots will be traveling to the task and therefore task execution has been halted. Unless drastic scenarios are considered, such cases do not arise and are not observed in the experiments conducted for this chapter. Mostly, one or two robots from the current task are reassigned to another task based on optimality considerations.

Also, the tasks undergoing execution by the robots are allowed to be interrupted only when at least one task has finished. The motivation behind allowing this feature is to reassign the robots that recently finished their task. All the robots are rescheduled, and not just those who got freed up after finishing their task. This helps to explore the search space for high-quality solutions as fully as possible.

This procedure can also be interpreted as if robot schedules are getting interrupted and reassigned as soon as any task is finished. For finding optimal reassignments, one must account for robot positions before rescheduling carefully. Those robots that got freed up after finishing their task must still be at the location of their current task. But the other robots may be working at their respective task locations or in transit to their task locations.

It is required to compute the amount of workload remaining for each feasible task. Then the robots can be optimally rescheduled based on the updated workload remaining for each task. In such cases, the robots have worked on their assigned task until the time it took for the most recent task to finish, and this time is known. This requires solving of the inverse problem of the fractional task completion model: given a fixed duration T_1 for the execution of task τ, how much workload has been finished by its assigned coalition C_τ?

If the allowed duration $T_1 < \delta t_a(r_{\pi(\rho_{min})})$, meaning the duration lapses even before the first ρ_{min} robots arrive at the task location, then the full workload is remaining. Otherwise, the approach is similar to Section 6.4.1, where robots that are arriving at the task location are tracked one by one, and they contribute to the completion of the task's workload. Once a new robot has

arrived, it combines with the previous robots. So, the workload finished by this extended team before the arrival of another robot must be computed.

Suppose that this extended team requires time T_2 to finish the task whereas the next robot will arrive in time T_3. If $T_1 < T_2 < T_3$ or $T_1 < T_3 < T_2$ then the task is partially finished by the coalition up to the allowed duration T_1. The robot positions are also updated because some of them may be at intermediate positions while traveling to the task location. If $T_2 < T_3 < T_1$ then all robots have arrived and finished the task. If $T_2 < T_1 < T_3$ then the intermediate position of the robots is updated and the task is finished. If $T_3 < T_1 < T_2$ or $T_3 < T_2 < T_1$, then the arrival of the next robot is recorded, the intermediate positions of the robots that have not yet arrived are updated, and the remaining workload is computed. Then, the same procedure is repeated until the allotted duration T_1 is exhausted. A succinct implementation of this methodology is shown in Algorithm 1.

6.4.3 Task Scheduling Strategies

Algorithms in processor scheduling literature provide insights in designing techniques for multi-robot scheduling. These algorithms operate through two successive phases: task prioritization and robot selection. These algorithms can be modified for MRTA problems based on the following analogy. The processing of tasks on processors is analogous to the execution of tasks by robot coalitions, and inter-processor communication is analogous to inter-task traveling by robots. Strategies $\mathcal{S}_1 - \mathcal{S}_5$ are inspired from the processor scheduling literature [122]. However, coalitions rather than individual robots are assigned to the tasks in the second phase. This is because this chapter assumes multi-robot (MR) tasks as noted in the beginning.

6.4.3.1 Heuristics Employed

List scheduling algorithms use heuristics that provide an important method for developing fast algorithms for such intractable problems. Greedy heuristics are very common which order the

tasks in the increasing (or decreasing) order of their execution durations. However, there are more advanced heuristics designed in the processor scheduling literature to outperform the greedy methods. We shall adopt these methods in the multi-robot context and compare their performance.

Strategies $\mathcal{S}_3$ and $\mathcal{S}_4$ use *static levels* to prioritize the tasks. Static levels for the mission tasks can be computed by summing up the execution durations of all the tasks along the longest path to the terminating tasks in the task precedence graph. Tasks with higher static levels are sequenced before tasks with lower static levels.

If the execution duration of all the tasks is assumed to be one and then compute the static levels, it constitutes strategy $\mathcal{S}_3$, and this heuristic is based on the Levelized Min-Time (LMT) scheduling algorithm. On the other hand, if the actual execution duration provided for the mission tasks beforehand is used, then that strategy is $\mathcal{S}_4$, and this heuristic is based on the Highest Levels First with Estimated Times (HLFET) scheduling algorithm.

Strategy $\mathcal{S}_5$ computes upward (or downward) ranks of the tasks in such a way that the time spent in traveling to the task's location is also included in the heuristic used for prioritizing the tasks. This is based on the Heterogeneous Earliest Finish Time scheduling (HEFT) scheduling algorithm. If one computes upward ranks, then the tasks are ordered according to the decreasing order of their upward ranks and assign the coalitions which finish the tasks earliest in that order.

All the mission tasks that form the *leaf* nodes in the task network (having zero out-degree) are assigned upward ranks as $rank_u(\tau_E) = \overline{e(\tau_E)}$. Here, the execution time of task τ_E is obtained using the function $e(\cdot)$. Let $\overline{e(\cdot)}$ denote the average execution time of a task over all feasible coalitions. For other tasks in general, the ranks may be computed using the equation below

$$\text{rank}_u(\tau) = \overline{e(\tau)} + \max_{\tau' \in \text{succ}(\tau)} (\text{rank}_u(\tau') + a(\tau, \tau')) \tag{6.10}$$

Where $\text{succ}(\tau)$ refers to the direct successors of task τ in the precedence constraint graph, and $a(\tau, \tau')$ refers to the travel time from task τ to task τ'.

Algorithm 2 Strategy $\mathcal{S}_0$ without task division

Input: $R, \mathcal{T}, P_m, \alpha$
Output: $\mathcal{M}_n$

1: $\mathcal{M}_n \leftarrow \emptyset$
2: **while** $\mathcal{T}$ not empty **do**
3: compute feasible task-set $\mathcal{T}_f \subset \mathcal{T}$ from graph P_m
4: **while** $\mathcal{T}_f$ not empty **do**
5: $\mathcal{D} \leftarrow$ all distributions of robots among tasks $\tau \in \mathcal{T}_f$
6: compute set $\mathcal{D}_r \subset \mathcal{D}$ of distributions violating task-resource constraints
7: $\mathcal{D} \leftarrow \mathcal{D}_r \setminus \mathcal{D}$
8: $t^* \leftarrow \infty$
9: **for all** $d = (c, \tau) \in \mathcal{D}$ **do**
10: $t \leftarrow \infty$
11: **if** $\rho_{min}(\tau) \leq |c| \leq \rho_{max}(\tau)$ **then**
12: **if** $|R \setminus c| \geq \sum_{\tau' \in \mathcal{T}_f \setminus \tau} \rho_{min}(\tau')$ **then**
13: $t \leftarrow \text{FTCM}(\tau, c)$
14: **end if**
15: **end if**
16: **if** $t < t^*$ **then**
17: $d^* \leftarrow d$
18: $c^* \leftarrow c$
19: $t^* \leftarrow t$
20: $\tau^* \leftarrow \tau$
21: **end if**
22: **end for**
23: $\mathcal{M}_n \leftarrow \mathcal{M}_n \cup d^*$
24: $R \leftarrow R \setminus c^*$
25: $\mathcal{T}_f \leftarrow \mathcal{T}_f \setminus \tau^*$
26: $\mathcal{T} \leftarrow \mathcal{T} \setminus \tau^*$
27: update robot positions
28: **end while**
29: **end while**

6.4.3.2 Task Allocation Without Task Division

A naive strategy $\mathcal{S}_0$ would be to perform exhaustive enumeration over all possible coalition-task pairs and use this brute-force technique (Algorithm 2) to compute the optimal schedule. After computing the set of tasks whose preconditions are satisfied (Line 3), all the possible feasible distributions (Lines $5-7$) of robots among the tasks ($|\mathcal{T}_f| \leq |R|$) are computed. In Lines $8-23$, the optimal coalition-task pair is found and then both the task and the assigned coalition are removed from further consideration (Line $24-26$). The positions of the robots forming the

Table 6.3: Overview of strategies used for task allocation

Strategy	**Task prioritization heuristic**
$\mathcal{S}_1$	Pick tasks with lower execution duration first
$\mathcal{S}_2$	Pick tasks with higher execution duration first
$\mathcal{S}_3$	Start picking tasks with lower static levels computed by assuming unit execution duration
$\mathcal{S}_4$	Start picking tasks with lower static levels computed for actual execution duration
$\mathcal{S}_5$	Pick tasks with lower upward ranks first

assigned coalition are updated so that when the next batch of feasible tasks is generated, the optimal coalitions are selected appropriately.

Using n robots, 2^n coalitions could be formed. Not all of them will be considered for task assignment because the resource constraints are used to discard the violating ones. Given a set of feasible tasks $\mathscr{T}_f$, all the distributions of n robots among the feasible tasks are computed. Since such computations are of the order $O(|\mathscr{T}_f|^n)$, computers run out of memory for a high number of robots or tasks. For example, 16 GB of memory is insufficient, even for nine tasks and nine robots. One could improve this further by only considering such distributions of the robots among tasks where at least one robot is assigned to a task because $\rho_{min} > 1 \,\forall\, \tau \in \mathscr{T}$. This leads to $n! * S(n, |\mathscr{T}_f|)$ distributions where $S(\cdot, \cdot)$ denotes the Stirling number of the second kind. This still hogs the entire 16 GB of RAM for nine tasks and eleven robots.

Given a set of feasible tasks $\mathscr{T}_f$, those distributions of n robots among the feasible tasks are considered which do not violate resource constraints. If the problem specifications are within RAM's capacity, then strategy $\mathcal{S}_0$ computes the optimal mission assignment in the deterministic version of the problem that serves as the baseline. However, the memory requirements placed by strategy $\mathcal{S}_0$ are very restrictive. To efficiently solve problems of larger sizes, heuristics are used to search the space of high-quality solutions quickly. The heuristics used in Sections 6.4.3.2 and 6.4.3.3 are described using Table 6.3. Based on heuristics described in Section 6.4.3.1, strategies $\mathcal{S}_1 - \mathcal{S}_5$ are designed, which can handle problems of larger sizes much more efficiently.

Two additional strategies, strategies $\mathcal{S}_6$ and $\mathcal{S}_7$, are also introduced for benchmarking purpose. They do not comprise a task prioritization phase. They directly start selecting coalitions while handling the interference of coalitions among themselves. Two coalitions interfere if at least one robot overlaps between them. Strategies $\mathcal{S}_6$ and $\mathcal{S}_7$ are based on the *MinStepSum* and *MinInterfere* heuristics described in [1]. The implementation details of strategies $\mathcal{S}_1 - \mathcal{S}_5$ are provided in the context of task divisions in the next section.

6.4.3.3 Task Allocation with Task Division

It can be seen that when task divisions are allowed, lower mission completion times may be achieved because this only results in reducing the duration for which the robots remain idle. This is explained in detail in the context of strategies $\mathcal{S}_1 - \mathcal{S}_5$.

Strategy $\mathcal{S}_1$ (described as Algorithm 3) greedily orders the feasible tasks generated in the ascending order of their execution duration every time they are generated (Lines $3 - 4$). Then, the task with the highest priority is selected for assigning its optimal coalition. The set of all coalitions that can be formed using the available robots is computed and the coalition sizes are restricted to the minimum between the maximum robots allowed for the task under consideration and the number of robots available for assignment (Lines $7 - 8$).

For the coalition selection phase, another simple heuristic is used. The coalition that takes the minimum time to finish the chosen task is then computed (Lines $17 - 20$). The time taken by the coalition to finish the task using the fractional task completion model is computed (Line 14). The selected coalition should satisfy the resource constraints (Line 12). The selected coalition should be such that the number of robots available after the robots in the selected coalition are used up is at least as much as the sum of minimum numbers of robots required for the remaining tasks (Line 13).

Once the coalition assignments $\mathcal{M}_n$ have been computed (Line 22), the task τ^* which will get finished the earliest is determined (Line 25). Task executions are paused once any task gets

Algorithm 3 Strategy $\mathcal{S}_1$ (and $\mathcal{S}_2$) with task division

Input: $R, \mathcal{T}, P_m, \alpha$
Output: $\mathcal{M}_n$

1: $\mathcal{M}_n \leftarrow \emptyset$
2: **while** $\mathcal{T}$ not empty **do**
3: compute feasible task-set $\mathcal{T}_f \subset \mathcal{T}$ from graph P_m
4: sort $\mathcal{T}_f$ in ascending (descending for $\mathcal{S}_2$) order by task duration
5: **while** $\mathcal{T}_f$ not empty **do**
6: $\tau \leftarrow \mathcal{T}_f$.pop()
7: $N_f \leftarrow \min(\rho_{max}(\tau), |R|)$
8: $\mathcal{C} \leftarrow$ set of all coalitions in set R up to N_f robots
9: $t^* \leftarrow \infty$
10: **for all** $c \in \mathcal{C}$ **do**
11: $t \leftarrow \infty$
12: **if** $\rho_{min}(\tau) \leq |c| \leq \rho_{max}(\tau)$ **then**
13: **if** $|R \setminus c| \geq \sum_{\tau' \in \mathcal{T}_f \setminus \tau} \rho_{min}(\tau')$ **then**
14: $t \leftarrow \text{FTCM}(\tau, c, \alpha)$
15: **end if**
16: **end if**
17: **if** $t < t^*$ **then**
18: $c^*(\tau) \leftarrow c$
19: $t^*(\tau) \leftarrow t$
20: **end if**
21: **end for**
22: $\mathcal{M}_n \leftarrow \mathcal{M}_n \cup (c^*, \tau)$
23: $R \leftarrow R \setminus c^*$
24: **end while**
25: $\tau^* \leftarrow \arg\min_{\tau} \ t^*(\tau)$
26: $\mathcal{T} \leftarrow \mathcal{T} \setminus \tau^*$
27: **for all** $\tau \in \mathcal{T}_f \setminus \tau^*$ **do**
28: $\Delta(\tau) \leftarrow \text{CTCF}(\tau, c^*(\tau), t^*(\tau^*))$
29: **end for**
30: update robot positions
31: **end while**

finished and all the robots are redistributed among the new set of feasible tasks $\mathcal{T}_f^{new}$. This is done to prevent robots from being idle while waiting for the last task in the original set of the feasible tasks $\mathcal{T}_f^{old}$ to be finished.

Instead, the coalition assignment $\mathcal{M}_n$ is only adopted until the completion of the earliest task. Then task τ^* is marked as finished (Line $26 - 27$) and compute the workload remaining for other tasks belonging to the set $\mathcal{T}_f^{old} \setminus \tau^*$ (Lines $28 - 30$). This entire procedure of computing the new set of feasible tasks $\mathcal{T}_f^{new}$ and finding its corresponding coalition assignment is then repeated. By

Algorithm 4 Task prioritization phase for strategies $\mathcal{S}_3, \mathcal{S}_4, \mathcal{S}_5$ with task division

1: $\mathcal{M}_n \leftarrow \emptyset$
2: $\mathcal{T}^p \leftarrow$ prioritized $\mathcal{T}$ (Section 6.4.3.3)
3: **while** $\mathcal{T}$ not empty **do**
4: compute feasible task-set $\mathcal{T}_f \subset \mathcal{T}$ from graph P_m
5: $\mathcal{T}_f \leftarrow \mathcal{T}^p \cap \mathcal{T}_f$
6: **while** $\mathcal{T}_f$ not empty **do**
7: REST IS SAME AS ALGORITHM 3
8: **end while**
9: **end while**

striving to reduce the amount of time spent by robots being idle, task divisions lead to mission completing faster.

Other than the task prioritization heuristic, strategies $\mathcal{S}_3 - \mathcal{S}_5$ also differ slightly regarding how they prioritize the tasks as shown in Algorithm 4. These strategies prioritize the tasks (Line 2) even before the currently feasible tasks are computed, i.e., before Line 3.

When the feasible tasks are computed, then they are ordered by taking an intersection with the prioritized task list $\mathcal{T}^p$ while preserving the order in the task-list $\mathcal{T}^p$. Rest of the steps are the same as in Algorithm 3.

The strategies described in this section are derived from the list scheduling algorithms by adding the features of coalition formation, fractional task completion, and task division.

6.4.4 Accounting for Uncertainty

Robots suffer from controller noise. Accurate models of tasks are generally not available beforehand. Environmental factors cause deviation from conditions assumed at the beginning of mission execution for its planning. Due to the interplay between these characteristics of robots, tasks, and environment, uncertainty associated with mission execution must be addressed.

The time taken by robots to arrive at their designated locations to execute tasks becomes uncertain and is modeled as Gaussian distributed variables. For simplicity, it may be assumed that the variance of the associated normal distribution is the same for all robots regardless of the

tasks and environment. Execution durations of the mission tasks are also modeled as Gaussian distributed variables whose variances are again assumed to only depend on the tasks for simplicity.

The task allocation methods discussed above explore different possible mission assignments and choose the best solution based on some optimizing criteria. Mission *makespan* is generally used as the metric to be minimized. Using the principles of risk-neutral decision-making, that mission assignment is chosen, which requires minimum expected mission completion time for its completion.

When the mission tasks are divisible and stochastic in nature, mission completion time is computed as follows. Generally, a task precedence graph decomposes the mission into stages. In each stage multiple tasks are simultaneously being finished. One has to assign a coalition to each of the feasible tasks of that stage, say for the s^{th} stage $\tau_1, \cdots, \tau_{n_s}$.

To select the optimal coalition, the distribution of the time taken by the coalition to complete the task is computed. For concreteness, let $\mathscr{C}_k = [r_1, r_2, \cdots, r_{n_k}]$ be the k^{th} coalition being considered for mission task τ_j. Robot r_i must travel distance d_{ij} between robot position $\mathbf{r}(r_i)$ and spatial location of the task $\mathbf{r}(\tau_j)$ at mean speed $\overline{v}$.

The time taken to arrive at the task's spatial location is modeled as a Gaussian variable $T_{ij}^a \sim \mathcal{N}(d_{ij}/\overline{v}, \sigma_a^2)$. Thereafter it must perform the task for duration $T_{ij}^e \sim \mathcal{N}(t_{ij}^e, \sigma_{ej}^2)$ which can be calculated using the fractional task completion model.

The total task completion time of robot r_i is $T_{ij}^k = T_{ij}^a + T_{ij}^e$. The total task completion time for the coalition is given by

$$ {}^sT_j^k = \max(T_{1j}, \cdots, T_{n_k j}) \tag{6.11} $$

We can approximate the maximum of two Gaussian distributed variables to be another Gaussian distributed variable. This does not hold true in the strict sense but works for approximate calculations.

Assuming the Pearson correlation coefficient ρ_{corr} between the two Gaussian variables $X \sim \mathcal{N}(\mu_X, \sigma_X^2)$ and $Y \sim \mathcal{N}(\mu_Y, \sigma_Y^2)$ to be zero, their maximum is the distribution $Z = \max(X, Y)$ whose mean and standard deviation can be computed using [225].

$$\begin{aligned}\mu_Z = \mu_X \Phi\left(\frac{\mu_X - \mu_Y}{\theta}\right) + \mu_Y \Phi\left(\frac{\mu_Y - \mu_X}{\theta}\right) \\ + \theta\phi\left(\frac{\mu_X - \mu_Y}{\theta}\right)\end{aligned} \tag{6.12}$$

The value for mean can be computed as shown above. The value of the standard deviation can be computed as follows

$$\begin{aligned}\sigma_Z^2 = (\mu_X^2 + \sigma_X^2)\Phi\left(\frac{\mu_X - \mu_Y}{\theta}\right) \\ + (\mu_Y^2 + \sigma_Y^2)\Phi\left(\frac{\mu_Y - \mu_X}{\theta}\right) \\ + (\mu_X + \mu_Y)\theta\phi\left(\frac{\mu_X - \mu_Y}{\theta}\right) - \mu_Z^2\end{aligned} \tag{6.13}$$

where

$$\Phi(x) = \frac{1}{2}\left[1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right] \tag{6.14}$$

$$\phi(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right) \tag{6.15}$$

$$\theta = \sqrt{\sigma_X^2 + \sigma_Y^2 - 2 * \rho_{corr} * \sigma_X * \sigma_Y} \tag{6.16}$$

The maximum of multiple Gaussian variables is computed based on the approximation [226] given below.

$${}^{s}T_j = \max \cdots \max(\max(T_{1j}, T_{2j}), T_{3j}) \cdots T_{n_k j}) \tag{6.17}$$

The coalition $\mathscr{C}_{k^*}$ is assigned to task τ_j where

$$k^* = \arg \min_k(\mu(T_j^k)) \tag{6.18}$$

Stage completion times are calculated by minimizing over the stage tasks because as soon as any stage task finishes, all the robots are rescheduled and, hence another stage begins. To calculate the mission completion time, one must sum over the stage completion times.

$$ {}^{s}T = \min(T_1, \cdots, T_{n_s}) \tag{6.19} $$

The minimum of two Gaussian variables $X \sim \mathcal{N}(\mu_X, \sigma_X^2)$ and $Y \sim \mathcal{N}(\mu_Y, \sigma_Y^2)$ is the distribution $W = \min(X, Y)$ which is given as follows

$$ \begin{aligned} \mu_W = \mu_X \Phi\left(\frac{\mu_Y - \mu_X}{\theta}\right) + \mu_Y \Phi\left(\frac{\mu_X - \mu_Y}{\theta}\right) \\ - \theta\phi\left(\frac{\mu_Y - \mu_X}{\theta}\right) \end{aligned} \tag{6.20} $$

$$ \begin{aligned} \sigma_W^2 = (\mu_X^2 + \sigma_X^2)\Phi\left(\frac{\mu_Y - \mu_X}{\theta}\right) \\ + (\mu_Y^2 + \sigma_Y^2)\Phi\left(\frac{\mu_X - \mu_Y}{\theta}\right) \\ - (\mu_X + \mu_Y)\theta\phi\left(\frac{\mu_Y - \mu_X}{\theta}\right) - \mu_W^2 \end{aligned} \tag{6.21} $$

And the minimum of multiple Gaussian variables is approximated by using the iterative method that was used for calculating maximum over Gaussian variables.

6.5 Incorporating Potential Contingency Tasks

6.5.1 Background

Three different approaches to incorporate potential contingency tasks in mission planning were previously discussed. The conservative approach adds the contingency tasks into the queue if they have been reported with any non-zero probabilities. On the other hand, a reactive approach is based on just-in-time behavior. According to this approach, if there is a non-zero probability for

Table 6.4: Different types of penalty associated with available actions for contingency tasks

Action on potential contingency task τ_c	τ_c does not affect mission	τ_c affects mission
Defer action on τ_c for now	Case A (correct choice)	Case B (penalty type 1)
Immediately act on τ_c	Case D (penalty type 2)	Case C (correct choice)

contingency task τ_c to impact mission task τ, then just when task τ becomes feasible the planner checks whether contingency task τ_c presents a clear danger. If task τ_c requires execution based on the query at that instant, then the uninterrupted fraction of the task is first completed, next the contingency task is finished and finally, the remaining portion of the task is finished.

However, the proactive approach explicitly takes into account the probabilistic nature of the contingency tasks to decide which contingency tasks should be incorporated in the mission plan and which should be ignored till beliefs about the occurrence of contingency tasks change. Whenever a new potential contingency task is reported, or the beliefs associated with already reported contingency tasks change, the entire decision-making process for all contingency tasks must be redone. If a contingency task does not affect the mission, then the reactive approach will always perform better than or at least as well as the proactive approach. However, if the contingency task impacts the mission, the reactive approach pays a higher penalty and performs worse than the proactive approach.

Two actions are available for every contingency task τ_c: either immediately incorporate it in the task network or defer its incorporation. The planner evaluates all options and assesses their impact on the expected mission completion time. Table 6.4 shows different outcomes associated with the two actions based on the real-world scenario. To decide whether to defer incorporation or not, the hypothesis testing approach is followed. If one decides to defer action on task τ_c and it ends up impacting the mission, a penalty is incurred for keeping the robots associated with

Algorithm 5 Computation of contingency task networks through refinement of nominal task network

Input: $\mathcal{T}_c$, $\mathcal{T}_c^{cr}$, P_m
Output: P_c

$\mathcal{T}_c' \leftarrow \mathcal{T}_c \setminus \mathcal{T}_c^{cr}$
$\mathcal{W} \leftarrow \left[\{\top\}^{|\mathcal{T}_c^{cr}|}, \{\top, \bot\}^{|\mathcal{T}_c'|}\right]$
$\mathcal{C}_d \leftarrow \mathcal{W} \times \mathcal{W}$
$P_c \leftarrow \emptyset$
for all $(a, w) \in \mathcal{C}_d$ **do**
  $P_c' \leftarrow P_m$
  for all $\tau_c \in \mathcal{T}_c$ **do**
    $N \leftarrow$ node in graph P_c' for the mission task impacted by task τ_c
    if a defers action on τ_c and τ_c does not impact w **then**
      leave graph P_c' unchanged (case A)
    else if a defers action on τ_c and τ_c impacts w **then**
      add τ_c serially between the two task fractions (case B)
    else if a acts on τ_c and τ_c impacts w **then**
      refine network as in Figure 6.3 (case C)
    else if a acts on τ_c and τ_c does not impact w **then**
      $\mu(t_c^e) = \mu(t_c^e) * \kappa$ ($\kappa = 70 - 100\%$)
      refine network as in Figure 6.3 (case D)
    end if
  end for
  $P_c \cdot \text{add}(P_c')$
end for

the interrupted mission task idle because the contingency task becomes a bottleneck (Type 1 penalty). On the other hand, immediate action on τ_c may incur a penalty if contingency action did not impact the mission tasks because some resources were unnecessarily diverted which could have been used to continue mission workflow (Type 2 penalty).

If the number of contingency tasks that are reported is m_c, then there are $N_c = 2^{m_c}$ possible decision outcomes. There are also N_c different world scenarios that could materialize when robots are completing the assigned mission depending on whether a particular contingency task impacts the mission.

A naive exhaustive method proceeds as follows. For each decision, one could compute the expected mission completion times for all the N_c scenarios. This leads to $N_c \times N_c$ score matrix $\mathcal{S}$. To compute the expected mission completion time for a particular action-world pair (a, w), the fractional task completion model is applied on a task network derived from the original task network

excluding the contingency tasks. The set of all possible task networks $P_c = \left[P_{c1}, \cdots, P_{c(N_c-1)}\right]$ is computed when contingency tasks are incorporated. Each graph $P_{ci} = (\mathfrak{T}_c \cup \mathfrak{T}, E_c)$ is also a directed acyclic graph. The task networks P_m and P_c comprise of all N_c decision alternatives. To obtain set P_c, Algorithm 5 based on the case-by-case analysis in Table 6.4 is used. It uses the two types of graph refinement shown in Figures 6.3 and 6.4.

It is assumed that if mission task τ is interrupted by contingency task τ_c, then a single robot is commanded to resolve the contingency task. According to this operational model, $\rho_{min} = \rho_{max} = 1$ for all contingency tasks.

Given the probability vector $P = [p_1, \cdots, p_{N_c}]$ of the different world scenarios and the matrix $\mathcal{S}$, the vector $T_d = [T_{d1}, \cdots, T_{dN_c}]$ storing expected mission completion times for each decision is computed where $T_{di} \sim \mathcal{N}(\mu, \sigma^2)$. The optimal choice of decision i^* is risk-neutral.

$$T_{di} = \sum_{j=1}^{N_c} p_j \mathcal{S}_{ij} \tag{6.22}$$

$$i^* = \arg\min_i \mu(T_{di}) \tag{6.23}$$

6.5.2 Search-space Pruning Heuristic

In this section, a heuristic is proposed to reduce the computations in the exhaustive method. For every contingency task, $\delta_1^*(\tau_{ci})$ is computed by taking the difference between expected mission completion time for deferring and executing it while all other contingency tasks are deferred. Similarly, $\delta_2^*(\tau_{ci})$ is the difference between expected mission completion time when deferring and executing it while all other contingency tasks are executed. A contingency task that satisfies $\min(\delta_1^*, \delta_2^*) - 0.1 * t_c^e(\tau_{ci}) > 0$ is considered *urgent* and must be immediately incorporated in the task network. A contingency task that satisfies $\max(\delta_1^*, \delta_2^*) + 0.1 * t_c^e(\tau_{ci}) < 0$ is considered *deferred* and must be revisited when its probability changes. For other regular contingency tasks,

a combinatorial search is performed to determine their optimal actions. The rationale for the heuristic is described next.

6.5.3 Analysis of Pruning Heuristic

Using certain illustrative cases, the validity of the heuristic is demonstrated. Analysis for a general task network encompassing all possible scenarios cannot be performed, it is analytically shown for five nominal cases that this heuristic holds. In all these cases, contingency tasks are assumed to take more time to finish than mission tasks, but the observations hold even if the assumption is reversed. Consider a scenario involving three high-probability contingency tasks such that only one contingency task impacts the mission per stage of mission execution. Then the expected mission completion time for the decision to defer all contingency tasks is represented as $T_M(000)$. Suppose the time to finish the mission according to the nominal task network is $\overline{T}$. Then for each contingency task, the nominal execution is modified according to case B of Algorithm 5 implying that the network is serially extended by each contingency task so $T_M(000) \sim \overline{T}+t^e_{c1}+t^e_{c2}+t^e_{c3}$ where t^e_{ci} is the time taken to finish task τ_{ci}. To compute $T_M(100)$, observe that the first contingency task will be treated according to case C, so the first stage of task execution is extended by time $t^e_{c1} - t^{max}$ where t^{max} represents the maximum time taken by the mission tasks of the first stage to finish. Hence $T_M(100) \sim \overline{T} + (t^e_{c1} - t^{max}) + t^e_{c2} + t^e_{c3}$ and $\delta^*_1(\tau_{c1}) = T_M(000) - T_M(100) \sim t^{max}$.

Even if one chooses an ordering for the contingency task execution times, the conclusion holds regardless of which ordering is chosen. Suppose $t^e_{c1} \geq t^e_{c2} \geq t^e_{c3}$. To compute $T_M(011)$ the first stage adds $t^e_{c2} - t^{max}$ to $\overline{T}$ because τ_{c2} is clearly the bottleneck task. Similarly, $T_M(111) \sim \overline{T} + (t^e_{c1} - t^{max})$ and $\delta^*_2(\tau_{c1}) = T_M(011) - T_M(111) \sim t^e_{c2}$. The other pairwise differences, viz. $T_M(010) - T_M(110)$ and $T_M(001) - T_M(101)$ are t^e_{c2} and t^e_{c3} respectively. These values are clearly contained in the range $[\delta^*_1, \delta^*_2]$. The same is observed when analysis is performed for the other two tasks.

Suppose all contingency tasks have low probability. Then $T_M(000) \sim \overline{T}$ because this corresponds to application of case A. To compute $T_M(100)$ one can invoke case D for task τ_{c1} and also assume $\kappa = 1$ for simplicity so $T_M(100) \sim \overline{T} + (t^e_{c1} - t^{max})$. Therefore $\delta^*_1 \sim -(t^e_{c1} - t^{max})$ and similarly $\delta^*_2 \sim -(t^e_{c1} - t^e_{c2})$. The other pairwise differences are $-(t^e_{c1} - t^e_{c2})$ and $-(t^e_{c1} - t^e_{c3})$ which clearly lie in the range between the first two values.

The third case considers four high-probability contingency tasks where one stage has two mission tasks impacted by tasks τ_{c1} and τ_{c2} and other contingency tasks only impact one mission task per stage ($t^e_{c1} \geq t^e_{c2} \geq t^e_{c4} \geq t^e_{c3}$). The fourth case has two stages where two mission tasks are impacted by a total of four high-probability contingency tasks: τ_{c1} and τ_{c2} in one stage, and τ_{c3} and τ_{c4} in another ($t^e_{c1} \geq t^e_{c3} \geq t^e_{c4} \geq t^e_{c2}$). The fifth case differs from fourth case only in assuming them to be low-probability ($t^e_{c4} \geq t^e_{c3} \geq t^e_{c1} \geq t^e_{c2}$). The results for these are summarized in Table 6.5. Case 3.1 represents the analysis for task τ_{c1} of third case. The first two entries in the second column for each row are δ^*_1 and δ^*_2 followed by other pairwise differences.

Thus, all possible pairwise differences between deferring and doing a contingency task seem to lie inside the range defined by δ^*_1 and δ^*_2. Statistical experiments were performed to verify this pattern using randomly generated task networks and found this bound to be overshot at most by 10% of the considered contingency task duration. This slight overshot is well expected because approximations were used in the above analysis.

6.5.4 Contingency Handling Algorithm

The above heuristic helps in choosing the right action for urgent and deferred contingency tasks which total to m^* out of m_c contingency tasks. For the rest $m_c - m^*$ regular contingency tasks, a sampling-based combinatorial search is performed. For each of the $N^*_c = 2^{(m_c - m^*)}$ possible decisions, the expected mission completion times over K world scenarios out of the N^*_c that are possible (for experiments in this chapter, choose $K = 50$ if $m_c > 5$ else $\min(N^*_c, 2^4)$) is

Case	Difference between deferring and executing contingency task
1.1	$t^{max}, t^e_{c2}, t^e_{c2}, t^e_{c3}$
1.2	$t^{max}, t^e_{c2}, t^e_{c2}, t^e_{c3}$
1.3	$t^{max}, t^e_{c3}, t^e_{c3}, t^e_{c3}$
2.1	$-(t^e_{c1} - t^{max}), -(t^e_{c1} - t^e_{c2}), -(t^e_{c1} - t^e_{c2}), -(t^e_{c1} - t^e_{c3})$
2.2	$-(t^e_{c2} - t^{max}), 0, 0, -(t^e_{c2} - t^e_{c3})$
2.3	$-(t^e_{c2} - t^{max}), 0, 0, 0$
3.1	$-(t^e_{c2} - t^{max}), t^e_{c2}, t^e_{c2}, -(t^e_{c2} - t^e_{c4}), t^e_{c2},$ $-(t^e_{c2} - t^e_{c4}), -(t^e_{c2} - t^e_{c4}), t^e_{c2}$
3.2	$-(t^e_{c2} - t^{max}), t^e_{c2}, -(t^e_{c2} - t^e_{c4}), t^e_{c2}, t^e_{c2},$ $t^e_{c2}, -(t^e_{c2} - t^e_{c3}), -(t^e_{c2} - t^e_{c4})$
3.3	$t^{max}, t^e_{c3}, t^e_{c3}, t^e_{c3}, t^e_{c3}, t^e_{c3}, t^e_{c3}, t^e_{c3}$
3.4	$t^{max}, t^e_{c4}, t^e_{c3}, t^e_{c4}, t^e_{c4}, t^e_{c4}, t^e_{c4}, t^e_{c4}$
4.1	$-(t^e_{c2} - t^{max}), t^e_{c3}, t^e_{c2}, -(t^e_{c2} - t^e_{c4}), t^e_{c4},$ $t^e_{c3} - t^e_{c2}, t^e_{c4} - t^e_{c2}, t^e_{c3}$
4.2	$-(t^e_{c2} - t^{max}), t^e_{c2}, 0, t^e_{c2}, t^e_{c2}, t^e_{c2}, 0, 0$
4.3	$-(t^e_{c2} - t^{max}), t^e_{c3}, t^e_{c4}, t^e_{c3} - t^e_{c4},$ $t^e_{c3} - t^e_{c4}, t^e_{c3}, -(t^e_{c4} - t^e_{c2}), t^e_{c4}$
4.4	$-(t^e_{c4} - t^{max}), t^e_{c4}, t^e_{c4}, 0, 0, t^e_{c4},$ $-(t^e_{c4} - t^e_{c2}), t^e_{c4}$
5.1	$-(t^e_{c1} - t^{max}), 0, -(t^e_{c1} - t^e_{c2}), 0, 0, 0, 0, 0$
5.2	$-(t^e_{c2} - t^{max}), 0, 0, 0, 0, 0, 0, 0$
5.3	$-(t^e_{c3} - t^{max}), 0, 0, -(t^e_{c3} - t^e_{c1}), -(t^e_{c3} - t^e_{c1}),$ $0, -(t^e_{c3} - t^e_{c2}), 0$
5.4	$-(t^e_{c4} - t^{max}), -(t^e_{c4} - t^e_{c3}), -(t^e_{c4} - t^e_{c3}),$ $-(t^e_{c4} - t^e_{c1}), -(t^e_{c4} - t^e_{c1}), -(t^e_{c4} - t^e_{c3}),$ $-(t^e_{c4} - t^e_{c2}), -(t^e_{c4} - t^e_{c3})$

Table 6.5: Analysis of various task networks

computed. The set of task networks $P_c = [P_{c1}, P_{c2}, \cdots, P_{cK}]$ representing each action-world scenario is computed using Algorithm 5 based on the case-by-case analysis in Table 6.4.

The probabilities of these world scenarios are computed using the product rule. Then, the probabilistic sum of the expected mission completion time over K scenarios is computed. Thus, the vector $T_d = [T_{d1}, \cdots, T_{dN^*_c}]$ storing expected mission completion times for each decision is obtained where $T_{di} \sim \mathcal{N}(\mu, \sigma^2)$. That choice of contingency actions is selected which minimizes the mean time.

For each contingency task, one must evaluate K task networks four times to compute δ^*_1 and δ^*_2. Thus, the first phase of proactive planning is of the order $O(Km_c)$. If a total of m^* urgent and deferred contingency tasks were identified, then only $2^{m_c - m^*}$ decisions need to be analyzed.

Algorithm 6 Proactive planning for contingency tasks

Input: $\mathcal{T}_c, P_m$
Output: Optimal decision x^*

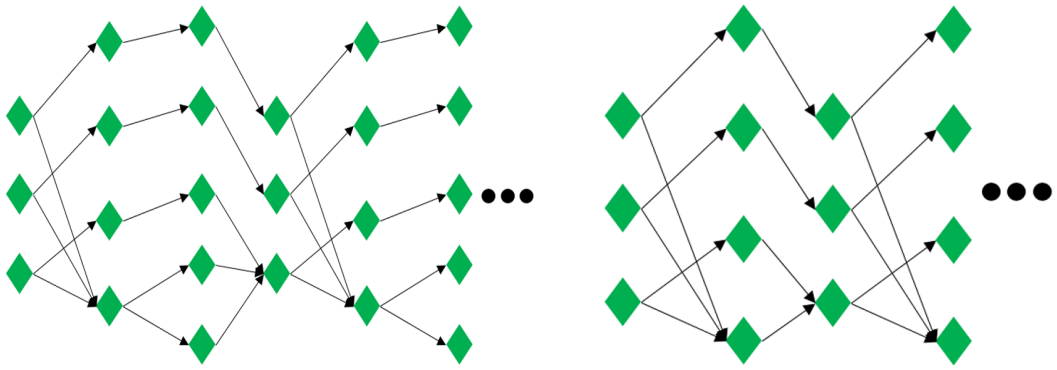
for all $\tau_c \in \mathcal{T}_c$ **do**
  if $\min(\delta_1^*, \delta_2^*) - 0.1 * t_c^e(\tau_c) > 0$ **then**
    $x^*(\tau_c) = \top$
  else if $\max(\delta_1^*, \delta_2^*) + 0.1 * t_c^e(\tau_c) < 0$ **then**
    $x^*(\tau_c) = \bot$
  end if
end for
Compute P_c using Algorithm 5
for all N_c^* decisions **do**
  for all K sampled world scenarios **do**
    Compute T^m using FTCM
    $\mathcal{S}_{xj} \leftarrow T^m$
  end for
end for
$T_{dx} \leftarrow \sum_{j=1}^{K} p_j \mathcal{S}_{xj}$
$x^* \leftarrow \arg\min_x \mu(T_{dx})$

The complexity of the second phase is clearly $O(K2^{m_c - m^*})$ as described above. Thus the overall complexity of the proactive approach (Algorithm 6) is $O(K(m_c + 2^{m_c - m^*})) = O(2^{m_c - m^*})$. For example, if a mission is reported to concurrently have ten contingency tasks and four contingency tasks can be pruned so that they do not need to be considered during combinatorial analysis, then this reduces the number of decisions to be considered from 1024 to 64, and thus enables the proposed approach to handle around ten contingency tasks.

If an action on a particular contingency task is deferred, one must generate replanning trigger conditions and monitor the associated probability values being updated for the ignored contingency tasks in case they may become critical in the future. The replanning trigger condition computed for the deferred tasks is basically a range of probability values $[p_l^r, p_u^r] \subset [0, 1]$. The lower bound is not of much importance and is set as the current probability of the contingency task. It is assumed that if the probability associated with the contingency task decreases, then the execution of the contingency task continues to get deferred. So one has to find the increase in the probability that is critical for replanning. The upper bound however indicates that when

the actual probability of the contingency task exceeds this value replanning of the mission must be performed because low-risk contingency tasks may have become critical for successful mission completion. The range is determined using a heuristic method of sensitivity analysis. All the other information regarding the environment is kept fixed. The current probability value, say p_c of the contingency task, is increased in suitable step-sizes until a change in decision occurs at p^*. The trigger condition is then given by the range $[p_c, p^*]$.

The methods of incorporating contingency tasks are illustrated with two examples. Figures 6.12 and 6.13 show a simple and more complicated task network respectively followed by their corresponding execution schedules. Observe that this approach helps robots spend less time traveling and remaining idle (gray and black bars).



(a) (b)

Figure 6.6: Task networks based on neural networks used for simulation experiments in Table 6.8

6.6 Results and Illustrations

6.6.1 Multi-robot Task Allocation

The strategies listed in Section 6.4 can be first compared without introducing the feature of task divisions. In Table 6.6, the optimized mission completion time is shown for ten different mission scenarios whose task network topologies were manually generated, but all other features were randomly generated some of which are described in Table 6.7. The task networks chosen for these

Table 6.6: Benchmarking strategy $\mathcal{S}_2$ against $\mathcal{S}_6$ and $\mathcal{S}_7$. Table entries are mission makespan in minutes.

	$\mathcal{S}_0$	$\mathcal{S}_2$	$\mathcal{S}_6$	$\mathcal{S}_7$
1	298.28	308.63	325.04	322.77
2	589.80	610.47	648.09	642.30
3	888.92	917.81	968.34	957.79
4	1190.44	1226.04	1295.36	1281.81
5	94.75	116.11	119.41	124.54
6	303.65	372.93	364.50	379.13
7	489.86	616.13	612.72	639.86
8	676.26	863.86	850.31	878.72
9	1779.97	1831.29	1924.50	1916.47
10	2163.44	2222.54	2343.84	2336.89

experiments were random repetitions of graphs derived from artificial neural networks (ANNs), as shown in Figure 6.6. Task execution durations are randomly sampled for each task. Resource constraints are tight meaning that the equality in Inequality 6.9 is satisfied or the corresponding inequality is satisfied by a tiny margin.

As expected, the exhaustive method achieves the overall minimum. The second column shows the time achieved by strategy $\mathcal{S}_2$. This is because of all the list scheduling algorithms described, strategy $\mathcal{S}_2$ performs significantly better. All five strategies were simulated for the ten mission scenarios. It was found that strategy $\mathcal{S}_2$ performed the best and strategy $\mathcal{S}_4$ was the second best. The mean percentage by which the mission completion time computed by strategy $\mathcal{S}_2$ was less than that of strategy $\mathcal{S}_4$ over the ten experiments was 3.28% with a standard deviation of 0.78%. Strategy $\mathcal{S}_2$ performs better because it takes into account the task execution durations in a local way, i.e., after the feasible tasks have been identified for parallel execution. On the other hand, strategy $\mathcal{S}_4$ prioritizes the tasks globally before task allocation begins. However, once tasks are allowed to be interrupted and reassigned to robots, strategy $\mathcal{S}_4$ starts working better. A detailed explanation for the case of task division follows in the next section. The last two columns represent the results for strategies $\mathcal{S}_6$ and $\mathcal{S}_7$.

Strategy $\mathcal{S}_2$ outperforms strategies $\mathcal{S}_6$ and $\mathcal{S}_7$ in seven cases out of ten. This is because in these cases, resource constraints are tighter, and there are fewer robots. Having tight resource

constraints means equality in Equation 6.9 or the inequality holding by a tiny margin which diminishes the advantage provided by using same robots for finishing multiple tasks (as in strategies $\mathcal{S}_6$ and $\mathcal{S}_7$). When this is not the case as in the other four cases, the list scheduling heuristics still outperform strategy $\mathcal{S}_7$ but not $\mathcal{S}_6$.

However, the runtime data shows that the list scheduling strategies are much faster because the mean ratio of runtime for strategy $\mathcal{S}_6$ to $\mathcal{S}_2$ over the ten experiments is 62.47 and the standard deviation is 62.5 and $\mathcal{S}_6$ is faster than $\mathcal{S}_7$. Given the motivation to develop fast contingency handling techniques, one must devote as less time as possible during nominal mission planning. The last two strategies were only included for benchmarking purposes.

All seven strategies are compared explicitly. In Figure 6.7a, the optimized mission completion time reported for five different experiments is plotted. The exhaustive method takes the least time. The second element that is plotted is the minimum achieved by the list scheduling strategies. The last two elements represent the results for strategies $\mathcal{S}_6$ and $\mathcal{S}_7$. The list scheduling strategies outperform strategies $\mathcal{S}_6$ and $\mathcal{S}_7$.

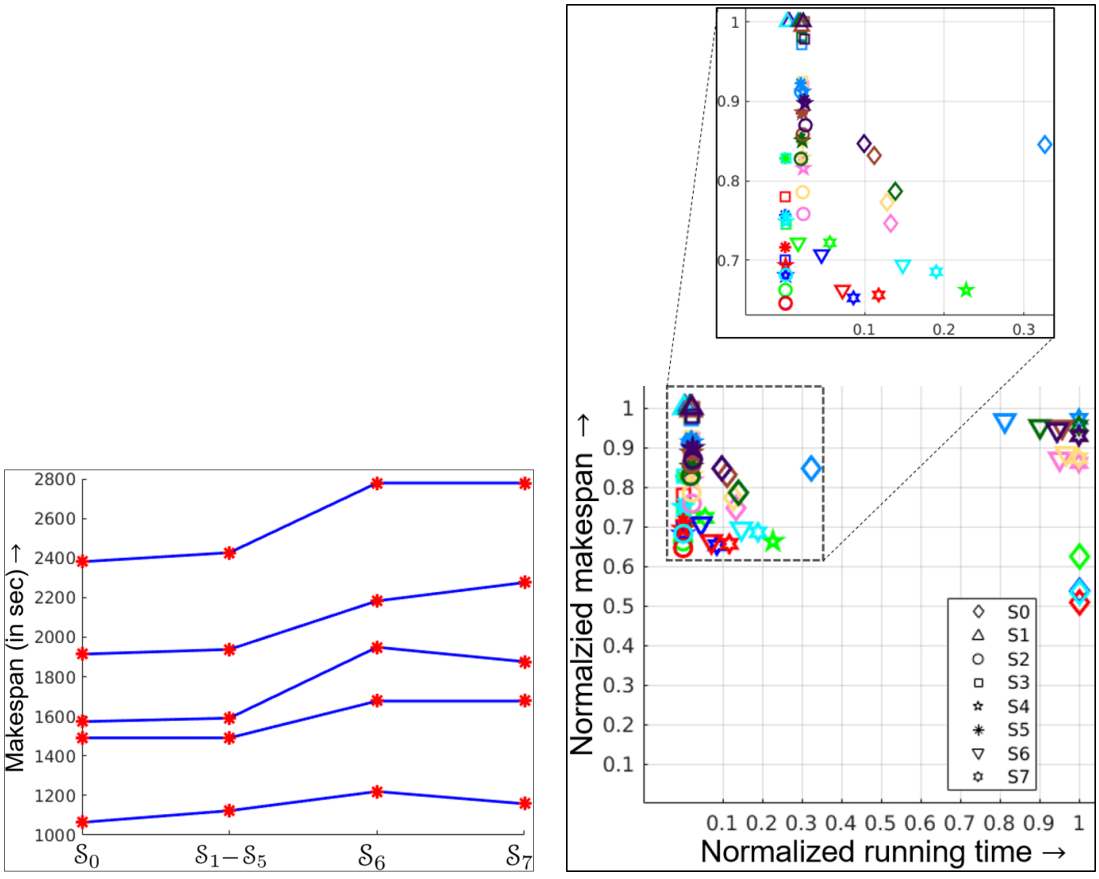
In Figure 6.7b, ten different scenarios were simulated and their normalized mission completion time as well as running time are reported for all the seven strategies. A Pareto frontier is found to be taking shape because reducing running time produces more suboptimal results. Note that strategies $\mathcal{S}_1 - \mathcal{S}_5$ take less time to complete than strategies $\mathcal{S}_6$ and $\mathcal{S}_7$. In fact, some runs show that the last two strategies take even more time than the exhaustive strategy $\mathcal{S}_0$.

A simple coalition selection heuristic was chosen for strategies $\mathcal{S}_1 - \mathcal{S}_5$. The major weightlifting was done by the task prioritization heuristics. On the other hand, strategies $\mathcal{S}_6$ and $\mathcal{S}_7$ used more sophisticated coalition selection heuristics to handle coalition interferences and discarded the task prioritization phase. The overall effect is that strategies $\mathcal{S}_1 - \mathcal{S}_5$ perform faster and produce better results than the last two strategies. This is because tight resource constraints downgrade the possibility of using overlapping coalitions for finishing concurrently feasible tasks. The last two strategies were included only for benchmarking purposes and so can be safely removed from

Table 6.7: Description of scenarios used for experiments in Table 6.8

	$\lvert\mathcal{T}\rvert$	$\lvert R\rvert$	$\lvert E(\mathcal{P})\rvert$
1	175	8	246
2	350	8	496
3	525	8	746
4	700	8	996
5	120	8	155
6	360	8	475
7	600	8	795
8	840	8	1115
9	1080	10	1435
10	1320	10	1755

further consideration based on the above observations. The exhaustive strategy is also excluded from further consideration because it does not scale efficiently and restricts the problem sizes.



(a) (b)

Figure 6.7: Performance of list scheduling heuristics compared to *MinStepSum* and *MinInterfere* from [1]

Table 6.8: Results for ten simulation runs with task division

	T^c (in seconds)					T^m (in seconds)				
	$\mathcal{S}_1$	$\mathcal{S}_2$	$\mathcal{S}_3$	$\mathcal{S}_4$	$\mathcal{S}_5$	$\mathcal{S}_1$	$\mathcal{S}_2$	$\mathcal{S}_3$	$\mathcal{S}_4$	$\mathcal{S}_5$
1	2.38	2.27	2.26	2.38	2.30	10510	9890	10265	9370	9950
2	13.52	13.51	13.51	13.52	13.53	21235	20795	21192	19376	20819
3	41.41	41.21	41.28	41.17	41.23	33138	31145	31181	28365	30823
4	93.08	93.60	93.08	93.06	93.44	43905	40952	41184	38949	40812
5	1.25	1.05	1.11	1.08	1.07	7276	6702	6537	5961	6582
6	14.80	14.91	14.97	14.77	14.86	22214	19734	19561	18139	19463
7	59.57	59.75	59.52	60.12	60.52	41269	36895	34296	33212	34556
8	155.76	164.67	166.98	173.03	183.42	55175	51759	47996	48023	47992
9	371.18	368.08	367.98	369.31	340.42	65144	60541	56752	54735	56361
10	612.50	622.37	627.38	623.98	592.20	80335	75028	69324	69413	68750

Next, the feature of task division is introduced and strategies $\mathcal{S}_1 - \mathcal{S}_5$ are compared. Ten experiments were run whose specifications are given in Table 6.7. Again the task networks chosen for these experiments are based on the artificial neural networks architecture as shown in Figure 6.6. All five strategies were applied on the ten scenarios. Here, the running times are denoted by T^c and optimized mission completion times are denoted by T^m. The results are shown in Table 6.8.

Since the complexity of the first seven experiments is significantly lower compared to the last three (Table 6.7), the running times for the strategies in these cases do not differ significantly. The last three experiments show significantly different running times.

From the data collected in Table 6.8, strategies which could be further discarded are decided. A strategy $\mathcal{S}_i$ dominates another strategy $\mathcal{S}_j$ if:

i. both $T^m(\mathcal{S}_i) \leq T^m(\mathcal{S}_j)$ and $T^c(\mathcal{S}_i) \leq T^c(\mathcal{S}_j)$

ii. either $T^m(\mathcal{S}_i) < T^m(\mathcal{S}_j)$ or $T^c(\mathcal{S}_i) < T^c(\mathcal{S}_j)$

Out of the ten experiments, strategy $\mathcal{S}_4$ is non-dominated eight times which is double that of the runner-up. However, no strategy is dominated in every experiment so none can be discarded based on Pareto efficiency. Strategy $\mathcal{S}_4$ outperforms the other four strategies except in experiments 8 and 10 where it was dominated by strategies $\mathcal{S}_3$ and $\mathcal{S}_5$ respectively. If a strategy is dominated in

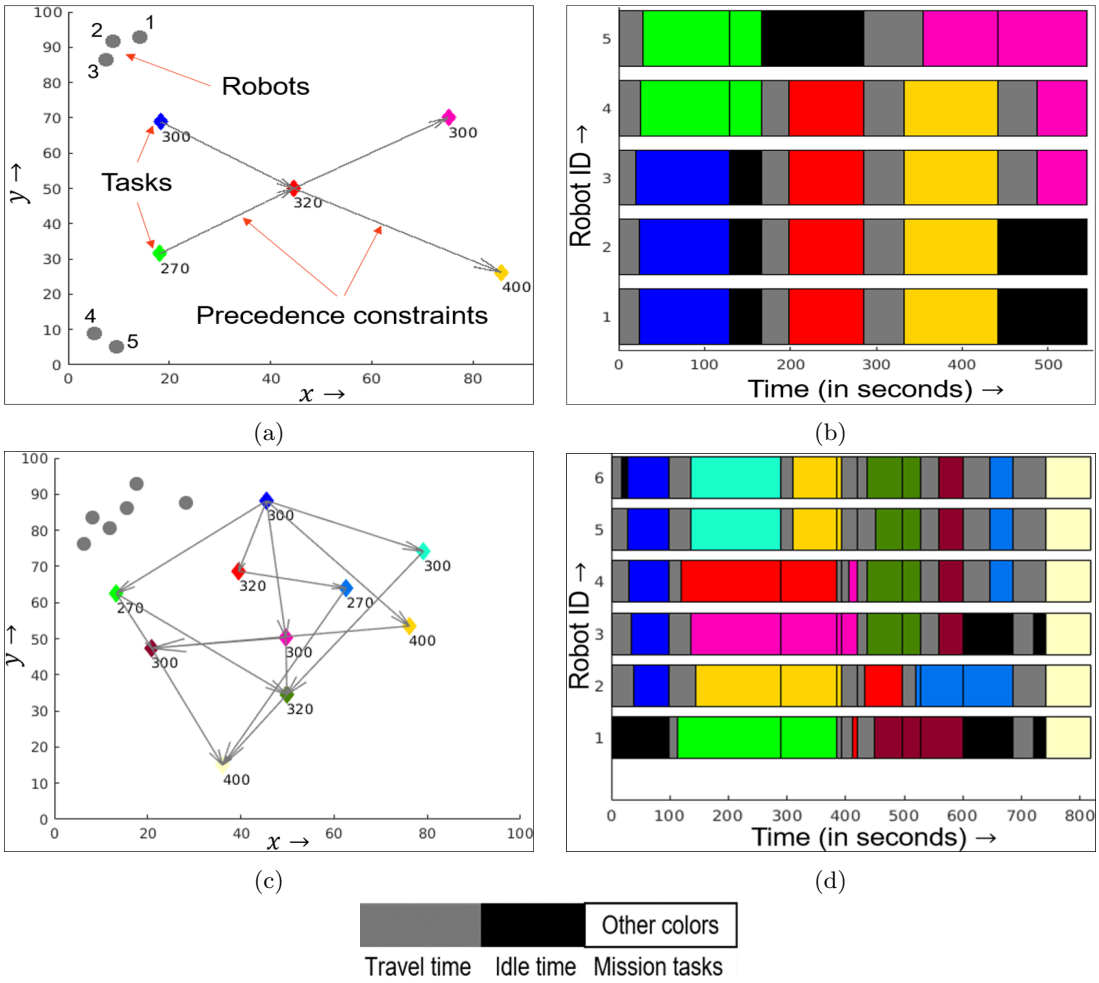


Figure 6.8: Application of strategy $\mathcal{S}_4$ on task networks (a,c), and corresponding robot schedules (b,d)

every experiment, then it is discarded. However, no such strategy exists based on the collected data.

The rationale for the above observations is as follows: The missions considered here are based on spatially separated tasks that may be generally encountered by mobile robots. It is reasonable to assume that the time spent by the robots during traveling is significantly less than the time spent by them during task execution, and therefore, high task duration was assigned to all the mission tasks. Strategy $\mathcal{S}_5$ includes traveling durations in its guiding heuristic. Strategies $\mathcal{S}_1$ and $\mathcal{S}_2$ first identify the stage-wise feasible tasks and then order them using task execution durations.

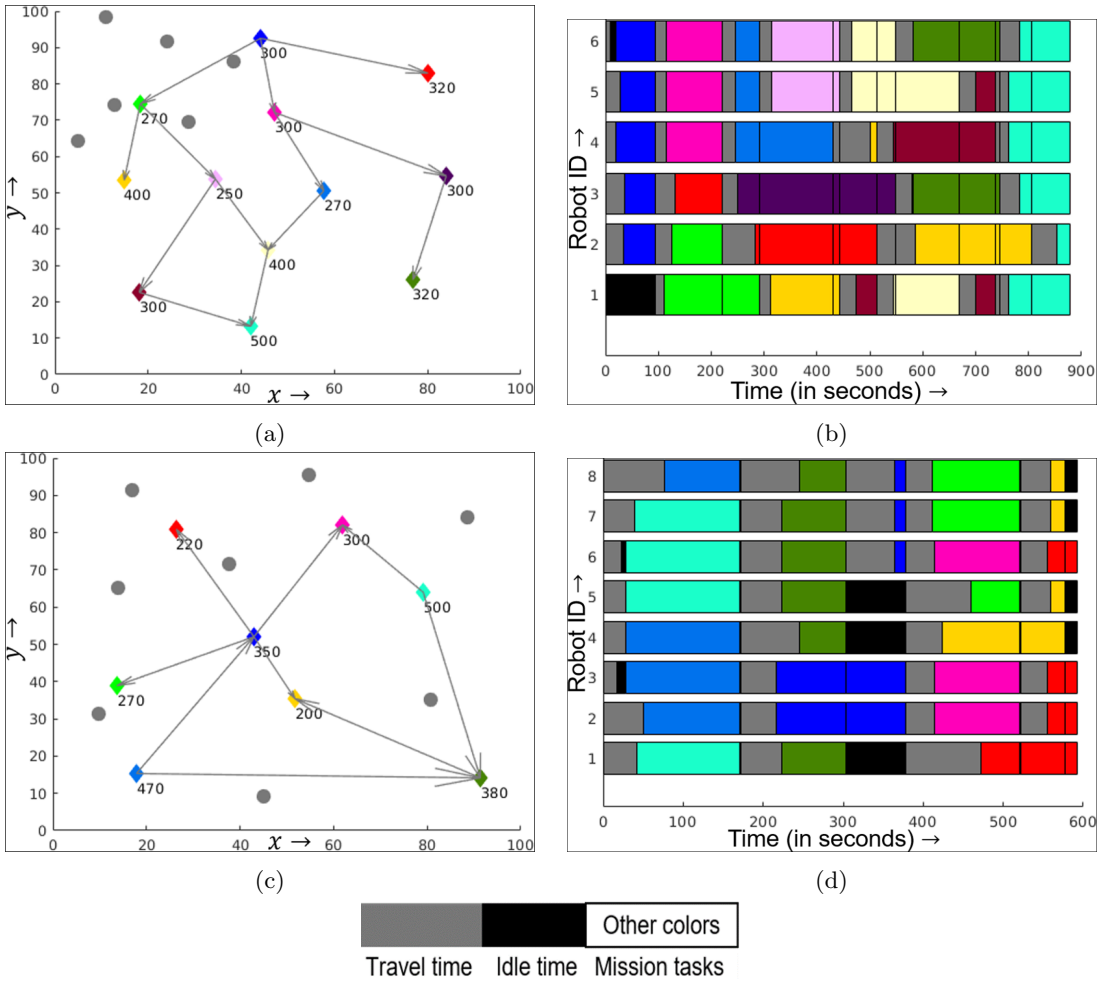


Figure 6.9: Application of strategy $\mathcal{S}_4$ on task networks (a,c), and corresponding robot schedules (b,d)

But strategy $\mathcal{S}_4$ orders tasks based on their execution durations first and the identification of feasible tasks and assignment of their optimal coalitions proceed later. In terms of running time, it is approximately equal to others. Based on these arguments, strategy $\mathcal{S}_4$ is chosen as the optimal choice for contingency handling.

The solutions generated by strategy $\mathcal{S}_4$ are illustrated for four different mission scenarios in Figures 6.8 and 6.9. Task networks differing from ANNs are efficiently handled. Color coding is used for better visualization of the solutions produced. Having robots idle for large amounts of time is not beneficial for the mission, so such a duration is colored black. One can also consider

traveling as a task of low significance and color traveling duration gray. The significance of this coloring scheme is that when one looks at the solutions, the collective quantities of gray and black blocks are expected to be smaller than those representing the execution of the mission tasks. The task networks define a particular color for each mission task which is used in the Gantt chart for the mission execution.

Computational analysis is performed for strategy $\mathcal{S}_4$. In both Figures 6.11a and 6.11b, the running time is recorded by varying the number of mission tasks and robots respectively. In both

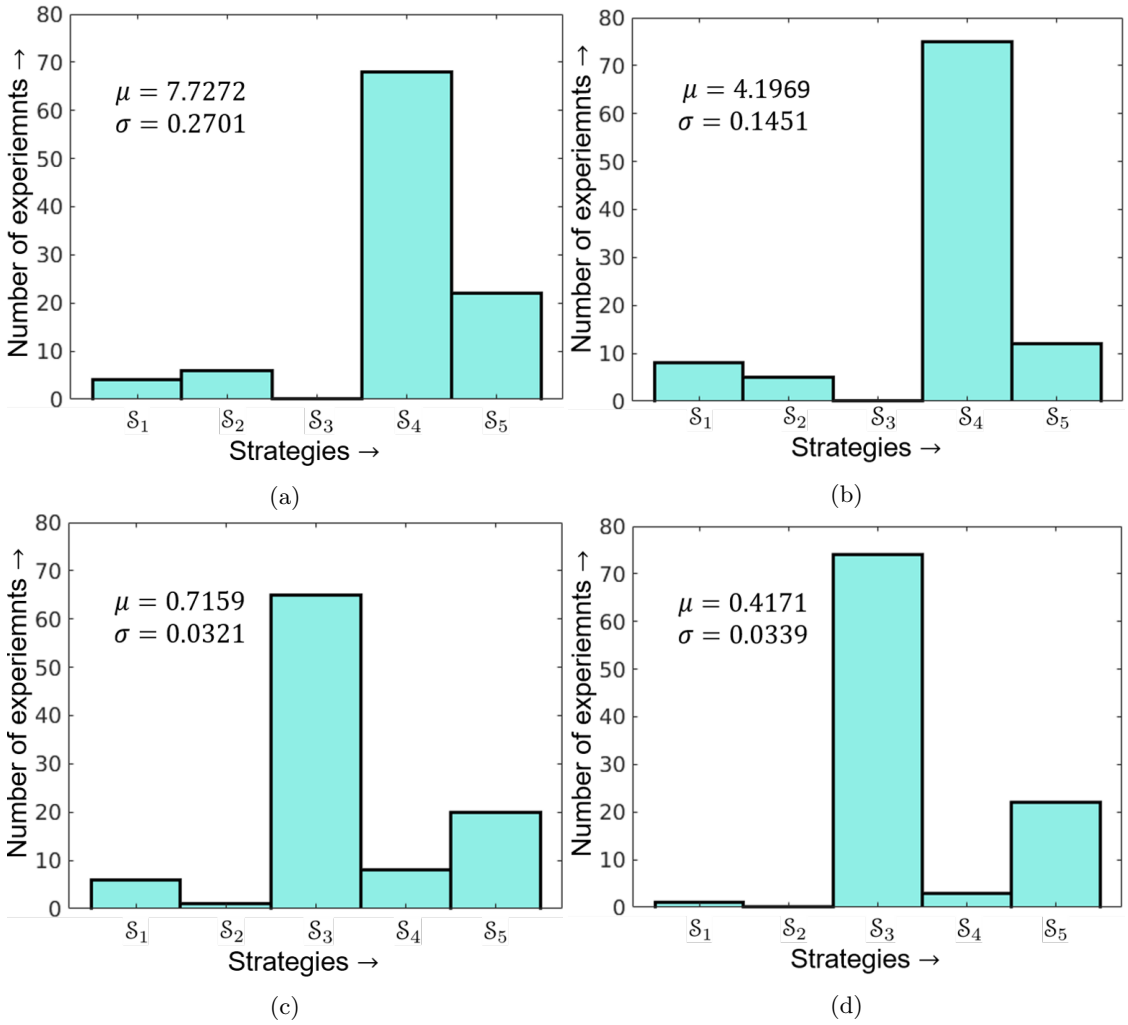
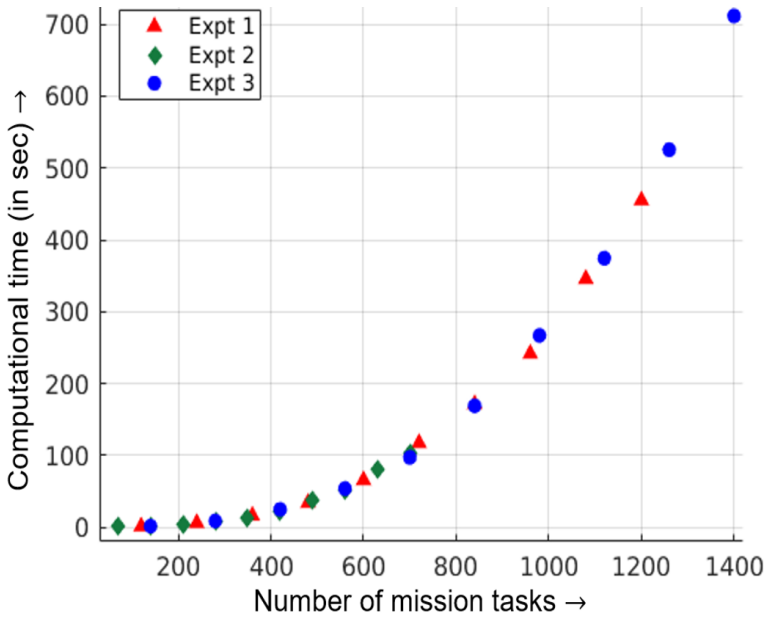
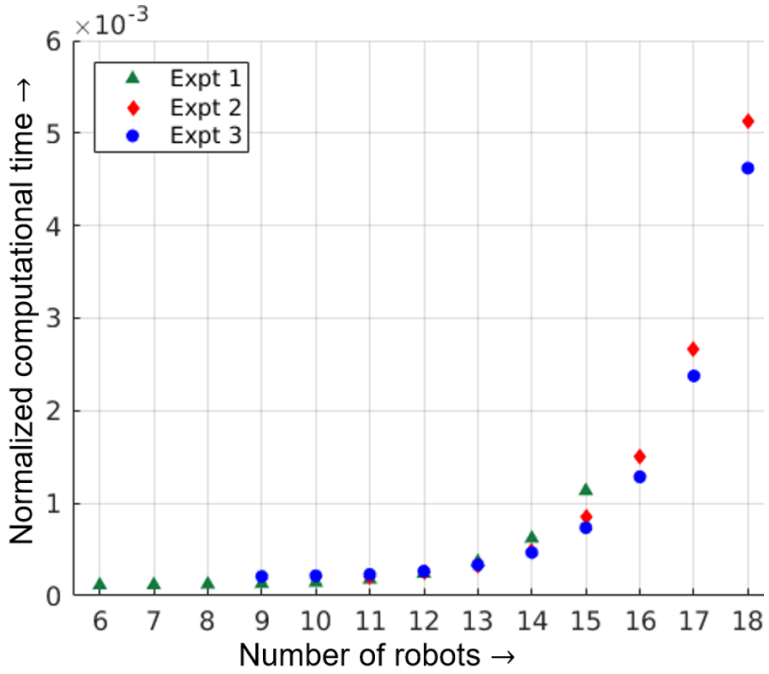


Figure 6.10: Effect on performance of strategies by varying task duration compared to traveling duration



(a)



(b)

Figure 6.11: Computational performance of strategy $\mathcal{S}_4$

Figures, the running time data is plotted for three different cases. In Figure 6.11b, on the $y-$axis, the running time normalized by the square of the number of mission tasks is shown (because $O(m^2)$ is a good approximation, and doing this fits all three data-sets in one plot).

Table 6.9: Effect of uncertainty on mission makespan

	σ_a	σ_e	T_1^m	T_2^m	$\lvert T_1^m - T_2^m \rvert$	$\uparrow$ (as %)
1	3	0.1	7427	7693	267	3.59
2	5	0.1	7363	7793	431	5.85
3	10	0.1	7613	8440	827	10.87
4	12	0.1	7564	8452	887	11.73
5	15	0.1	7585	8721	1136	14.97
6	0.1	10	7438	8309	870	11.70
7	0.1	20	7544	8910	1366	18.10
8	0.1	30	7578	9460	1882	24.84
9	0.1	40	7644	9935	2291	29.97
10	0.1	50	7702	10038	2336	30.32

For the experiments of Table 6.8, it is assumed that task execution duration is significantly higher than travel times because real-life tasks have similar characterizations and observed that strategy $\mathcal{S}_4$ is a better optimizer in most cases. But then one must also study the effect of varying task execution duration with respect to traveling time. The randomly generated ten task networks are used as before. Specifically, the ratio of average task duration $\langle t_e \rangle$ to average travel time $\langle t_a \rangle$ is varied. For strategy s, the mean $\langle t_e \rangle$ is computed over the number of tasks. So, the mean $\langle t_a \rangle$ is computed over the number of tasks. For a given problem scenario and strategy s, the ratio $r_s = \langle t_e \rangle / \langle t_a \rangle$ is computed. One can then compute the average ratio for the entire scenario as $\langle r_s \rangle = (\sum r_s)/5$ because five strategies $\mathcal{S}_1 - \mathcal{S}_5$ are considered.

These experiments are repeated on 100 problem scenarios and the underlying task network is chosen from among the ten generated task networks. The task duration is randomly varied within specific bounds such that the computed average ratio for each problem scenario is roughly the same. Then the mean as well as the standard deviation of the average ratios computed for the 100 experiments is reported (Figure 6.10). The histogram representing the frequency with which the strategies outperform others is also plotted. This is repeated for four different scenarios as shown in Figure 6.10a-6.10d by reducing the task-travel ratio successively. For high task durations, strategy $\mathcal{S}_4$ wins as already discussed above. For all ten task networks considered here for benchmarking purposes, similar patterns were observed.

However, when task duration is comparable to travel time or even much shorter, then strategy $\mathcal{S}_4$ is outperformed by strategy $\mathcal{S}_3$. When travel duration becomes more significant, it is expected that strategy $\mathcal{S}_4$ which relies partly on task duration as its task prioritizing heuristic will perform worse than strategy $\mathcal{S}_3$ which disregards task duration by assuming it to be one for all tasks. Intuitively, strategy $\mathcal{S}_5$ is expected to perform better in cases where there is significant traveling involved because it tries to include traveling duration in its heuristic. But this strategy naively averages travel times over all possible routes the robots may take. The experiments show this is not good enough to beat strategy $\mathcal{S}_3$.

For analyzing uncertainty results, a scenario comprising of 120 tasks, 9 robots is simulated. The task network is similar to Figure 6.6b and has 155 edges. The average task execution duration for the 120 tasks is around 450 seconds. The average time spent by robots to travel to task locations is computed by dividing the total time spent by all robots during traveling by the number of mission tasks and is around 70 seconds.

In Table 6.9, the standard deviation associated with noise in traveling times and execution durations are allowed to reach up to $10 - 20\%$ of their average values. T_1^m denotes the expected mission completion time computed by only considering the mean time expected for traveling and mission tasks. If the durations are instead modeled as Gaussian distributed variables and both the mean and variance information are used, then the computed expected mission completion time is denoted by T_2^m. The last column records the disparity caused by ignoring the variance information associated with task durations. Ignoring the effects of uncertainty leads to highly inaccurate decisions.

6.6.2 Contingency Management

The methods of incorporating contingency tasks are illustrated through two examples in Figures 6.12 and 6.13. The problem scenarios for the two examples are shown in Figures 6.12a and 6.13a respectively. Results of nominal mission planning without taking contingency tasks into

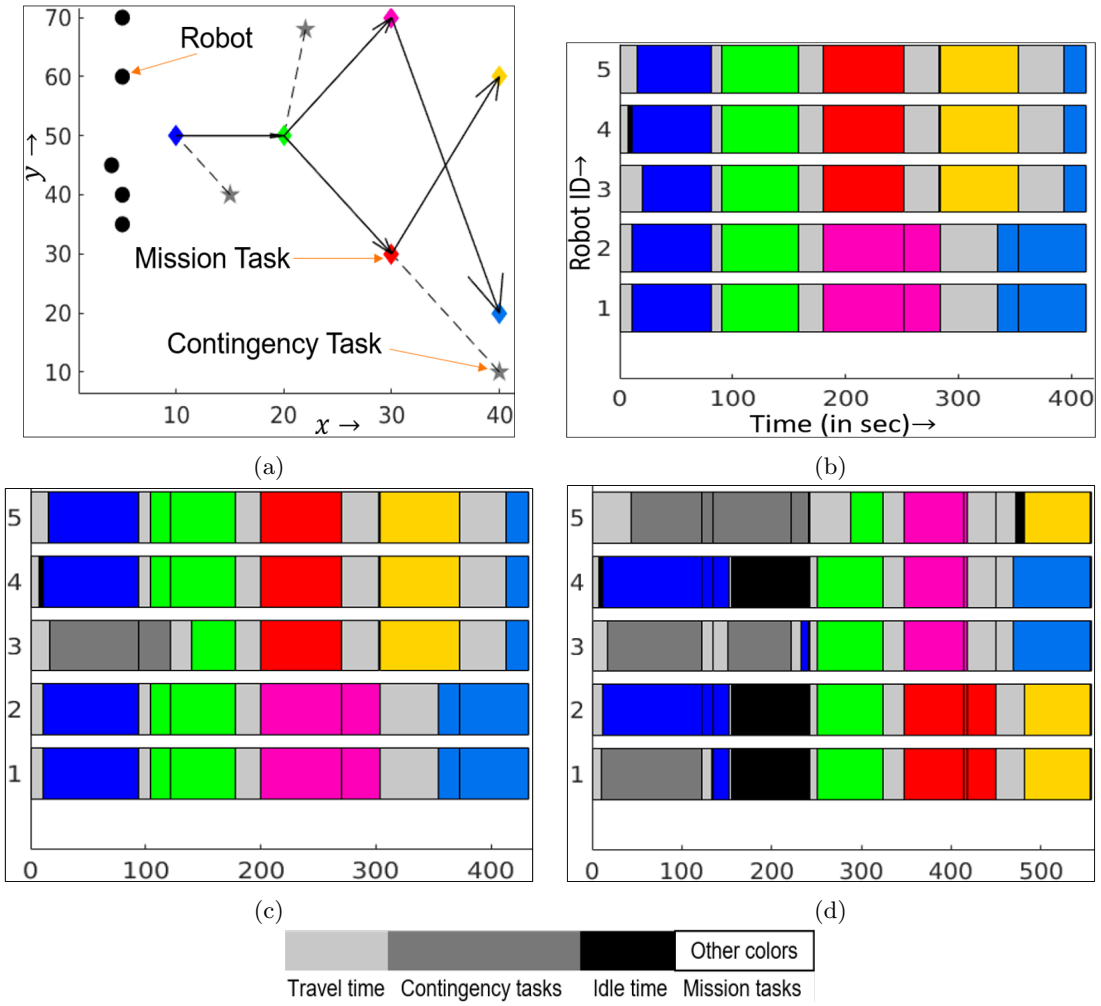


Figure 6.12: (a) Mission scenario, (b) Robot schedules obtained without considering contingency tasks, (c) Incorporating reported contingency tasks in robot schedules, (d) Replanning schedules for robots when probability of contingency tasks exceed desired limits

consideration are shown in Figures 6.12b and 6.13b respectively. When contingency tasks are considered, the resulting robot schedules are shown in Figures 6.12c and 6.13c respectively. When the probability values of the contingency tasks change so much that they trigger the replanning conditions, the newer schedules for robots are computed based on the updated information in Figures 6.12d and 6.13d respectively.

In Figure 6.14, the time spent by robots during task execution is compared with the time spent in other activities including traveling, remaining idle or performing contingency tasks. Figures

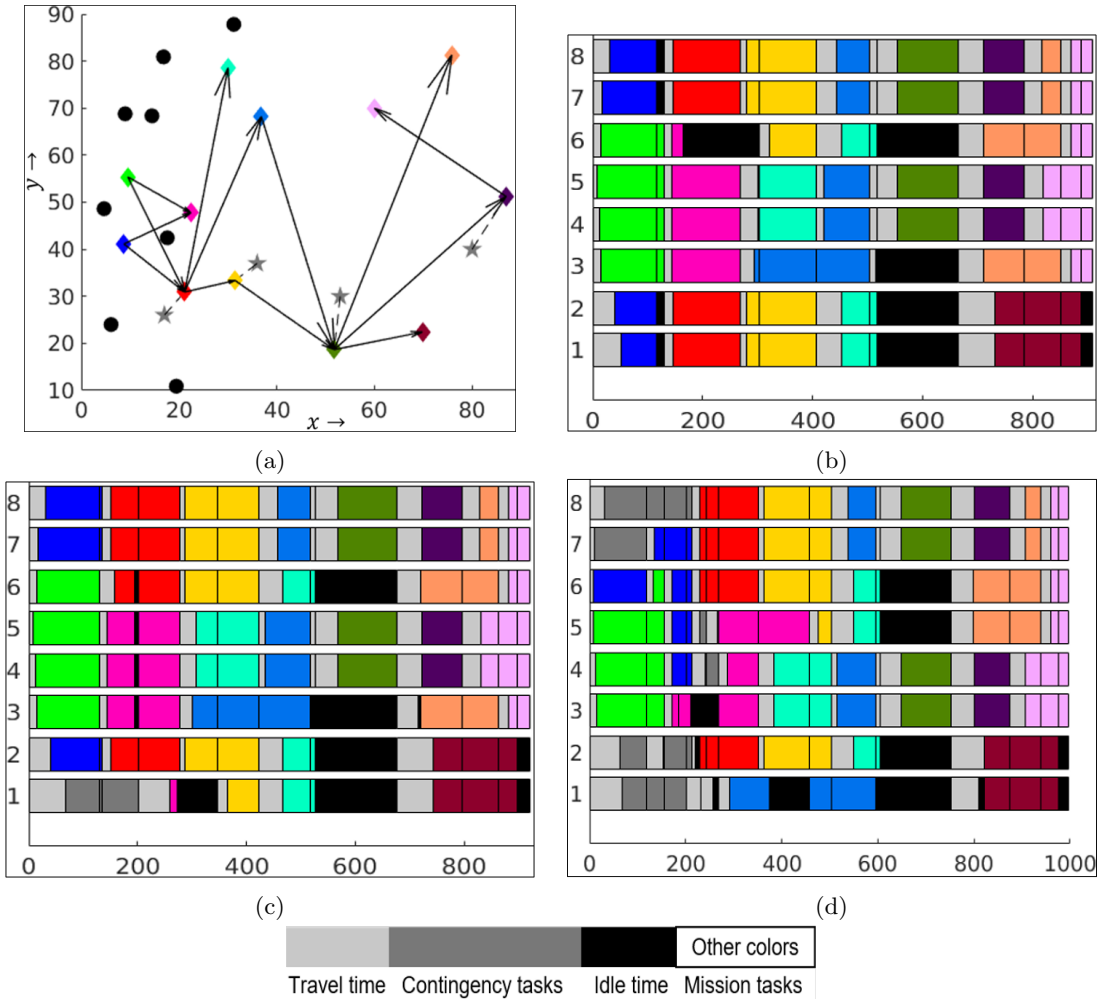


Figure 6.13: (a) Mission scenario, (b) Robot schedules obtained without considering contingency tasks, (c) Incorporating reported contingency tasks in robot schedules, (d) Replanning schedules for robots when probability of contingency tasks exceed desired limits

6.14a and 6.14b show the doubly stacked bar-plots for the missions of Figures 6.12a and 6.13a respectively. The durations are normalized by dividing with the nominal mission completion time for each case. The set of left stacked columns in Figures 6.14a and 6.14b refer to the nominal scenarios of Figures 6.12b and 6.13b respectively. The set of right stacked columns in Figures 6.14a and 6.14b refer to the contingency scenarios of Figures 6.12d and 6.13d respectively. The average idle times of the robots are also computed for the scenarios shown in Table 6.8 and found to be well under 1% of the mission time.

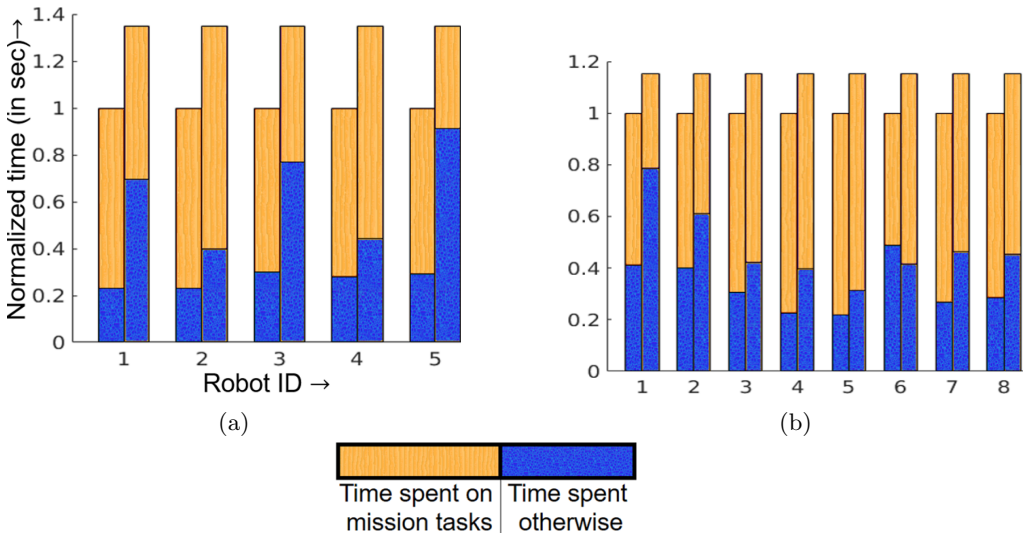


Figure 6.14: Comparison of time spent during task execution and otherwise for problems of (a) Fig. 6.12a, (b) Fig. 6.13a

This optimization scheme should ensure that the robots spend more time during task execution as opposed to other activities. All activities colored with a shade of gray are sought to be minimized by minimizing makespan.

In Figure 6.14, note that most robots spend higher times executing tasks than otherwise. Note also that the contingency tasks adversely impact the robots because their engagement in mission-related activities gets prolonged.

The sensitivity analysis results are shown for the contingency tasks based on six experiments. In Figure 6.17, the upper bounds are shown that are supposed to trigger replanning for those contingency tasks whose immediate execution was deferred. These experiments suggest that if the probability of contingency tasks adversely impacting the mission is higher than 50%, then it is highly likely that its execution is not deferred.

However, the upper thresholds also depend on other characteristics of the contingency tasks impacting the mission like their locations, durations, and uncertainties. Therefore, contingency tasks can also have much smaller upper threshold, as shown in the first three experiments.

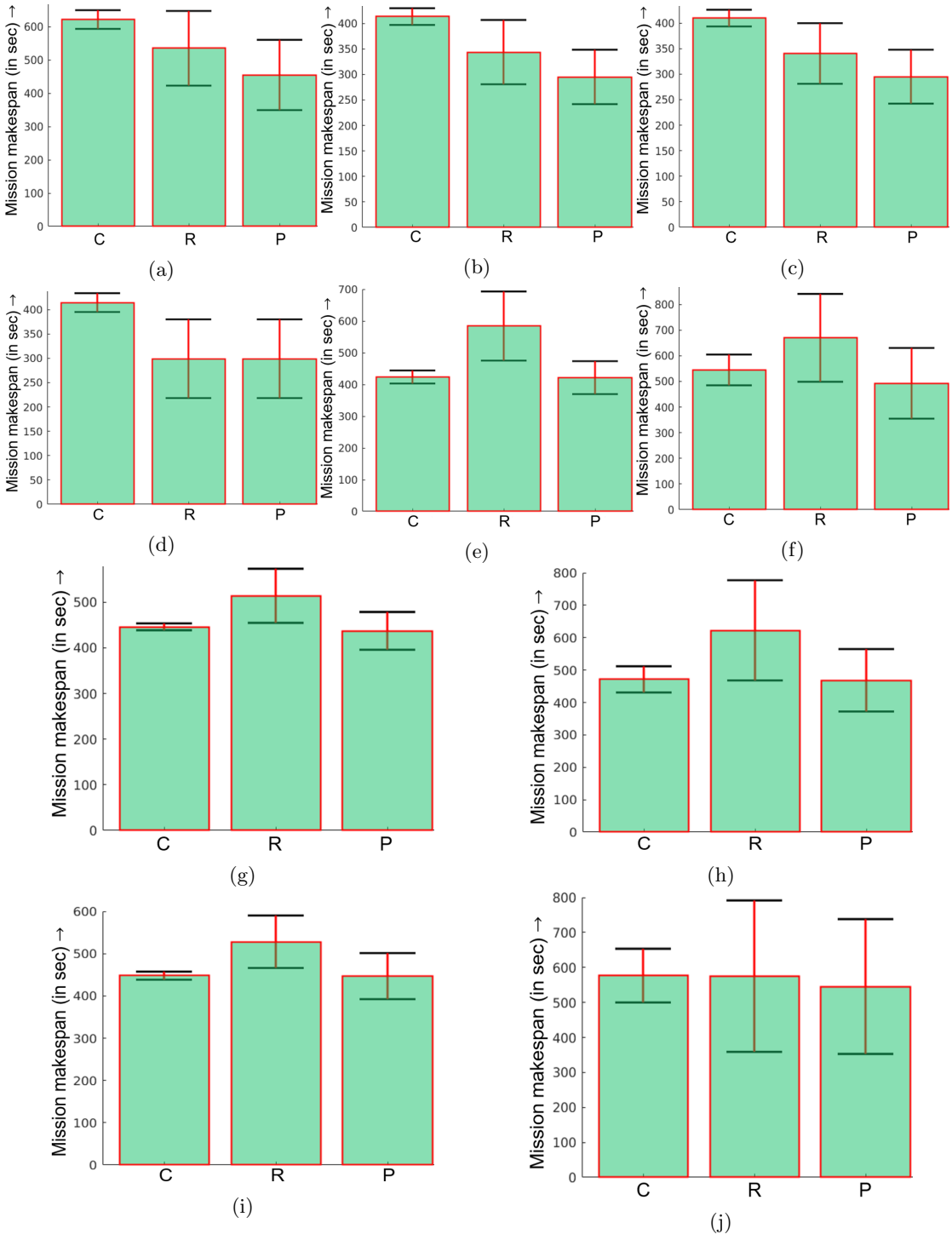
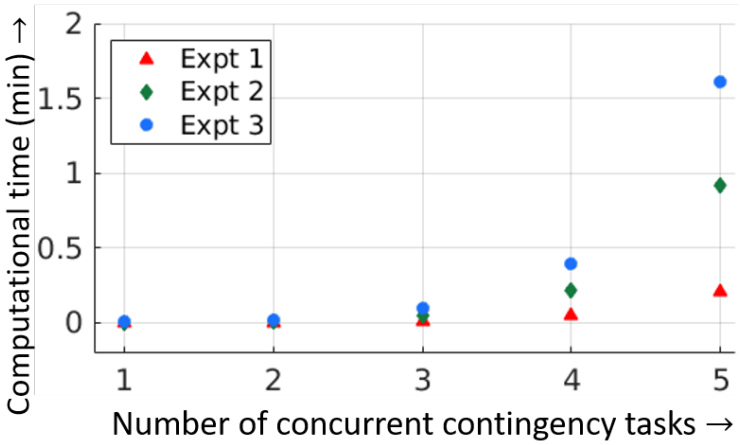


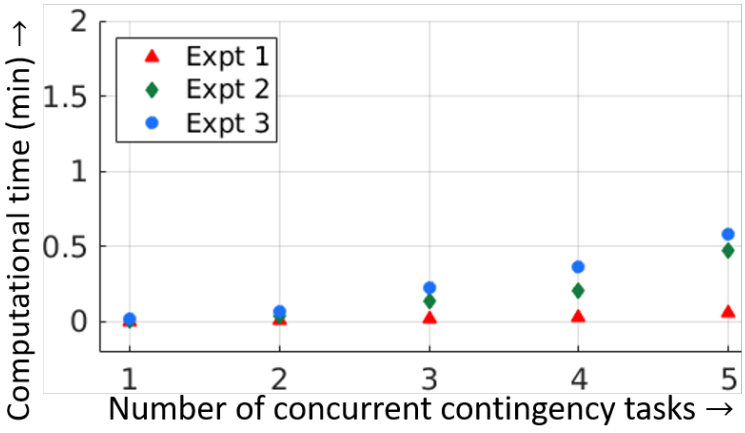
Figure 6.15: Comparison of conservative (C), reactive (R) and proactive (P) approaches

Figure 6.15 compares the conservative, reactive, and proactive approaches for ten different problems. Generally, the proactive approach is expected to perform better than both the conservative and reactive approaches. In Figures 6.15a−6.15c, proactive approach outperforms conservative approaches by around 30% and reactive approaches by around 15%.

However, if the probability of contingency tasks impacting the mission is low, then the reactive approach is much better than the conservative approach, and the proactive approach will at least be as good as the reactive approach on average (Figure 6.15d). On the other hand, if



(a) Exhaustive method



(b) Pruning-based method

Figure 6.16: Variation of running time with number of concurrently reported contingency tasks for strategy $\mathcal{S}_4$

the probability is high, then the conservative approach will beat the reactive approach, and the proactive approach will at least be as good as the conservative approach on average (Figures 6.15e−6.15i). There may also be cases where the three approaches perform approximately the same (Figure 6.15j). The three approaches were also compared on the ten task networks from Table 6.7 and it was found that the proactive approach is at least as good as the other two approaches on average.

In Figure 6.16, the pruning-based method outperforms the exhaustive method by reducing the running times. The number of contingency tasks pruned for each of the experiments in serial order for the case of five contingency tasks is: $2, 1, 2$.

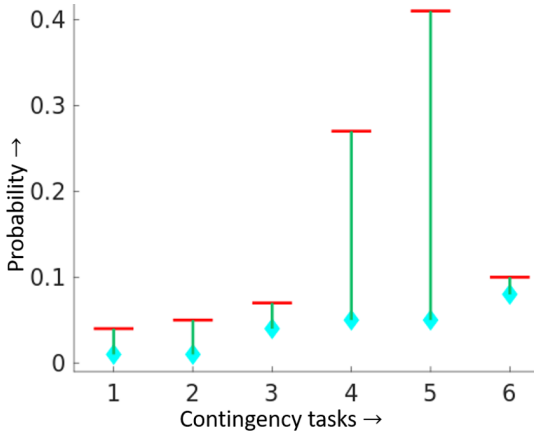


Figure 6.17: Replanning trigger conditions obtained from sensitivity analysis

6.7 Summary

The current chapter studies an approach of forming coalitions among robots and scheduling them for the execution of spatially distributed tasks having inter-task precedence constraints while incorporating contingency tasks. Uncertainty associated with traveling and task execution is taken into account while making decisions for the contingency tasks. In the present mission model, partial teams can commence task execution depending on the resource constraints of different mission tasks. Also, the robots assigned to a task can be interrupted mid-way and

rescheduled to another task. It can be concluded that the strategy that orders the tasks based on their execution durations using the concept of static levels provides the best trade-off. Running time is around 5 minutes for approximately 1000 tasks and 10 robots in MATLAB. Thus, it can efficiently handle a large number of mission tasks during nominal mission planning. For a scenario involving six robots, ten mission tasks, and five contingency tasks, the running time was around 20 seconds. In real missions involving multiple USVs, it is not feasible to consider higher numbers at least for centralized approaches. This should also hold for complex missions involving other kinds of mobile robots.

Chapter 7

Decomposition of Collaborative Surveillance Tasks with Uncertainty in Environmental Conditions and Robot Availability

In this chapter [1], an optimization approach is presented for partitioning a single, complex task among multiple robots.

7.1 Introduction

Most categories of collaborative tasks involving one or more team(s) of robots require spatial partitioning of the region of interest. For example, for tasks like search and rescue, patrolling, and landmine detection, the robots need to sweep and clear the region of interest as early as possible. In such scenarios, it will be quite difficult for a single robot to accomplish the task in an acceptable amount of time. It will be advantageous to use a team of collaborative robots that can efficiently decompose the task as early as possible.

Decomposition of tasks among the teams of robots is a very challenging problem because a large variety of factors need to be considered. Some of these factors may be imperfect information of the environment, changing environmental conditions, varying performance of the robot as a result

[1] The work in this chapter is derived from the work published in [227, 228]

Figure 7.1: This is an illustration of a port scenario where a region has been demarcated for exploration by multiple USVs.

of its interaction with the environment or risk of collision with external entities (e.g., civilian traffic). The algorithm used to decompose the task has to ensure that the workload has been effectively distributed among all the robots according to their capabilities. Also, the algorithm should be intelligent enough to re-plan dynamically when it receives more information about the robots' operating condition as well as environmental conditions.

Figure 7.1 shows an illustrative problem scenario. Suppose one is interested in exploring some region inside the port efficiently using multiple USVs. The region of interest (green polygon) may have obstacles (red polygons). For an efficient exploration of the region to take place, one can partition the environment among the USVs to ensure that when the USVs start covering their assigned sub-regions, they complete exploring the region of interest efficiently. Once such a partitioning has been completed, planning for the motion of the USVs can be performed using methods described in [229–232].

The proposed method helps to decompose the exploration task for a given region among multiple USVs. It is different from the geometrical area partitioning because one has to take into

account the different constraints to USV motion in the region of interest. To achieve this, the time taken by the bottleneck USV is minimized, which in turn helps in achieving workload balancing among all the USVs. Such cases are also considered where information regarding environmental conditions is unknown.

7.2 Problem Formulation

7.2.1 Definitions

Let A be the planar region that is to be surveyed by the team of USVs. Region A is the closure of a subset of the Euclidean space $\mathbb{R}^2$ in the usual topological sense.

$$A = \left[(x, y) \mid (x, y) \in A \subset \mathbb{R}^2\right] \tag{7.1}$$

It is assumed that the region A is a simply-connected region of the Euclidean space $\mathbb{R}^2$. Simply-connected means there exists a path completely contained in A between any two arbitrary locations that lie inside the region. In other words, the planar region A does not consist of any disjoint regions.

Let, $\mathbf{O}_A$ be the set of all non-traversable or obstacle regions that lie inside the planar region A. These are the land areas that cannot be traversed by the USVs or port areas that must be avoided. Each obstacle $O_i \in \mathbf{O}$ is a simply-connected region like the region of interest A. Now, the traversable area within the area of interest A can be computed as $A_p = A \setminus O_A$. Here, A_p is the traversable area that needs to be surveyed by the team of USVs. The simply-connected property of region A remains intact even after removal of the simply-connected obstacles $\mathbf{O}$ from the region. Thus, it can be stated that the region A_p is also a simply-connected traversable area. The boundary (denoted by ∂A_p) of the region A_p is assumed to be non-traversable wherever it intersects with obstacles $\mathbf{O}$.

It may happen that for a given dynamic model of a USV, a path respecting nonholonomic constraints may not exist between two points in the region A_p even if a geometrical path exists. This problem only occurs when each USV starts coverage of its claimed sub-regions. In such cases, the USVs should ignore covering areas unreachable physically and then return the non-traversed regions to the mission planner so that the area can be reassigned to another team of USVs for exploration if required. If the obstacle-deducted region is not path-connected, a mission planner should have assigned the unconnected regions to different teams in the first place. However, since the current chapter is mainly focused on exploratory task decomposition among multiple USVs, the nonholonomic coverage problem is not addressed here.

Let the team of unmanned surface vehicles (USVs) be denoted by $\mathbf{T}_n$, where n is the total number of available USVs that may be used to survey the region A_p. Each USV in the team is denoted by $U_i \in \mathbf{T}_n$, where $i \leq n$. In this chapter, the primary focus is on studying a partitioning problem where a team of USVs $\mathbf{T}_n$ is assigned to perform a collaborative exploration task of region A_p. Here, it is assumed that the USVs are homogeneous and have the same physical capabilities. It is also assumed that a functioning global communication network exists among the USVs in the assigned team.

While performing the collaborative exploration task, each USV $U_i \in \mathbf{T}_n$ will explore a subset of region A_p. Hence, the team of USVs has to divide the exploration tasks among themselves in an optimal manner. Essentially, this particular case reduces to a variant of area partitioning problem. Each USV claims sub-regions, $A_{p1}, A_{p2}, ..., A_{p|T_n|}$, in region A_p such that the disjoint union of each sub-region sums up to the original area to be explored.

$$A_p = A_{p1} \cup A_{p2} \cup \cdots \cup A_{p|T_n|} \tag{7.2}$$

Equation 7.2 implies that all locations within the region of interest A_p are explored by one USV only. Moreover, the USVs are not allowed to revisit the location it has already visited. This

is enforced in the way each USV is allowed to expand its claimed area, which will be discussed later. It is assumed that when a region A_p was assigned to team T_p, then the region was small enough so that by the expected amount of time in which the exploration has been completed, the conditions in the region A_p have not changed significantly. Hence the area of region A_p suitable for exploration by a team of USVs will vary from port to port as well as within the port.

During the exploration task, the motion of each USV will be affected by the surface currents, sea depth, and wakes generated by other civilian vessel and USVs. These factors introduce uncertainty in the positions of the USVs and also lead to noise in the controller. The positional uncertainties and controller noises increase with the increase in the USV's speed. Thus, the USVs operating at high speeds have large uncertainty in their positions and reduced controllability of the vessel, and vice versa. Therefore, the USVs operating at high speeds have a higher probability of collision with other civilian vessels and the static obstacles $\mathbf{O}$. To handle the collision-risk, a velocity-map $V(a_i)$ is introduced that decides the maximum operating speed for the USV operating in the region $a_i \in A_p$ by keeping the collision-risk within acceptable bounds. The maximum operating speeds of a USV governs the rate at which the USV can explore the sub-region. In the regions with higher operating speeds (or lower risk), the USV will cover larger areas as compared to the regions with lower operating speeds (or higher risk).

7.2.2 State Space Representation

The continuous region of interest A_p is discretized with a uniform grid of δl. During the division of region A_p into sub-regions, it is assumed that each USV can traverse at least one grid cell in a unit time interval. Thus, the size of δl is decided by the length of duration of discrete time interval (δt) and the minimum operating speed of USV (v_{min}), i.e., $\delta l = v_{min} \cdot \delta t$. The discretized region of interest is denoted by $A_{p,d}$. Each cell in $A_{p,d}$ is represented by state $\mathbf{s} = (x, y) \in \mathbb{Z}^2$, where $1 \leq x \leq x_{max}$ and $1 \leq y \leq y_{max}$. Let $\mathcal{S}$ be the set of all the states from region $A_{p,d}$.

Neighbors of each state $\mathbf{s}$ are determined by performing a single step of Manhattan moves from $\mathbf{s}$ in each direction, i.e., North, South, East, and West. Let $Neigh(\mathbf{s})$ be a function that returns all the feasible neighbors of state $\mathbf{s}$. Finally, the discrete region $A_{p,d}$ is approximated by a polygon B_A, such that all the cells of region $A_{p,d}$ intersect with or lie in the interior region of B_A. The discrete velocity-map $V_d(\mathbf{s})$ provides the maximum velocity the USV can operate at state $\mathbf{s}$.

7.2.3 Problem Statement

The goal is to seek an optimal division of area such that the area is covered in minimum time and work-load is optimally balanced among the USVs. This is achieved by minimizing the time taken by the bottleneck USV based on Minimax style of decision-making [233].

Formally, given (a) Region of interest $A_{p,d}$, (b) Team of USVs, T_n, and (c) Velocity-map, $V_d(\cdot)$, then one must compute optimal partitions of the region $\mathbf{P}_i \in \mathscr{P}$ for each USV $U_i \in T_n$, where $i \in [1, n]$. Each partitioned region $\mathbf{P}_i \in \mathscr{P}$ is computed such that, the time taken by the team of USVs T_n is minimized to explore then entire region $A_{p,d}$.

7.3 Overview of Approach

Below, the technique is described in detail. It is used to partition a region among multiple USVs based on a spatially distributed speed profile stored in a velocity-map. In this method, different initial placements of the USVs along the boundary of the polygon give rise to different partitions for the same region. So one must find that initial placement of USVs for which the time taken by the last USV to finish claiming its area is minimum. No other initial placement should result in lower time for the last USV to finish. It is with regards to this criterion that a particular partitioning result will be referred to as optimal or not.

The rest of the chapter is organized as follows: Section 7.3.1 describes how each USV grows its area based on speed constraints. Section 7.3.2 gives the objective function to be minimized.

Section 7.4.1 describes how the area partitioning simulations are implemented for known velocity-maps. Then, some helpful illustrations (Sec. 7.4.1) and computational performance plots (Sec. 7.4.2) are provided. In Section 7.5, the strategies are employed to handle unknown velocity-maps.

7.3.1 Partitioning the Region of Interest

The discretized region of interest $A_{p,d}$ consists of discrete states $\mathbf{s} \in \mathcal{S}$. Let N_{total} be the total number of discrete states in $\mathcal{S}$. It is desirable to compute the partitioned region for each USV $U_i \in T_n$ while minimizing the time taken by bottleneck USV during its exploration of the region $A_{p,d}$. A partition $\mathbf{P}$ of the region is a collection of n mutually exclusive and exhaustive subsets of discrete states $\mathcal{S}$, here n is total number of USVs in team T_n.

Algorithm 7 Determination of discretized region of interest

Input: Polygon representing region of interest, obstacles, and grid structure
Output: Set of grid-cells $A_{p,d}$

1: Compute all cell-centers that collide with polygon and store as Boolean matrix
2: Use matrix to calculate which cell has at least one vertex intersecting with the polygon and mark such cells as part of discrete polygon $A_{p,d}$
3: **for all** cells not in discrete polygon **do**
4: **for all** vertices in polygon **do**
5: Check intersection of polygon vertex with cell
6: **if** intersection found **then**
7: Add cell to discrete polygon $A_{p,d}$
8: **end if**
9: **end for**
10: **end for**
11: Repeat Lines $3-10$ for all obstacle polygons
12: **for all** cells in discrete polygon $A_{p,d}$ **do**
13: **if** cell not intersecting with an obstacle as found in Step 11 **then**
14: Cell remains in $A_{p,d}$
15: **else**
16: Remove cell from $A_{p,d}$
17: **end if**
18: **end for**

Computing the discretized region of interest $A_{p,d}$ involves using polygonal collision checking routines. $A_{p,d}$ denotes a contiguous set of cells in the grid imposed on top of the continuous region. The recipe for calculating this set of cells is shown in Algorithm 7 which is described as follows. In computational geometry, point-in-polygon (PIP) algorithms are computationally very

efficient and can quickly decide whether a given point in the 2-D plane lies inside the boundary of a polygon or not.

One call of such a software routine may be used to determine all grid cells whose centers lie inside the polygon and store the information from this collision-checking in the form of a Boolean matrix. Using standard geometric tricks, that matrix may be used to determine all cells whose corners may be inside the polygon. All these cells are marked as included in the set $A_{p,d}$. Of the excluded cells, one must again determine computationally as to whether any vertex of the polygon representing region A lies inside these cells. All those cells with which at least one polygon vertex intersects is also included in the set $A_{p,d}$. But now one must remove all those cells which are intersecting with the obstacle cells. For this purpose, the same procedure is repeated that was carried out using the polygon representing region A, but this time using the polygons representing each obstacle sequentially. This generates a set of cells that intersect with the obstacle polygons. All these *obstacle* cells must be excluded from the previous set of cells identified only using the polygonal region A. Then only the set of cells $A_{p,d}$ must be explored using the USVs.

Algorithm 8 Boundary Characterization

Input: $n_A \times 2$ matrix of polygon vertices
Output: Array of edge lengths l_i, polygon perimeter $peri(A)$
1: $peri(A) \leftarrow 0$
2: **for all** edges of polygon **do**
3: Compute and store edge length l_i
4: $peri(A) \leftarrow peri(A) + l_i$
5: **end for**

The boundary of the polygon representing the region A_p is parameterized by variable $\rho \in [0, 1)$. Here, $\rho = 0$ represents the starting point of the closed polygon. Note that $\rho = 1$ is excluded because it corresponds to the same point as $\rho = 0$, i.e., the starting point on the boundary. The boundary itself is also discretized with a suitable step-size $\delta\rho$. This essentially reduces the boundary to an array of vertices from where area exploration is started by a team of USVs. For example, if $\delta\rho = 0.1$ then a USV may start from the boundary point denoted by $\rho = 0.9$. Note a finer assignment such as $\rho = 0.92$ is not possible because of the chosen scheme of boundary

discretization. So choosing a correct step-size to discretize the boundary is an important design decision taken to bring the approximate optimal boundary assignments to the USV as close as possible to the true optima without increasing the computational costs by a huge margin. For computing the partition of a USV that has been assigned the boundary point $\rho = 0.9$, one would require further characterization of the boundary. This is shown in Algorithm 8. Considering the polygonal region A consisting of $|A| = n_A$ vertices, the perimeter of the polygonal boundary is computed by traversing over n_A edges while also storing the individual length of each edge. This is important as then one can use this information to map a particular ρ value to its abscissa and ordinate. One needs to track the distance measuring $\rho \times peri(A)$ units from the starting vertex $\rho = 0$. The edge lengths are used to determine which edge this vertex will lie in, and then the unitary method is used to determine the exact intermediate point on this edge.

Once each USV has been assigned a boundary point, one readily obtains the starting states for each USV, $\mathbf{s}_{i,init}$. However, the mapping from $\rho-$space to states in grid-region $A_{p,d}$ is generally not injective though this also depends on the step size chosen to discretize the grid. Also, those boundary points which are inside obstacles will never be assigned as starting points for any of the USVs.

At each time-step, the USVs can only move at speeds prescribed by the velocity-map, $V_d(\mathbf{s})$. The results shown below are for velocity-maps based on Cartesian, or polar system as well as contour lines. The speed is expressed in the units of number of cells traversed per unit time. Suppose for the USV starting at grid cell $\mathbf{s}$, the maximum allowed velocity is denoted as $V_d(\mathbf{s})$. So, the USV U_i will attempt to claim $V_d(\mathbf{s})$ states next time step to augment its currently claimed area. Also, denote the area claimed by USV U_i at iteration t by $a(t, i)$.

Initially, all the USVs are assigned states along the boundary polygon B_A of the region $A_{p,d}$. All states are labeled *open*. At this point, each USV $U_i \in T_n$ has claimed the state $\mathbf{s}_{i,init}$ at which they are initialized. Each USV has claimed area $a(1, i) = 1$. In order to partition the region $A_{p,d}$ into n convex shaped simply-connected regions, it will be advantageous for each USV to select

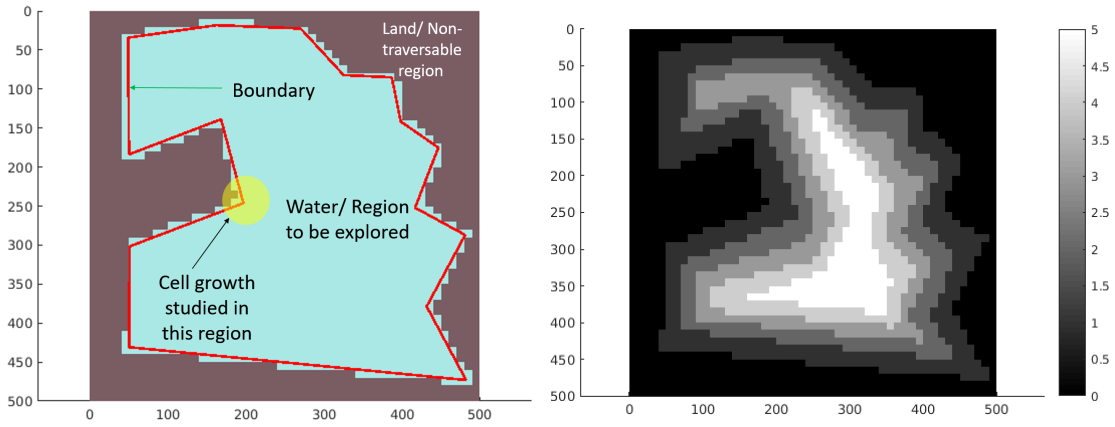


Figure 7.2: (a) This image shows the example scenario discussed in Figure 7.3, (b) Contour-lines based velocity-map is used to model low speeds for USVs near land.

states that are connected and nearest to current state of the USV. Let, C_{U_i} be the cache of all the possible states that USV U_i can claim in next time-step by maintaining the convexity of the partitioned region A^{U_i}. Here, A^{U_i} be the partitioned region for USV U_i and $\mathcal{S}_i$ be the set of all states belonging to region A^{U_i}. In the initial step, $\mathcal{S}_i$ will contain $\mathbf{s}_{i,init}$, and C_{U_i} will contain all the neighboring states provided by $Neigh(\mathbf{s}_{i,init})$. Let $c_{1,i}$ denote the maximum number of states that USV U_i can claim at time step $t = 2$. Clearly, $c_{1,i} = V_d(\mathbf{s}_{i,init})$. The cache must have at least $c_{1,i}$ states. Initially, cache of each USV, $C_U(i)$, is empty so at time $t = 1$, neighbors of state $\mathbf{s}_{i,init}$ are added to the cache $C_U(i)$.

Once the state $\mathbf{s}_{i,init}$ has been claimed by one of the USVs and its neighbors are generated, the state is labeled as *closed*. Since the starting state is a boundary point, at least one neighboring state will lie outside the grid-region $A_{p,d}$. But because non-convex polygons are allowed, it is possible to have $\mid Neigh(\mathbf{s}_{i,init}) \mid \leq 3$. If $c_{1,i} \leq Neigh(\mathbf{s}_{i,init})$, then the partitioning procedure continues with claiming states next time step from the current cache. States unclaimed at time $t = 2$ from the cache will remain in cache for future times. But if $c_{1,i} > Neigh(\mathbf{s}_{i,init})$, then those states in the cache, $C_U(i)$ that are *open* have their neighbors generated and stored in the cache. This process continues till the cache population, $|C_U(i)|$, equals or exceeds $c_{1,i}$.

The actual number of states $c'_{2,i}$ that the USV claims for time step $t = 2$ depends on conflicts with other USVs and also the availability of free states in its neighborhood to explore. Thus,

$$0 \leq c'_{2,i} \leq c_{1,i} \tag{7.3}$$

The frontier cells may be described as follows. At any time t, for a USV U_i it refers to the $c'_{t,i}$ states it actually claims. Thus, the set of states that the USV U_i has claimed at time step $t = 2$ can be denoted by $\mathbf{s}_{i,2} = [\mathbf{s}^1_{i,2}, \mathbf{s}^2_{i,2}, \cdots, \mathbf{s}^{c'_{2,i}}_{i,2}]$. These form the *frontier cells* for the USV U_i at time step $t = 2$. These frontier cells augment the current claimed area for the USV to $a(2, i) = a(1, i) + c'_{2,i}$.

The first cell $\mathbf{s}^1_{i,2}$ is chosen from the contending open states in the cache such that it augments the current claimed area of USV U_i with minimum loss of convexity. The chosen state should have minimum center-to-center *distance* from the center of the starting state of USV U_i. Such an expansion in an obstacle-free and unconstrained square lattice invariably leads to a circle-like expansion. So, this heuristic is also expected to give good results for non-convex regions.

The *distance* heuristic referred to above is not based on the classical Euclidean metric because the boundary contour itself maybe non-convex or contain non-convex polygonal obstacles inside it. Euclidean distance heuristic works well only in the absence of these concavities and holes in the map world.

Instead, the Dijkstra's algorithm is applied [234] to every lattice point, thereby computing the lengths of shortest collision-free paths through the region, A_p between each pair of lattice points as shown in Algorithm 9. The input for this algorithm is the undirected connectivity graph of the grid world along with its weighted edges. Weights are the Euclidean distances between neighboring states, hence either 1 or $\sqrt{2}$.

It is prudent here to allow diagonal connectivity as well to compute paths more tuned to the local concavities. The distances generated using such a look-up table is used as the heuristic in

guiding the growth of USV's claimed area, $a(t, i)$. The sub-regions claimed by USVs will not be convex in general because of the non-convex free region A_p.

Algorithm 9 Compute all-pairs shortest path-lengths

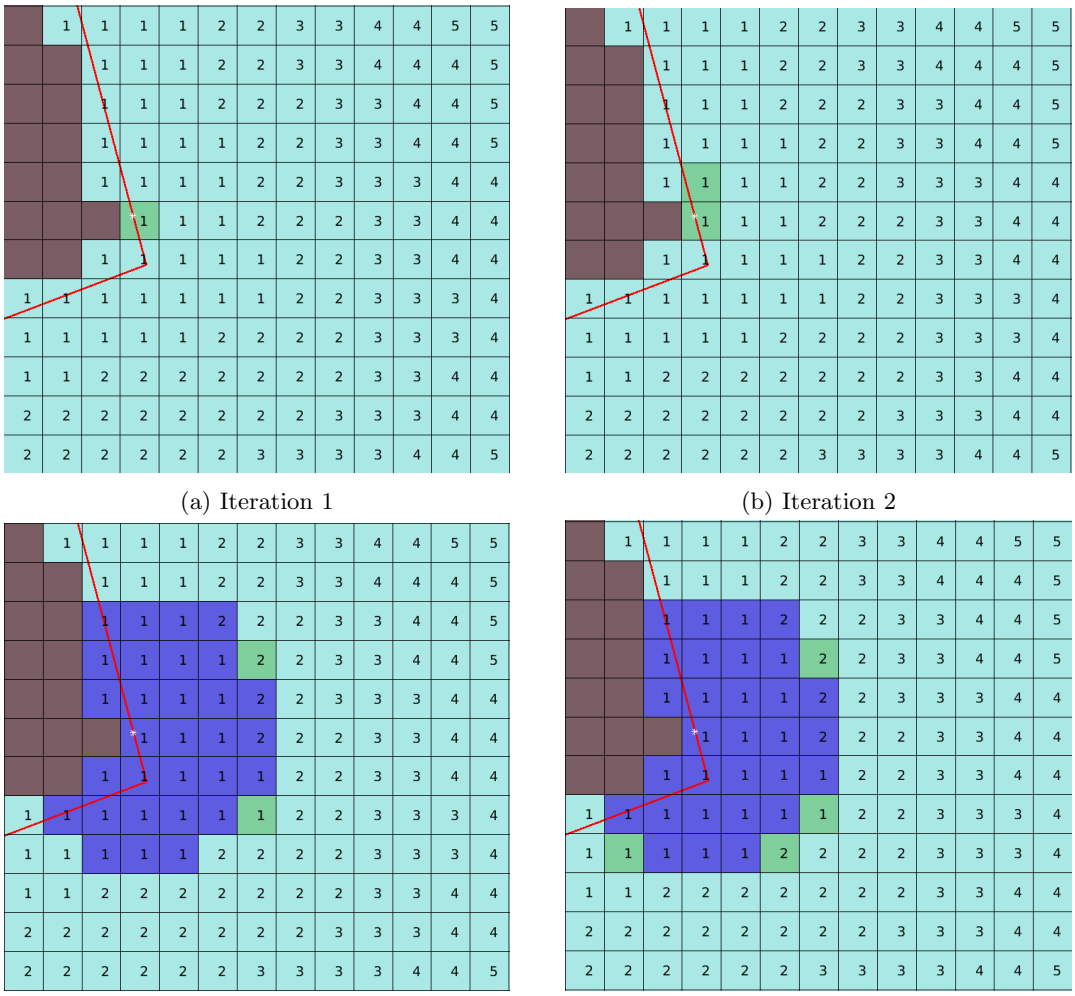
Input: Discretized region of interest $A_{p,d}$
Output: Look-up table
1: Initialize an empty undirected graph $\mathcal{G}$
2: **for all** cells in $A_{p,d}$ **do**
3: Add cell as a node in the graph $\mathcal{G}$
4: Compute neighbors and add as nodes if they lie in $A_{p,d}$
5: Add weighted edges to neighbors from cells
6: **end for**
7: Compute all-pairs shortest distances by applying Dijkstra's algorithm on $\mathcal{G}$
8: Store the distances in the form of a look-up table

Figure 7.3 demonstrates how a USV grows its area. The values shown represent the maximum speed allowed at each cell. Figure 7.3a shows the snapshot of iteration 29 where the frontier cells appear as green. There are 2 frontier cells having velocity values of 1 and 2. Thus their average is 1.5, which is rounded to 2. So, in the next iteration, the USV tries to claim 2 more states among the free states such that maximum convexity is maintained. Thus, in Figure 7.3b, 2 more states have been added to the existing claimed area of the USV. This procedure is summed up in equations below.

In the second iteration, there are possibly multiple frontier cells. Each cell corresponds to a value of maximum speed that can be attained around that cell. So one can take an average of these multiple maximum speeds to determine how many states should the USV U_i claim next time step. Hence, the following equation is obtained:

$$c_{2,i} = \frac{\sum_{k=1}^{c'_{2,i}} V_d(\mathbf{s}^k_{i,2})}{c'_{2,i}} \tag{7.4}$$

Thereafter, new neighbors are generated as before from the current frontier cells in a breadth-first search pattern. If more states are needed, then other open states in the cache can be expanded.



(a) Iteration 1

(b) Iteration 2

(c) Iteration 29

(d) Iteration 30

Figure 7.3: In the above images, the brown cells are outside the region of interest, the light blue cells represent the free, unexplored region and the dark blue cells represent regions already claimed by an USV for exploration. (c) The green cells represent the frontier cells for iteration 29. The scheme used for the above exploration is explained in detail in Section 7.3.1.

This process continues till the grid-region $A_{p,d}$ has been completely explored. Thus, the number of cells claimed evolve for each USV as follows:

$$c_{1,i} = V_d(\mathbf{s}_{i,init}) \tag{7.5}$$

As pointed out earlier, the actual number of claimed cells $c'_{t,i}$ may not be equal to the maximum number of cells that can be claimed, $c_{t,i}$.

$$c_{t,i} = \frac{\sum_{k=1}^{c'_{t,i}} V_d(\mathbf{s}^k_{i,t})}{c'_{t,i}} \qquad\qquad c'_{t,i} = f(c_{t-1,i}) \tag{7.6}$$

The exact number $c'_{t,i}$ is however computed by doing the simulations which is represented by function $f(\cdot)$ above. Also, area claimed by USV U_i evolves as follows:

$$a(1,i) = 1 \qquad\qquad a(t,i) = a(t-1,i) + c'_{t,i} \tag{7.7}$$

Next, a summary of this elaborate procedure is provided from the implementation point-of-view in Algorithm 10 to compute the partitions of a region given the initial boundary assignments to the USVs.

Area partitioning proceeds in iterations, and for each iteration, a random ordering of USVs is generated, and USVs start claiming their cells in that order. For this purpose, each USV maintains its own *open* list. This list is populated firstly by the starting cell of each USV.

In the first iteration, each USV claims the only cell in the open list, which is its starting cell. This is a trivial step. Each USV then adds the feasible neighbors of the starting cell into the open list. Each USV also computes the number of cells that it should claim in the second iteration in accordance with the velocity-map.

In subsequent iterations, each USV first populates its open list with enough cells to claim in that iteration. It does this by adding the neighbors of the cells in the open list. If there are still not enough cells in the open list, then one must include the neighbors of the cells in the open list at the start of this iteration.

Algorithm 10 Compute Partitions

Input: Discretized polygon, USV starting points
Output: Partitioned polygons

while region not explored **do**
  O_U = random ordering of USVs
  $iter \leftarrow 1$
  for USV u_i in O_U **do**
    if $iter == 1$ **then**
      remove current cell from ith open list
      assign current cell to USV i
      mark current cell as claimed
      generate neighbors of current cell and populate ith open list
      compute number of cells n_i to claim next time step using velocity map
    else
      populate open list with enough cells to claim
      if success in populating **then**
        sort unexplored cells in open list by shortest path-lengths
        for first n_i cells **do**
          remove current cell from ith open list
          assign current cell to USV i
          mark current cell as explored
        end for
        compute number of cells n_i to claim next time step using velocity map
      else
        for all unexplored cells in the ith open list **do**
          remove current cell from ith open list
          assign current cell to USV i
          mark current cell as claimed
        end for
        compute number of cells n_i to claim next time step using velocity map
      end if
    end if
  end for
  check if all cells in $A_{p,d}$ have been claimed
  $iter \leftarrow iter + 1$
end while

The number of times each USV may recursively keep exploring its neighbors in this fashion to populate its open list is kept fixed. For the simulations conducted for the present chapter,

partitioning proceeded smoothly if this procedure of expanding neighbors recursively stopped at the fourth level.

Mostly the open list gets populated way before reaching the fourth level of neighbor expansion. So neighbor search gets terminated mostly in first or second levels. If velocity maps assign very high area growth rates, then one may have to allow the possibility of higher levels of neighbor expansion.

If the open list is successfully populated with enough cells, then one must sort them in increasing order according to their distance in the geodesic sense from the starting cell for each USV and then select the cells in the open list for the corresponding USV according to this new ranking.

Then the number of cells that may be claimed by each USV in the next iteration is computed based on the velocity-map. It may happen that the open list of some of the USVs does not get populated as desired; then those USVs claim all the cells in their open lists.

7.3.2 Optimizing Partition

The states in the region $A_{p,d}$ are divided among multiple USVs in a finite number of time steps, t_{finish}. At each time step, the USVs search their neighboring unclaimed states and keep expanding their claimed sub-region for exploration. The iterations continue until all free states have been claimed.

Depending on the shape and size of the region $A_{p,d}$, each USV $U_i \in T_n$ will take a different amount of time to claim its sub-region. The USV which claims the last unclaimed state is considered to be the bottleneck agent. In the scenario where multiple USVs finish in the final iteration, any one of them can be considered the bottleneck agent. The time taken by each USV to claim all the possible free states depends upon the initial location of the USV on the boundary B_A. It is assumed that each USV is allotted a different initial state.

Suppose USV U_i takes time t_{if} to complete claiming states for exploration. Then the time step at which the process of area partitioning terminates as well as the bottleneck USV U_{i^*} is determined as follows:

$$t_{finish} = \max_i(t_{if}) \qquad\qquad i^* = \arg\max_i(t_{if}) \tag{7.8}$$

The initial starting states map to an $n-$dimensional vector, $\rho_U = [\rho_1, \cdots \rho_n]$. The vector elements should be sufficiently spaced so that two or more USVs do not claim to the same initial state. If the minimum time taken by the bottleneck USV is allowed to only be a function of vector ρ_U, then this problem can be formulated in terms of optimal initial placement ρ_U^* as follows

$$\rho_U^* = \arg\min_{\rho_U}\max_i(t_{if}) \qquad\qquad t_{finish}^* = \min_{\rho_U}\max_i(t_{if}) \tag{7.9}$$

One can also modify the objective function to take care of initial locations of the USVs. The USV U_i will be at some state outside region $A_{p,d}$ before exploration of the region has actually begun. The distance it will travel to reach state $\mathbf{s}_{i,init}$ should also be included in the cost incurred. Based on the maximum speed allowed (from velocity-map) at each state that will be traversed by USV U_i along the shortest path to reach state $\mathbf{s}_{i,init}$, one can calculate an optimistic estimate of time taken, t_{i0}.

$$\rho_U^* = \arg\min_{\rho_U}\max_i(t_{if} + t_{i0}) \tag{7.10}$$

7.4 Area Partitioning Using Known Velocity-map

7.4.1 Approach

While implementing the partitioning algorithm, the USVs are not implemented to behave like concurrently operating agents that claim states in the region of interest simultaneously. However,

if the USVs sequentially claim states, then there will be a bias in the partitions created among the USVs in favour of the USV which is assigned at the top of the list of USVs. Instead, a random order is generated for each discrete time-step using which the USVs claim the states from the set of free available states. This is the only source of randomness in this task decomposition technique.

During area partitioning, this protocol will lead to dead-locks for those time steps when multiple USVs lay claims on the same cell. To resolve such deadlocks, one again makes use of the random ordering of the USVs assigned to each iteration of the simulation. In case of a deadlock, the USV with higher priority at that time step wins the bid to its claim.

The scenarios used to evaluate the performance of the developed approach were assigned increasing degree of concavity to represent regions in ports such as narrow channels or small, enclosed regions in the harbor (see Figures 7.5, 7.6, 7.7 and 7.8). A scenario of size 500×500 meters is selected and discretized with a uniform grid-size of 10 meters.

Four types of velocity-maps are generated for testing this area partitioning scheme. The simplest example of such a velocity-map would be a uniform, constant map such that all states have the same value of maximum speed allowed. Then, maps where velocities for each state are defined as a function of its Cartesian (Figure 7.5a) or polar (Figure 7.7a) coordinates are also generated. The former was based on using linear gradients to signify speed zones. By applying a linear gradient horizontally and superposing it with another linearly graded risk profile in a vertical direction, one can obtain the velocity-map, as shown in Figure 7.5a. The horizontal profile prescribes maximum risk on the left, whereas the vertical profile prescribes maximum risk to the top of the area.

Figure 7.7a uses a velocity-map defined in terms of polar coordinates such that the risk constraints are described in terms of radius and azimuth rather than the usual Cartesian coordinates. The most useful is the contour map (Figure 7.9a), which assigns lower speeds near land and higher speeds away from land.

Topographic maps use contour lines to denote the elevation. It is not only an excellent visualization tool but also a very convenient way of recording features of the terrain. In the present chapter, the velocity constraints and underlying risk factors at different locations in the port region A_p are caused by features of the seabed such as depth, and features of the sea surface such as current.

Traffic and unaccounted objects flowing on sea surface also should be taken into account. Velocity-maps can also be represented as contour maps. One such velocity-map along with its example is shown in Figure 7.9.

The time taken by the bottleneck USV is only optimized in the present chapter over the boundary points assigned to each USV. This decision variable has already been defined by the $\rho-$vector, $\rho_U = [\rho_1, \rho_2, \cdots \rho_n]$. For a given scenario, the initial choice of n boundary points determines the partitions along with the randomness introduced in prioritizing the USVs at each time step.

Other factors like world size and region of interest are held constant in the current simulation. Thus, the mission completion time is minimized over the vector of starting states ρ_U. The objective function, $ap(\rho_U) = \max_i(t_{if}(\rho_U) + t_{i0})$ is non-linear. The elements of the decision vector are bounded in the subset $[0, 1)$. The objective itself is an ill-defined mapping as far as gradient-based optimization methods are concerned.

$$ap : (\rho_U \in [0, 1]^n) \rightarrow \mathbb{Z} \tag{7.11}$$

Since the mapping associates every bounded vector to a positive integer, it is expected not to be differentiable. Classical gradient-based methods are expected to perform poorly. Therefore, the PSO method is employed [235] to minimize the time taken by bottleneck USVs to explore the region. PSO is used because it efficiently handles higher dimensions and does not require the objective function to be differentiable.

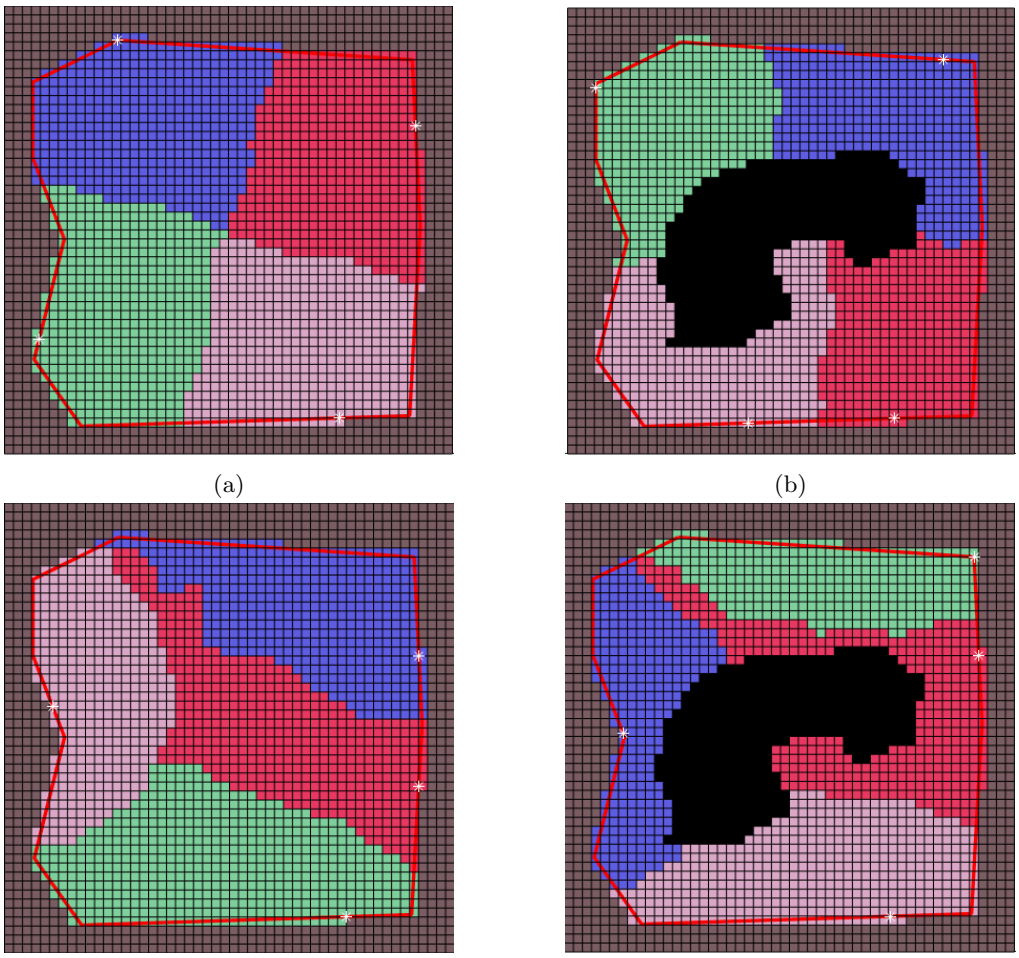
In the present chapter, the cognitive trust parameter is chosen to be 1.7 and the social trust parameter to be 2. A population size of 12 was chosen for optimization over four USVs, and PSO was run for 75 iterations. The tolerance for the convergence of the solution was chosen to be 0.001. The initial population is chosen as the uniformly spaced placement of USVs along the polygonal contour in terms of the ρ parameter, viz: 0, 0.25, 0.5, 0.75.

The above-discussed scheme of decomposing exploration tasks among multiple USVs is coded in MATLAB(R) software [236] on Ubuntu MATE 18.04.1 operating system. The workstation used was Dell(R) Precision Tower 3620 including eight Intel(R) Xeon(R) E3-1245V5 CPUs having 3.5GHz speed and a total of 32GB RAM. Computation time for generating partitioned regions for a team of four USVs exploring the region (see Figure 7.4c) and velocity-map (see Figure 7.5a) is approximately 17 seconds.

The partitioning of the area by the USVs to minimize the time taken by the bottleneck USV depends heavily on the velocity-map available. This is because velocity-map determines the maximum velocity allowed at each cell, and hence the maximum number of states that can be claimed by the USVs at each time step.

A constant velocity-map is compared with the linear velocity-map in Figure 7.4. For the constant risk case such that maximum speed allowed is $4\,m/s$ for each cell, as in Figure 7.4a, the optimal bottleneck time obtained through simulation is 113 seconds whereas for Figure 7.4b it is 89 seconds. The addition of obstacle reduces the number of states to be explored and hence results in less time taken for the latter case. Also, the constant velocity-map leads to four fairly equally divided regions for the four USVs. The number of USVs is only representative here and can be varied as needed.

In Figure 7.4c, the total number of iterations required for dividing the region among multiple USVs is 131 whereas for Figure 7.4d it is 103. Again as expected, the linear case without obstacles requires larger time than the corresponding constant-map case. This is due to the risk profile in Figure 7.5a, which prescribes maximum speeds of magnitudes as low as $2\,m/s$. In contrast, the



(a) (b)

(c) (d)

Figure 7.4: (a-b) Uniform, constant velocity-map, (c-d) Linear velocity-map

constant velocity-map prescribes speeds of magnitude $4\,m/s$, irrespective of the cell location. And, though some states in the non-uniform case do achieve speeds of magnitude as high as 6, it is clear from the simulation that the constant-map case completes faster. This is because only a few cells with such high speeds fall inside the region of interest. This pattern is observed in corresponding obstacle cases as well (Figure 7.4b and 7.4d). Note that in the obstacle cases, the region of interest is not divided equally among all the four USVs. This is because the presence of obstacle results in high concavities in the free region that will be explored. Similar behavior will be seen in the absence of obstacles if the region of interest A_p itself is highly concave. Also, since the top and

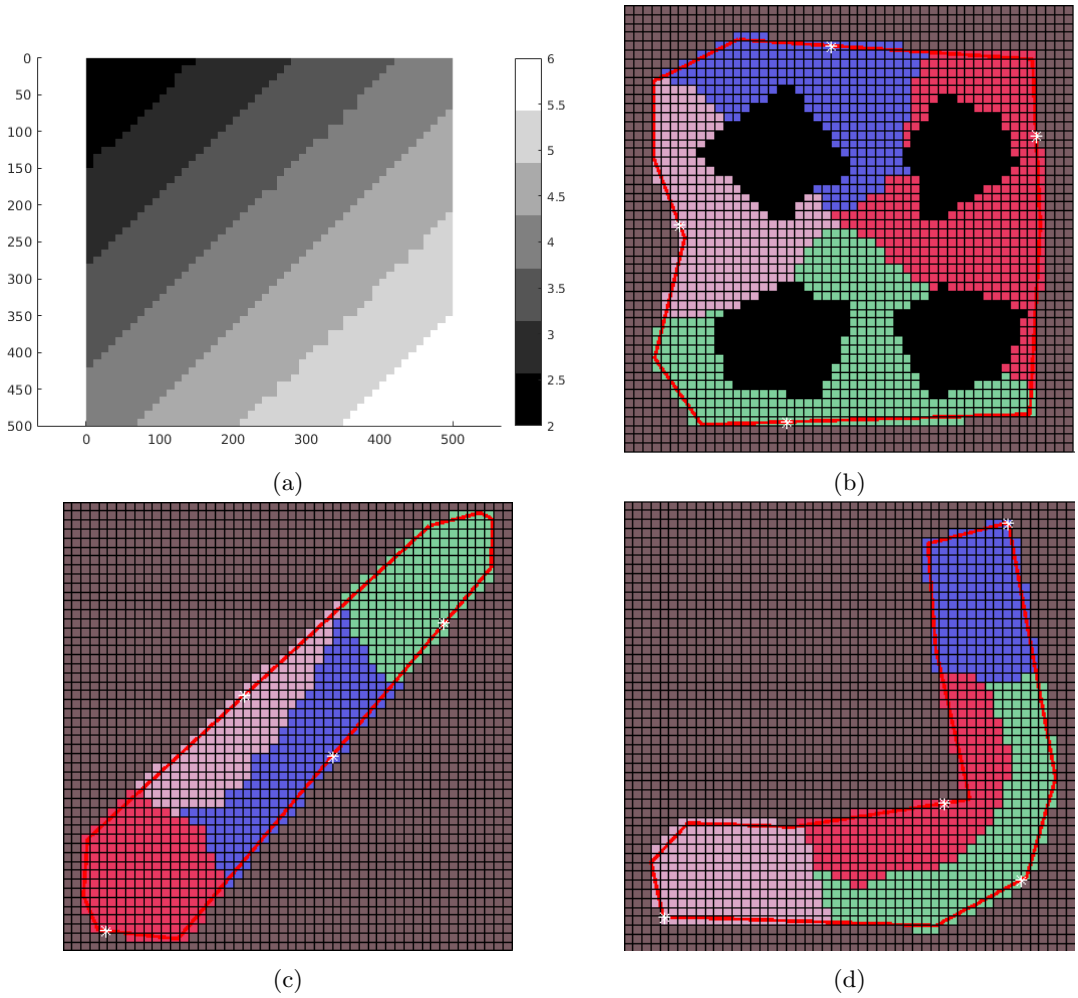


Figure 7.5: (a) Linearly graded velocity-map, (b-d) Application on manually created regions.

left states of the map have lower speeds while the right and bottom states have higher speeds, so USVs originating in the lower right zones claim larger area for exploration than the USVs starting on the left in Figure 7.4c.

Referring back to Figures 7.4a and 7.4c, if the polar map is applied to the same region in Figure 7.7b, then the optimal partition in this case is very different. In Figure 7.5(c-d), the linear map is applied to a convex and non-convex region. Then, the map is applied to regions modeled using real scenarios obtained from Google(R) Maps in Figure 7.6.

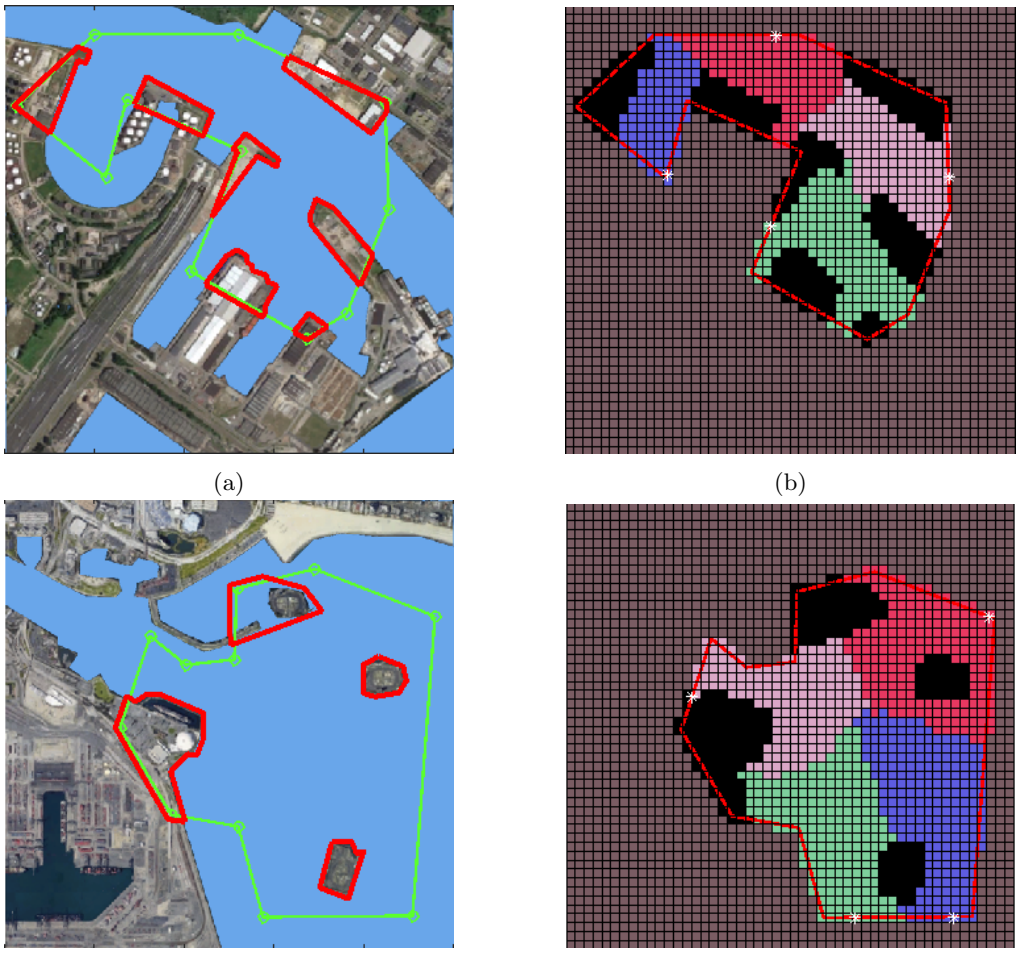


Figure 7.6: Application of the linearly graded velocity-map on regions based on real maps.

In Figure 7.7(c-d), the polar map is applied to two highly non-convex regions. The polar map is then also applied to two regions obtained from Google(R) Maps in Figure 7.8 but a smaller spatial discretization of 5 meters is chosen to model the narrow channels correctly. Otherwise, they get treated as obstacles when spatial step size is chosen to be 10 meters.

7.4.2 Computational Results

In this section, the effectiveness of the algorithm is evaluated by varying the number of USVs from 2 to 10. Minimization of time taken by the bottleneck USV requires each USV in the team

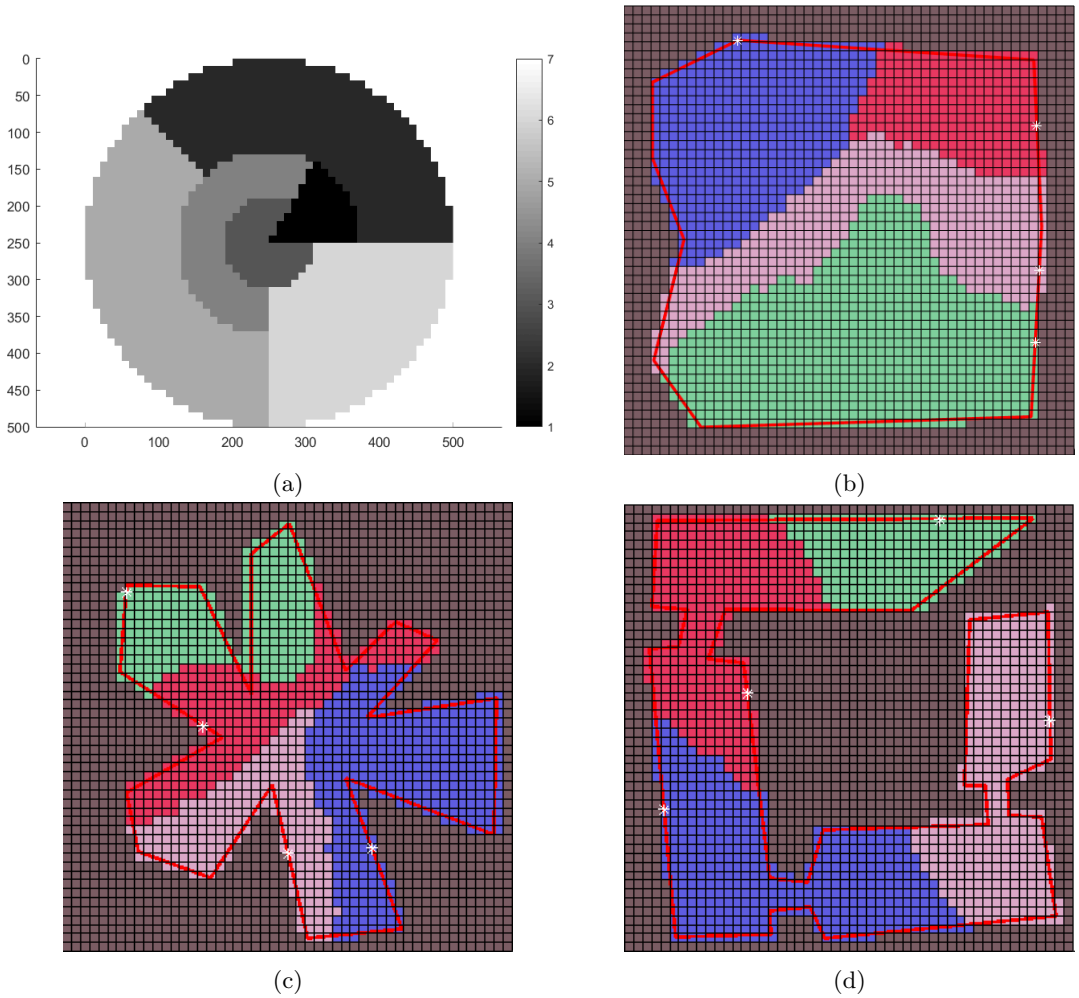
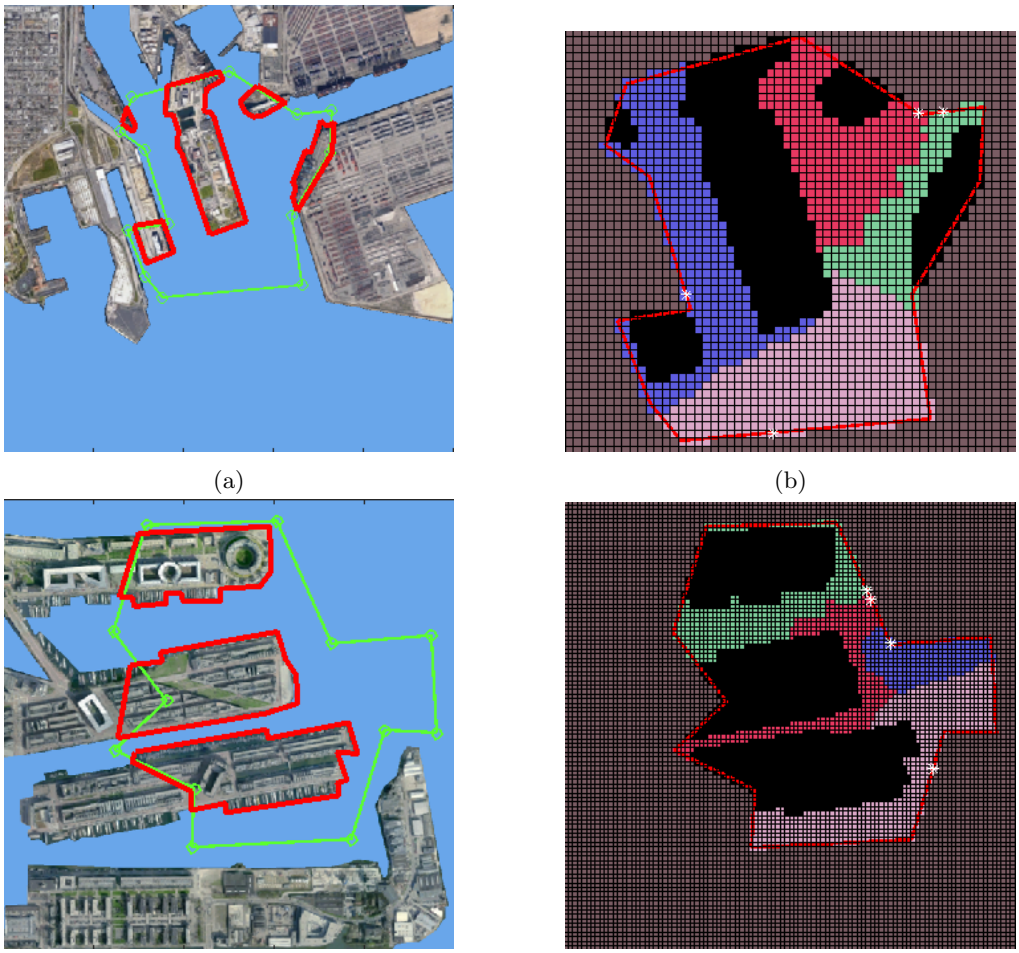


Figure 7.7: (a) Polar velocity-map, (b-d) Application on manually created regions.

to finish claiming all the possible remaining unexplored states at the same time as the bottleneck USV. Figure 7.10(a) shows a box-plot for different ranges of the final exploration time required by the team of USVs over 200 simulations. The range is computed by taking a difference between the maximum and minimum value of the data. Theoretically, the range values for the final exploration times of the USVs should be zero, but non-convexity associated with a region leads to deviation from ideal results as can be seen in the simulation data recorded. The values shown on the y-axis (see Figure 7.10(a)) are discrete values of time-step. In Figure 7.10(a), the median of the data is 2 while the minimum is zero, which is expected. The maximum range is capped at 6, and the



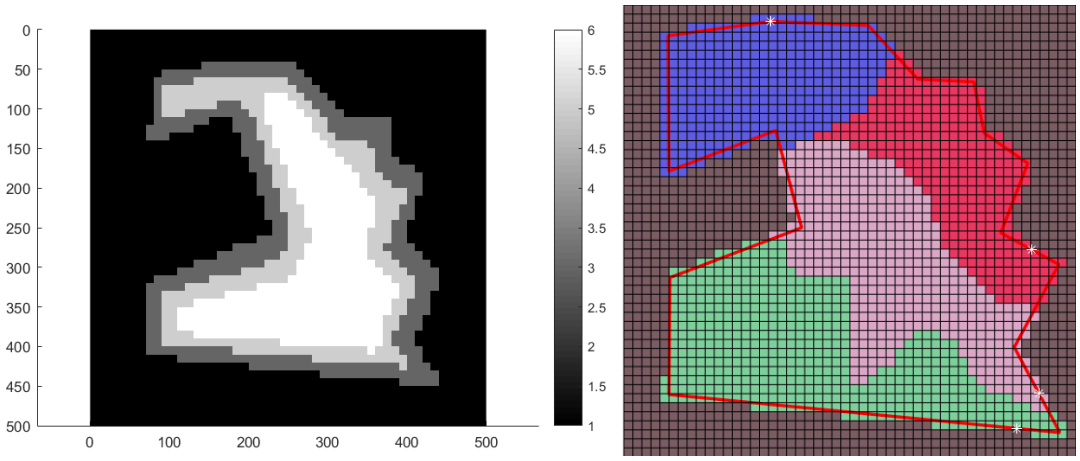
(a) (b)

(c) (d)

Figure 7.8: Application of the polar velocity-map on regions based on real maps.

interquartile range is 2. These results point to the successful minimization of time taken by the bottleneck USV.

The reasoning behind the zero range is related to the idea of optimal load balancing. If the bottleneck USV takes much higher time to compute its explorable region compared to other USVs, it is implied that other USVs could be prompted to claim more area to reduce the load on the bottleneck USV. This process can then ideally be continued until all USVs finish at approximately the same time. However, this still does not guarantee that the area will be equally divided. In particular, if a USV is in a high-risk zone, then it will cover less area than another USV operating

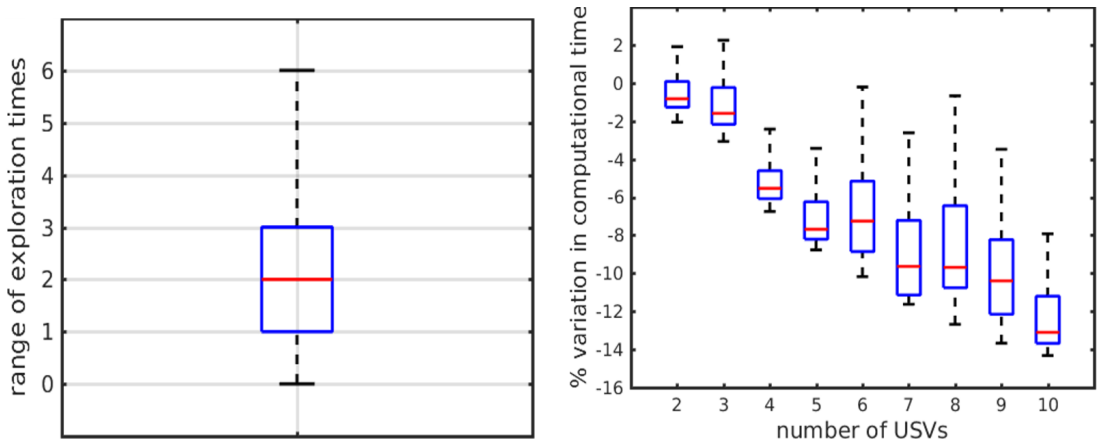


(a) (b)

Figure 7.9: (a) Contour-lines based velocity-map, (b) Corresponding area partition

in a low-risk zone for the same amount of time. Additionally, if a region is highly non-convex, then it may not be possible to partition the area among the USVs such that they finish almost at the same time. In such cases, the optimal bottleneck time may significantly be higher than the finish times of the other USVs. In spite of such issues, optimal load balancing is possible for most cases by computing the appropriate initial placement of the USVs.

Figure 7.10(b) shows the box-plot of the computational times obtained for partitioning the area by varying the number of USVs. The y-axis plots the percentage reduction in computation time with respect to the mean computation time required by the algorithm to compute the optimal partitions for two USVs. Ordinarily, the optimization with more number of USVs should take more computational time. However, with the increase in the number of USVs, the areas assigned to each USV reduce. This results in a decrease in the number of time-steps required to compute optimal partitions for each USV. So, the time to compute optimum partitions of the region for each optimization iteration reduces. This effect is more significant and leads to a reduction in overall computation time.



(a) (b)

Figure 7.10: (a) This image shows the range of times required by the USVs for exploration of given regions from over 200 simulations in the form of box-plot, (b) Here, the variation of computational time required for generating area partitions with respect to the number of USVs involved in exploration is shown.

7.5 Area Partitioning Using Unknown Velocity-map

7.5.1 Approach

In this section, the robustness of the area partitioning algorithm is evaluated when the velocity-map is unknown initially. The algorithm does not have perfect knowledge of the velocity-map while computing the initial exploration plan. However, while the USVs are exploring the region, they gather more information about the achievable maximum velocity and obtain the correct velocity-map. So, two approaches are developed to handle such scenarios of imperfect or unknown velocity-maps.

In the first approach, the optimal starting boundary points ρ_U^* for the USVs is determined based on uniform, constant velocity-map. USVs begin covering the region starting from the boundary points ρ_U^* but claim states based on speeds from the correct velocity-map. This is because even if the USVs do not know about the correct velocity-map but while exploring the region, USVs will move according to the actual environmental conditions. While the USVs are covering the areas in real-time, the central mission planner waits for the correct velocity-map to

be received. The planner then uses the correct velocity-map to compute the optimal partitions for the remaining region. During this computation, the planner excludes the states of the region $A_{p,d}$ that have already been covered by the USVs.

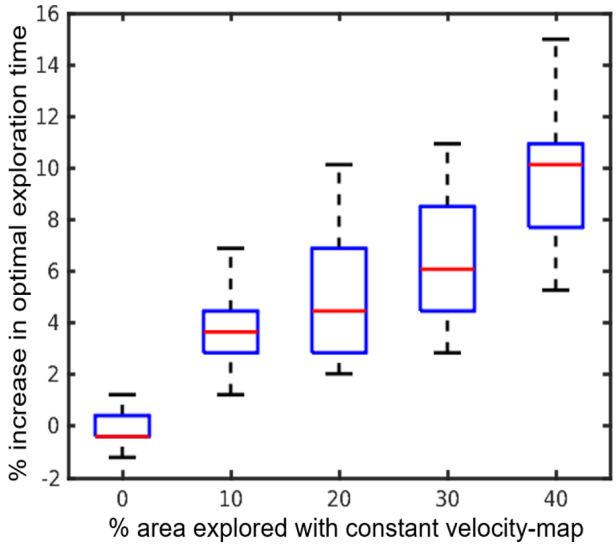
In the second approach, the USVs proceed with a noisy estimate of velocity-map until the correct velocity-map is obtained. The noisy velocity-map is assumed to be the best estimate of the correct velocity-map based on the sensor data that is available at that moment.

It is assumed that the time taken by the USVs to estimate the correct velocity-map is $t_{estimate}$. The optimal execution time required by the team of USVs to explore the region $A_{p,d}$ with perfect information of the velocity-map is denoted by $t^*_{optimal}$. Finally, the optimal execution time required by the team of USVs for the region remaining after time $t_{estimate}$ using the updated velocity-map is given by $t^*_{remaining}$. So one can say that $t^*_{optimal} < t^*_{remaining} + t_{estimate}$. If the values of the estimation time $t_{estimate}$ are low, then one can also state that $t^*_{optimal} \approx t^*_{remaining}$.

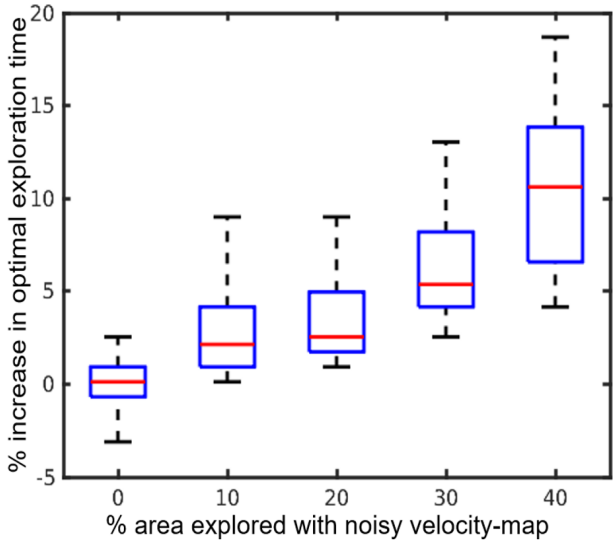
7.5.2 Computational Results

In this section, computational experiments are performed by varying the time taken, $t_{estimate}$, by the USVs to estimate the correct velocity-map. To simplify the implementation, the results are presented by varying the percentage area covered by the USVs before it acquires the correct estimate of the velocity-map (shown on the x-axis of the graphs in Figure 7.11). Figure 7.11 plots the percentage increase in execution time taken for exploration by the USVs from the true-optimal time $t^*_{optimal}$ on the y-axis. The simulation data presented in the form of box-plots are taken over 100 simulations with medians denoted by the red lines. The simulation scenario used for the experiments is shown in Figure 7.4.

Figure 7.11 gives plots that indicate that with more delay in estimating the correct values of the velocity map, there is a steady increase in the execution time incurred by the team of USVs. Both the approaches perform quite similar when the correct estimates of velocity-map are received before the USVs cover about 10% of the area. However, when the USVs cover more



(a)



(b)

Figure 7.11: In the absence of correct velocity-map, one can either use (a) constant map, or (b) noisy map; and record the time taken to complete partitioning the region based on how much time into the area partitioning process the correct map becomes available. The observed trend is explained in detail in Section 7.5.

than 20% of the area without correct estimates, then the approach which computes the initial plan using the noisy estimates performs marginally poor as compared to the first approach which starts with a uniform, constant map. This suggests that it is better to use the mean estimate of the velocity-map computed over time as compared to using currently available noisy estimates.

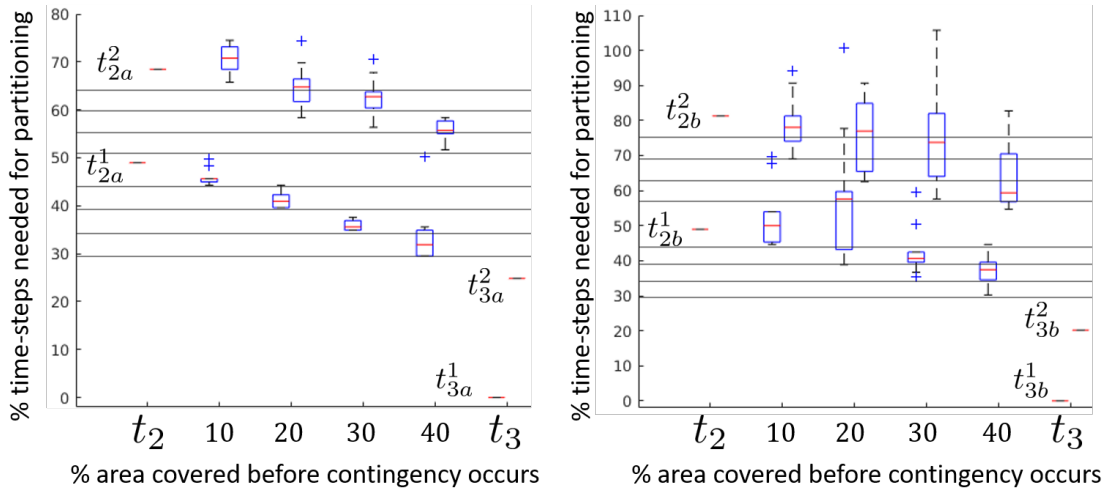
It is seen from the graph (shown in Figure 7.11) that the % increase in execution time taken by the USVs from the true-optimal time dips below zero. This is because the median time taken by the team of USVs when correct velocity-map is available from the beginning is set to zero.

7.6 Area Partitioning Under Variable Robot Availability

In this section, the effects of robots becoming suddenly unavailable in the middle of task execution is studied. It must be analyzed how robust this method is against such contingencies, and what are the effects of such events on the resulting task decomposition.

Suppose three USVs are assigned to a region for exploration. This region is partitioned and assigned to the USVs for exploration. Assume that while exploring, one of the USVs becomes unavailable. So, the rest of the region must again be partitioned between only the two remaining USVs. Let the time-steps it would take two USVs to partition the region completely be t_2, and for three USVs be t_3. Suppose that $\eta\%$ of the region was covered by the time one of the USVs becomes unavailable. Then the steps to partition the region with three USVs up to $\eta\%$, and with two USVs for the rest could be linearly approximated as $\eta t_3 + (1-\eta)t_2$. At least, this is what would be expected for a constant unit map and a convex region of interest. Following simulation experiments are run to analyze the effects of such robot unavailability.

Figure 7.12a shows plots for a scenario involving a convex region of interest R_a and a uniform velocity map with a constant speed of 4 cells per time steps. If this map is denoted with an index of 1, then t^1_{2a} denotes the time taken by two robots to cover region R_a using the $4-$uniform map. A nonuniform map which is similar to a linearly graded map shown before is used and denoted by index of 2. It has speeds varying from 2 to 6 cells per time step. Figure 7.12a shows two pairs of four horizontal black lines. These horizontal lines indicate the linear approximations obtained from the weighted averaging as shown above for both map scenarios.



(a) (b)

Figure 7.12: Effect of a robot becoming unavailable in the middle of task execution on the time to partition a (a) convex , and (b) nonconvex region

The $y-$axis shows the time-steps taken to partition the given region. It is represented in the percentage form with respect to t_{3a}^1. The $x-$axis shows the percentage of area covered after which one of the agent becomes unavailable. The plots show that transitioning from a uniform map to a nonuniform map worsens the time required to partition the region but it is well within 30% in each case. Again this is expected because the linear approximation will break for nonuniform maps. In the case of the nonuniform map, the deviation from the horizontal line is clearly seen to be more than the uniform case. Figure 7.12b shows similar patterns for a manually created nonconvex region R_b. It is expected that the linear approximation will perform even worse for a nonconvex region. Thus, the time to partition the region worsens between the uniform and nonuniform map but still stays around 30% with respect to the time taken by three robots to finish the task without interruption for the uniform map scenario. Thus, despite unfavourable scenarios the task completion times do not deviate erratically and remain bounded within reasonable limits.

7.7 Summary

In this chapter, an algorithm is developed that solves the area partitioning problem while taking into account the physical constraints applied to the USVs. The algorithm can optimally partition a large spatial region among multiple USVs using underlying velocity-maps well under one minute.

Chapter 8

Conclusions

This chapter presents the expected intellectual contributions and anticipated benefits from the work proposed in this dissertation.

8.1 Intellectual Contributions

This dissertation built computational foundations for handling contingencies proactively during missions involving multiple robots. Nominal mission features can be handled using fast heuristic methods. But for handling uncertain contingency information, optimization-based methods must be employed. Thus, to proactively incorporate contingencies into nominal mission plan in real-time, the problem must be partitioned to strike the right balance between optimization and heuristic domain. The right interplay between the use of optimization and heuristics can help obtain solution with high quality in a computationally efficient manner. This approach is used in Chapters 6 and 7. Another approach to create efficient contingency-aware plans is to develop a multi-layered structure for handling different levels of the problem. This was demonstrated through Chapters 3 to 5 where a model checking framework was used for encoding the mission model and queries using formal methods. On top of this, a sampling-based Monte Carlo simulator was used to quickly evaluate the performance of different contingency resolution strategies. And

finally a high-level reasoning framework was deployed to perform aggregate analysis for problems of larger sizes. Such multi-layered and combined approaches can be applied to solve other computationally challenging problems arising in multi-robot mission planning domain.

The key contributions according to the three main problems dealt within this dissertation are summarized in the following sections.

8.1.1 Modelling and Verification of Contingency Resolution Strategies for Multi-robot Missions

Chapter 3 describes a framework to do modeling of multi-robot missions with deterministic and probabilistic transitions based on a nominal mission description. The modeling of contingency resolution strategies is also demonstrated with specific examples of contingencies for two case studies. In the first case study, the correctness of contingency resolution strategies is verified, and NuSMV is used to generate mission execution plans for different scenarios. In the second case study, PRISM is used as a computational tool to provide a probabilistic assessment for mission success and analyze the effect of relevant mission variables on mission execution. It is demonstrated that subsystems can be analyzed individually, and their results can be aggregated to evaluate the performance of the proposed operations plan for the entire system, thereby helping to design more optimized systems. A modeling approach is presented for a production line set up across multiple robotic assembly cells. PRISM software is used to combine probabilistic modeling techniques with temporal logic based evaluation of the system. It is demonstrated how the cell settings could be modified based on feedback from the encoded mission model to optimize production performance.

8.1.2 Incorporation of Contingency Tasks in Nominal Task Allocation

The multi-robot task allocation approach developed in Chapter 6 efficiently handles contingency tasks by incorporating probabilistic analysis coupled with pruning to reduce the combinatorial explosion of computational requirements. The proactive approach is shown to outperform the conservative and reactive approaches to handling contingency tasks. This dissertation presents an algorithm for incorporating contingency tasks in a mission plan. It is expected that the proposed algorithm will be feasible for planning up to 10 contingency tasks because it takes under a minute in MATLAB implementation for five contingency tasks, and one could reasonably expect at least $80-90\%$ reduction for a parallelized C++ implementation. A higher number of contingency tasks such as those that were considered in this work will generally not be encountered in such missions.

8.1.3 Decomposition of Collaborative Surveillance Tasks

Chapter 7 develops an algorithm that solves the area partitioning problem while taking into account the constraints on the rates at which USVs can perform exploration tasks. The algorithm can partition a large spatial region among multiple USVs using underlying velocity-maps well under one minute. It is shown that by increasing the number of USVs, the computational time reduces. The performance of the algorithm was evaluated in scenarios where the velocity-map is unknown initially for a given length of time. Subsequently, the accurate velocity-maps becomes available after the USVs have explored part of the desired region. It is observed that the percentage increase in the optimal partitioning time remains below 5% even when the accurate velocity-map becomes available after $10-20\%$ of the area has been explored.

8.2 Anticipated Benefits

This dissertation presents methods to incorporate unexpected, contingency tasks that may adversely impact the progress of multi-robot missions. Autonomous systems are being deployed

rapidly in various scenarios to automate the operations and increase efficiency in the long run. However, the deployment of autonomous systems is not possible without guarantees against contingency events. Reactive methods of handling contingency events cannot provide such guarantees. Contingencies need to be handled proactively, which implies that it must be incorporated in nominal mission planning tasks like model verification, task allocation, and task decomposition. The approach in this dissertation is to set up simulation models that incorporate contingency information into nominal mission planning techniques in a deliberative and proactive manner, and then study and analyze the performance of this approach. Seeking such solutions provide a robust degree of confidence and guarantee to proceed with the operations of autonomous systems, especially when the operation site is complex and is frequently traversed by humans.

8.3 Future Work

The present work can be extended in the following directions:

- The work presented in this dissertation does not take unanticipated contingencies into consideration. Enabling the ability to process environmental data and predict new types of contingencies that may occur can help such automated contingency handling systems achieve much better performance than existing state-of-the-art systems. This would possibly require humans in the decision loop but computational tasks may still be carried out by the algorithms.

- A potential research direction for extending this dissertation is to develop interaction techniques for humans to effectively supervise robot teams and provide input on contingency handling. It would be also interesting to integrate the formal modeling approach presented in this dissertation with scheduling software packages or custom-designed algorithms. This is because scheduling allows the investigation of the behavior of involved agents in a more

detailed manner. In scheduling literature, operation performance is enhanced by studying the agent behavior, identifying undesirable behaviors, and addressing them.

- Limitations of the approach presented in this dissertation include the use of manually designed heuristics to plan for multi-robot missions. One could deploy machine learning algorithms for discovering task allocation heuristics. One could also deploy machine learning algorithms so that newer and better contingency resolution strategies are automatically generated based on sensory feedback and the history of human interventions. This will be effective for scenarios where contingencies occur at high frequencies amid rapidly changing mission conditions.

- Another current limitations of the approach presented in this dissertation is that robot teams of large sizes are not taken into consideration. With the advancement of technologies, it is however possible to deploy large teams of robots to accomplish complex and higher duration missions. One could develop scalable techniques that account for contingency occurrence in large robot teams using techniques from swarm robotics. Future work could also model issues arising for heterogeneous teams.

Reference List

[1] Yu Zhang and Lynne E Parker. Multi-robot task scheduling. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 2992–2998, 2013.

[2] James S Jennings, Greg Whelan, and William F Evans. Cooperative search and rescue with a team of mobile robots. In *Proceedings of IEEE International Conference on Advanced Robotics (ICAR)*, pages 193–200, 1997.

[3] Antonio Barrientos, Julian Colorado, Jaime del Cerro, Alexander Martinez, Claudio Rossi, David Sanz, and João Valente. Aerial remote sensing in agriculture: A practical approach to area coverage and path planning for fleets of mini aerial robots. *Journal of Field Robotics*, 28(5):667–689, 2011.

[4] Joseph L Baxter, EK Burke, Jonathan M Garibaldi, and Mark Norman. Multi-robot search and rescue: A potential field based approach. In *Autonomous robots and agents*, pages 9–16. Springer, 2007.

[5] George Kantor, Sanjiv Singh, Ronald Peterson, Daniela Rus, Aveek Das, Vijay Kumar, Guilherme Pereira, and John Spletzer. Distributed search and rescue with robot and sensor teams. In *Field and Service Robotics*, pages 529–538. Springer, 2003.

[6] Michal Pěchouček and Vladimír Mařík. Industrial deployment of multi-agent technologies: review and selected case studies. *Autonomous agents and multi-agent systems*, 17(3):397–431, 2008.

[7] John Enright and Peter R Wurman. Optimization and coordinated autonomy in mobile fulfillment systems. In *Proceedings of AAAI Workshop on Automated Action Planning for Autonomous Mobile Robots*, pages 33–38, 2011.

[8] Simon Bogh, Casper Schou, Thomas Rühr, Yevgen Kogan, Andreas Dömel, Manuel Brucker, Christof Eberst, Riccardo Tornese, Christoph Sprunk, Gian Diego Tipaldi, et al. Integration and assessment of multiple mobile manipulators in a real-world industrial production facility. In *Proceedings of International Symposium on Robotics (ISR/Robotik)*, pages 1–8, 2014.

[9] Hadi Ardiny, Stefan Witwicki, and Francesco Mondada. Construction automation with autonomous mobile robots: A review. In *Proceedings of RSI International Conference on Robotics and Mechatronics (ICROM)*, pages 418–424, 2015.

[10] Qin Li, Alexander Pogromsky, Teun Adriaansen, and Jan Tijmen Udding. A control of collision and deadlock avoidance for automated guided vehicles with a fault-tolerance capability. *International Journal of Advanced Robotic Systems*, 13(2):64, 2016.

[11] Brual C Shah and Satyandra K Gupta. Long distance path planning for unmanned surface vehicles in complex marine environment. *IEEE Journal of Oceanic Engineering*, Accepted for publication.

[12] Ariyan Kabir, Alec Kanyuck, Rishi K. Malhan, Aniruddha V. Shembekar, Shantanu Thakar, Brual C. Shah, and Satyandra K. Gupta. Generation of synchronized configuration space trajectories of multi-robot systems. In *IEEE International Conference on Robotics and Automation (ICRA)*, Montreal, Canada, May 2019.

[13] Wolfram Burgard, Mark Moors, Dieter Fox, Reid Simmons, and Sebastian Thrun. Collaborative multi-robot exploration. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, pages 476–481, 2000.

[14] Nicholas Roy and Gregory Dudek. Collaborative robot exploration and rendezvous: Algorithms, performance bounds and observations. *Autonomous Robots*, 11(2):117–136, 2001.

[15] Ioannis M Rekleitis, Gregory Dudek, and Evangelos E Milios. Multi-robot collaboration for robust exploration. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, volume 4, pages 3164–3169, 2000.

[16] Ashley Stroupe, Terry Huntsberger, Avi Okon, Hrand Aghazarian, and Matthew Robinson. Behavior-based multi-robot collaboration for autonomous construction tasks. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1495–1500, 2005.

[17] Dalong Wang, Dikai Liu, and Gamini Dissanayake. A variable speed force field method for multi-robot collaboration. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2697–2702, 2006.

[18] Y Uny Cao, Alex S Fukunaga, and Andrew Kahng. Cooperative mobile robotics: Antecedents and directions. *Autonomous robots*, 4(1):7–27, 1997.

[19] J-H Kim, H-S Shim, H-S Kim, M-J Jung, I-H Choi, and J-O Kim. A cooperative multi-agent system and its real time application to robot soccer. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, pages 638–643, 1997.

[20] R. K. Malhan, A. M. Kabir, B. Shah, T. Centea, and S. K. Gupta. Determining feasible robot placements in robotic cells for composite prepreg sheet layup. In *ASMEs 14th Manufacturing Science and Engineering Conference*, Erie, PA, USA, June 2019.

[21] R. K. Malhan, A. M. Kabir, A. V. Shembekar, B. C. Shah, T. Centea, and S. K. Gupta. Hybrid cells for multi-layer prepreg composite sheet layup. In *IEEE International Conference on Automation Science and Engineering (CASE)*, Munich, Germany, Aug 2018.

[22] R. K. Malhan, A. M. Kabir, B. C. Shah, T. Centea, and S. K. Gupta. Automated prepreg sheet placement using collaborative robotics. In *2018 SAMPE*, Long Beach, CA, USA, 2018.

[23] Prahar M. Bhatt, Ariyan M. Kabir, Rishi K. Malhan, Brual C. Shah, Aniruddha V. Shembekar, Yeo J. Yoon, and Satyandra K. Gupta. A robotic cell for multi-resolution additive manufacturing. In *IEEE International Conference on Robotics and Automation (ICRA)*, Montreal, Canada, May 2019.

[24] Prahar M Bhatt, Ariyan M Kabir, Max Peralta, Hugh A Bruck, and Satyandra K Gupta. A robotic cell for performing sheet lamination-based additive manufacturing. *Additive Manufacturing*, 2019.

[25] Prahar M Bhatt, Rishi K Malhan, and Satyandra K Gupta. Computational foundations for using three degrees of freedom build platforms to enable supportless extrusion based additive manufacturing. In *International Manufacturing Science and Engineering Conference*. ASME, 2019.

[26] A. M. Kabir, K. N. Kaipa, J. Marvel, and S. K. Gupta. Automated planning for robotic cleaning using multiple setups and oscillatory tool motions. *IEEE Transactions on Automation Science and Engineering*, 14(3):1364–1377, July 2017.

[27] A. M. Kabir, J. D. Langsfeld, S. Shriyam, V. S. Rachakonda, C. Zhuang, K. N. Kaipa, J. Marvel, and S. K. Gupta. Planning algorithms for multi-setup multi-pass robotic cleaning with oscillatory moving tools. In *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 751–757, Fort Worth, Texas, USA, Aug 2016.

[28] J. D. Langsfeld, A. M. Kabir, K. N. Kaipa, and S. K. Gupta. Integration of planning and deformation model estimation for robotic cleaning of elastically deformable objects. *IEEE Robotics and Automation Letters*, 3(1):352–359, Jan 2018.

[29] P. Rajendran, T. Moscicki, J. Wampler, and S. K. Gupta K. D. von Ellenrieder. Wave-aware trajectory planning for unmanned surface vehicles operating in congested environments. In *IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, Philadelphia, PA, U.S.A., August 2018.

[30] Pradeep Rajendran, Shantanu Thakar, and Satyandra Gupta. User-guided path planning for redundant manipulators in highly constrained work environments. In *IEEE International Conference on Automation Science and Engineering (CASE)*, Vancouver, Canada, August 2019.

[31] Pradeep Rajendran, Shantanu Thakar, Ariyan Kabir, Brual Shah, and S. K. Gupta. Context-dependent search for generating paths for redundant manipulators in cluttered environments. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, Macau, China, November 2019.

[32] Sarah Al-Hussaini, Jason M. Gregory, and Satyandra K. Gupta. A policy synthesis-based framework for robot rescue decision-making in multi-robot exploration of disaster sites. In *IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, Philadelphia, PA, U.S.A., August 2018.

[33] Sarah Al-Hussaini, Jason M. Gregory, and Satyandra K. Gupta. Generation of context-dependent policies for robot rescue decision-making in multi-robot teams. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Madrid, Spain, October 2018.

[34] Lynne E Parker. Multiple mobile robot systems. *Springer Handbook of Robotics*, pages 921–941, 2008.

[35] Yi Guo, Lynne E Parker, and Raj Madhavan. Towards collaborative robots for infrastructure security applications. In *Proceedings of International Symposium on Collaborative Technologies and Systems*, volume 2004, pages 235–240, 2004.

[36] HR Everett, RT Laird, DM Carroll, GA Gilbreath, and TA Heath-Pastore. Multiple resource host architecture (mrha) for the mobile detection assessment response system (mdars) revision a. Technical report, Space And Naval Warfare Systems Center San Diego CA, 2000.

[37] Lynne E Parker. Alliance: An architecture for fault tolerant multirobot cooperation. *IEEE transactions on robotics and automation*, 14(2):220–240, 1998.

[38] Cai Luo, Andre Possani Espinosa, Danu Pranantha, and Alessandro De Gloria. Multi-robot search and rescue team. In *Proceedings of IEEE International Symposium on Safety, Security, and Rescue Robotics*, pages 296–301, 2011.

[39] Rachid Alami, Sara Fleury, Matthieu Herrb, Félix Ingrand, and Frédéric Robert. Multi-robot cooperation in the martha project. *IEEE Robotics & Automation Magazine*, 5(1):36–47, 1998.

[40] Raffaele Limosani, Raffele Esposito, Alessandro Manzi, Giancarlo Teti, Filippo Cavallo, and Paolo Dario. Robotic delivery service in combined outdoor–indoor environments: technical analysis and user evaluation. *Robotics and Autonomous Systems*, 103:56–67, 2018.

[41] Ashley Stroupe, Avi Okon, Matthew Robinson, Terry Huntsberger, Hrand Aghazarian, and Eric Baumgartner. Sustainable cooperative robotic technologies for human and robotic outpost infrastructure construction and maintenance. *Autonomous Robots*, 20(2):113–123, 2006.

[42] Ali Narenji Sheshkalani and Ramtin Khosravi. Verification of visibility-based properties on multiple moving robots in an environment with obstacles. *International Journal of Advanced Robotic Systems*, 15(4):1729881418786657, 2018.

[43] Alan FT Winfield, Jin Sa, Mari-Carmen Fernández-Gago, Clare Dixon, and Michael Fisher. On formal specification of emergent behaviours in swarm robotic systems. *International Journal of Advanced Robotic Systems*, 2(4):39, 2005.

[44] Matthew L Bolton, Ellen J Bass, and Radu I Siminiceanu. Using formal verification to evaluate human-automation interaction: A review. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 43(3):488–503, 2013.

[45] Yash Vardhan Pant, Houssam Abbas, Rhudii A Quaye, and Rahul Mangharam. Fly-by-logic: control of multi-drone fleets with temporal logic objectives. In *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems*, pages 186–197. IEEE Press, 2018.

[46] Zhiyu Liu, Bo Wu, Jin Dai, and Hai Lin. Distributed communication-aware motion planning for multi-agent systems from STL and SpaTeL specifications. In *Proceedings of IEEE Conference on Decision and Control (CDC)*, pages 4452–4457, 2017.

[47] Zhiyu Liu, Jin Dai, Bo Wu, and Hai Lin. Communication-aware motion planning for multi-agent systems from signal temporal logic specifications. In *Proceedings of IEEE American Control Conference (ACC), 2017*, pages 2516–2521, 2017.

[48] Alexandros Nikou, Dimitris Boskos, Jana Tumova, and Dimos V Dimarogonas. Cooperative planning for coupled multi-agent systems under timed temporal specifications. In *Proceedings of IEEE American Control Conference (ACC)*, pages 1847–1852, 2017.

[49] Yuchen Zhou, Dipankar Maity, and John S Baras. Optimal mission planner with timed temporal logic constraints. In *Proceedings of IEEE European Control Conference (ECC)*, pages 759–764, 2015.

[50] Joseph Kim, Christopher J Banks, and Julie A Shah. Collaborative planning with encoding of users' high-level strategies. In *Proceedings of AAAI Conference on Artificial Intelligence*, pages 955–962, 2017.

[51] Douglas C MacKenzie, Ronald C Arkin, and Jonathan M Cameron. Multiagent mission specification and execution. In *Robot colonies*, pages 29–52. Springer, 1997.

[52] Alessio Lomuscio and Franco Raimondi. Model checking knowledge, strategies, and games in multi-agent systems. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 161–168. ACM, 2006.

[53] Yunus Emre Sahin, Petter Nilsson, and Necmiye Ozay. Synchronous and asynchronous multi-agent coordination with cLTL+ constraints. In *Proceedings of IEEE Conference on Decision and Control (CDC)*, pages 335–342, 2017.

[54] Jonathan A DeCastro, Javier Alonso-Mora, Vasumathi Raman, Daniela Rus, and Hadas Kress-Gazit. Collision-free reactive mission and motion planning for multi-robot systems. In *Robotics Research*, pages 459–476. Springer, 2018.

[55] Lars Lindemann, Christos K Verginis, and Dimos V Dimarogonas. Prescribed performance control for signal temporal logic specifications. *arXiv preprint arXiv:1703.07094*, 2017.

[56] Yu Ke, Antonios Tsourdos, Brian White, and Peter Silson. Dynamic mission planning of multiple unmanned autonomous platforms for sensing and autonomous urban reconnaissance. In *Proceedings of AIAA Guidance, Navigation, and Control Conference*, page 6216, 2009.

[57] Gordon Fraser, Franz Wotawa, and Paul E Ammann. Testing with model checkers: a survey. *Software Testing, Verification and Reliability*, 19(3):215–261, 2009.

[58] Georgios E Fainekos, Antoine Girard, Hadas Kress-Gazit, and George J Pappas. Temporal logic motion planning for dynamic robots. *Automatica*, 45(2):343–352, 2009.

[59] Morteza Lahijanian, Joseph Wasniewski, Sean B Andersson, and Calin Belta. Motion planning and control from temporal logic specifications with probabilistic satisfaction guarantees. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 3227–3232, 2010.

[60] Cameron Finucane, Gangyuan Jing, and Hadas Kress-Gazit. LTLMoP: Experimenting with language, temporal logic and robot control. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1988–1993, 2010.

[61] Savas Konur, Clare Dixon, and Michael Fisher. Analysing robot swarm behaviour via probabilistic model checking. *Robotics and Autonomous Systems*, 60(2):199–213, 2012.

[62] Fujio Miyawaki, Ken Masamune, Satoshi Suzuki, Kitaro Yoshimitsu, and Juri Vain. Scrub nurse robot system-intraoperative motion analysis of a scrub nurse and timed-automata-based model for surgery. *IEEE Transactions on Industrial Electronics*, 52(5):1227–1235, 2005.

[63] Thao Nguyen Van, Nugroho Fredivianus, Huu Tam Tran, Kurt Geihs, and Thi Thanh Binh Huynh. Formal verification of ALICA multi-agent plans using model checking. In *Proceedings of the Ninth International Symposium on Information and Communication Technology*, pages 351–358. ACM, 2018.

[64] Rong Gu, Raluca Marinescu, Cristina Seceleanu, and Kristina Lundqvist. Formal verification of an autonomous wheel loader by model checking. In *Proceedings of IEEE/ACM International FME Workshop on Formal Methods in Software Engineering (FormaliSE)*, pages 74–83, 2018.

[65] Sertac Karaman and Emilio Frazzoli. Linear temporal logic vehicle routing with applications to multi-UAV mission planning. *International Journal of Robust and Nonlinear Control*, 21(12):1372–1395, 2011.

[66] Meng Guo and Dimos V Dimarogonas. Task and motion coordination for heterogeneous multiagent systems with loosely coupled local tasks. *IEEE Transactions on Automation Science and Engineering*, 14(2):797–808, 2017.

[67] Philipp Schillinger, Mathias Bürger, and Dimos V Dimarogonas. Simultaneous task allocation and planning for temporal logic goals in heterogeneous multi-robot systems. *The International Journal of Robotics Research*, page 0278364918774135, 2017.

[68] Yasser Shoukry, Pierluigi Nuzzo, Ayca Balkan, Indranil Saha, Alberto L Sangiovanni-Vincentelli, Sanjit A Seshia, George J Pappas, and Paulo Tabuada. Linear temporal logic motion planning for teams of underactuated robots using satisfiability modulo convex programming. In *Proceedings of IEEE Conference on Decision and Control (CDC)*, pages 1132–1137, 2017.

[69] Maryam Kamali, Louise A Dennis, Owen McAree, Michael Fisher, and Sandor M Veres. Formal verification of autonomous vehicle platooning. *Science of Computer Programming*, 148:88–106, 2017.

[70] Mathilde Machin, Jérémie Guiochet, Hélene Waeselynck, Jean-Paul Blanquart, Matthieu Roy, and Lola Masson. SMOF: A safety monitoring framework for autonomous systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 48(5):702–715, 2018.

[71] Liangguo Liu, Jun Peng, Xiaoyong Zhang, Rui Zhang, Bin Chen, Kai Gao, and Yingze Yang. Reactive behavioral strategy for unmanned ground vehicle under liner temporal logic specifications. In *Proceedings of IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 133–140, 2018.

[72] Ankur M Mehta, Joseph DelPreto, Kai Weng Wong, Scott Hamill, Hadas Kress-Gazit, and Daniela Rus. Robot creation from functional specifications. In *Robotics Research*, pages 631–648. Springer, 2018.

[73] Jiyoung Choi, Seungkeun Kim, and Antonios Tsourdos. Verification of heterogeneous multi-agent system using mcmas. *International Journal of Systems Science*, 46(4):634–651, 2015.

[74] Laura Humphrey. Model checking UAV mission plans. In *Proceedings of AIAA Modeling and Simulation Technologies Conference*, page 4723, 2012.

[75] Clayton Rothwell, Alexa Eggert, Michael J Patzek, George Bearden, Gloria L Calhoun, and Laura R Humphrey. Human-computer interface concepts for verifiable mission specification, planning, and management. In *Proceedings of AIAA Infotech@ Aerospace (I@ A) Conference*, page 4804, 2013.

[76] Javier Alonso-Mora, Jonathan A DeCastro, Vasumathi Raman, Daniela Rus, and Hadas Kress-Gazit. Reactive mission and motion planning with deadlock resolution avoiding dynamic obstacles. *Autonomous Robots*, 42(4):801–824, 2018.

[77] Hasan Sinan Bank, Sandeep D'souza, and Aditya Rasam. Temporal logic (TL)-based autonomy for smart manufacturing systems. *Procedia Manufacturing*, 26:1221–1229, 2018.

[78] Philipp Schillinger, Mathias Bürger, and Dimos V Dimarogonas. Multi-objective search for optimal multi-robot planning with finite LTL specifications and resource constraints. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 768–774, 2017.

[79] Matthew R Maly, Morteza Lahijanian, Lydia E Kavraki, Hadas Kress-Gazit, and Moshe Y Vardi. Iterative temporal motion planning for hybrid systems in partially unknown environments. In *Proceedings of the international conference on Hybrid systems: computation and control*, pages 353–362. ACM, 2013.

[80] Indranil Saha, Rattanachai Ramaithitima, Vijay Kumar, George J Pappas, and Sanjit A Seshia. Automated composition of motion primitives for multi-robot systems from safe LTL specifications. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1525–1532, 2014.

[81] Meng Guo, Karl H Johansson, and Dimos V Dimarogonas. Motion and action planning under LTL specifications using navigation functions and action description language. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 240–245, 2013.

[82] Yi Li, Jing Sun, Jin Song Dong, Yang Liu, and Jun Sun. Planning as model checking tasks. In *Proceedings of IEEE Software Engineering Workshop*, pages 177–186, 2012.

[83] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[84] Dong Liu, Ming Cong, Qiang Zou, and Yu Du. A biological-inspired episodic cognitive map building framework for mobile robot navigation. *International Journal of Advanced Robotic Systems*, 14(3):1729881417705922, 2017.

[85] Shengbing Jiang and Ratnesh Kumar. Diagnosis of repeated failures for discrete event systems with linear-time temporal-logic specifications. *IEEE Transactions on Automation Science and Engineering*, 3(1):47–59, 2006.

[86] Krishnendu Chatterjee, Martin Chmelík, Raghav Gupta, and Ayush Kanodia. Qualitative analysis of pomdps with temporal logic specifications for robotics applications. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 325–330, 2015.

[87] Bruno Lacerda, David Parker, and Nick Hawes. Optimal and dynamic planning for Markov decision processes with co-safe LTL specifications. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1511–1516, 2014.

[88] Igor Cizelj, Xu Chu Ding, Morteza Lahijanian, Alessandro Pinto, and Calin Belta. Probabilistically safe vehicle control in a hostile environment. In *Presented at World Congress of the International Federation of Automatic Control (IFAC)*, 2011.

[89] Xuchu Ding, Stephen L Smith, Calin Belta, and Daniela Rus. Optimal control of markov decision processes with linear temporal logic constraints. *IEEE Transactions on Automatic Control*, 59(5):1244–1257, 2014.

[90] Fatma Faruq, Bruno Lacerda, Nick Hawes, and David Parker. Simultaneous task allocation and planning under uncertainty. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[91] Lu Feng, Clemens Wiltsche, Laura Humphrey, and Ufuk Topcu. Synthesis of human-in-the-loop control protocols for autonomous systems. *IEEE Transactions on Automation Science and Engineering*, 13(2):450–462, 2016.

[92] Marius Kloetzer and Cristian Mahulea. LTL-based planning in environments with probabilistic observations. *IEEE Transactions on Automation Science and Engineering*, 12(4):1407–1420, 2015.

[93] Yushan Chen, Jana Tumová, and Calin Belta. LTL robot motion control based on automata learning of environmental dynamics. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 5177–5182, 2012.

[94] Xuchu Dennis Ding, Brendan Englot, Alessandro Pinto, Alberto Speranzon, and Amit Surana. Hierarchical multi-objective planning: From mission specifications to contingency management. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 3735–3742, 2014.

[95] Tichakorn Wongpiromsarn and Richard M Murray. Distributed mission and contingency management for the DARPA urban challenge. In *Proceedings of International Workshop on Intelligent Vehicle Control Systems (IVCS)*, 2008.

[96] Jerry L Franke, Stephen M Jameson, Rosemary D Paradis, and Robert J Szczerba. Hierarchical contingency management system for mission planners, December 13 2011. US Patent 8,078,319.

[97] Anders Lyhne Christensen, Rehan O'Grady, Mauro Birattari, and Marco Dorigo. Fault detection in autonomous robots based on fault injection and learning. *Autonomous Robots*, 24(1):49–67, 2008.

[98] Steve McGuire, P Michael Furlong, Christoffer Heckman, Simon Julier, Daniel Szafir, and Nisar Ahmed. Failure is not an option: Policy learning for adaptive recovery in space operations. *IEEE Robotics and Automation Letters*, 3(3):1639–1646, 2018.

[99] Matthew T Long, Robin R Murphy, and Lynne E Parker. Distributed multi-agent diagnosis and recovery from sensor failures. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2506–2513, 2003.

[100] Patrick Ulam and Ronald C Arkin. When good communication go bad: communications recovery for multi-robot teams. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, volume 4, pages 3727–3734, 2004.

[101] Siddharth Mayya and Magnus Egerstedt. Safe open-loop strategies for handling intermittent communications in multi-robot systems. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 5818–5823, 2017.

[102] Liang Tang, Gregory J Kacprzynski, Kai Goebel, Abhinav Saxena, Bhaskar Saha, and George Vachtsevanos. Prognostics-enhanced automated contingency management for advanced autonomous systems. In *Proceedings of International Conference on Prognostics and Health Management (PHM)*, pages 1–9, 2008.

[103] Liang Tang, Bin Zhang, Jonathan DeCastro, and Eric Hettler. An integrated health and contingency management case study on an autonomous ground robot. In *Proceedings of IEEE International Conference on Control and Automation (ICCA)*, pages 584–589, 2011.

[104] Dong-Hyun Lee, Sheir Afgen Zaheer, and Jong-Hwan Kim. A resource-oriented, decentralized auction algorithm for multirobot task allocation. *IEEE Transactions on Automation Science and Engineering*, 12(4):1469–1481, 2015.

[105] Changjoo Nam and Dylan A Shell. Assignment algorithms for modeling resource contention in multirobot task allocation. *IEEE Transactions on Automation Science and Engineering*, 12(3):889–900, 2015.

[106] Lingzhi Luo, Nilanjan Chakraborty, and Katia Sycara. Distributed algorithms for multirobot task assignment with task deadline constraints. *IEEE Transactions on Automation Science and Engineering*, 12(3):876–888, 2015.

[107] Amanda Whitbrook, Qinggang Meng, and Paul WH Chung. Reliable, distributed scheduling and rescheduling for time-critical, multiagent systems. *IEEE Transactions on Automation Science and Engineering*, 15(2):732–747, 2018.

[108] Shudong Liu, Huayu Wu, Shili Xiang, and Xiaoli Li. Mobile robot scheduling with multiple trips and time windows. In *Proceedings of International Conference on Advanced Data Mining and Applications*, pages 608–620, 2017.

[109] Chihyun Jung and Tae-Eog Lee. An efficient mixed integer programming model based on timed petri nets for diverse complex cluster tool scheduling problems. *IEEE Transactions on Semiconductor Manufacturing*, 25(2):186–199, 2012.

[110] Mary Koes, Illah Nourbakhsh, and Katia Sycara. Constraint optimization coordination architecture for search and rescue robotics. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 3977–3982, 2006.

[111] Eduardo Feo Flushing, Luca M Gambardella, and Gianni A Di Caro. Simultaneous task allocation, data routing, and transmission scheduling in mobile multi-robot teams. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1861–1868, 2017.

[112] Chongjie Zhang and Julie A Shah. Co-optimizating multi-agent placement with task assignment and scheduling. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3308–3314, 2016.

[113] Xavier Lorca, Charles Prud'Homme, Aurélien Questel, and Benoît Rottembourg. Using constraint programming for the urban transit crew rescheduling problem. In *Proceedings of International Conference on Principles and Practice of Constraint Programming*, pages 636–649, 2016.

[114] Ridvan Gedik, Emre Kirac, Ashlea Bennet Milburn, and Chase Rainwater. A constraint programming approach for the team orienteering problem with time windows. *Computers & Industrial Engineering*, 107:178–195, 2017.

[115] Kyle EC Booth, Goldie Nejat, and J Christopher Beck. A constraint programming approach to multi-robot task allocation and scheduling in retirement homes. In *Proceedings of International Conference on Principles and Practice of Constraint Programming*, pages 539–555, 2016.

[116] Juan M Novas and Gabriela P Henning. Integrated scheduling of resource-constrained flexible manufacturing systems using constraint programming. *Expert Systems with Applications*, 41(5):2286–2299, 2014.

[117] Moacir Godinho Filho, Clarissa Fullin Barco, and Roberto Fernandes Tavares Neto. Using genetic algorithms to solve scheduling problems on flexible manufacturing systems (FMS): a literature survey, classification and analysis. *Flexible Services and Manufacturing Journal*, 26(3):408–431, 2014.

[118] J Jerald, Prabaharan Asokan, G Prabaharan, and R Saravanan. Scheduling optimisation of flexible manufacturing systems using particle swarm optimisation algorithm. *The International Journal of Advanced Manufacturing Technology*, 25(9-10):964–971, 2005.

[119] S Meysam Mousavi and Reza Tavakkoli-Moghaddam. A hybrid simulated annealing algorithm for location and routing scheduling problems with cross-docking in the supply chain. *Journal of Manufacturing Systems*, 32(2):335–347, 2013.

[120] Torbjørn S Dahl, Maja Matarić, and Gaurav S Sukhatme. Multi-robot task allocation through vacancy chain scheduling. *Robotics and Autonomous Systems*, 57(6-7):674–687, 2009.

[121] Shiyuan Jin, Guy Schiavone, and Damla Turgut. A performance study of multiprocessor task scheduling algorithms. *The Journal of Supercomputing*, 43(1):77–97, 2008.

[122] Haluk Topcuoglu, Salim Hariri, and Min-you Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE transactions on parallel and distributed systems*, 13(3):260–274, 2002.

[123] Ernesto Nunes, Mitchell McIntire, and Maria Gini. Decentralized multi-robot allocation of tasks with temporal and precedence constraints. *Advanced Robotics*, 31(22):1193–1207, 2017.

[124] Ernesto Nunes, Marie Manner, Hakim Mitiche, and Maria Gini. A taxonomy for task allocation problems with temporal and ordering constraints. *Robotics and Autonomous Systems*, 90:55–70, 2017.

[125] Kenli Li, Xiaoyong Tang, Bharadwaj Veeravalli, and Keqin Li. Scheduling precedence constrained stochastic tasks on heterogeneous cluster systems. *IEEE Transactions on computers*, 64(1):191–204, 2015.

[126] Binghai Zhou and Jiahui Xu. An adaptive SVM-based real-time scheduling mechanism and simulation for multiple-load carriers in automobile assembly lines. *International Journal of Modeling, Simulation, and Scientific Computing*, 8(04):1750048, 2017.

[127] Jeffrey R Peters and Luca F Bertuccelli. Robust scheduling strategies for collaborative human-UAV missions. In *Proceedings of American Control Conference (ACC), 2016*, pages 5255–5262, 2016.

[128] Matthew C Gombolay, Ronald Wilcox, and Julie A Shah. Fast scheduling of multi-robot teams with temporospatial constraints. In *Proceedings of Robotics: Science and Systems*, 2013.

[129] Lantao Liu and Dylan A Shell. Optimal market-based multi-robot task allocation via strategic pricing. In *Proceedings of Robotics: Science and Systems*, volume 9, pages 33–40, 2013.

[130] Drew Wicke, David Freelan, and Sean Luke. Bounty hunters and multiagent task allocation. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pages 387–394, 2015.

[131] Michael Otte, Michael Kuhlman, and Donald Sofge. Multi-robot task allocation with auctions in harsh communication environments. In *Proceedings of IEEE International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, pages 32–39, 2017.

[132] Han-Lim Choi, Luc Brunet, and Jonathan P How. Consensus-based decentralized auctions for robust task allocation. *IEEE transactions on robotics*, 25(4):912–926, 2009.

[133] Wanqing Zhao, Qinggang Meng, and Paul WH Chung. A heuristic distributed task allocation method for multivehicle multitask problems and its application to search and rescue scenario. *IEEE transactions on cybernetics*, 46(4):902–915, 2016.

[134] Gautham P Das, Thomas M McGinnity, Sonya A Coleman, and Laxmidhar Behera. A distributed task allocation algorithm for a multi-robot system in healthcare facilities. *Journal of Intelligent & Robotic Systems*, 80(1):33–58, 2015.

[135] Carla Mouradian, Jagruti Sahoo, Roch H Glitho, Monique J Morrow, and Paul A Polakos. A coalition formation algorithm for multi-robot task allocation in large-scale natural disasters. In *Proceedings of IEEE International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 1909–1914, 2017.

[136] Anton Dukeman and Julie A Adams. Hybrid mission planning with coalition formation. *Autonomous Agents and Multi-Agent Systems*, 31(6):1424–1466, 2017.

[137] Bing Xie, Shaofei Chen, Jing Chen, and LinCheng Shen. A mutual-selecting market-based mechanism for dynamic coalition formation. *International Journal of Advanced Robotic Systems*, 15(1):1729881418755840, 2018.

[138] Onn Shehory and Sarit Kraus. Methods for task allocation via agent coalition formation. *Artificial intelligence*, 101(1-2):165–200, 1998.

[139] José Guerrero and Gabriel Oliver. Multi-robot coalition formation in real-time scenarios. *Robotics and Autonomous Systems*, 60(10):1295–1307, 2012.

[140] Tyler Gunn and John Anderson. Dynamic heterogeneous team formation for robotic urban search and rescue. *Journal of Computer and System Sciences*, 81(3):553–567, 2015.

[141] Zhen Yang Lim, SG Ponnambalam, and Kazuhiro Izui. Multi-objective hybrid algorithms for layout optimization in multi-robot cellular manufacturing systems. *Knowledge-Based Systems*, 120:87–98, 2017.

[142] Chao Lu, Liang Gao, Xinyu Li, Quanke Pan, and Qi Wang. Energy-efficient permutation flow shop scheduling problem using a hybrid multi-objective backtracking search algorithm. *Journal of cleaner production*, 144:228–238, 2017.

[143] Elkin Castro and Sanja Petrovic. Combined mathematical programming and heuristics for a radiotherapy pre-treatment scheduling problem. *Journal of Scheduling*, 15(3):333–346, 2012.

[144] Wei Tan and Behrokh Khoshnevis. Integration of process planning and scheduling - a review. *Journal of Intelligent Manufacturing*, 11(1):51–63, 2000.

[145] Vipul Jain and Ignacio E Grossmann. Algorithms for hybrid MILP/CP models for a class of optimization problems. *INFORMS Journal on computing*, 13(4):258–276, 2001.

[146] Haitao Li and Keith Womer. Scheduling projects with multi-skilled personnel by a hybrid MILP/CP benders decomposition algorithm. *Journal of Scheduling*, 12(3):281, 2009.

[147] Qiao Zhang, Hervé Manier, and M-A Manier. A genetic algorithm with tabu search procedure for flexible job shop scheduling with transportation constraints and bounded processing times. *Computers & Operations Research*, 39(7):1713–1723, 2012.

[148] Stefano Di Alesio, Lionel Briand, Shiva Nejati, and Arnaud Gotlieb. Combining genetic algorithms and constraint programming to support stress testing of task deadlines. *ACM Transactions on Software Engineering & Methodology*, 25(1), 2015.

[149] Xinyan Ou, Qing Chang, Nilanjan Chakraborty, and Junfeng Wang. Gantry scheduling for multi-gantry production system by online task allocation method. *IEEE Robotics and Automation Letters*, 2(4):1848–1855, 2017.

[150] Amanda Whitbrook, Qinggang Meng, and Paul WH Chung. A novel distributed scheduling algorithm for time-critical multi-agent systems. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6451–6458, 2015.

[151] Pedro M Shiroma and Mario FM Campos. CoMutaR: A framework for multi-robot coordination and task allocation. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4817–4824. IEEE, 2009.

[152] Matthew Turpin, Nathan Michael, and Vijay Kumar. CAPT: Concurrent assignment and planning of trajectories for multiple robots. *The International Journal of Robotics Research*, 33(1):98–112, 2014.

[153] Ernesto Nunes and Maria L Gini. Multi-robot auctions for allocation of tasks with temporal constraints. In *Proceedings of AAAI Conference on Artificial Intelligence*, pages 2110–2116, 2015.

[154] Abderraouf Maoudj, Brahim Bouzouia, Abdelfetah Hentout, and Redouane Toumi. Multi-agent approach for task allocation and scheduling in cooperative heterogeneous multi-robot team: Simulation results. In *Proceedings of IEEE International Conference on Industrial Informatics (INDIN)*, pages 179–184, 2015.

[155] Anshul Kanakia, Behrouz Touri, and Nikolaus Correll. Modeling multi-robot task allocation with limited information as global game. *Swarm Intelligence*, 10(2):147–160, 2016.

[156] Smriti Chopra and Magnus Egerstedt. Heterogeneous multi-robot routing. In *Proceedings of IEEE American Control Conference (ACC)*, pages 5390–5395, 2014.

[157] Yueyue Deng, Pierre-Philippe J Beaujean, Edgar An, and Edward Carlson. Task allocation and path planning for collaborative autonomous underwater vehicles operating through an underwater acoustic network. *Journal of Robotics*, 2013, 2013.

[158] Alejandro R Mosteo, Luis Montano, and Michail G Lagoudakis. Multi-robot routing under limited communication range. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 1531–1536, 2008.

[159] Bilal Y Kaddouh, William J Crowther, and Peter Hollingsworth. Dynamic resource allocation for efficient sharing of services from heterogeneous autonomous vehicles. *Journal of Aerospace Information Systems*, pages 450–474, 2016.

[160] Jacopo Banfi, Jérôme Guzzi, Francesco Amigoni, Eduardo Feo Flushing, Alessandro Giusti, Luca Gambardella, and Gianni A Di Caro. An integer linear programming model for fair multitarget tracking in cooperative multirobot systems. *Autonomous Robots*, pages 1–16, 2018.

[161] Chen K Tham and Richard W Prager. A modular q-learning architecture for manipulator task decomposition. In *Proceedings of International Conference on Machine Learning*, pages 309–317. Elsevier, 1994.

[162] Naiqi Wu, Ning Mao, and Yanming Qian. An approach to partner selection in agile manufacturing. *Journal of Intelligent Manufacturing*, 10(6):519–529, 1999.

[163] Nabil NZ Gindy and Tsvetan M Ratchev. Cellular decomposition of manufacturing facilities using resource elements. *Integrated Manufacturing Systems*, 8(4):215–222, 1997.

[164] Shimon Whiteson, Nate Kohl, Risto Miikkulainen, and Peter Stone. Evolving soccer keepaway players through task decomposition. *Machine Learning*, 59(1-2):5–30, 2005.

[165] Peter Stone and Manuela Veloso. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*, 110(2):241–273, 1999.

[166] Chau Su and Yuan F Zheng. Task decomposition for a multilimbed robot to work in reachable but unorientable space. *IEEE transactions on robotics and automation*, 7(6):759–770, 1991.

[167] Jeffrey Too Chuan Tan, Feng Duan, Ye Zhang, and Tamio Arai. Task decomposition of cell production assembly operation for man-machine collaboration by hta. In *Proceedings of IEEE International Conference on Automation and Logistics (ICAL)*, pages 1066–1071, 2008.

[168] Jacob Huckaby and Henrik I Christensen. Toward a knowledge transfer framework for process abstraction in manufacturing robotics. In *ICML Workshop on Theoretically Grounded Transfer Learning*, 2013.

[169] Susan Hert and Vladimir Lumelsky. Polygon area decomposition for multiple-robot workspace division. *International Journal of Computational Geometry & Applications*, 8(04):437–466, 1998.

[170] Hannah Bast and Susan Hert. The area partitioning problem. In *Proceedings of Canadian Conference on Computational Geometry*, 2000.

[171] Jyh-Ming Lien and Nancy M Amato. Approximate convex decomposition of polygons. *Computational Geometry*, 35(1-2):100–123, 2006.

[172] Marco Pavone, Alessandro Arsie, Emilio Frazzoli, and Francesco Bullo. Distributed algorithms for environment partitioning in mobile robotic networks. *IEEE Transactions on Automatic Control*, 56(8):1834–1848, 2011.

[173] Praneel Chand and Dale A Carnegie. Mapping and exploration in a hierarchical heterogeneous multi-robot system using limited capability robots. *Robotics and autonomous Systems*, 61(6):565–579, 2013.

[174] Vittorio A Ziparo, Alexander Kleiner, Bernhard Nebel, and Daniele Nardi. Rfid-based exploration for large robot teams. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 4606–4613, 2007.

[175] Arnoud Visser and Bayu A Slamet. Including communication success in the estimation of information gain for multi-robot exploration. In *Proceedings of IEEE International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless (WiOPT)*, pages 680–687, 2008.

[176] Laëtitia Matignon, Laurent Jeanpierre, and Abdel-Illah Mouaddib. Distributed value functions for multi-robot exploration: a position paper. In *Multi-Agent Sequential Decision Making in Uncertain Multi-Agent Domain (MSDM)(AAMAS workshop)*, 2012.

[177] Jing Yuan, Yalou Huang, Tong Tao, and Fengchi Sun. A cooperative approach for multi-robot area exploration. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1390–1395, 2010.

[178] Kian Hsiang Low, Geoffrey J Gordon, John M Dolan, and Pradeep Khosla. Adaptive sampling for multi-robot wide-area exploration. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 755–760, 2007.

[179] Kian Hsiang Low, John M Dolan, and Pradeep Khosla. Adaptive multi-robot wide-area exploration and mapping. In *Proceedings of International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 23–30, 2008.

[180] Mazda Ahmadi and Peter Stone. A multi-robot system for continuous area sweeping tasks. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1724–1729, 2006.

[181] Jie Zhao, Xiangguo Su, and Jihong Yan. A novel strategy for distributed multi-robot coordination in area exploration. In *Proceedings of International Conference on Measuring Technology and Mechatronics Automation*, volume 2, pages 24–27, 2009.

[182] Nicola Basilico and Francesco Amigoni. Exploration strategies based on multi-criteria decision making for searching environments in rescue operations. *Autonomous Robots*, 31(4):401, 2011.

[183] Agusti Solanas and Miguel Angel Garcia. Coordinated multi-robot exploration through unsupervised clustering of unknown space. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 717–721, 2004.

[184] Kai M Wurm, Cyrill Stachniss, and Wolfram Burgard. Coordinated multi-robot exploration using a segmentation of the environment. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1160–1165, 2008.

[185] Yuanteng Pei, Matt W Mutka, and Ning Xi. Coordinated multi-robot real-time exploration with connectivity and bandwidth awareness. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 5460–5465, 2010.

[186] Reid Simmons, David Apfelbaum, Wolfram Burgard, Dieter Fox, Mark Moors, Sebastian Thrun, and Håkan Younes. Coordination for multi-robot exploration and mapping. In *Proceedings of Conference on Innovative Applications of Artificial Intelligence*, pages 852–858, 2000.

[187] Arnaud Doniec, Noury Bouraqadi, Michael Defoort, Serge Stinckwich, et al. Distributed constraint reasoning applied to multi-robot exploration. In *Proceedings of IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 159–166, 2009.

[188] Weihua Sheng, Qingyan Yang, Jindong Tan, and Ning Xi. Distributed multi-robot coordination in area exploration. *Robotics and Autonomous Systems*, 54(12):945–955, 2006.

[189] Jose Vazquez and Chris Malcolm. Distributed multirobot exploration maintaining a mobile network. In *Proceedings of IEEE Conference on Intelligent Systems*, volume 3, pages 113–118, 2004.

[190] Yehuda Elmaliach, Noa Agmon, and Gal A Kaminka. Multi-robot area patrol under frequency constraints. *Annals of Mathematics and Artificial Intelligence*, 57(3-4):293–320, 2009.

[191] Andrew Howard, Maja J Matarić, and Gaurav S Sukhatme. Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem. In *Proceedings of International Symposium on Distributed Autonomous Robotic Systems (DARS)*, pages 299–308. 2002.

[192] Maxim A Batalin and Gaurav S Sukhatme. Spreading out: A local approach to multi-robot coverage. In *Proceedings of International Symposium on Distributed Autonomous Robotic Systems (DARS)*, pages 373–382. 2002.

[193] Ali Marjovi, João Gonçalo Nunes, Lino Marques, and Aníbal de Almeida. Multi-robot exploration and fire searching. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1929–1934, 2009.

[194] KS Senthilkumar and KK Bharadwaj. Multi-robot exploration and terrain coverage in an unknown environment. *Robotics and Autonomous Systems*, 60(1):123–132, 2012.

[195] Robert Zlot, Anthony Stentz, M Bernardine Dias, and Scott Thayer. Multi-robot exploration controlled by a market economy. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 3, pages 3016–3023, 2002.

[196] Julian De Hoog, Stephen Cameron, and Arnoud Visser. Role-based autonomous multi-robot exploration. In *Proceedings of IEEE Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns*, pages 482–487, 2009.

[197] Julian De Hoog, Stephen Cameron, Arnoud Visser, et al. Selection of rendezvous points for multi-robot exploration in dynamic environments. In *Proceedings of International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2010.

[198] Ruofei Ouyang, Kian Hsiang Low, Jie Chen, and Patrick Jaillet. Multi-robot active sensing of non-stationary gaussian process-based environmental phenomena. In *Proceedings of International Conference on Autonomous Agents and Multi-agent Systems*, pages 573–580, 2014.

[199] Mahdi Hassan, Dikai Liu, Shoudong Huang, and Gamini Dissanayake. Task oriented area partitioning and allocation for optimal operation of multiple industrial robots in unstructured environments. In *Proceedings of International Conference on Control Automation Robotics & Vision (ICARCV)*, pages 1184–1189, 2014.

[200] Markus Jager and Bernhard Nebel. Dynamic decentralized area partitioning for cooperating cleaning robots. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, volume 4, pages 3577–3582, 2002.

[201] John Gunnar Carlsson. Dividing a territory among several vehicles. *INFORMS Journal on Computing*, 24(4):565–577, 2012.

[202] Juan Camilo Gamboa Higuera and Gregory Dudek. Fair subdivision of multi-robot tasks. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 3014–3019, 2013.

[203] A. Agarwal, Meng-Hiot Lim, and Meng Joo Er. Proportional partition of holed rectilinear region amongst multiple uravs. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 1779–1784, 2005.

[204] Brian Yamauchi. A frontier-based approach for autonomous exploration. In *Proceedings of IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA)*, pages 146–151, 1997.

[205] Weihua Sheng, Qingyan Yang, Song Ci, and Ning Xi. Multi-robot area exploration with limited-range communications. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 2, pages 1414–1419, 2004.

[206] Wolfram Burgard, Mark Moors, and Frank Schneider. Collaborative exploration of unknown environments with teams of mobile robots. In *Advances in Plan-Based Control of Robotic Agents*, pages 52–70. Springer, 2002.

[207] Xin Ma, Qin Zhang, and Yibin Li. Genetic algorithm-based multi-robot cooperative exploration. In *Proceedings of IEEE International Conference on Control and Automation (ICCA)*, pages 1018–1023, 2007.

[208] Yiheng Wang, Alei Liang, and Haibing Guan. Frontier-based multi-robot map exploration using particle swarm optimization. In *Proceedings of IEEE Symposium on Swarm Intelligence (SIS)*, pages 1–6, 2011.

[209] Yi Zhou, Kai Xiao, Yiheng Wang, Alei Liang, and Aboul Ella Hassanien. A pso-inspired multi-robot map exploration algorithm using frontier-based strategy. *International Journal of System Dynamics Applications (IJSDA)*, 2(2):1–13, 2013.

[210] Yeun-Soo Jung, Kong-Woo Lee, Seong-Yong Lee, Myoung Hwan Choi, and Beom-Hee Lee. An efficient underwater coverage method for multi-auv with sea current disturbances. *International Journal of Control, Automation and Systems*, 7(4):615–629, 2009.

[211] Marta Kwiatkowska, Gethin Norman, and David Parker. Stochastic model checking. In *Proceedings of International School on Formal Methods for the Design of Computer, Communication and Software Systems*, pages 220–270. Springer, 2007.

[212] Shaurya Shriyam and Satyandra K Gupta. Incorporating potential contingency tasks in multi-robot mission planning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA, Brisbane, Australia, May 21-25)*, 2018.

[213] Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. NuSMV 2: An opensource tool for symbolic model checking. In *Proceedings of International Conference on Computer Aided Verification*, pages 359–364. Springer, 2002.

[214] Shaurya Shriyam and Satyandra K Gupta. Modeling and analysis of subsystem interactions in robotic assembly. In *Proceedings of ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, 2019.

[215] Abdullah Alsharhan, Ariyan Kabir, Rishi Malhan, Brual Shah, and Satyandra K Gupta. A flexible hybrid cell for low production volume assembly. `https://www.youtube.com/watch?v=yqT2XHwkpOo`, 2017.

[216] Shantanu Thakar, Pradeep Rajendran, Vivek Annem, Ariyan Kabir, and Satyandra K. Gupta. Accounting for part pose estimation uncertainties during trajectory generation for part pick-up using mobile manipulators. In *IEEE International Conference on Robotics and Automation (ICRA)*, Montreal, Canada, May 2019.

[217] Shantanu Thakar, Liwei Fang, Brual Shah, and Satyandra K. Gupta. Towards time-optimal trajectory planning for pick-and-transport operation with a mobile manipulator. In *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, pages 981–987. IEEE, 2018.

[218] Vivek Annem, Pradeep Rajendran, Shantanu Thakar, and Satyandra K. Gupta. Towards remote teleoperation of a semi-autonomous mobile manipulator system in machine tending tasks. In *ASME Manufacturing Science and Engineering Conference (MSEC)*, Erie, USA, June 2019.

[219] Shantanu Thakar, Ariyan Kabir, Prahar Bhatt, Rishi Malhan, Pradeep Rajendran, Brual Shah, and Satyandra K. Gupta. Task assignment and motion planning for bi-manual mobile manipulation. In *IEEE International Conference on Automation Science and Engineering (CASE)*, Vancouver, Canada, August 2019.

[220] Marta Kwiatkowska, Gethin Norman, and David Parker. Prism 4.0: Verification of probabilistic real-time systems. In *Proceedings of International conference on computer aided verification*, pages 585–591. Springer, 2011.

[221] Shaurya Shriyam and Satyandra K Gupta. Task assignment and scheduling for mobile robot teams. In *Proceedings of the ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC-CIE, Quebec City, Quebec, August 26-29)*, 2018.

[222] G Ayorkor Korsah, Anthony Stentz, and M Bernardine Dias. A comprehensive taxonomy for multi-robot task allocation. *The International Journal of Robotics Research*, 32(12):1495–1512, 2013.

[223] Brian P Gerkey and Maja J Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, 23(9):939–954, 2004.

[224] International Maritime Organization. *COLREG: Convention on the International Regulations for Preventing Collisions at Sea, 1972*. International Maritime Organization, 2002.

[225] Saralees Nadarajah and Samuel Kotz. Exact distribution of the max/min of two Gaussian random variables. *IEEE Transactions on very large scale integration (VLSI) systems*, 16(2):210–212, 2008.

[226] Charles E Clark. The greatest of a finite set of random variables. *Operations Research*, 9(2):145–162, 1961.

[227] Shaurya Shriyam, Brual C Shah, and Satyandra K Gupta. On-line task decomposition for collaborative surveillance of marine environment by a team of unmanned surface vehicles. In *Proceedings of the ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC-CIE, Cleveland, Ohio, August 6-9)*, 2017.

[228] Shaurya Shriyam, Brual C Shah, and Satyandra K Gupta. Decomposition of collaborative surveillance tasks for execution in marine environments by a team of unmanned surface vehicles. *Journal of Mechanisms and Robotics*, 10(2):025007, 2018.

[229] Brual C Shah, Petr Švec, Ivan R Bertaska, Armando J Sinisterra, Wilhelm Klinger, Karl von Ellenrieder, Manhar Dhanak, and Satyandra K Gupta. Resolution-adaptive risk-aware trajectory planning for surface vehicles operating in congested civilian traffic. *Autonomous Robots*, 40(7):1139–1163, 2016.

[230] Petr Švec, Atul Thakur, Eric Raboin, Brual C Shah, and Satyandra K Gupta. Target following with motion prediction for unmanned surface vehicle operating in cluttered environments. *Autonomous Robots*, 36(4):383–405, 2014.

[231] Brual C Shah and Satyandra K Gupta. Speeding up A* search on visibility graphs defined over quadtrees to enable long distance path planning for unmanned surface vehicles. In *Proceedings of International Conference on Automated Planning and Scheduling*, pages 527–535, 2016.

[232] Eric Raboin, Petr Švec, Dana S Nau, and Satyandra K Gupta. Model-predictive asset guarding by team of autonomous surface vehicles in environment with civilian boats. *Autonomous Robots*, 38(3):261–282, 2015.

[233] Stuart Russell, Peter Norvig, and Artificial Intelligence. A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, 25:27, 1995.

[234] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[235] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, 1995.

[236] MATLAB. *Version 9.5 (R2018b)*. The MathWorks Inc., Natick, Massachusetts, 2018.