Robust, Goal-directed Plan Execution with Bounded Risk

by

Masahiro Ono

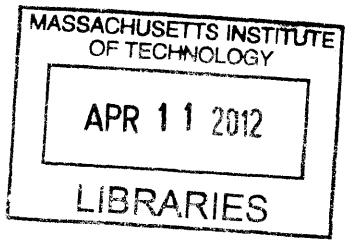
Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2012



ARCHIVES

Author .....
Department of Aeronautics and Astronautics
February 2, 2012

Certified by .....
Prof. Brian C. Williams
Thesis Supervisor

Certified by .....
Prof. Emilio Frazzoli
Thesis Committee Member

Certified by .....
Dr. Lars Blackmore
Thesis Committee Member

Accepted by .....
Prof. Eytan H. Modiano
Chair, Graduate Program Committee

Robust, Goal-directed Plan Execution with Bounded Risk

by

Masahiro Ono

Submitted to the Department of Aeronautics and Astronautics
on February 2, 2012, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

There is an increasing need for robust optimal plan execution for multi-agent systems in uncertain environments, while guaranteeing an acceptable probability of success. For example, a fleet of unmanned aerial vehicles (UAVs) and autonomous underwater vehicles (AUVs) are required to operate autonomously for an extensive mission duration in an uncertain environment. Previous work introduced the concept of a model-based executive, which increases the level of autonomy, elevating the level at which systems are commanded. This thesis develops model-based executives that reason explicitly from a stochastic plant model to find the optimal course of action, while ensuring that the probability of failure is within a user-specified risk bound.

This thesis presents two robust mode-based executives: *probabilistic Sulu* or *p-Sulu*, and *distributed probabilistic Sulu* or *dp-Sulu*. The objective for p-Sulu and dp-Sulu is to allow users to command continuous, stochastic multi-agent systems in a manner that is both intuitive and safe. The user specifies the desired evolution of the plant state, as well as the acceptable probabilities of failure, as a temporal plan on states called a *chance-constrained qualitative state plan* (*CCQSP*). An example of a CCQSP statement is "go to A through B within 30 minutes, with less than 0.001% probability of failure." p-Sulu and dp-Sulu take a CCQSP, a continuous plant model with stochastic uncertainty, and an objective function as inputs, and outputs an optimal continuous control sequence, as well as an optimal discrete schedule. The difference between p-Sulu and dp-Sulu is that p-Sulu plans in a centralized manner while dp-Sulu plans in a distributed manner. dp-Sulu enables robust CCQSP execution for multi-agent systems.

We solve the problem based on the key concept of risk allocation, which achieves tractability by allocating the specified risk to individual constraints and mapping the result into an equivalent deterministic constrained optimization problem. Risk allocation also enables a distributed plan execution for multi-agent systems by distributing the risk among agents to decompose the optimization problem. Building upon the risk allocation approach, we develop our first CCQSP executive, p-Sulu, in four spirals. First, we develop the Convex Risk Allocation (CRA) algorithm, which can solve a CCQSP planning problem with a convex state space and a fixed schedule, highlighting the capability of optimally allocating risk to individual constraints. Second, we develop the Non-convex Iterative Risk Alloca-

tion (NIRA) algorithm, which can handle non-convex state space. Third, we build upon NIRA a full-horizon CCQSP planner, p-Sulu FH, which can optimize not only the control sequence but also the schedule. Fourth, we develop p-Sulu, which enables the real-time execution of CCQSPs by employing the receding horizon approach.

Our second CCQSP executive, dp-Sulu, is developed in two spirals. First, we develop the Market-based Iterative Risk Allocation (MIRA) algorithm, which can control a multi-agent system in a distributed manner by optimally distributing risk among agents through the market-based method called tâtonnement. Second and finally, we integrate the capability of MIRA into p-Sulu to build the robust model-based executive, dp-Sulu, which can execute CCQSPs on multi-agent systems in a distributed manner.

Our simulation results demonstrate that our executives can efficiently execute CCQSP planning problems with significantly reduced suboptimality compared to prior art.

Thesis Supervisor: Prof. Brian C. Williams

父と母に捧ぐ

This thesis is for my parents, Akira and Chisetsu.

Acknowledgments

First, I would like to express my most profound gratitude for my advisor, Prof. Brian C. Williams. I learned from him not only about how to conduct interesting research, but also how to tell a compelling story in writing and presentations. I am also very thankful to him for granting me a large degree of freedom in my research. He encouraged me to be creative, rather than commanding me to do specific tasks. He treated me as a mature researcher, rather than merely a research assistant. With his guidance, support and trust, I had a wonderful research experience at MIT.

I would also like to thank my committee members and readers, Prof. Emilio Frazzoli, Dr. Lars Blackmore, Prof. Nicholas Roy, and Dr. Howard E. Shrobe, for giving me helpful feedback on my thesis and defense. I particularly thank Lars for having been a great mentor and collaborator throughout my PhD research. His advice helped me greatly in improving this thesis.

I would like to thank all of my colleagues and collaborators. Alborz Geramifard, David Wang, Hui Li, Julie Shah, and Peng Yu in the MERS group worked hard with me late into the night in order to produce successful demonstrations for the Boeing project. Ronald Provine and Scott Smith at the Boeing Company gave us wonderful feedback and support on the project. I also thank all of my friends. I particularly thank Yoshiaki Kuwata for encouraging me to come to MIT, as well as for being a great mentor after I arrived.

I thank the Voyger 2 spacecraft for inspiring me to pursue space engineering. Her rendezvous with Neptune in 1989 was an unforgettable event in my childhood. I wish her good luck in her never-ending journey out of the Solar System, as well as in her mission to deliver the Golden Record to an extraterrestrial civilization. May the Force be with her.

I would not have been able to complete my PhD degree without all the love, support, and respect I have received from my family members. My father has been the greatest teacher in my life. He bought me a telescope, with which we spent hours every weekend to discover the wonders of the universe, such as the craters on the Moon, the rings of Saturn,

and the tail of a comet. He gave me a box of electrical components, such as transistors, resistors, and capacitors, and taught me how to put them together to create a radio. He also gave me my first computer, which ran on the i486 CPU and MS-DOS, as well as a book on the C Language. All of these childhood experiences become the cornerstones of my achievements at MIT. My mother always cared about me, more than anyone else. She never forgets to send me a birthday present, even after I left home and came to the U.S. She often sends me packages filled with Japanese snacks, foods, and teas, and handwritten letters. They helped me a lot to to go through tough times at MIT. My sister, Asumi, has been a great friend of mine for nearly 25 years. My grandfathers and grandmothers are the greatest cheerleaders in everything I do. My dog, Chiko, and my cat, Mimi, were my mascots who comforted me a lot.

Finally and most importantly, I thank my wonderful wife, Eriko, who was a classmate at MIT and became my lifelong partner. She have always greatly encouraged and supported me, even though we have lived separately in Japan and the U.S. since last June, She watched my thesis defense over the internet at 2 a.m. in Japan time, and wept with joy when I passed it. Her love and trust are the essential ingredients of my success.

This research was funded by Boeing Company under contract MIT-BA-GTA-1 and the National Science Foundation under Grant No. IIS-1017992. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the sponsoring agencies.

Contents

Chapter 1

Introduction

1.1 Motivation

The vision of our research is to develop technologies that overcome the risks of technologies. Why do airplanes have a risk of crashing? Why do power grids have a risk of blackouts? We specify human error and uncertainty as two major factors in such risks. For example, in 2004, pilot error was listed as the primary cause of 75.5% of fatal general aviation accidents in the U.S., according to the 2005 Joseph T. Nall Report [3]. The 1987 Tokyo blackout, which affected 2.8 million households, was due to an unexpected surge of electricity demand [76]. An essential cause of those risks is that, while our society is growing ever more complex and becoming increasingly sensitive to uncertainty, optimal decision making is becoming intractable for humans. Hence, an effective approach to overcome those risks is to support human decision-making, which is largely based on intuitions and experiences, by autonomous systems that optimize the behaviors by explicitly considering statistical models of uncertainty.

A motivating example for this thesis is the Boeing concept of a future aerial personal transportation system (PTS), as shown in Figure 1-1. PTS is a concept of a flying taxi service. It consists of a fleet of small personal aerial vehicles (PAV) that enable the flexible point-to-point transportation of individuals and families. Automated path planning, scheduling, collision avoidance, and traffic management will significantly improve the safety of PTS, as well as its efficiency. The challenges to operating such a system include

adapting to uncertainties in the environment, such as storms and turbulence, and cooperatively controlling a large number of vehicles, while satisfying various needs of users, such as the arrival time, the preferred route, and the acceptable level of risk.

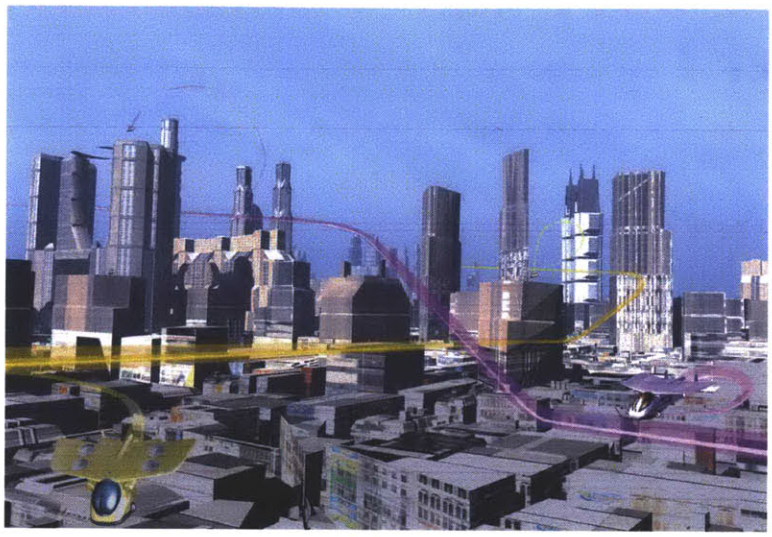


Figure 1-1: Personal Transportation System (PTS). (Courtesy of the Boeing Company)

Figure 1-2 shows a sample PTS scenario. A passenger of PAV 1 starts in Provincetown, MA and wants to go to Bedford within 60 minutes. The passenger also wants to go through a scenic area and remain there between 5 and 10 minutes during the flight. There is a no-fly zone (NFZ) and a storm that must be avoided. However, the storm's future location is uncertain; the vehicle's location is uncertain as well, due to control error and exogenous disturbances. Thus there is a risk of penetrating the NFZ or the storm. The passengers want to limit such risk to at most 0.01%.

Meanwhile, another passenger on PAV 2 starts in Hyannis, and wants to go straight to Bedford. Since the passenger would like to arrive at the destination as soon as possible, she accepts a relatively high 0.1% of risk to allow the vehicle to go through the narrow corridor between the NFZ and the storm. Since the paths of the two vehicles cross, they are required to limit the probability of collision to 0.001%. Additionally, the control tower of Bedford Airport requires at least a five-minute interval between the two landings.

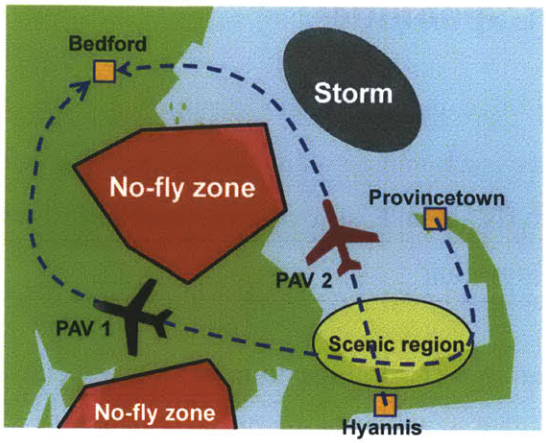


Figure 1-2: A sample plan for personal aerial vehicle (PAV)

1.2 Objective

The objective of this thesis is to develop robust plan executives that can operate real-world systems, such as PTS, and achieve the temporally extended goals specified by a given plan. The executives explicitly consider statistical model of uncertainty. This includes the centralized goal directed control of individual agents, and the distributed control of teams of agents. The resulting solutions must guarantee that the probability of failure is within the specified risk bound.

There is a substantial body of work on plan execution under uncertainty that is relevant. However, our approach is distinctive in three key respects. First, our executives allows users to explicitly limit the probability of constraint violation. This capability is particularly important for risk-sensitive missions where the impact of failure is significant. Second, the executive is goal-directed, by which we mean that it achieves temporally extended goals within user-specified temporal constraints. Third, the executive works in a continuous state space. A continuous state space representation fits naturally to many real-world applications, such as planning for aerial and underground vehicles.

1.3 Required Capabilities

In order to meet the objective, the robust plan executives provides the following five capabilities: 1) goal-directed execution in a continuous domain, 2) stochastic plan optimization, 3) robust execution with risk bounds, 4) real-time execution with receding planning horizon, and 5) distributed execution.

Goal-directed execution in a continuous domain The executives must execute actions with continuous effects that achieve temporally extended goals specified by users. For example, in the case of the PTS scenario in Figure 1-2, PAV 1 must sequentially achieve two temporally extended goals, called episodes: going through the scenic area and then arriving at Bedford. There are additional temporal constraints on the goals that are inherent to the scenario; some temporal constraints come from physical limitations, such as fuel capacity, and others come from passenger requirements.

Stochastic plan optimization Cost reduction and performance improvement are important issues for any system. In the PTS scenario, passengers may want to minimize the trip time or fuel usage. The executives optimize the control sequence and schedule according to the user-defined objective function, while satisfying given constraints.

Robust planning with risk bounds Real-world systems are subject to various uncertainties, such as state estimation error, modeling uncertainty, and exogenous disturbance. In the case of PAVs, the position and velocity of the vehicle estimated by the Kalman filter typically involve Gaussian-distributed uncertainties; the system model used for planning and control is not perfect; and the vehicles are subject to unpredictable disturbances such as turbulence. Under such uncertainty, the executed result of a plan inevitably deviates from the original plan and hence involves risk of constraint violation. Deterministic plan execution is particularly susceptible to risk when it is optimized in order to minimize a given cost function, since the optimal plan typically pushes against one or more constraint boundaries, and hence leaves no margin for error. For example, the shortest path in the PTS scenario shown in Figure 1-2 cuts in close to the NFZs and the storm, or more generally,

constraint boundaries. Then, a tiny perturbation to the planned path may results in a penetration into the obstacles. Such risk can be reduced by setting a safety margin between the path and the obstacles, at a cost of longer path length. However, it is often impossible to guarantee zero risk, since there is typically a non-zero probability of having a disturbance that is large enough to push the vehicle out of the feasible region. Therefore, passengers of the vehicle must accept some risk, but at the same time they need to limit it to a certain level. More generally, users of an autonomous system under uncertainty should be able to specify their bounds on risk. The planner must guarantee that the system is able to operate within these bounds. Such constraints are called *chance constraints* [27].

Real-time execution with receding planning horizon The executives must generate control sequences in real time, instead of preplanning, in order to adapt to the changes in the environment. In order to be tractable for plans with long durations, the executives employ the receding horizon approach, meaning that they plan for a finite duration at each planning cycle.

In PTS, for example, the PAVs must continuously update the planned paths by considering the latest position of storms, as well as the latest predictions of future storm locations, since the storms move over time. Real-time execution is particularly important when controlling uncertain systems, since uncertainty accumulates over time. By updating information continuously, the executive can minimize the uncertainty involved in the plan. In order to intuitively explain this effect, look at the 5-day forecast of a cyclone in Figure 1-3, which shows the 70% probability circles of the center position. Figure 1-3-(a) is the forecast as of 11 pm (UTC), July 19, 2011, while Figure 1-3-(b) is the one as of 11 pm (UTC), July 20, 2011. Note that the size of the circles grows over time in each forecast, showing a typical tendency that a forecast for the distant future involves more uncertainty than a forecast for the near future. Next, compare the probability circles for the same point of time in the two predictions (find the circles with the same label; for example, "23/18 UTC"). The newer forecast, Figure 1-3-(b), always has a smaller probability circle. This is because the uncertainty of the forecast for a specific point of time decreases over time as new information becomes available. The plan executive can take advantage of this feature of uncertainty and

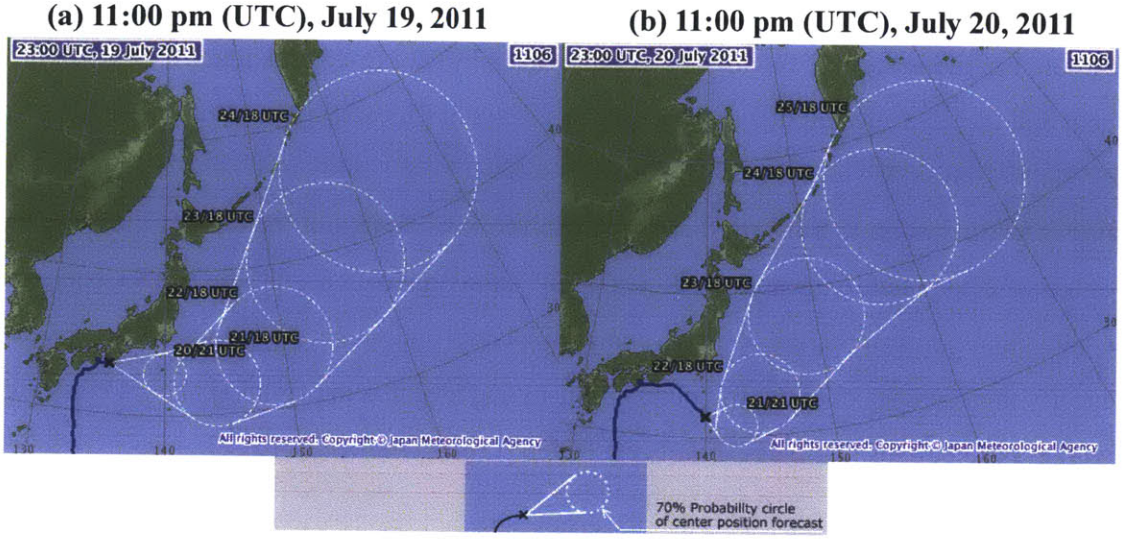


Figure 1-3: Five-day forecasts of a tropical cyclone. The blue solid lines shows the past trajectories of the storm's center, the crosses are the current positions as of the times of the forecasts, and the dotted circles are the 70% probability circles of the center position. The two numbers next to each probability circle mean day/time. Images are from the Japan Meteorological Agency.

potentially achieve higher optimality by continuously replanning the control sequence.

The receding-horizon approach has an advantage with respect to tractability over full-horizon planning, which can easily become intractable for a problem with a long duration. Receding-horizon planners generate control sequences for a fixed, short duration at each planning cycle. For example, in the PTS scenario, at $t = 0$, each PAV generates a control sequence for the duration between $t = 0$ and $= 10$. It replans the path at $t = 3$, generating another 10-minute control sequence from $t = 3$ till $t = 13$. This process is repeated until the plan is completed. The two challenges are to guide the plant state towards goals beyond the planning horizon, and to guarantee satisfaction of chance constraints.

Distributed Execution The PTS involves hundreds of PAVs flying at the same time, like taxis on the street. Centralized control approaches can hardly scale to such large multi-agent systems. Rather, the computational burden must be distributed among all agents. Each agent in a system is responsible for generating its own control sequence and schedule. The main technical challenge of distributed plan execution is the coordination of multiple

agents. The control sequences and schedules, which are generates in a distributed manner, must satisfy coupled state, chance, and temporal constraints. For example, in the PTS scenario, the schedules of the two vehicles must be coordinated so that there is at least a five-minute interval between their landing times. The paths of the two vehicles must be planned so that the probability of collision is below the specified risk bound.

1.4 Introduction of Robust Plan Executives

In order to provide the five capabilities specified in Section 1.3, we develop two robust plan executives in this thesis. One is a centralized executive called *probabilistic Sulu*[1] *(p-Sulu)*, and the other one is a distributed executive called *distributed probabilistic Sulu (dp-Sulu)*. The focus of p-Sulu is to control single-agent systems, while dp-Sulu is to scale to systems with a large number of agents.

This thesis consists of two parts. The first part is concerned with p-Sulu, while the second part deals with dp-Sulu. In this chapter, Sections 1.5 to 1.7 discuss p-Sulu, while Sections 1.8-1.10 are concerned with dp-Sulu. Technical components of p-Sulu are developed through Chapters 5-8. Chapters 9 and 10 describe additional technical innovations that enable distributed plan execution by dp-Sulu.

1.5 Overview of p-Sulu

We build p-Sulu upon prior work on the continuous-state model-based plan executive called Sulu [62]. p-Sulu takes as an input a plan representation called a *chance-constrained qualitative state plan (CCQSP)*[17], which encodes both temporally extended goals and chance constraints. A CCQSP is an extension of the deterministic qualitative state plans (QSP), introduced by [62]. p-Sulu employs a continuous-state probabilistic planner that outputs an optimal executable control sequence, an optimal state sequence, and an optimal schedule that minimize cost given the state and chance constraints of the CCQSP. A control sequence

[1] Sulu is a deterministic plan executive developed by [61]. The name of the executive was taken from Hikaru Sulu, a character in the science fiction drama *Star Trek*. In the story, Sulu serves as a helmsman of the starship USS Enterprise. The plan executive was named after him because its role is to "steer a ship" in order to achieve a given plan.

is an assignment to real-valued control variables, while a schedule is an assignment of discrete execution time to events. In the remainder of this section we describe the inputs and outputs informally. They are rigorously defined in Chapter 2.

1.5.1 Inputs

Initial Condition p-Sulu plans a control sequence starting from the current state, which is typically estimated from noisy sensor measurements. Therefore, p-Sulu takes the probability distribution, instead of the point estimate, of the current state as the initial condition.

Stochastic Plant Model p-Sulu takes as an input a discrete-time, continuous-state stochastic plant model, which specifies probabilistic state transitions in a continuous domain. This is a stochastic extension of the continuous plant model used in [62]. Although we limit our focus to Gaussian-distributed uncertainty, the algorithms presented in this paper can be extended to broader classes of distributions.

Chance-constrained qualitative state plan (CCQSP) A CCQSP is our formalism to express temporally extended goals and chance constraints. It is an extension of qualitative state plan (QSP), developed and used by [18, 43, 62]. CCQSP specifies a desired evolution of the plant state over time, and is defined by a set of discrete *events*, a set of *episodes*, which impose constraints on the plant state evolution, a set of *temporal constraints* between events, and a set of *chance constraints* that specify reliability constraints on the success of sets of episodes in the plan.

A CCQSP may be depicted as an acyclic directed graph, as shown in Figure 1-4. The circles represent events and squares represent episodes. Flexible temporal constraints are represented as a simple temporal network (STN) [32], which specifies upper and lower bounds on the duration of episodes (shown as the pairs of numbers in parentheses). The figure describes the plan for PAV 1 in the PTS scenario depicted in Figure 1-4, which can be stated informally as:

> "Start from Provincetown, reach the scenic region within 30 time units, and remain there for between 5 and 10 time units. Then end the flight in Bedford.

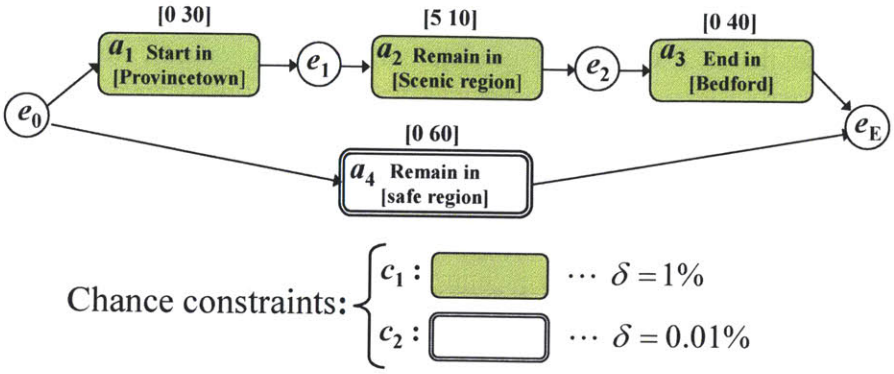


Figure 1-4: An example of a CCQSP for a personal transportation system (PTS) planning problem for PAV 1, illustrated in Figure 1-2. Passengers of the PAV would like to go from Provincetown to Bedford, and fly over the scenic region on the way. The "safe region" means the entire state space except the obstacles. Risk of the episodes must be within the risk bounds specified by chance constraints.

> The probability of failure of these episodes must be less than 1%. At all times, remain in the safe region by avoiding the no-fly zones and the storm. Limit the probability of penetrating such obstacles to 0.01%. The entire flight must take at most 60 time units."

A formal definition of CCQSP is given in Section 2.4.3.

1.5.2 Output

Optimal executable control sequence One of the two outputs of p-Sulu is an executable control sequence that minimizes a given cost function and satisfies all constraints specified by the input CCQSP. In the case of the PTS scenario, the outputs are the vehicle's actuation inputs, such as acceleration and ladder angle, that result in the nominal paths shown in Figure 1-2. In order for the control sequence to be executable, it must be dynamically feasible. For example, the curvature of the PAV's path must not exceed the vehicles' maneuverability.

Optimal schedule The other output of p-Sulu is the optimal schedule, a set of execution time steps for events in the input CCQSP. In the case of the PTS scenario shown in Figure

1-4, a schedule specifies when to leave the scenic region and when to arrive at Bedford, for example. Note that a CCQSP involves simple temporal constraints, which only impose upper and lower bounds on the durations of episodes, instead of specifying exact execution time steps of events. For example, a CCQSP requires arrival at the destination *within* 30 minutes, instead of specifying the arrival time as 10:11 p.m. Hence, a CCQSP allows flexibility in schedule of plan events. p-Sulu finds a schedule that satisfies all the simple temporal constraints specified by the CCQSP, and minimizes the cost function.

The two outputs – the optimal control sequence and the optimal schedule – must be consistent with each other: the temporally extended goals are achieved on the optimal schedule by applying the optimal control sequence to the given initial conditions.

1.6 Approach to p-Sulu

1.6.1 Risk Allocation

All the algorithms presented in this thesis are built upon a novel concept called *risk allocation*. Risk allocation reformulates a chance-constrained optimization problem into a resource allocation problem by distributing risk to individual constraints.

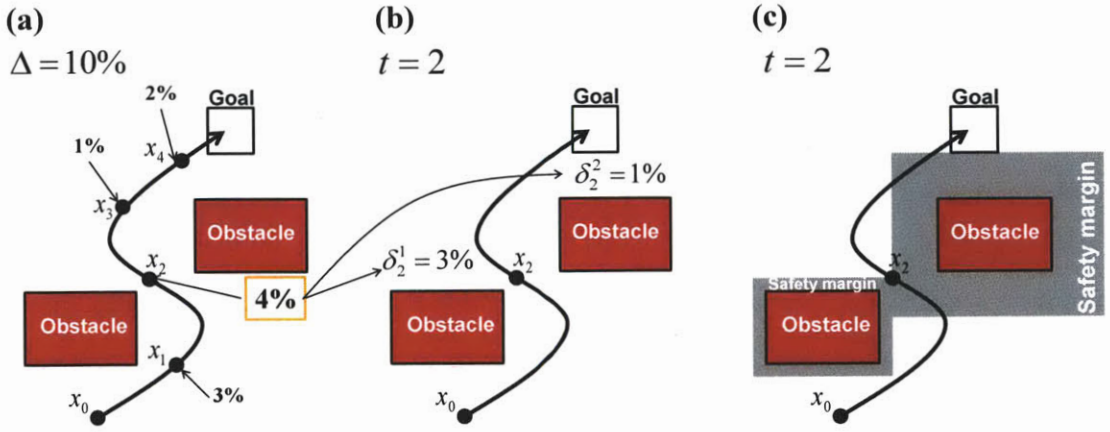


Figure 1-5: Overview of risk allocation approach. (a) The 10% of risk is allocated to time steps in the plan. (b) The 4% of risk allocated to $t = 2$ is distributed again among constraints (i.e., obstacles). (c) Safety margin is set around the constraints according to the risk allocation.

Figure 1-5 gives an overview of the risk allocation approach in a simple path planning problem. A chance constraint requires the vehicle to limit the probability of crashing into the obstacles to 10%. In other words, the vehicle is allowed to take 10% risk throughout the plan. The risk allocation approach distributes this risk among time steps in the plan, as shown in Figure 1-5-(a). If the probabilities of a crash at each time step are within these distributed risk bounds, then the original 10% risk bound is guaranteed to be satisfied. The 4% of risk that is allocated to the time step $t = 2$ is again distributed among constraints (i.e., obstacles) as in Figure 1-5-(b). Such a decomposition is obtained from Boole's inequality [79]. For example, let A be an event of colliding with the left obstacle at $t = 2$, and B be an event of colliding with the right obstacle at $t = 2$. Then,

$$\Pr(A) + \Pr(B) \geq \Pr(A \cup B).$$

Therefore, by allocating the risk A and B so that the left-hand side is equal to 4%, the right hand side is guaranteed to be less than or equal to 4%.

Once risk is allocated to every single constraint, a safety margin that guarantees the risk bound can be found as in Figure 1-5-(c). By planning the nominal path outside of the safety margin for all time steps, the executive can guarantee the satisfaction of the original chance constraint. Note that planning a *nominal* path outside of safety margins is a deterministic optimization problem, which can be solved efficiently using existing approaches. Such a decomposition of a chance constraint makes the stochastic optimization significantly easier, hence allowing the executive to compute the optimal control sequence efficiently.

The sensitivity of a constraint to risk varies from constraint to constraint and from time step to time step. Hence, allocation of risk must be optimized in order to obtain the best performance. In Section 4.2, we develop a novel algorithm called *iterative risk allocation (IRA)*, which can efficiently obtain near-optimal risk allocation. The subproblem solved in each iteration is a deterministic optimization problem. Hence, IRA can be implemented on top of existing deterministic optimizers, and turns them into chance-constrained stochastic optimizers.

1.6.2 Spiral Development of p-Sulu

Robust CCQSP execution is a very difficult problem to solve optimally since it is a stochastic problem that involves both combinatorial optimization of a discrete schedule and non-convex optimization of a continuous control sequence. We develop the robust plan executive, p-Sulu, in four spirals. In the first spiral, we develop the *convex risk allocation (CRA)* algorithm, which solves a special case of the full-horizon CCQSP planning problem where the feasible state space is convex and the schedule is fixed, as shown in Figure 1-6-(a). A path planning problem without obstacles typically has a convex state space. In the second spiral, we develop the *non-convex iterative risk allocation* (NIRA) algorithm, which extends the capability of CRA to solve problems with a *non-convex* state space. Hence, NIRA can solve a robust path planning problem with obstacles, as shown in Figure 1-6-(b). The limitation of NIRA is that it cannot handle a problem with a flexible schedule. The third spiral removes this limitation by developing p-Sulu FH, a general full-horizon CCQSP planner, which can solve problems with a *flexible* schedule and obstacles, as shown in Figure 1-6-(c). In the fourth and the final spiral, we develop p-Sulu, which executes CCQSP in real time with a receding planning horizon, as in Figure 1-6-(d).

1.6.3 CRA: full-horizon CCQSP planning with a convex state space, a fixed schedule

Starting with the first spiral, when the schedule is fixed and there are no obstacle in the environment, the problem is reduced to a joint chance-constrained finite-horizon optimal control problem. The first difficulty in solving this problem is that evaluating joint chance constraints requires the computation of an integral of a multi-variable probability distribution function over an arbitrary feasible state region. This computation cannot be carried out in closed form, and approximate techniques such as sampling are time-consuming and introduce approximation error [16].

Our approach to solve the problem is two-fold. In the first step, we use risk allocation to reformulate a chance constraint over a conjunction of state constraints (joint chance constraint) into a conjunction of individual chance constraints, each of which only involves

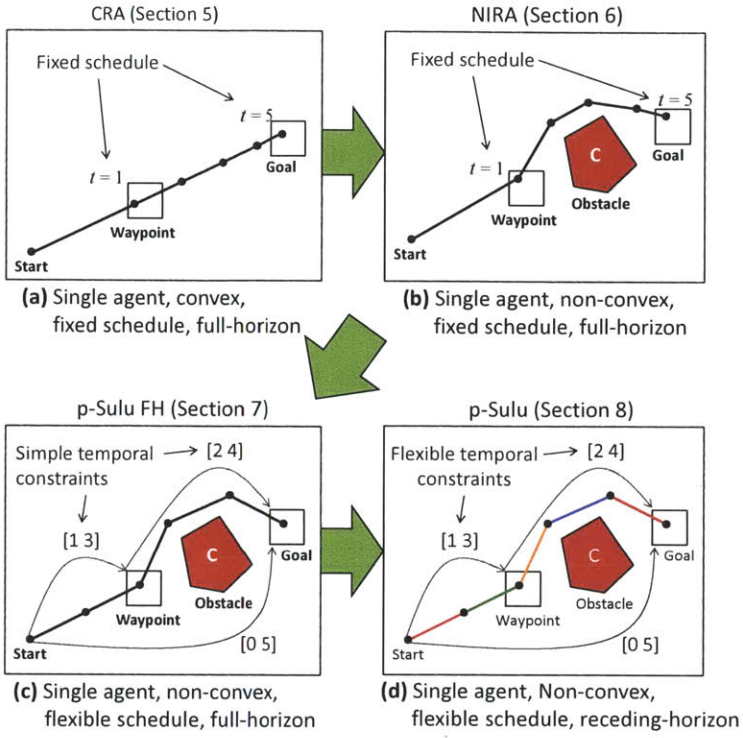


Figure 1-6: Spiral development approach to p-Sulu

a univariate probability distribution. Integrals over univariate probability distributions can be evaluated accurately and efficiently. Hence, we can obtain the equivalent deterministic form of the individual chance constraints. As a result, a stochastic optimization problem is transformed into a deterministic optimization problem.

In the second step, we optimize the control sequence, as well as the risk allocation, by solving the deterministic version of the convex chance-constrained optimal control problem. We show that the deterministic optimization problem is a convex problem. The convex risk allocation (CRA) algorithm optimizes the control sequence and schedule together by using a convex programming solver. The algorithm is explained in detail in Chapter 5.

Although these algorithms are very efficient, their convexity requirement, as well as the fixed schedule assumption, strongly limits their application for CCQSP planning problems.

1.6.4 NIRA: full-horizon CCQSP planning with a non-convex state space, a fixed schedule

Turning to our second spiral, the non-convex iterative risk allocation (NIRA) algorithm, which is presented in Chapter 6, removes the above convexity requirement. This means that the planner can plan a path that avoids obstacles, although the time steps to go through waypoints must be fixed, as in Figure 1-6-(b).

Our approach to solve the problem is again two-fold. First, we decompose a chance constraint on non-convex feasible state regions through risk allocation and *risk selection.* Risk selection is added to address non-convexity and reformulates each chance constraint over a disjunctive clause into a disjunction of individual chance constraints. If one of the individual chance constraints is satisfied, the original chance constraint is guaranteed to be satisfied. Intuitively, this decomposition allows the optimizer to select the individual chance constraint on which to impose the risk bound. Once the non-convex chance constraints are decomposed into a set of individual chance constraints, we apply the same deterministic transformation as in the convex, fixed schedule case. This bounding approach was originally proposed by [18].

Second, we solve the resulting deterministic version of the non-convex chance-constrained optimal control problem by the NIRA algorithm. The algorithm employs a branch-and-bound approach, where each subproblem is a convex chance-constrained optimal control problem. NIRA uses the IRA algorithm or a deterministic, convex program solver as a subroutine to solve each subproblem. For example, when generating a path plan with obstacles, the planner specifies for each subproblem the side of the obstacles that vehicle must be on at each time step, so that the subproblem has only a convex set of constraints. The branch-and-bound algorithm then searches for the side of obstacles at each time step that minimizes the given cost function.

However, although the branch-and-bound algorithm is guaranteed to find an optimal solution, it requires many subproblems to be solved. Since in our case the convex subproblems are nonlinear programs, this means that the overall computation time can be large. In order to speed up the algorithm, we propose a novel method called fixed risk relaxation

(FRR). An FRR of each subproblem is typically a linear program. FRRs can be solved efficiently and gives a tight lower bound on an objective function value. NIRA obtains a strictly optimal solution of Problem 10 by solving the subproblems *exactly* without FRR at unpruned leaf nodes of the search tree, while other subproblems are solved approximately with FRR in order to reduce the computation time.

We also present the **NIRA+BoostLP** algorithm, which achieves further speed-up. It solves FRRs *approximately* by using a regression-based LP solver called BoostLP [10]. While NIRA+BoostLP solves the FRRs of the subproblems using BoostLP to enhance computation speed, it solves the original subproblems without approximation at unpruned leaf nodes in order to obtain an exact optimal solution. As a result, NIRA+BoostLP achieves 10- to 25-fold reduction in computation time without any compromise in the optimality.

A limitation of the NIRA and NIRA+BoostLP algorithms is that the time steps to achieve the temporally extended goals of a CCQSP must be pre-specified. However, in real-world problems such as the PTS scenario presented in Section 1.1, the schedule is required to be flexible. We consider this next.

1.6.5 p-Sulu FH: full-horizon CCQSP planning with a non-convex state space and a flexible schedule

Turning to the third spiral, p-Sulu FH extends the general CCQSP planning capability by allowing flexibility in the schedule. As shown in Figure 1-6-(c), it optimizes the schedule to achieve the temporally extended goals within user-specified simple temporal constraints. The p-Sulu FH algorithm is explained in detail in Chapter 7.

p-Sulu FH employs a branch-and-bound approach in two places. The first one is to solve the scheduling problem, introduced by the addition of flexible temporal constraints. The subproblems of the first branch and bound are non-convex chance-constrained optimal control problems. Each of the subproblems is solved by the NIRA algorithm, which employs the second instance of the second branch-and-bound in order to deal with the non-convex constraints. The first instance of branch-and-bound searches for the optimal schedule by

incrementally assigning execution time steps to each event in a depth-first manner. For each subproblem, it fixes a schedule by specifying at which time step the vehicle achieves each goal. Lower bounds on the objective function value are obtained by solving fixed-schedule CCQSP planning problems with partial assignments to a schedule. p-Sulu FH minimizes the search space by dynamically pruning the domain through forward-checking. More specifically, after an execution time is assigned to an event at each iteration of the branch-and-bound search, it runs a shortest-path algorithm to tighten the real-valued upper and lower bounds on the execution time step of unassigned events according to the newly assigned execution time step.

In order to enhance the speed of the algorithm, we propose the variable ordering where the episodes with a convex feasible region are always considered before the episodes with a non-convex feasible region. As long as only the convex constraints are considered in a subproblem, p-Sulu FH solves the FRR of the subproblem, instead of the non-convex chance-constrained optimization problem.

1.6.6 p-Sulu: receding-horizon CCQSP execution with a non-convex state space and a flexible schedule

Turning to the fourth spiral, p-Sulu enables real-time execution of CCQSP using the receding horizon approach. As illustrated in Figure 1-6-(d), the entire plan is divided into multiple segments or *planning horizons*, each of which is generated on the fly. The risk needs to be distributed among those planning horizons. However, since the executive does not generate plans beyond the current planning horizon, the risk allocation among planning horizons cannot be decided at the beginning. Therefore, we develop an extension of risk allocation, called *risk budgeting*. For example, consider the chance constraint c_2 in the CC-QSP shown in Figure 1-4, which imposes a 0.01% risk bound on PAV 1 to avoid obstacles. We interpret this chance constraint to mean that PAV 1 has a 0.01% budget of risk at the beginning. The vehicle "consumes" the risk budget by allocating risk to planning cycles. For example, if PAV 1 allocates 0.002% of risk to the first path segment, then it has 0.008% of remaining risk budget at the second planning cycle. The amount of risk allocated to the

current planning cycle is decided heuristically.

The optimization problem solved by each planning cycle in p-Sulu is essentially the same as the full-horizon CCQSP planning problem, except for the following three differences. First, the fixed planning horizon is replaced with a receding horizon. Second, instead of imposing hard constraints that require executing all the episodes within the current horizon, p-Sulu imposes soft constraints that allow the executive to postpone the execution of episodes for later time horizons, with a penalty. Third, a cost-to-go function is added to the objective function in order to guide the plant state toward goals beyond the current planning horizon.

In real-time execution, it is more important to obtain a feasible solution in a fixed computation time, rather than pursuing optimality. Such a capability is provided by our IRA algorithm. IRA is a feasible descent optimization algorithm, meaning that the solutions to the subproblems are feasible throughout the iterations, and the cost function value monotonically decreases throughout the iterations. Each subproblem of IRA is a deterministic QSP planning problem without uncertainty, which is encoded in a mixed integer linear programming (MILP) and efficiently solved by a commercial solver [61]. Hence, the computation burden of one iteration of the IRA algorithm is equivalent to the deterministic QSP planning problem. Once a feasible solution is obtained, IRA iterates the solution to improve it until the time is up, instead of continuing the iteration until convergence to the optimal solution.

1.7 Innovations for p-Sulu

In the development of p-Sulu, we have created nine major innovations presented in this thesis.

First, in order to decompose a chance constraint over a conjunctive clause into a conjunction of individual chance constraints, we introduce the risk allocation approach (Section 4.1).

Second, in order to optimize risk allocation on a wide range of problems, we develop the IRA algorithm, which can be implemented on top of existing deterministic optimizers and

give them a capability to solve corresponding chance-constrained optimization problems (Section 4.2).

Third, in order to obtain lower bounds for the branch-and-bound search in NIRA, we develop fixed risk relaxation (FRR), a linear program relaxation of the subproblems (Section 6.4.2).

Fourth, in order to enhance the solution time of NIRA, we integrate a regression-based optimizer, BoostLP, into the branch-and-bound algorithm to solve a chance-constrained optimization problem. (Section 6.5).

Fifth, we minimize the search space for the optimal schedule in p-Sulu FH by introducing a forward-checking method that combines a d-graph with a branch-and-bound algorithm (Section 7.2).

Sixth, in order to minimize the number of non-convex subproblems solved in the branch-and-bound search, we introduce a variable ordering heuristic, namely the convex-episode-first (CEF) heuristic, which explores the episodes with a convex feasible state region before the ones with a non-convex state region (Section 7.2.2).

Seventh, in order to enhance the computation time of schedule optimization in p-Sulu FH, we introduced a method to obtain a lower bound for the branch-and-bound by solving fixed-schedule planning problems with a partial assignment of a schedule. (Section 7.3)

Eighth and finally, in order to find a feasible risk allocation in real-time execution in p-Sulu, we develop a risk budgeting approach (Section 8.1).

1.8 Overview of dp-Sulu

Next, we turn to dp-Sulu, our *distributed* robust CCQSP executive. Although p-Sulu can theoretically control multi-agent systems in a centralized manner, its computation time exponentially grows as the number of the agent increases since p-Sulu involves non-convex programmings. Therefore, it does not scale to systems with large numbers of agents in practice. dp-Sulu addresses this issue by distributing the computational burden among agents. Distributed planning also has an advantage in robustness since it can eliminate a single point of failure from the system. Moreover, it is relatively easy to achieve plug-and-

play of system components in a distributed system.

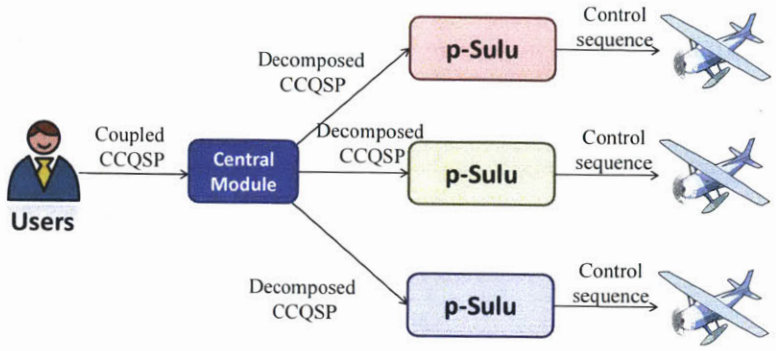


Figure 1-7: Overall Structure of dp-Sulu

dp-Sulu takes the same inputs as p-Sulu: initial conditions, stochastic plant models, and a CCQSP. Its outputs are also the same as p-Sulu: optimal executable control sequences and schedules. The difference is that the computation of the output is conducted in a distributed manner. Figure 1-7 shows the overall structure of dp-Sulu. The input CCQSP is decomposed by the central module and delivered to agents. The plan executive of each agent takes the decomposed CCQSP as an input, and outputs a control sequence and a schedule as outputs.

Figure 1-8 shows an example of a CCQSP for a personal transportation system (PTS) planning problem, shown in Figure 1-2. The CCQSP involves plan specifications for both PAVs 1 and 2. It can be stated informally in plain English as:

> "PAV 1 must start from Provincetown, reach the scenic region within 30 time units, and remain there for between 5 and 10 time units. It must end the flight in Bedford. At all times, PAV 1 must remain in the safe region by avoiding the no-fly zones and the storm. It must limit the probability of penetrating such obstacles to 0.01%. The entire flight must take at most 60 time units.
>
> "Meanwhile, PAV 2 must start from Hyannis and go to Bedford within 40 time units. At all times, it must remain in the safe region by avoiding the no-fly zones and the storm, and must limit the probability of penetrating such obstacles to 0.1%.

"Both vehicles must achieve the specified temporally-extended goals. The risk of failure to achieve at least one of the goals must be less than 1%. PAV 2 must land at Bedford at least five time units before PAV 1's landing. Finally, both vehicles must limit the risk of collision to 0.001%."

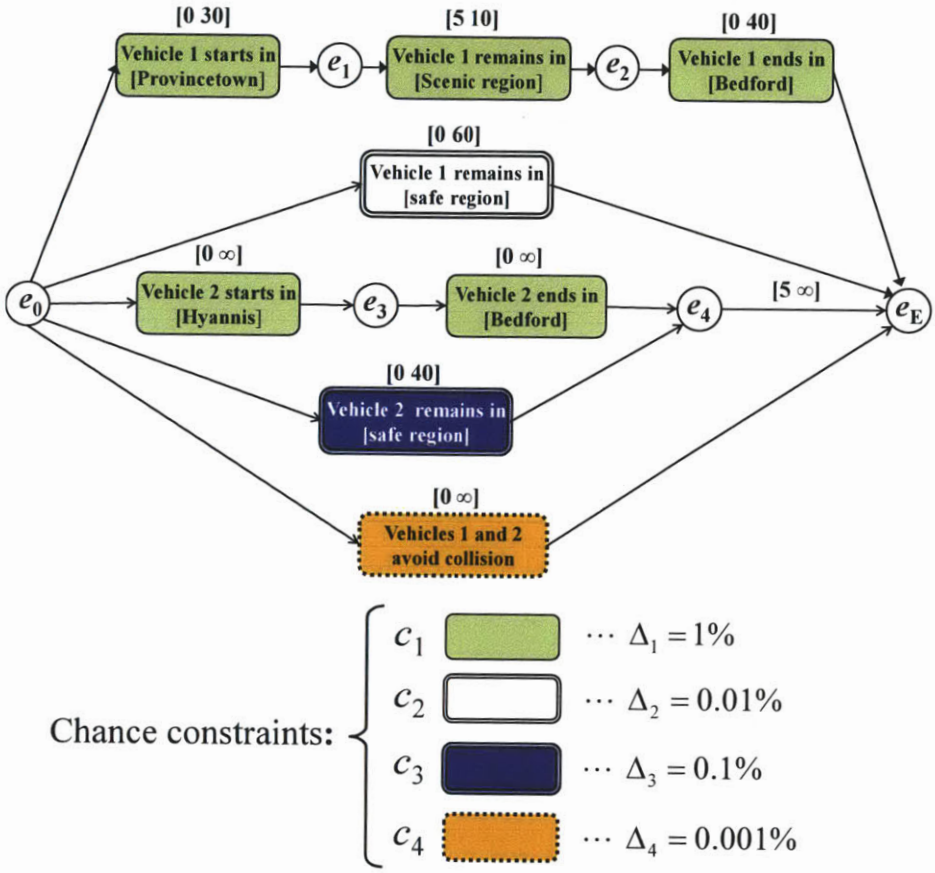


Figure 1-8: An example of a multi-agent CCQSP for a personal transportation system (PTS) planning problem, illustrated in Figure 1-2.

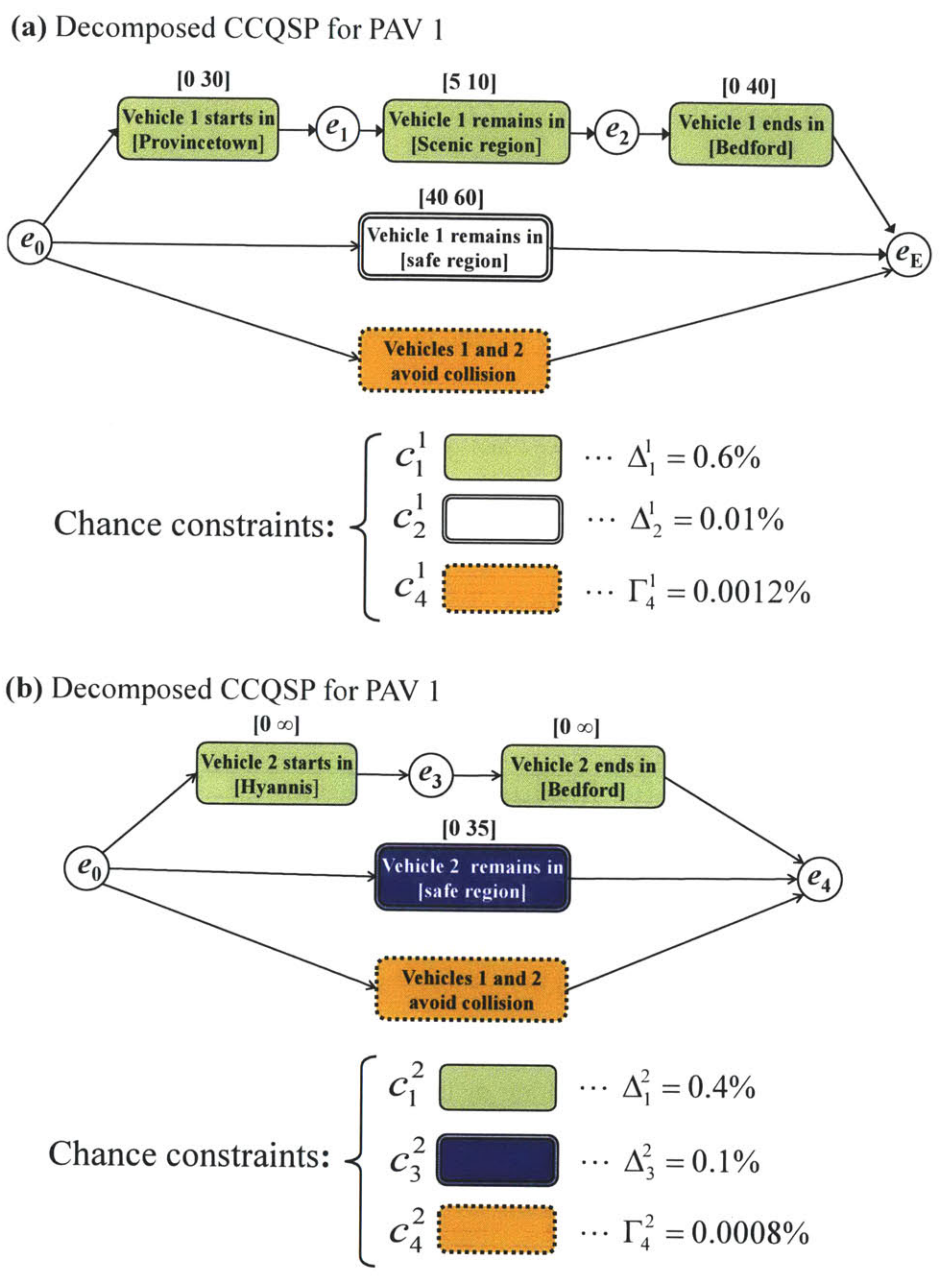


Figure 1-9: The decomposition of the CCQSP shown in Figure 1-8. (a) Decomposed CCQSP for PAV 1; (b) decomposed CCQSP for PAV 2.

Note that the two vehicles are coupled in three ways, through chance. First, there is a coupling through the chance constraint c_1 since it involves both vehicles. Second, two vehicles are coupled through the temporal constraint that requires a five time-unit interval between the two landings. Third, there is also a coupling through the collision avoidance

requirement.

dp-Sulu decouples such coupling constraints, and provide agents with decomposed CCQSPs. Figure 1-9 shows the decomposed CCQSPs. The decoupling is conducted so that the satisfaction of the coupling constraints is guaranteed if each vehicle satisfies its own decomposed CCQSP. For example, the decomposed CCQSP for PAV 1, shown in Figure 1-9-(a), requires PAV 1 to land at Bedford after 40 time units, while the CCQSP for PAV 2, shown in Figure 1-9-(b), requires PAV 2 to land at Bedford before 35 time units. If both vehicles satisfy these requirements, then the coupled temporal constraints that require a five time-unit interval between two landings are guaranteed to be satisfied. The next section presents the key insights underlying the decomposition. Chapter 10 explains in detail how to obtain such decomposed CCQSPs.

1.9 Approach to dp-Sulu

1.9.1 Risk Allocation for Multi-agent Systems

The concept of risk allocation can be naturally extended to multi-agent systems. Consider an example where two agents work together to complete a mission. Failure of one agent results in a failure of the mission. The user of the multi-agent system imposes 1% risk bound on the mission failure. Such risk can be distributed among agents. The two agents can split the risk in many ways. As long as the risk allocations of both agents add up to 1%, the risk of mission failure is guaranteed to be less than 1%. If one agent has higher sensitivity to risk than the other, then the overall performance of the multi-agent system can be enhanced by allocating larger amounts of risk to the one with higher sensitivity. The multi-agent extension of the risk allocation concept is discussed in detail in Section 9.1.1.

1.9.2 Spiral Development of dp-Sulu

We develop the distributed robust plan executive, dp-Sulu, in two spirals. In the first spiral, we develop the *market-based iterative risk allocation (MIRA)* algorithm, which solves a full-horizon multi-agent CCQSP planning problem with a convex state space and a fixed

schedule in a distributed manner, as illustrated in Figure 1-10-(a). In the second spiral, we build the distributed robust CCQSP executive, dp-Sulu, upon p-Sulu. dp-Sulu executes a multi-agent CCQSP in real time, with a non-convex state space and a flexible schedule, as shown in Figure 1-10-(b).

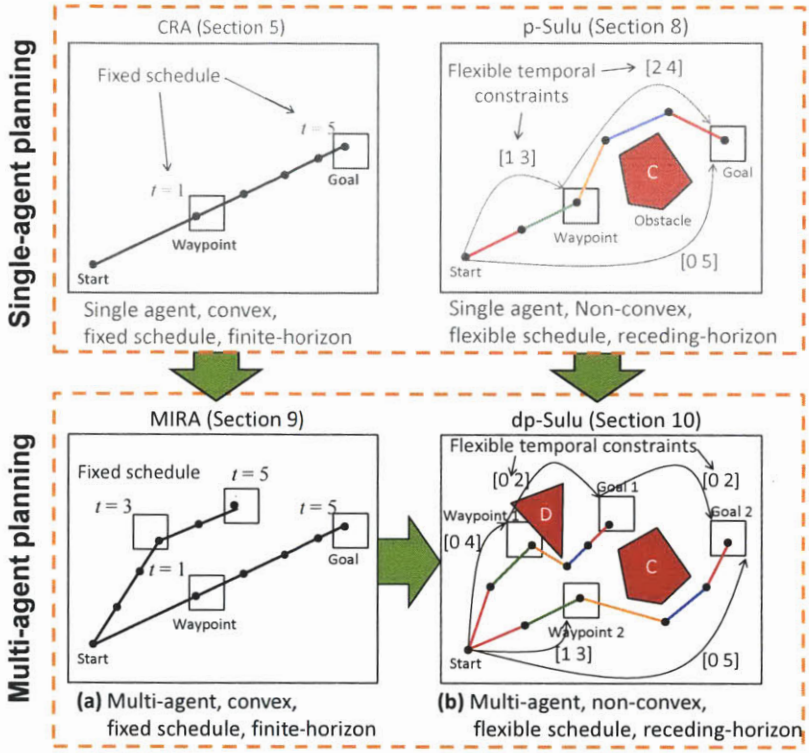


Figure 1-10: Spiral development approach to dp-Sulu

1.9.3 MIRA: full-horizon multi-agent CCQSP planning with a convex state space and a fixed schedule

The first spiral deals with a special case of CCQSP planning problem, where there is no obstacle in the environment and the schedule is fixed. We assume that the agents are coupled only through chance constraints. The challenge here is how to find the optimal risk allocation in a distributed manner.

MIRA addresses this challenge through a market-based approach. It creates a computational market of risk, where each agent can purchase risk in order to improve its own

performance. Agents are price takers. Given the price, each agent computes the optimal amount of risk to take (i.e., *demand for risk*) by solving a convex optimization problem. The optimal action sequence and the internal risk allocation are also determined by solving the optimization problem, just as in the CRA algorithm. The demand from each agent can be seen as a function of the price of risk (*demand curve*). Typically, the higher the price is, the less each agent demands. Each agent has a different demand curve according to its sensitivity to risk. The supplier of risk is the user. She supplies the fixed amount of risk by specifying the upper bound on risk that the system can take.

The price of risk must be adjusted so that the total demand (*aggregate demand*) becomes equal to the supply. The equilibrium price is found by an iterative process called *tâtonnement* or *Walrasian auction* [92] as follows:

- Increase the price if aggregate demand exceeds supply,
- Decrease the price if supply exceeds aggregate demand, and
- Repeat until supply and demand are balanced.

Our method obtains the price increment in each iteration by computing one step of Brent's method, which is a commonly-used root-finding algorithm with fast and guaranteed convergence [6]. The details of MIRA are described in Chapter 9.

To explain this approach mathematically, the price of risk corresponds to the dual variable (or the Lagrange multiplier) for the risk allocation constraint. Each agent computes the dual objective function value, while the central module optimizes the dual variables. Such a decomposition approach is called dual decomposition.

1.9.4 dp-Sulu: receding-horizon multi-agent CCQSP execution with a non-convex state space and a flexible schedule

Turning to the second and final spiral, we develop the distributed robust CCQSP executive, dp-Sulu, which executes multi-agent CCQSP in real time with obstacles and a flexible schedule. As is explained in Section 1.8, dp-Sulu consists of centralized and distributed modules. The central module decouples the coupling constraints in a given CCQSP, and provide agents with decomposed CCQSPs. Given the decomposed CCQSP, each agent runs

p-Sulu to generate its own control sequence and schedule (see Figure 1-7). The challenge here is how to decompose couplings through chance, state, and temporal constraints.

The coupled chance constraints are handled by the *Multi-agent Heuristic Risk Allocation* approach. Multi-agent Heuristic Risk Allocation takes a set of coupled chance constraints as an input, and outputs a set of decomposed chance constraints for every agent (Line 4 in Algorithm 14).

We propose a new approach called Bi-stage Robust Collision Avoidance (BRCA), in order to address coupling through state constraints (collision avoidance constraints) efficiently. In BRCA, each agent solves the path planning problem without the collision avoidance constraints in the first iteration. We call the resulting solution the proposal solution. If the proposal solution satisfies the collision avoidance constraints, the executive executes the control sequence of the proposal solution. Otherwise, the central module constructs decomposed chance constraints based on the proposal solution. Each agent plans the path again with the decomposed constraints. We call the new solution the adjusted solution. The adjusted solution is guaranteed to satisfy the collision avoidance constraints. Hence, each agent executes the control sequence of the adjusted solution.

Temporal constraints couple agents essentially because they are binary constraints, meaning that they impose upper and lower bounds on the *duration* between the execution times of two events. Our novel approach, *Robust Feasible Temporal Constraint Decomposition*, derives a set of unary temporal constraints for each agent by solving a linear programming. The set of unary temporal constraints is a sufficient condition of the original temporal constraints. The linear programming problem is guaranteed to have a feasible solution if the given binary temporal constraints are feasible. Our approach is suitable for real-time execution because it does not involve iteration or backtracking.

We present dp-Sulu in detail in Chapter 10.

1.10 Innovations for dp-Sulu

In the development of dp-Sulu, we have developed four major innovations presented in this thesis.

First, in order to decompose a chance constraint over multiple agents, we develop a multi-agent extension of the risk allocation approach (Section 9.1.1).

Second, in order to optimize risk allocation in a full-horizon CCQSP problem, we develop a novel dual-decomposition algorithm MIRA (Section 9.3).

Third, in order for dp-Sulu to decompose collision avoidance constraints without iterations, we develop a Bi-stage Robust Collision Avoidance method (Section 10.4).

Fourth, in order to decompose coupled temporal constraints without iterations or backtracking, we develop a Robust Feasible Temporal Constraint Decomposition method (Section 10.5).

1.11 Summary of Empirical Results

In Chapter 11 of this thesis, we deploy the proposed planners and executives on aerial, underwater, and space vehicles, as well as various benchmark problems.

The results chapter of this thesis consists of three parts. In the first part, we evaluate the performance of each of the proposed planners and executives by comparing them with prior art methods. The benchmark problems used for the evaluation are relatively simple compared to real-world problems. Simulations are run multiple times with randomized settings. Overall, our algorithms result in more efficient solutions (i.e., less cost) than prior art methods. This is because all of our algorithms are build upon the risk allocation approach. By optimally allocating risk among agents, time steps, and constraints, our planners and executives achieve improved solution efficiency compared to previously proposed algorithms that do not flexibly allocate risk. However, in general the risk allocation-based algorithms take relatively longer computation time than existing algorithms with fixed risk bounds. Hence, a number of methods are proposed throughout this thesis to reduce the computation time. As a result, the empirical results show that our algorithms are tractable for many applications.

In the second part we apply our planners and executives to PTS. This part consists of three spirals. In Spiral 1, we use p-Sulu FH as a stand-alone controller. In this spiral we show that our approach can scale to real-world problems. In Spiral 2, we integrate p-Sulu

with other planners to create Integrated Plan Executive (IPE). Besides p-Sulu, IPE includes Kirk (a temporally-flexible contingent planner) [37, 44, 50], Collaborative Diagnosis System (a plan conflict fixer) [107], and Dialog Management System (a natural language interpreter) [63, 64]. The simulation results of this spiral demonstrate p-Sulu's capability of adapting to environmental changes, such as the change in the location of storms. In Spiral 3, we replace p-Sulu in Spiral 2 with dp-Sulu. The simulation results of this spiral demonstrate dp-Sulu's capability of controlling a multi-agent system in a distributed manner. The simulation results are visualized by a flight simulator called X-Plane, as shown in Figure 1-11.

Figure 1-11: Visualization of dp-Sulu execution result by the X-Plane flight simulator

Finally, in the third part, we apply our planners and executives to the control of underwater and space vehicles, both of which have significantly different plant models than the PTS. The objective of this section is to demonstrate that our approach is applicable to a wide range of real-world systems. For the underwater scenario, we used the plant model of a Dorado-class autonomous underwater vehicle (AUV) [52] developed and operated by Monterey Bay Aquarium Research Institute. For the space scenario, the plant model takes orbital mechanics into account. We simulate an autonomous rendezvous of a cargo vehicle, H-II Transfer Vehicle (HTV), which resupplies the International Space Station (ISS) [105]. We also apply dp-Sulu for a space formation flight scenario. In all of these scenarios, the empirical results demonstrate that our algorithms can control the vehicles with improved efficiency and bounded risk.

1.12 Related Work

1.12.1 Plan Execution with Temporally Extended Goals

Recall that the CCQSP planning problem is distinguished by its use of temporally extended goals, continuous states and actions, stochastic optimal solutions and chance constraints. While the planning and control disciplines have explored aspects of this problem, its solution in total is novel, and our approach to solving this problem efficiently through risk allocation is novel.

More specifically, there is an extensive literature on planning with discrete actions to achieve temporally extended goals (TEGs), such as TLPlan [7] and TALPlan [60], which treat TEGs as temporal domain control knowledge and prune the search space by progressing the temporal formula. However, since these TEG planners assume discrete state spaces, they cannot handle problems with continuous states and effects without discretization. Ignoring chance constraints, the representation of temporally extended goals used by TLPlan and p-Sulu is similar. TlPlan uses a version of metric interval temporal logic (MITL) [5] applied to discrete states, while p-Sulu uses qualitative state plans (QSPs) [43, 62, 66] over continuous states. [66] shows that, for a given state space, any QSP can be expressed in MITL. The key difference that defines p-Sulu is the addition of chance constraints, together with its use of continuous variables.

Several planners, particularly those that are employed as components of model-based executives, command actions in continuous state space. For example, Sulu [62] takes as input a deterministic linear model and QSP, which specifies a desired evolution of the plant state as well as flexible temporal constraints, and outputs a continuous control sequence. Its approach is to encode the QSP execution problem into a mixed integer linear program (MILP) and solve it using the CPLEX optimizer. Chekhov [43] also takes as input a QSP and a nonlinear deterministic system model, and outputs a continuous control sequence. Chekhov's design is driven by the need for compliance and fast response, hence it precomputes *flow tubes*, the sets of continuous state trajectories that end in the goal regions specified by the given plan. A plan is executed by following a trajectory within each flow tube. This approach is particularly effective for systems with complicated dynamics, such

as biped robots, and the need for fast response. Kongming [66] provides a generative planning capability for hybrid systems, involving both continuous and discrete actions. It employs a compact representation of hybrid plans, called a Hybrid Flow Graph, which combines the strengths of a Planning Graph for discrete actions and flow tubes for continuous actions. In artificial intelligence (AI) planning literatures, a planning domain description language, PDDL+ [40], supports mixed discrete-continuous planning domains. A PDDL planning problem with continuous linear numeric change can be efficiently solved by a temporal planner, Colin [31].

1.12.2 Planning under Uncertainty

Plan executives mentioned in the previous subsection adapt to the effects of uncertainty, but do not explicitly reason about the effects of uncertainty during planning. For example, Sulu employs a receding horizon approach, which continuously replans the control sequence using the latest measurements. Chekhov's flow tube representation of feasible policies allows the executive to generate new control sequences in response to disturbances on-line. p-Sulu is distinct from these continuous planners in that it plan with a model of uncertainty in the continuous dynamics, instead of just reacting to it. Its solution guarantees the user-specified probability of success by explicitly reasoning about the effects of uncertainty.

Most work within the AI community on probabilistic planning has focused on planning in discrete domains and builds upon the Markov decision process (MDP) framework. A growing sub-community has focused on extensions of MDPs to the continuous domain. However, tractability is an issue, since they typically require partitioning or approximation of the continuous state space. A straightforward partitioning of the continuous state and action spaces into discrete states and actions often leads to an exponential blow-up of running time. Furthermore, when the feasible state space is unbounded, it is impossible to partition the space into a finite number of compact subspaces. An alternative approach is to approximate the value function using linear or nonlinear function approximators [21], but it lacks the guarantee of convergence [8, 13]. Time-dependent MDPs [22, 38] can do efficient partitioning of continuous state space, but make a relatively strong assumption that the transition model and action space are discrete, and the state space is bounded. p-Sulu is

similar to MDPs in that it discretizes time. However, our approach is essentially different from the MDP approaches in that the continuous state and control variables are directly optimized through convex optimization instead of discrete dynamic programming. Therefore, p-Sulu does not require any discretization of the continuous state space. Hence, the continuity of the state space does not harm the tractability of p-Sulu.

A second point of comparison is the treatment of risk. Like p-Sulu, the MDP framework offers an approach to marrying utility and risk. However, most MDP algorithms balances the utility and risk by assigning a large negative utility to the event of constraint violation. Such an approach cannot guarantee bounds on the probability of constraint violation. The constrained MDP approach [4] can explicitly impose constraints. [34] showed that stationary deterministic policies for constrained MDPs can be obtained by solving a mixed integer linear program (MILP). However, the constrained MDP framework can only impose bounds on the *expected value* of costs, and again, cannot guarantee strict upper bounds on the probability of constraint violation. In contrast, p-Sulu FH allows users to impose chance constraints, which explicitly restrict the probability of constraint violation. As far as we know, [42] is the only work that considers MDPs with chance constraints. They developed a reinforcement learning algorithm for MDPs with a constraint on the probability of entering error states. Our work is distinct from theirs in that p-Sulu FH is goal-directed, by which we mean that it achieves temporally extended goals within user-specified temporal constraints.

1.12.3 Receding Horizon Control

p-Sulu's capability for planning in continuous domains is built upon receding horizon control, also known as model predictive control (MPC). Initially developed for process industries, MPC is a closed-loop control approach that, at each time step, solves a finite-horizon optimal control problem from the current initial state, executes the first step in the resulting optimal control sequence, and then resolves at the next time step [41]. It has also been successfully applied to vehicle path planning problems in continuous state space [24, 49, 58, 80, 81]. MPC is distinguished from classical control approaches by its capability to explicitly consider state constraints. This feature of MPC enables its application to

plan execution in continuous state spaces, since temporally extended goals can be encoded into inequality constraints on the continuous plant state. Sulu is the first continuous plan executive that applies the MPC approach [61]. Discrete plan executives that have employed this approach include Remote Agent [74] and Aspen/Casper [29, 53]. Our plan executives, p-Sulu and dp-Sulu, extend Sulu to deal with stochastic uncertainty.

Recently, a significant effort in the MPC community has been devoted to the development of robust model predictive control (RMPC) algorithms. The main purpose of RMPC is to guarantee stability under the existence of uncertainty. Most literature on RMPC assumes a set-bounded model of uncertainty, meaning that the level of uncertainty at each time step is upper-bounded [48, 54, 59, 67]. Under moderate conditions, set-bounded RMPC is proved to be asymptotically stable, and a feasible solution is guaranteed to be found at every planning cycle (this feature is called resolvability). An extensive review of the literature on set-bounded RMPCs is given in [71]. Our algorithms are different from the set-bounded RMPC approaches in that we use a *stochastic* model of uncertainty, as is explained in detail in the following subsection.

1.12.4 Chance-constrained Optimal Control

Although set-bounded uncertainty is a common assumption in RMPC, stochastic uncertainty is a more natural model for exogenous disturbances in practice. For example, Figure 1-12 shows the histogram of the wind speed observed in Uccle, Belgium, over a year. Here, it is difficult to find a hard upper bound on wind speed. Rather, it is more plausible to model the wind speed as a stochastic variable that follows a certain probability distribution. In fact, Vallée et al. [96] found that the Weibull distribution, shown by the dashed line in Figure 1-12, gives a good approximation of the distribution of wind speed. Barr et al. [11] found that low-altitude wind and turbulence can be approximated by Gaussian distributions.

However, with stochastic uncertainty, it is often impossible to guarantee that state constraints are satisfied, since there is typically a non-zero probability of having a disturbance that is large enough to push the state out of the feasible region. An effective framework to address robustness with stochastic uncertainty is optimization with chance constraints.

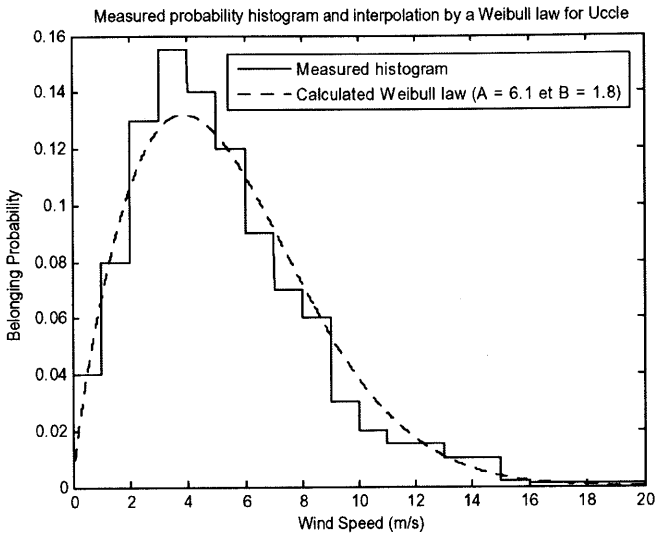


Figure 1-12: Probabilistic histogram of the wind speed observed in Uccle, Belgium, over a year (solid line), and an approximation with Weibull distribution (dashed line). The plots are from Vallée et al., 2007 [96]

.

Chance constraints require that the probability of violating the state constraints (i.e. the probability of failure) is below a user-specified risk bound.

Previous work [16, 25, 75, 97] studied a specific form of chance constraint, called a *joint chance constraint*, which is defined on the *conjunction* of linear state constraints. The method proposed by [97] turns a stochastic problem into a deterministic problem using a very conservative ellipsoidal relaxation. Although this algorithm is computationally efficient, its result is highly suboptimal, since the ellipsoidal relaxation produces a very conservative bound. [16] propose a sampling-based method called Particle Control, which can directly optimize the control sequence without using a conservative bound. The advantage of Particle Control over other approaches is that it can handle arbitrary distributions. However, it is slow since the dimension of the decision vector grows proportionally with the number of samples. [18] and [75] employed Boole's inequality to decompose a joint chance constraint into individual chance constraints. Although Boole's inequality is less conservative than the ellipsoidal relaxation, their approach still has non-negligible conservatism, since it fixes each individual risk bound to a uniform value. Our risk allocation approach builds upon this approach, with modifications to optimize individual risk bounds

in order to improve utility. The Bernstein approximations proposed by Prekopa [79] uses Bonferroni's inequality, a generalizations of Boole's inequality with higher binomial moments. A special case of his approach with the first-order moment also involves optimization of the individual risk bounds. This special case of [79] corresponds to a special case of our risk allocation approach with convex state constraints. Hence, both our work and [79] are generalization of the convex risk allocation but in different directions. In our work the risk allocation approach is extended to non-convex problems, as well as to planning with temporally extended goals. We also develop the IRA algorithm, which enables different deterministic planners under the risk allocation. [69] proposed a chance-constrained rapidly-exploring random tree (CC-RRT) approach. The randomized algorithm efficiently generates a dynamically feasible and probabilistically safe trajectory to the specified goal. However, since its focus is to obtain a feasible trajectory quickly, it does not guarantee the optimality of the resulting trajectory.

1.12.5 Distributed Planning and Distributed MPC

MIRA is build upon a market-based approach called tâtonnement. Although tâtonnement has drawn less attention than other market-based approaches such as auctions, it has been successfully applied to various problems such as the distribution of heating energy in an office building [101], control of electrical power flow [45], and resource allocation in communication networks [47]. In economics, a simple linear price update rule has long been the main subject of study, but the convergence of price can only be guaranteed under a quite restrictive condition [92]. In order to substitute the linear price update rule, various root-finding methods have been employed in agent-based resource allocation algorithms, such as the bisection method [103], Newton's method [106], and Broyden's method [101]. However, in general, it is difficult to guarantee quick and stable convergence to the equilibrium price. We employ Brent's method [6] to provide guaranteed convergence with a superlinear rate of convergence by exploiting the fact that a risk, which is treated as a resource in our problem formulation, is a scalar value.

Another thread of research that is relevant is distributed model predictive control, which is a branch in the larger area of cooperative control. For example, [36] developed a dis-

tributed MPC algorithm that cooperatively controls multiple agents, whose dynamics and constraints are decoupled, but whose state vectors are coupled through the objective function. This algorithm is extended by [35] in order to consider coupling through dynamics. [58] proposed a cooperative path planning algorithm, where agents solve their subproblems in sequence in order to generate robust collision-free paths. Our work is distinct from such distributed MPC approaches in that we explicitly consider stochastic uncertainty and impose chance constraints. Recent research in cooperative control is summarized in [73].

1.13 Thesis Organization

Six algorithms are presented in this thesis. In order to organize the presentation, we characterize the six algorithms by three properties, as shown in Table 1.1.

Table 1.1: Mappings of problems and solution approaches to the algorithms presented in this thesis.

	Centralized		Decentralized	
	Full horizon	Receding horizon	Full horizon	Receding horizon
Convex fixed schedule (Problem 4)	CRA (Chapter 5)	-	MIRA (Chapter 9)	-
Non-convex, fixed schedule (Problem 3)	NIRA (Chapter 6)	-	-	-
Non-convex, flexible schedule (Problem 2)	p-Sulu FH (Chapter 7)	p-Sulu (Chapter 8)	-	dp-Sulu (Chapter 10)

The first property is the level of generality of the CCQSP and environmental constraints addressed. The most general problems considered in this thesis allow a non-convex state space (hence allowing obstacles in environment) and a flexible schedule. p-Sulu FH, p-Sulu, and dp-Sulu solve this type of problem. The second general form of problem also allows a non-convex state space, but with a fixed schedule. NIRA solves a problem in this category. The most restrictive problem only deal with a convex state space and a fixed schedule. CRA and MIRA solve this class of problem. **Chapter 2** provides the formal problem statement of the most general problem. **Chapter 3** derives the encoding of the

general problem as a chance-constrained optimization problem (Problem 2), as well as the encodings of the two limited versions of the problem (Problems 3 and 4).

The second property is the type of planning horizon: full horizon or receding horizon. The former is typically used for preplanning, while the latter is used for real-time plan execution. Hence, we call the algorithms with full planning horizon (CRA, NIRA, MIRA, and p-Sulu FH) *planners*, while calling the receding horizon algorithms (p-Sulu and dp-Sulu) *plan executives*.

The third property is the optimization approach: centralized or distributed. The centralized algorithms include CRA, NIRA, p-Sulu FH, and p-Sulu, while the distributed algorithms include MIRA and dp-Sulu.

All of the six algorithms are built upon the risk allocation approach, which is explained in **Chapter 4**. The chapter also presents the general risk allocation optimization problem, Iterative Risk Allocation (IRA). IRA is a key component of p-Sulu and dp-Sulu.

Figure 1-13 shows the technical dependencies between the six algorithms, as well as risk allocation and IRA. We first present the four centralized algorithms. Starting from the algorithm that solves the most restrictive problem, **Chapter 5** presents the CRA algorithm. Building upon CRA, we move towards more general solvers: **Chapter 6** presents NIRA, and **Chapter 7** presents p-Sulu FH. The centralized CCQSP executive, p-Sulu, is presented in **Chapter 8**.

We then present the two decentralized algorithms. **Chapter 9** presents the simpler one, MIRA. Then, finally, **Chapter 10** presents our final product, dp-Sulu, the decentralized robust CCQSP executive.

We present the empirical results in **Chapter 11**, and conclude the thesis and discuss future work in **Chapter 12**.

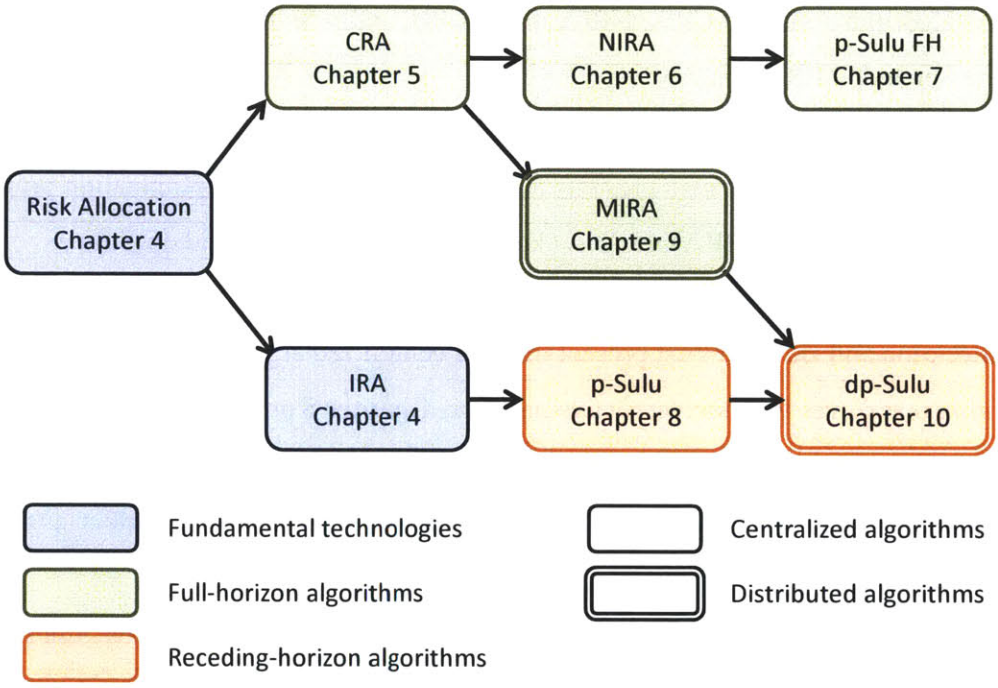


Figure 1-13: Technical dependencies of the algorithms presented in this thesis.

Chapter 2

Problem Statement

As introduced in the preceding chapter, our plan executives, p-Sulu and dp-Sulu, generate an executable optimal control sequence that achieves a set of temporally extended goals within specified risk levels. In particular, the executives take as input linear stochastic plant models, which specify the effects of actions; initial state descriptions, describing probability distributions over initial states of the all agents in the system; a CCQSP, which specifies desired evolutions of the state variables, as well as acceptable levels of risk; and an objective function. Their outputs are the optimal executable control sequences and schedules.

Recall that the difference between p-Sulu and dp-Sulu, or the difference between CRA and MIRA, is not in the problem that the algorithms solve, but in the solution approach: centralized or decentralized. Problem formulations of a single-agent problem and a multi-agent problem are essentially the same. Therefore, in this chapter, we state single-agent and multi-agent problems in a unified way.

The full-horizon planners (CRA, NIRA, MIRA, p-Sulu FH) and receding-horizon executives, introduced in this thesis, solve the same problem, but again in a different approach. The former solves the planning problem in one shot, while the latter solves it by repeatedly solving subproblems with a shorter time horizon. Each subproblem of the receding horizon approach is essentially the same as the full-horizon planning problem, with several modifications. Such modifications are described in detail in Section 8.2.

We first define the variables and events used in the problem formulations. Then we define elements contained in inputs and outputs. Notations defined in this chapter are

summarized in the Appendix.

2.1 Definition of time step

We consider a series of discretized finite time steps $t = 0, 1, 2, \cdots N$ with a fixed time interval ΔT, where integer N is the size of the planning horizon. Since the time interval ΔT can take any positive real value, it suffices to consider time steps with only integer indices to approximate the system's dynamics. We use the term "time step" to mean an integer index of the discretized time steps, while using the term "time" to mean a real-valued time. We define a set $\mathbb{T}$ as follows:

$$\mathbb{T} := \{0, 1, 2, \cdots N\}. \tag{2.1}$$

We denote by $\mathbb{T}^-$ the set of all time step in $\mathbb{T}$ except for the last time step:

$$\mathbb{T}^- := \{0, 1, 2, \cdots N-1\}.$$

We require that the duration of the given plan is upper bounded by the given CCQSP (Assumption 2). When such a requirement is satisfied, we can always find a finite positive integer N so that $\mathbb{T}$ covers the entire plan. The justification for the assumption is described in Section 2.4.3.

2.2 Definitions of Events

An event denotes the start or end of an episode of behavior in our plan representation.

Definition 1. *An **event** $e \in \mathcal{E}$ is a instance that is executed at a certain time step in $\mathbb{T}$.*

We define two special events, the start event e_0 and the end event e_E. We assume that e_0 is always executed at $t = 0$. The end event e_E represents the termination of the entire plan.

2.3 Definitions of variables

Variables used in our problem formulation involves a discrete schedule, a continuous state vector, and a continuous control vector.

2.3.1 Schedule

A schedule is a set of assignments of execution time steps to all the events in $\mathcal{E}$ as follows:

Definition 2. *A **schedule** $s : \mathcal{E} \mapsto \mathbb{T}$ is a map from events to their execution time steps.*

We denote by $s(e)$ the assignment of execution time steps to the event $e \in \mathcal{E}$. By definition, the start event is executed at $t = 0$ i.e, $s(e_0) = 0$.

In Section 7, we solve relaxed optimization problems with a partially assigned schedule in order to obtain lower bounds of the optimal objective value. We define a *partial schedule* as follows:

Definition 3. *A **partial schedule** $\sigma : \mathcal{E}_\sigma \mapsto \mathbb{T}$ is a map from a subset of events in a CC-QSP to their execution time steps, where $\mathcal{E}_\sigma \subseteq \mathcal{E}$.*

Note that a fully assigned schedule is also a partial schedule. Following the notation of a schedule, we denote by $\sigma(e)$ the execution time of an event $e \in \mathcal{E}_\sigma$. See also the definition of a schedule (Definition 2).

2.3.2 state vector

We consider a continuous state space, where a state vector is defined as follows:

Definition 4. *A **state vector** $\boldsymbol{x}_t \in \mathbb{R}^{n_x}$ is a real-valued vector that represents the state of the plant at time step t.*

Note that $\boldsymbol{x}_t$ is a random variable, in order to model a stochastic plant dynamics. We also define a state sequence as follows:

Definition 5. *A **state sequence** $\boldsymbol{x}_{0:N} := [\boldsymbol{x}_0 \cdots \boldsymbol{x}_N]$ is a vector of state variables from time 0 to N.*

2.3.3 Control vector and control sequence

Our actions are assignments to continuous control inputs:

Definition 6. *A **control vector** $\boldsymbol{u}_t \in \mathbb{U}$ is a real-valued vector that represents the control input to the system at time step t, where $\mathbb{U} \subset \mathbb{R}^{n_u}$ is a compact set that represents the continuous domain of the feasible control inputs.*

Recall that one of two outputs of p-Sulu is the optimal control sequence:

Definition 7. *A **control sequence** $\boldsymbol{u}_{0:N-1} := [\boldsymbol{u}_0 \cdots \boldsymbol{u}_{N-1}]$ is a vector of control inputs from time 0 to $N-1$.*

2.4 Definitions of inputs

This subsection defines the four inputs of p-Sulu: an initial condition, a stochastic plant model, a CCQSP, and an objective function.

2.4.1 Initial condition

We assume that the initial state has a Gaussian distribution with a known mean $\bar{\boldsymbol{x}}_0$ and a variance $\Sigma_{\boldsymbol{x}_0}$:

$$\boldsymbol{x}_0 \sim \mathcal{N}(\bar{\boldsymbol{x}}_0, \boldsymbol{\Sigma}_{\boldsymbol{x}_0}). \tag{2.2}$$

The parameters in (2.2) are specifed by an *initial condition*, which is defined as follows:

Definition 8. *An **initial condition** I is a pair $I = \langle \bar{\boldsymbol{x}}_0, \boldsymbol{\Sigma}_{\boldsymbol{x}_0} \rangle$, where $\bar{\boldsymbol{x}}_0$ is the mean initial state and $\boldsymbol{\Sigma}_{\boldsymbol{x}_0}$ is the covariance matrix of the initial state.*

2.4.2 Stochastic plant model

Traditional AI planning systems perform discrete actions that affect the values of discrete variables or conditions, and whose behaviors are specified as state transitions, using a language like Planning Domain Definition Language (PDDL) [39, 72]. p-Sulu controls dynamical systems in which actions correspond to the settings of continuous control variables, and whose effects are on continuous state variables. p-Sulu specifies these actions

and their effects through the plant model. p-Sulu employs a continuous-state, discrete-time linear dynamics with additive disturbance. This is a standard plant model used in chance-constrained stochastic optimal control [27, 75, 77, 97]. In Chapters 5-10, we assume that the disturbance has a zero-mean Gaussian distribution with known variance. Our use of Gaussian distribution is justified because many naturally occurring disturbances can be well approximated by Gaussian distributions. We do *not* assume any specific type of probability distribution in Chapter 4. Although we assume Gaussian distribution in most part of this thesis, our methods can be extended to non-Gaussian disturbances.

We consider the following stochastic, discrete-time, continous-state linear plant model:

$$\forall t \in \mathbb{T}, \quad \boldsymbol{x}_{t+1} = \boldsymbol{A}_t \boldsymbol{x}_t + \boldsymbol{B}_t \boldsymbol{u}_t + \boldsymbol{w}_t, \tag{2.3}$$

where $\boldsymbol{w}_t \in \mathbb{R}^{n_x}$ is the disturbance at t-th time step that has a zero-mean Gaussian distribution with covariance matrix Σ_{w_t}:

$$\boldsymbol{w}_k \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma}_{w_t}). \tag{2.4}$$

The parameters in (2.3) and (2.4) are specified by a *stochastic plant model*, which is defined as follows:

Definition 9. *A **stochastic plant model** M is a three-tuple $M = \langle \boldsymbol{A}_{0:N-1}, \boldsymbol{B}_{0:N-1}, \boldsymbol{\Sigma}_{w_{0:N-1}} \rangle$, where $\boldsymbol{A}_{0:N-1}$ and $\boldsymbol{B}_{0:N-1}$ are sets of N matrices $\boldsymbol{A}_{0:N-1} := \{\boldsymbol{A}_0, \boldsymbol{A}_1, \cdots \boldsymbol{A}_{N-1}\}$, $\boldsymbol{B}_{0:N-1} := \{\boldsymbol{B}_0, \boldsymbol{B}_1, \cdots \boldsymbol{B}_{N-1}\}$, and $\Sigma_{w_{0:N-1}}$ is a set of N covariance matrices $\Sigma_{w_{0:N-1}} = \{\Sigma_{w_0}, \Sigma_{w_1}, \cdots, \Sigma_{w_{N-1}}\}$.*

Note that $\boldsymbol{x}_t$, as well as $\boldsymbol{w}_t$, is a random variable with Gaussian distribution, while $\boldsymbol{u}_t$ is a deterministic variable. Figure 2-1 illustrates our plant model. In a typical plant model, the probability circles grow over time since disturbance $\boldsymbol{w}_t$ is added at every time step, as drawn in the figure. This effect represents a commonly observed tendency that the distant future involves more uncertainty than the near future.

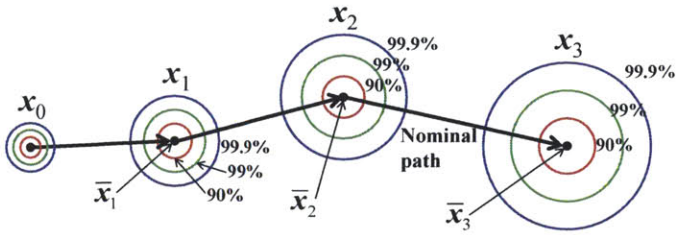


Figure 2-1: Illustration of the stochastic plant model used by p-Sulu. Note the similarity to Figure 1-3.

2.4.3 Chance-constrained qualitative state plan (CCQSP)

A qualitative state plan (QSP) [62] is a temporally flexible plan that specifies the desired evolution of the plant state. The activities of a QSP are called episodes and specify constraints on the plant state. A CCQSP, originally proposed by [17], is an extension of QSPs to stochastic plans that involve chance constraints. It is comprised of a set of events, episodes, temporal constraints, and chance constraints.

Definition 10. *A **chance-constrained qualitative state plan** (CCQSP) is a four-tuple* $P = \langle \mathcal{E}, \mathcal{A}, \mathcal{T}, \mathcal{C} \rangle$*, where* $\mathcal{E}$ *is a set of discrete events,* $\mathcal{A}$ *is a set of episodes,* $\mathcal{T}$ *is a set of simple temporal constraints, and* $\mathcal{C}$ *is a set of chance constraints.*

The four elements of a CCQSP are defined precisely in a moment. Like a QSP, a CCQSP can be illustrated diagrammatically by an acyclic directed graph in which the discrete events in $\mathcal{E}$ are represented by vertices, drawn as circles, and the episodes as arcs with ovals, as shown in Figure 1-4. A CCQSP has a start event e_0 and an end e_E, which corresponds to the beginning and the end of the mission, respectively.

For example, the CCQSP of the PTS scenario is shown in Figure 1-4. The state regions and obstacles in the CCQSP are illustrated in Figure 1-2. It involves four events: $\mathcal{E} = \{e_0, e_1, e_2, e_E\}$. Their meanings are described as follows.

1. The start event e_0 corresponds to the take off of the PAV from Provincetown.

2. The second event e_1 corresponds to the time step when PAV reaches the scenic region.

3. Event e_2 is associated with the time instant when the PAV has just left the scenic region.

4. The end event e_E corresponds to the arrival of the PAV in Bedford.

The CCQSP has four episodes $\mathcal{A} = \{a_1, a_2, a_3, a_4\}$ and two chance constraints $\mathcal{C} = \{c_1, c_2\}$.

A natural language expression of the CCQSP is:

> *" Start from Provincetown, reach the scenic region within 30 time units, and remain there for between 5 and 10 time units. Then end the flight in Bedford. The probability of failure of these activities must be less than 1%. At all times, remain in the safe region by avoiding the no-fly zones and the storm. Limit the probability of penetrating such obstacles to 0.0001%. The entire flight must take at most 60 time units."*

Below we formally define the three types of constraints - episodes, temporal constraints, and chance constraint.

Episodes Each *episode* $a \in \mathcal{A}$ specifies the desired state of the system under control over a time interval.

Definition 11. *An* ***episode*** $a = \langle e_a^S, e_a^E, \Pi_a(t_S, t_E), R_a \rangle$ *has an associated start event* e_a^S *and an end event* e_a^E. $R_a \in \mathbb{R}^N$ *is a region in a state space.* $\Pi_a \subseteq \mathbb{T}$ *is a set of time steps at which the state* x_t *must be in the region* R_a.

The feasible region R_a can be any subset of $\mathbb{R}^N$. We will approximate R_a with a set of linear constraints later in Section 3.1.1.

$\Pi_a(t_S, t_E)$ is a subset of $\mathbb{T}$ given as a function of the start time step t_S and the end time step t_E. Different forms of $\Pi_a(t_S, t_E)$ result in various types of episodes. The following three types of episodes are particularly of our interest:

1. *Start-in episode*: $\Pi_a(t_S, t_E) = \{t_S\}$

2. *End-in episode*: $\Pi_a(t_S, t_E) = \{t_E\}$

3. *Remain-in episode*: $\Pi_a(t_S, t_E) = \{t_S, t_S + 1, \cdots, t_E\}$

For a given episode a, the set of time steps at which the plant state must be in the region R_a is obtained by substituting $s(e_a^S)$ and $s(e_a^E)$, the execution time steps of the start event and the end event of the episode, into t_S and t_E. In other words, an episode a requires that the plant state is in R_a for all time steps in $\Pi_a\left(s(e_a^S), s(e_a^E)\right)$. For the rest of the thesis, we use the following abbreviated notation:

$$\Pi_a(s) := \Pi_a\left(s(e_a^S), s(e_a^E)\right).$$

Using this notation, an episode is equivalent to the following state constraint:

$$\bigwedge_{t \in \Pi_a(s)} \boldsymbol{x}_t \in R_a. \tag{2.5}$$

For example, in the CCQSP shown in Figure 1-4, there are four episodes: a_1 ("Start in [Provincetown]"), a_2 ("Remain in [Scenic region]"), a_3 ("End in Bedford"), and a_4 ("Remain in [safe region]").

In Section 7, we solve a relaxed optimization problem with a partial schedule (Definition 3) in order to obtain a lower bound of the optimal objective value. In such relaxed problems, only a subset of the episodes that are relevant to the given partial schedule are imposed. We formally define *a partial episode set* of a partial schedule σ as follows:

Definition 12. *Given a partial schedule σ, $\mathcal{A}(\sigma) \subseteq \mathcal{A}$ is its **partial episode set**, which is a subset of $\mathcal{A}$, that involves the episodes whose start event and end event are assigned execution time steps.*

$$\mathcal{A}(\sigma) = \left\{a \in \mathcal{A} \mid e_a^S \in \mathcal{E}_\sigma \wedge e_a^E \in \mathcal{E}_\sigma\right\}$$

Recall that $\mathcal{E}_\sigma$ is a set of events to which the execution time steps are assigned by the partial schedule σ (Definition 3).

Chance constraint Recall that a chance constraint is a probabilistic constraint that requires constraints to be satisfied within a user-specified probability. A CCQSP can have multiple chance constraints. A chance constraints is associated with at least one episodes.

Let $\Psi_c \subseteq \mathcal{A}$ be a set of episodes associated with the chance constraint c, and Δ_c be a user-specified risk bound.

Definition 13. *A **chance constraint** $c = \langle \Psi_c, \Delta_c \rangle$ is a constraint requiring that:*

$$\Pr\left[\bigwedge_{a \in \Psi_c} \bigwedge_{t \in \Pi_a(s)} \boldsymbol{x}_t \in R_a\right] \geq 1 - \Delta_c, \tag{2.6}$$

In the chance-constrained optimization community, 2.6 is classified as a *joint* chance constraint, meaning that it requires the satisfaction of *all* the associated constraints with a given probability. On the other hand, a *individual* chance constraint requires the satisfaction of a single constraint with a given probability. An individual chance constraint can be considered as a special case of joint chance constraint, where the chance constraint only involves one state constraint.

Every episode in a CCQSP must be associated with exactly one chance constraint. Any episode in $episodes$ must not be involved in more than one chance constraints or unassociated with any chance constraint. To put it another way, there must be a onto map from $\mathcal{A}$ to $\mathcal{C}$.

For example, the CCQSP shown in Figure 1-4 has two chance constraints, c_1 and c_2. Their associated episodes are $\Psi_{c_1} = \{a_1, a_2, a_3\}$ and $\Psi_{c_2} = \{a_4\}$. Therefore, c_1 requires that the probability of satisfying the three episodes a_1, a_2, and a_3 (colored in green) is more than than 99%, while c_2 requires that the probability of satisfying the episode a_4 is more than 99.99999%.

We place the following requirement. This requirement is necessary in order to guarantee the convexity of constraints in Section 5.2.

Requirement 1.

$$\Delta_c \leq 0.5$$

In other words, the risk bounds are required to be less than 50%. We claim that this requirement does not constrain practical applications, since typically the user of an autonomous system would not accept more than the 50% of risk.

We note that the chance constraint (2.6) includes state constraints because our state variable $\boldsymbol{x}$ is a random variable due to the stochastic uncertainty, given as (2.2), (2.3), and (2.4). Temporal constraints cannot be included in (2.6) because the schedule s is not a random variable in our problem formulation. In a scheduling problem where stochastic uncertainty in execution time is considered, the chance constraint should include temporal constraints.

Temporal constraint CCQSP includes *simple temporal constraints (STCs)* [33], which impose upper and lower bounds on the duration of episodes and on the temporal distances between two events in $\mathcal{E}$.

Definition 14. *A **simple temporal constraint** $\tau = \langle e_\tau^S, e_\tau^E, b_\tau^{\min}, b_\tau^{\max} \rangle$ is a constraint, specifying that the duration from a start event e_τ^S to an end event e_τ^E be in the real-valued interval $[b_\tau^{\min}, b_\tau^{\max}] \subseteq [0, +\infty]$.*

Temporal constraints are represented diagrammatically by arcs between nodes, labeled with the time bounds $[b_\tau^{\min}, b_\tau^{\max}]$, or by labels over episodes. For example, the CCQSP shown in Figure 1-4 has four simple temporal constraints. One requires the time between e_0 and e_1 to be at most 30 time unites. One requires the time between e_1 and e_2 to be at least 5 units and at most 10 units. One requires the time between e_2 and e_E to be at most 40 time unites. One requires the time between e_0 and e_E to be at most 60 time unites.

A schedule s is feasible if it satisfies all temporal constraints in the CCQSP, $\mathcal{T}$. The number of feasible schedules is finite, since $\mathbb{T}$ is discrete and finite. We denote by $\mathcal{S}_F$ the domain of feasible schedules, which is formally defined as follows:

$$\mathcal{S}_F = \{s \in \mathbb{T}^{|\mathcal{E}|} \mid \forall_{\tau \in \mathcal{T}} \quad b_\tau^{\min} \leq \Delta T\{s(e_\tau^E) - s(e_\tau^S)\} \leq b_\tau^{\max}\}, \tag{2.7}$$

where $|\mathcal{E}|$ is the number of events in the CCQSP. The temporal duration is multiplied by the time interval ΔT because $b_\tau^{\min}$ and $b_\tau^{\min}$ are real-valued time while s is a time step in $\mathbb{T}$. Note that we separate the notion of time step from time (see Section 2.1).

We require that a CCQSP has a temporal constraint that imposes an upper bound on the duration of the entire plan. With a slight abuse, this requirement is stated as follows:

Requirement 2. *There is $\tau \in \mathcal{T}$ such that*

$$e_\tau^S = e_0 \wedge e_\tau^E = e_E \wedge b_\tau^{\max} < +\infty.$$

We place such a requirement because otherwise the solution to the planning problem often become trivial. For example, consider a problem that require a vehicle to reach a goal with minimum fuel consumption. If the plan duration is not bounded, then the optimal solution is to stay at the start forever, assuming that the vehicle will reach the goal in the infinite future.

2.4.4 Objective function

In this section, we formally define the objective function.

Definition 15. *Given sequences of input variables $\boldsymbol{u}_{0:N-1}$ and the expected state variables $\bar{\boldsymbol{x}}_{1:N}$, and given a schedule s for a set of events $\mathcal{E}$, an **objective function** $J : \mathbb{U}^N \times \mathbb{X}^N \times \mathcal{S}_F \mapsto \mathbb{R}$ is a real-valued function over $\boldsymbol{u}_{0:N-1}$, $\bar{\boldsymbol{x}}_{1:N}$, and s.*

We assume that J is a convex function over $\bar{\boldsymbol{x}}_{1:N}$ and $\boldsymbol{u}_{0:N-1}$. Note that J is defined over the *expectation* of $\boldsymbol{x}_{1:N}$, $\bar{\boldsymbol{x}}_{1:N}$, since $\boldsymbol{x}_{0:N}$ is a random variable. Since $\bar{\boldsymbol{x}}_{1:N}$ is a function of $\boldsymbol{u}_{0:N-1}$, J can be regarded as a function of $\boldsymbol{u}_{0:N-1}$.

A typical example of an objective function is the quadratic sum of control inputs, which requires the total control efforts to be minimized:

$$J(\boldsymbol{u}_{0:N-1}, \bar{\boldsymbol{x}}_{1:N}, s) = \sum_{t=0}^{N-1} ||\boldsymbol{u}_t||^2.$$

Another example is:

$$J(\boldsymbol{u}_{0:N-1}, \bar{\boldsymbol{x}}_{1:N}, s) = s(e_E), \tag{2.8}$$

which minimizes the total plan execution time, by requiring that the end event e_E of the qualitative state plan be scheduled as soon as possible.

2.5 Definitions of outputs

The output of p-Sulu is an *optimal solution*, which consists of an optimal control sequence $\boldsymbol{u}^{\star}_{0:N-1} \in \mathbb{U}^N$ and an optimal schedule $s^{\star} \in \mathcal{S}_F$.

Definition 16. *The optimal solution is a pair* $\langle \boldsymbol{u}^{\star}_{0:N-1}, s^{\star} \rangle$. *The solution satisfies all constraints in the given CCQSP (Definition 10), the initial condition* I, *and the stochastic plant model* M. *The solution minimizes the given objective function* $J(\boldsymbol{u}_{0:N-1}, \bar{\boldsymbol{x}}_{1:N}, s)$ *(Definition 15).*

2.6 Problem Statement

We now formally define the *robust optimal CCQSP planning* problem, which is solved by p-Sulu FH.

Problem 1: Robust Optimal CCQSP Planning/Execution Problem

Given a stochastic plant model $\mathcal{M} = \langle \boldsymbol{A}_{0:N-1}, \boldsymbol{B}_{0:N-1}, \boldsymbol{\Sigma}_{w_{0:N-1}} \rangle$, an initial condition $\mathcal{I} = \langle \bar{\boldsymbol{x}}_0, \boldsymbol{\Sigma}_{\boldsymbol{x}_0} \rangle$, a CCQSP $P = \langle \mathcal{E}, \mathcal{A}, \mathcal{T}, \mathcal{C} \rangle$, and an objective function $J(\boldsymbol{u}_{0:N-1}, \bar{\boldsymbol{x}}_{1:N}, s)$, a robust optimal CCQSP planning problem is to find an optimal solution $\langle \boldsymbol{u}^{\star}_{0:N-1}, s^{\star} \rangle$ for $\mathcal{M}$, $\mathcal{I}$, P, and J.

Chapter 3

Encoding

This section encodes the CCQSP planning and execution problem, stated in the preceding chapter, as a mathematical programming problem. Recall that we develop our CCQSP executives, p-Sulu and dp-Sulu, in spirals, as explained in Section 1.6.2 and Section 1.9.2. We start from planners that solve restricted instances of the CCQSP planning problem, and then build general planners and executives upon them. In this section, we first present the problem encoding of a general CCQSP planning/execution problem with a non-convex state space and a flexible schedule (Figure 1-6-(c)) in Subsection 3.1. Then we present the encodings of the two special cases of the CCQSP planning/execution problem in Subsections 3.2 and 3.3: one with a non-convex state space and a *fixed* schedule (Figure 1-6-(b)), and one with a *convex* state space and a fixed schedule (Figure 1-6-(a)). Each problem can be solved either in a centralized or decentralized manner; each can also be solved by either a full-horizon planner or a receding horizon executive. See the matrix shown in Table 1.1, which indicates the problem that each algorithm solves.

3.1 Encoding of a CCQSP Planning/Execution Problem with a Non-convex State Space and Flexible Schedule

3.1.1 Encoding of Feasible Regions

In order to encode Problem 1 into a mathematical programming problem, the geometric constraint in (2.6), $\boldsymbol{x}_t \in R_a$, must be represented by algebraic constraints. For that purpose, we approximate the feasible state regions R_a by a set of half-spaces, each of which is represented by a linear state constraint.

Figure 3-1 shows two simple examples. The feasible region of (a) is outside of the obstacle, which is approximated by a triangle. The feasible region of (b) is inside of the pickup region, which is again approximated by a triangle. Each feasible region is approximated by a set of linear constraints as follows:

$$\text{(a)} \quad \bigvee_{i=1}^{3} \boldsymbol{h}_i^T \boldsymbol{x} \leq \boldsymbol{g}_i$$

$$\text{(b)} \quad \bigwedge_{i=1}^{3} \boldsymbol{h}_i^T \boldsymbol{x} \geq \boldsymbol{g}_i$$

We approximate the feasible regions so that the set of linear constraints is a sufficient condition of the original state constraint $\boldsymbol{x}_t \in R_a$.

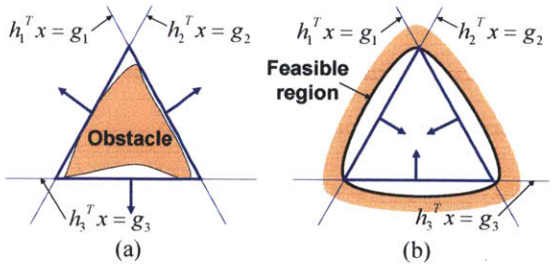


Figure 3-1: Representation of feasible regions by a set of linear constraints

We assume that the set of linear state constraints that approximates a feasible region

has been reduced to conjunctive normal form (CNF) as follows:

$$\bigwedge_{k\in\mathcal{K}_a}\bigvee_{j\in\mathcal{J}_{a,k}} \boldsymbol{h}_{a,k,j}^T\boldsymbol{x}_t - g_{a,k,j} \leq 0, \tag{3.1}$$

where $\mathcal{K}_a = \{1, 2, \cdots |\mathcal{K}_a|\}$ and $\mathcal{J}_{c,i} = \{1, 2, \cdots |\mathcal{J}_{c,i}|\}$ are sets of indices. By replacing $\boldsymbol{x}_t \in R_a$ in (2.6) by (3.1), a chance constraint c is encoded as follows:

$$\Pr\left[\bigwedge_{a\in\Psi_c}\bigwedge_{t\in\Pi_a(s)}\bigwedge_{k\in\mathcal{K}_a}\bigvee_{j\in\mathcal{J}_{a,k}} \boldsymbol{h}_{c,a,k,j}^T\boldsymbol{x}_t - g_{c,a,k,j} \leq 0\right] \geq 1 - \Delta_c, \tag{3.2}$$

where $\mathcal{K}_a = \{1, 2, \cdots |\mathcal{K}_a|\}$ and $\mathcal{J}_{c,i} = \{1, 2, \cdots |\mathcal{J}_{c,i}|\}$ are sets of indices. In order to simplify the notation, we merge indices $a \in \Psi_c$, $t \in \Pi_a(s)$, and $k \in \mathcal{K}_a$ into a new index $i \in \mathcal{I}_c(s)$, where $\mathcal{I}_c(s) = \{1, 2, \cdots |\mathcal{I}_c(s)|\}$ and $|\mathcal{I}_c(s)| = |\mathcal{K}_a| \cdot \sum_{a\in\Psi_c} |\Pi_a(s)|$. We let a_i, k_i, and t_i the indices that correspond to to the combined index i, and let $h_{c,i,j} = h_{c,a_i,k_i,j}$. Using these notations, the three conjunctions of (3.2) are combined into one, and we obtain the following encoding of a chance constraint:

$$\Pr\left[\bigwedge_{i\in\mathcal{I}_c(s)}\bigvee_{j\in\mathcal{J}_{c,i}} \boldsymbol{h}_{c,i,j}^T\boldsymbol{x}_{t_i} - g_{c,i,j} \leq 0\right] \geq 1 - \Delta_c. \tag{3.3}$$

The specification of chance constraints given in (3.3) requires that all $|\mathcal{I}_c(s)|$ disjunctive clauses of state constraints must be satisfied with a probability $1 - \Delta_c$. The i'th disjunctive clause of the c'th chance constraint is composed of $|\mathcal{J}_{c,i}|$ linear state constraints.

3.1.2 CCQSP Planning/Execution Problem Encoding

Using 3.3, a robust optimal CCQSP planning/execution problem (Problem 1) is encoded as follows:

Problem 2: General CCQSP Planning/Execution Problem

$$\min_{\boldsymbol{u}_{0:N-1}\in\mathbb{U}^N,s} \quad J(\boldsymbol{u}_{0:N-1},\bar{\boldsymbol{x}}_{1:N},s) \tag{3.4}$$

$$\text{s.t.} \quad s\in\mathcal{S}_F \tag{3.5}$$

$$\forall t\in\mathbb{T}^-, \quad \boldsymbol{x}_{t+1}=\boldsymbol{A}_t\boldsymbol{x}_t+\boldsymbol{B}_t\boldsymbol{u}_t+\boldsymbol{w}_t \tag{3.6}$$

$$\bigwedge_{c\in\mathcal{C}}\Pr\left[\bigwedge_{i\in\mathcal{I}_c(s)}\bigvee_{j\in\mathcal{J}_{c,i}}\boldsymbol{h}_{c,i,j}^T\boldsymbol{x}_{t_i}-g_{c,i,j}\le 0\right]\ge 1-\Delta_c. \tag{3.7}$$

$$\boldsymbol{x}_0\sim\mathcal{N}(\bar{\boldsymbol{x}}_0,\boldsymbol{\Sigma}_{\boldsymbol{x}_0}), \quad \boldsymbol{w}_k\sim\mathcal{N}(\boldsymbol{0},\boldsymbol{\Sigma}_{w_t}) \tag{3.8}$$

Recall that $\mathcal{S}_F$, formally defined in (2.7) , is the set of schedules that satisfy all temporal constraints in the given CCQSP. This CCQSP execution problem is a hybrid optimization problem over both discrete variables s (schedule) and continuous variables $\boldsymbol{u}_{0:N-1}$ (control sequence). Note that the temporal constraints Problem 2 is solved in Section 7.

3.2 Encoding of a CCQSP Planning/Execution Problem with a Non-convex State Space and Fixed Schedule

A restricted version of a CCQSP planning/execution problem with a *fixed* schedule is obtained by fixing s in Problem 2 as follows:

Problem 3: CCQSP Planning/Execution Problem with a Fixed Schedule

$$J^\star(s)=\min_{\boldsymbol{u}_{0:N-1}\in\mathbb{U}^N} \quad J'(\boldsymbol{u}_{0:N-1},\bar{\boldsymbol{x}}_{1:N}) \tag{3.9}$$

$$\text{s.t.} \quad \forall t\in\mathbb{T}^-, \quad \boldsymbol{x}_{t+1}=\boldsymbol{A}_t\boldsymbol{x}_t+\boldsymbol{B}_t\boldsymbol{u}_t+\boldsymbol{w}_t \tag{3.10}$$

$$\bigwedge_{c\in\mathcal{C}}\Pr\left[\bigwedge_{i\in\mathcal{I}_c(s)}\bigvee_{j\in\mathcal{J}_{c,i}}\boldsymbol{h}_{c,i,j}^T\boldsymbol{x}_{t_i}-g_{c,i,j}\le 0\right]\ge 1-\Delta_c, \tag{3.11}$$

$$\boldsymbol{x}_0\sim\mathcal{N}(\bar{\boldsymbol{x}}_0,\boldsymbol{\Sigma}_{\boldsymbol{x}_0}), \quad \boldsymbol{w}_k\sim\mathcal{N}(\boldsymbol{0},\boldsymbol{\Sigma}_{w_t}) \tag{3.12}$$

where $J^\star(s)$ is the optimal objective value of the CCQSP Planning/Execution problem with the schedule fixed to s. Note that the schedule s, which is a decision variable in Problem 2, is treated as a constant in Problem 3. Therefore, the objective function J' is now a function of only control sequence and mean state, since we have fixed the schedule. Since we assumed that J is a convex function regarding to $\boldsymbol{u}_{0:N-1}$ and $\bar{\boldsymbol{x}}_{1:N}$, J' is also a convex function. Section 6 solves Problem 3.

3.3 Encoding of a CCQSP Planning/Execution Problem with a Convex State Space and Fixed Schedule

A more restrictive version of a CCQSP planning/execution problem with a fixed schedule and a convex state space is obtained by removing the disjunctions in the chance constraints in Problem 3 as follows:

Problem 4: CCQSP Planning/Execution Problem with a Fixed Schedule and a Convex State Space

$$\min_{\boldsymbol{u}_{0:N-1}\in\mathbb{U}^N} \quad J'(\boldsymbol{u}_{0:N-1}, \bar{\boldsymbol{x}}_{1:N}) \tag{3.13}$$

$$\text{s.t.} \quad \forall t \in \mathbb{T}^-, \quad \boldsymbol{x}_{t+1} = \boldsymbol{A}_t\boldsymbol{x}_t + \boldsymbol{B}_t\boldsymbol{u}_t + \boldsymbol{w}_t \tag{3.14}$$

$$\bigwedge_{c\in\mathcal{C}} \Pr\left[\bigwedge_{i\in\mathcal{I}_c(s)} \boldsymbol{h}_{c,i}^T\boldsymbol{x}_{t_i} - g_{c,i} \le 0\right] \ge 1-\Delta_c. \tag{3.15}$$

$$\boldsymbol{x}_0 \sim \mathcal{N}(\bar{\boldsymbol{x}}_0, \boldsymbol{\Sigma}_{\boldsymbol{x}_0}), \quad \boldsymbol{w}_k \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma}_{w_t}) \tag{3.16}$$

Note that the disjunctions in Problem 3 are removed in Problem 3.3. Section 5 solves Problem 4.

3.4 Encoding of CCQSP Planning Problems with Multiple Agents

CCQSP planning problems involving multiple agents are encoded in the same formulation as Problems 2-4 by combining state and control variables of all agents into one state and control variables as follows:

$$\boldsymbol{x}_t := \begin{bmatrix} \boldsymbol{x}_t^1 \\ \boldsymbol{x}_t^2 \\ \vdots \\ \boldsymbol{x}_t^N \end{bmatrix}, \quad \boldsymbol{u}_t := \begin{bmatrix} \boldsymbol{u}_t^1 \\ \boldsymbol{u}_t^2 \\ \vdots \\ \boldsymbol{u}_t^N \end{bmatrix},$$

where $\boldsymbol{x}_t^i$ and $\boldsymbol{u}_t^i$ are the state and control variables of the ith agent at tth time step, respectively. Therefore, theoretically, any algorithms that can solve Problems 2-4 can also solve their multi-agent variants in a centralized manner. However, in practice, such centralized solutions of poorly scale to multi-agent problems with large number of agents. Moreover, centralized approach has an issue of robustness since the central controller is a single point of failure in terms of computation and communication. In order to address these issues, we develop distributed algorithms in Chapter 9 and 10.

The following chapters present algorithms that solve the CCQSP planning and execution problems defined above.

Chapter 4

Risk Allocation

This chapter presents the key concept of this thesis, risk allocation, as well as the iterative risk allocation (IRA) algorithm, a general algorithm that optimizes the allocation of risk. The concept of risk allocation can be intuitively explained by an analogy of resource allocation. When allocating resources, a limited amount of resource is allocated to individual entities. The overall cost is minimized by optimizing the allocation of the resource. Likewise, we show that risk can be allocated to individual risk factors, and that the allocation of risk needs to be optimized in order to minimize the overall cost. We then show that the IRA algorithm can improve the risk allocation through iteration.

This risk allocation approach provides a convenient quantitative tool to break down risk into multiple risk factors. Typically, one failure mode can be caused by multiple risk factors. Consider, for example, a multi-vehicle system in an environment with multiple obstacles, which conducts a mulit-day mission. A collision can occur by any agent in the system, with any obstacle in the environment, on any day during the mission. The risk of collision can be distributed among agents, obstacles, or days. Assuming there are two agents in the system, if a user wants to limit the probability of collision to 1%, she can assign 0.4% risk bound to Vehicle A and 0.6% risk bound to Vehicle B, for example. We will later show that the overall 1% risk bound is satisfied if both vehicles satisfies their own individual risk bounds. Alternatively, she can allocate the 0.4% of risk to Obstacle A and the 0.6% of risk to Obstacle B. By ensuring that the probability of each obstacle being hit by any vehicles is below its own risk bound, the overall risk bound is guaranteed to be

satisfied. She can also assign 0.4% risk bound to Day 1 and 0.6% risk bound to Day 2 (here we assume that there are two days in the mission). If the probability of collision on each day is within its individual risk bound, the overall risk bound for the two-day mission is guaranteed to be satisfied.

In order to maximize the performance of the system (i.e., minimize the system cost), the risk allocation must be optimized. For example, if Vehicle A's sensitivity to risk is higher than Vehicle B, allocating larger portion of risk to Vehicle A would result in better performance. The newly proposed IRA algorithm provides a method to automatically decide the allocation of risk to each risk factor in order to obtain better performance. The IRA algorithm can be implemented on top of existing deterministic optimizers and give them a capability to solve corresponding chance-constrained optimization problems. p-Sulu (Chapter 8) and dp-Sulu (Chapter 10) are built upon IRA. IRA is an anytime algorithm, meaning that it can return a feasible solution at any time, and the cost function value of the solution monotonically decreases through iteration. Although IRA outperforms existing algorithms in a wide range of practical problems, it does not have theoretical guarantee for optimality. On the other hand, CRA (Chapter 5), NIRA (Chapter 6), and MIRA (Chapter 9) are theoretically guaranteed to find optimal solution for specific full-horizon planning problems.[1].

Although this thesis focuses on the CCQSP planning problems, the risk allocation approach and the IRA algorithm can be applied to broader classes of problems with multiple risk factors. Hence, we consider a general joint chance-constrained optimization problems in this chapter. The application of risk allocation approach to specific CCQSP planning problems are discussed after Chapter 5.

[1]More precisely, The CRA, NIRA, and MIRA algorithms solve the deterministic approximation of chance constrained optimization problems optimally. The IRA algorithm gives a feasible, but non-optimal solution to the same deterministic approximation of chance constrained optimization problems

4.1 Risk Allocation Approach

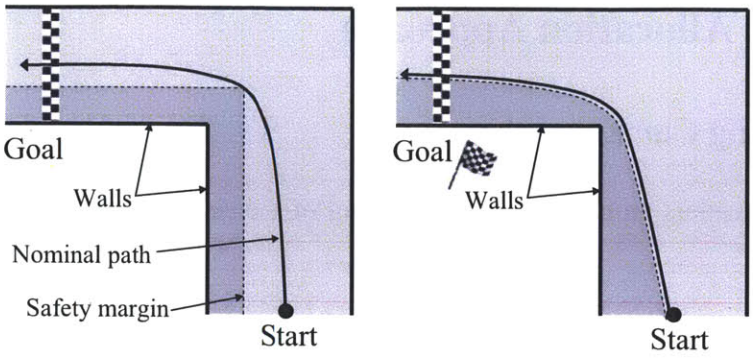
4.1.1 Racing Car Example

We first offer readers an intuitive understanding of the risk allocation approach using an example.

Consider the race car example, shown in Figure 4-1. The dynamics of the vehicle have Gaussian-distributed uncertainty. The task is to plan a path that minimizes the time to reach the finish line, with the guarantee that the probability of crashing into a wall during the race is less than 0.1% (*chance constraint*). Planning the control sequence is equivalent to planning the nominal path, which is shown by the solid lines in Figure 4-1. To limit the probability of crashing into the wall, a good driver might set a safety margin, which is colored in dark gray in Figure 4-1, and then plan the nominal path outside of the safety margin.

The driver wants to set the safety margin as small as possible to make the nominal path shorter. However, since the probability of crashing during the race is bounded, there need to be a certain lower bound on the size of the safety margin. We assume here that a lower bound on the total area of the safety margin is suitably set. Given this constraint, there are different strategies of setting a safety margin; in Figure 4-1(a) the width of the margin is uniform; in Figure 4-1(b) the safety margin is narrow around the corner, and wide at the other places.

An intelligent driver would take the strategy of (b), since he knows that going closer to the wall at the corner makes the path shorter, while doing so at the straight line does not. A key observation here is that taking risk (i.e., setting a narrow safety margin) at the corner results in a greater reward (i.e. time saving) than taking the same risk at the straight line. This gives rise to the notion of *risk allocation.* A good risk allocation strategy is to save risk when the reward is small, while taking it when the reward is great. As is illustrated in this example, the risk allocation must be optimized in order to minimize the objective function of a joint chance-constrained stochastic optimization problem.



(a) Uniform risk allocation (b) Optimal risk allocation

Figure 4-1: Risk allocation strategies on the racing car example

4.1.2 Formal Statement of the Risk Allocation Approach

General Problem Statement

We next formally present the risk allocation approach. Since risk allocation is a general approach whose application is not limited to CCQSP planning problems, in this chapter we consider a general joint chance-constrained programming problem, which is a generalization of the CCQSP planning problems stated in the previous chapter (Problems 2, 3, and 4). We construct the problem formulation by extending the following *deterministic* constrained optimization problem:

Problem 5: General Deterministic Constrained Optimization

$$\begin{aligned} \min_{u \in \mathbb{U}} \quad & p(u) \\ \text{s.t.} \quad & \bigwedge_{c \in \mathcal{C}} \bigwedge_{i \in \mathcal{I}_c} \bar{q}_{c,i}(u) \leq 0, \end{aligned} \tag{4.1}$$

where u is a deterministic decision vector, and p and $\bar{q}_{c,i}$ are deterministic real-valued functions of u. $\mathcal{C}$ is a set of constraints that are imposed on the optimization problem.

Next, we consider a stochastic version of Problem 5 where the constraint functions have

uncertainty. Let

$$q_{c,i}(u) = \bar{q}_{c,i}(u) + w_{c,i}, \tag{4.2}$$

where $w_{c,i}$ is a continuous random variable that has a zero-mean probability density function $f_{c,i}(w)$. Note that $q_{c,i}(u)$ is a random variable. The assumption that $w_{c,i}$ has a zero-mean distribution implies that the expected value of $q_{c,i}(u)$ is $\bar{q}_{c,i}(u)$:

$$\bar{q}_{c,i}(u) = \mathbb{E}\left[q_{c,i}(u)\right].$$

The general formulation of joint chance-constrained optimization can be obtained as follows:

Problem 6: General Joint Chance-Constrained Optimization

$$\min_{u \in \mathbb{U}} \quad p(u) \tag{4.3}$$

$$\text{s.t.} \quad \bigwedge_{c \in \mathcal{C}} \Pr\left[\bigwedge_{i \in \mathcal{I}_c} q_{c,i}(u) \leq 0\right] \geq 1 - \Delta_c, \tag{4.4}$$

where Δ_c is the upperbound on the probability of violating any of the constraints in the cth group.

Problem 6 is a generalization of the CCQSP Planning problems (Problems 2, 3, and 4). Note that, although we only include conjunction of constraints, the disjunctive constraints in Problems 2, 3 can be equivalently transformed into a single inequality constraint as follows:

$$\bigvee_{j \in \mathcal{J}} q_j(u) \leq 0 \Longleftrightarrow \min_{j \in \mathcal{J}} q_j(u) \leq 0.$$

The constraint on the right hand side is satisfied if at least one of the function $q_j(u)$ is less than or equal to zero; therefore, it is equivalent to a disjunctive constraint on the left. Hence, Problem 6 can be considered as a generalization of Problems 2, 3, and 4. In Section

6.1.1, we present the risk selection approach, which can handle such a disjunctive chance constraints. Also note that Problem 6 do not assume Gaussian distribution, as opposed to Problems 2, 3, and 4.

Risk Allocation Approach

Each joint chance constraint in (4.4) requires that the probability of violating *any* of multiple inequality constraints is less than the user-specified risk bound. Hence, such a chance constraint is called a *joint* chance constraint. Although the joint chance constraint is typically convex [79], evaluating its left hand side requires computing an integral of a multivariate probability distribution over an arbitrary region. Such an integral cannot be obtained in closed form. Although the sampling-based approach can compute the integral approximately, using sampling iteratively makes computation significantly slower [19].

We address this issue by decomposing the intractable joint chance constraint (4.4) into a set of *individual chance constraints*, each of which involves only a univariate probability distribution. The key feature of an individual chance constraint is that it can be transformed to an equivalent deterministic constraint, which can be evaluated analytically. As we discussed earlier, this decomposition can be considered as an allocation of risk. Through the decomposition, the risk bound of the joint chance constraint is distributed to the individual chance constraints.

More specifically, the risk allocation approach uses Boole's inequality, also known as the union bound, in order to obtain a set of individual chance constraints that is a sufficient condition of the original joint chance constraint. Let C_i be a proposition that is either true or false. Then the following lemma holds:

Lemma 1.

$$\Pr\left[\bigwedge_{i=1}^{N} C_i\right] \geq 1-\Delta \quad \Leftarrow \quad \exists \delta_i \geq 0,\ \bigwedge_{i=1}^{N} \Pr\left[C_i\right] \geq 1-\delta_i \ \wedge \ \sum_{i=1}^{N} \delta_i \leq \Delta$$

Proof:

$$\Pr\left[\bigwedge_{i=1}^{N} C_i\right] \geq 1-\Delta \;\Leftrightarrow\; \Pr\left[\bigvee_{i=1}^{N} \neg C_i\right] \leq \Delta \tag{4.5}$$

$$\Leftarrow \bigwedge_{c\in\mathcal{C}} \sum_{i=1}^{N} \Pr\left[\neg C_i\right] \leq \Delta \tag{4.6}$$

$$\Leftrightarrow \exists \delta_i \geq 0 \quad \bigwedge_{i=1}^{N} \Pr\left[\neg C_i\right] \leq \delta_i \wedge \sum_{i=1}^{N} \delta_i \leq \Delta$$

$$\Leftrightarrow \exists \delta_i \geq 0 \quad \bigwedge_{i=1}^{N} \Pr\left[C_i\right] \geq 1-\delta_i \wedge \sum_{i=1}^{N} \delta_i \leq \Delta. \tag{4.7}$$

We denote by $\neg C$ the negation of C. We use the following Boole's inequality to obtain (4.6) from (4.5):

$$\Pr\left[\bigvee_{i=1}^{N} C_{c,i}\right] \leq \sum_{i=1}^{N} \Pr[C_{c,i}].$$

■

The following result immediately follows from Lemma 1 by substituting an inequality constraint $q_{c,i}(u) \leq 0$ for C_i for each chance constraint c.

Corollary 1. *The following set of constraints is a sufficient condition of the joint chance constraint (4.4) in Problem 6:*

$$\exists \delta_{c,i} \geq 0 \quad \bigwedge_{c\in\mathcal{C}} \left\{ \bigwedge_{i\in\mathcal{I}_c} \Pr\left[q_{c,i}(u) \leq 0\right] \geq 1-\delta_{c,i} \;\wedge\; \sum_{i\in\mathcal{I}_c} \delta_{c,i} \leq \Delta_c \right\} \tag{4.8}$$

The newly introduced variables $\delta_{c,i}$ represent the upper bounds on the probability of violating each atomic constraint. We refer to them as *individual risk bounds*. The individual risk bound $\delta_{c,i}$ can be viewed as the amount of risk *allocated* to the i constraint. The fact that $\delta_{c,i}$ is a bound on probability implies that $0 \leq \delta_{c,i} \leq 1$. The second term of (4.8) requires that the total amount of risk is upper-bounded to the original risk bound Δ_c. Here we find an analogue to the resource allocation problem, where the allocation of a resource is optimized with an upper bound on the total amount of resource. Likewise, the allocation of risk $\delta_{c,i}$ must be optimized in order to minimize the cost. Therefore, we call this decomposition method a *risk allocation* [78].

Note that the set of constraints (4.8) is a *sufficient* condition for the original chance constraint (4.4). Therefore, any solution that satisfies (4.8) is guaranteed to satisfy (4.4). Furthermore, although (4.8) is a conservative approximation of (4.4), the conservatism introduced by risk allocation is typically small for practical applications. Section 4.1.5 explains the reason for the smallness of the conservatism. Our claim is validated by extensive empirical results presented in Section 11.

4.1.3 Conversion to Deterministic Constraints

Each individual chance constraint in (4.8) only involves a single inequality constraint, where $q_{c,i}(u)$ has a univariate probability distribution. Such an individual chance constraint can be transformed to an equivalent deterministic constraint.

We need some additional setup to proceed. We denote the cumulative distribution function of $w_{c,i}$ by $F_{c,i}(\cdot)$:

$$F_{c,i}(x) := \Pr[w_{c,i} \leq x] = \int_{-\infty}^{x} f_{c,i}(x')dx'.$$

We also denote by $F_{c,i}^{-1}(\delta)$ the inverse function of the cumulative distribution function:

$$F_{c,i}(x) = \delta \Longleftrightarrow F_{c,i}^{-1}(\delta) = x$$

Given these notations, the following lemma transforms an individual chance constraint into an equivalent deterministic constraint that involves the mean $\bar{q}_{c,i}(u)$, instead of the random variables $q_{c,i}(u)$:

Lemma 2. *The following two conditions are equivalent.*

$$\Pr\left[q_{c,i}(u) \leq 0\right] \geq 1 - \delta_{c,i} \quad \Leftrightarrow \quad \bar{q}_{c,i}(u) \leq -F_{c,i}^{-1}(1 - \delta_{c,i})$$

Proof:

$$\begin{aligned}
\Pr\left[q_{c,i}(u) \leq 0\right] \geq 1 - \delta_{c,i} \quad &\Leftrightarrow \quad \Pr\left[\bar{q}_{c,i}(u) + w_{c,i}(u) \leq 0\right] \geq 1 - \delta_{c,i} \\
&\Leftrightarrow \quad \Pr\left[w_{c,i}(u) \leq -\bar{q}_{c,i}(u)\right] \geq 1 - \delta_{c,i} \\
&\Leftrightarrow \quad F_{c,i}(-\bar{q}_{c,i}(u)) \geq 1 - \delta_{c,i} \\
&\Leftrightarrow \quad \bar{q}_{c,i}(u) \leq -F_{c,i}^{-1}(1 - \delta_{c,i})
\end{aligned}$$

The last equivalence is derived from the fact that a cumulative distribution function is always a non-decreasing function. ■

Figure 4-2 provides an intuition of this proof. The integral of the probability distribution function of $q_{c,i}(u)$ above zero represents the probability of constraint violation. In order to satisfy the individual chance constraint in (4.8), this area must be less than $\delta_{c,i}$. Such a condition can be met by leaving a margin between the mean and the constraint boundary, and the necessary width of the margin is $F_{c,i}^{-1}(1 - \delta_{c,i})$. This margin $F_{c,i}^{-1}(1 - \delta_{c,i})$ corresponds

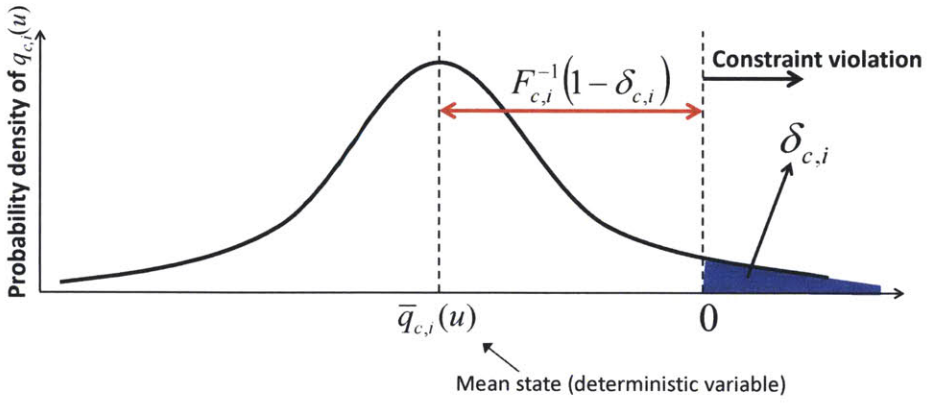


Figure 4-2: Transformation of an individual chance constraint into an deterministic constraint.

to the safety margin used in the Race Car Example in Section 4.1.1.

4.1.4 Deterministic Approximation of the Joint Chance-constrained Optimization Problem

We obtain a tractable approximation of the joint chance-constrained optimization problem (Problem 6) by replacing the joint chance constraints (4.4) with deterministic constraints using Corollary 1 and Lemma 2. We denote by δ the vector that contains risk allocations to all constraints:

$$\delta := \left[\delta_{1,1}, \delta_{1,2}, \cdots \delta_{|\mathcal{C}|,|\mathcal{I}_{|\mathcal{C}|}|}\right]$$

Let $\succ$ be the componentwise inequality. Then, the approximation of the joint-chance constraint optimization problem is given as follows:

Problem 7: Deterministic Approximation of Problem 6

$$\min_{u\in\mathbb{U},\delta\succ 0} \quad p(u) \tag{4.9}$$

$$\text{s.t.} \quad \bigwedge_{c\in\mathcal{C}} \bigwedge_{i\in\mathcal{I}_c} \bar{q}_{c,i}(u) \leq -F_{c,i}^{-1}(1-\delta_{c,i}) \tag{4.10}$$

$$\bigwedge_{c\in\mathcal{C}} \sum_{i\in\mathcal{I}_c} \delta_{c,i} \leq \Delta_c \tag{4.11}$$

It follows immediately from Corollaries 1 and 2 that a feasible solution to Problem 7 is always a feasible solution to Problem 6. When $p(u)$ and $\bar{q}_{c,i}(u)$ are convex functions, this approach corresponds to the Bernstein approximations [79].

Example

Consider the following joint chance-constrained optimization problem with two decision variables, $u_1, u_2 \in \mathbb{R}$, and two random variables w_1, w_2:

$$\min_{u_1,u_2\in\mathbb{R}} \quad u_1 + u_2$$

$$\text{s.t.} \quad \Pr\left[u_1 + w_1 \geq 0 \ \wedge \ u_2 + w_2 \geq 0\right] \geq 1 - \Delta.$$

We assume that the random variables, w_1 and w_2, have the standard Gaussian distribution:

$$w_1, w_2 \sim \mathcal{N}(0, 1).$$

The inverse of the cumulative distribution function of the standard Gaussian distribution is given as follows:

$$F^{-1}(\delta) = \sqrt{2}\,\mathrm{erf}^{-1}(2\delta - 1),$$

where $\mathrm{erf}^{-1}(\cdot)$ is the inverse of the Gauss error function. Using this F^{-1}, the joint chance-constrained optimization problem above is approximated by the following deterministic optimization problem:

$$\begin{aligned} \min_{u_1,u_2\in\mathbb{R},\delta_1,\delta_2\geq 0} \quad & u_1 + u_2 \\ \text{s.t.} \quad & u_1 \geq -F^{-1}(\delta_1) \ \wedge \ u_2 \geq -F^{-1}(\delta_2) \\ & \delta_1 + \delta_2 \leq \Delta. \end{aligned}$$

4.1.5 Conservatism of Risk Allocation Approach

As mentioned in Section 4.1.2, the risk allocation approach gives a conservative approximation (4.8) of the original chance constraint (4.4). This subsection evaluates the level of conservatism of the risk allocation approach.

Let P_{fail} be the true probability of failure, defined as the probability that a solution violates the constraints (4.1). Since (4.8) is a sufficient but not necessary condition for (4.4), P_{fail} is smaller than or equal to the risk bound Δ in general: $\Delta \geq P_{fail}$. Hence, the conservatism introduced by risk allocation is represented as

$$\Delta - P_{fail}.$$

The best-case scenario for the risk allocation approach is when the violations of all constraints are mutually exclusive, meaning that a solution that violates one constraint always satisfies all the other constraints. In that case, (4.8) becomes a *necessary and* sufficient

condition for (4.4) since

$$P_{fail} = \Pr\left[\bigwedge_{i\in\mathcal{I}_c} q_{c,i}(u) \leq 0\right] = \sum_{i\in\mathcal{I}_c} \Pr\left[q_{c,i}(u) \leq 0\right],$$

and hence, risk allocation does not involve any conservatism. If the chance constraints are active for the optimal solution, then

$$\Delta - P_{fail} = 0.$$

On the other hand, the worst-case scenario is when all constraints are equivalent, meaning that a solution that violates one constraint always violates all the other constraints. In such a case,

$$P_{fail} = \Pr\left[\bigwedge_{i\in\mathcal{I}_c} q_{c,i}(u) \leq 0\right] = \frac{1}{N}\sum_{i\in\mathcal{I}_c} \Pr\left[q_{c,i}(u) \leq 0\right],$$

where N is the number of constraints. Therefore, the worst-case conservatism is:

$$\Delta - P_{fail} = \frac{N-1}{N}\Delta.$$

Most practical problems lie somewhere between the best-case scenario and the worst-case scenario, but typically closer to the best-case than to the worst-case scenario. For example, if there are two separate obstacles in a path planning problem, collisions with the two obstacles are mutually exclusive events. Collision with an obstacle at one time step does not usually imply collisions at other time steps. A rough approximation of such a real-world situation is to assume that the satisfaction of constraints are probabilistically independent. With such an assumption, the true probability of failure is:

$$P_{fail} = \Pr\left[\bigwedge_{i\in\mathcal{I}_c} q_{c,i}(u) \leq 0\right] = \prod_{i\in\mathcal{I}_c} \Pr\left[q_{c,i}(u) \leq 0\right] \leq 1 - \prod_{i\in\mathcal{I}_c}(1-\delta_i).$$

In such a case, the risk bound of the risk allocation approach (4.8) gives the first-order approximation of the true probability of failure around $\delta_i = 0$. Also note that $\delta_i \leq \Delta$.

Therefore, the conservatism introduced by risk allocation is at the second order of Δ:

$$\Delta - P_{fail} \sim \mathcal{O}(\Delta^2).$$

For example, if $\Delta = 1\%$, the true probability of failure is approximately $P_{fail} \sim 0.99\%$. In most practical cases, the users prefer to set very small risk bounds, typically less than 1%. In such cases, the conservatism introduced by risk allocation becomes very small.

Comparison with existing bounding approaches

As far as we know, there are three existing approaches that can solve optimization problems with joint chance constraint (4.4): Particle Control, the fixed risk allocation approach, and the ellipsoid relaxation approach. We claim that the risk allocation approach has the least conservatism among the algorithms that guarantee satisfaction of chance constraints.

Particle Control [20], or a scenario approach [25], is a sampling-based method, which uses a finite number of samples to approximate the joint chance constraints in (4.4). Although this approach converges to the exact solution as the number of samples approaches infinity, it becomes intractable with only 100 to 1,000 samples in many practical cases. With a finite number of samples, the solution of Particle Control is not guaranteed to satisfy chance constraints. On the other hand, the solution of the risk allocation approach is guaranteed to satisfy chance constraints. This claim is empirically validated in Sections 11.1.5, 11.1.6, and 11.3.1.

Although **the Fixed risk allocation approach** employed by [18] and [75] is similar to the risk allocation approach in that it uses Boole's inequality to decompose joint chance constraints, it does *not* optimize risk allocation. Instead, it fixes the risk allocation to a uniform value. Hence, although its solution is guaranteed to satisfy chance constraints, it is more conservative than the risk allocation approach. The cost function values of the solutions of the fixed risk allocation approach are always higher than or equal to that of the risk allocation approach. This claim is empirically validated in Section 11.1.7.

Finally, **the elliptical relaxation approach**, developed by [97], turns joint chance constraints into deterministic constraints using a conservative ellipsoidal relaxation. Like the

risk allocation approach, the elliptical relaxation approach guarantees the satisfaction of chance constraints. Theoretically, this approach can result in less conservative solution than the risk allocation approach, particularly when state constraints are very tight. However, in most practical cases, it ends up with a significantly more conservative solution than the risk allocation approach. For example, in the simulations in Section 11.1.5 where $\Delta = 5\%$, the risk allocation approach resulted in a probability of failure $P_{fail} = 3.78\%$, while the elliptical relaxation approach results in $P_{fail} < 0.001\%$.

Figure 4-3 explains the reason behind this effect. In the example in the figure, two linear constraints on a 2-D variable u are considered: $h_A u \leq g_A$ and $h_B u \leq g_B$. The chance constraint (4.4) is equivalent to

$$\Pr[A \cup B] \leq \Delta,$$

where A and B correspond to the regions shown in the figure.

The risk allocation approach (4.8) imposes a conservative constraint:

$$\Pr[A] + \Pr[B] \leq \Delta.$$

Hence, the conservatism of risk allocation is represented by:

$$\Delta - P_{fail} = \Pr[A \cap B]. \tag{4.12}$$

On the other hand, the elliptical relaxation approach finds an ellipsoid E, as shown in Figure 4-3, that satisfies:

$$\Pr[E] = 1 - \Delta,$$

and requires E to be in the feasible state space. Therefore, the conservatism of the elliptical relaxation approach is represented by:

$$\Delta - P_{fail} = \Pr\left[\neg\left(A \cup B \cup E\right)\right], \tag{4.13}$$

as shown by the shadowed region in Figure 4-3-(b).

As can be seen from Figure 4-3, (4.13) typically results in larger probability than (4.12), particularly when the dimension of the decision variable is high. A typical CCQSP planning problem involves at least 10 time steps with two-dimensional control variables for each. Hence, the dimension of the decision variable is at least 20. This accounts for the significantly greater conservatism of the elliptical relaxation approach compared to the risk allocation approach. This claim is empirically validated in Sections 11.1.5, 11.1.6, and 11.3.1.

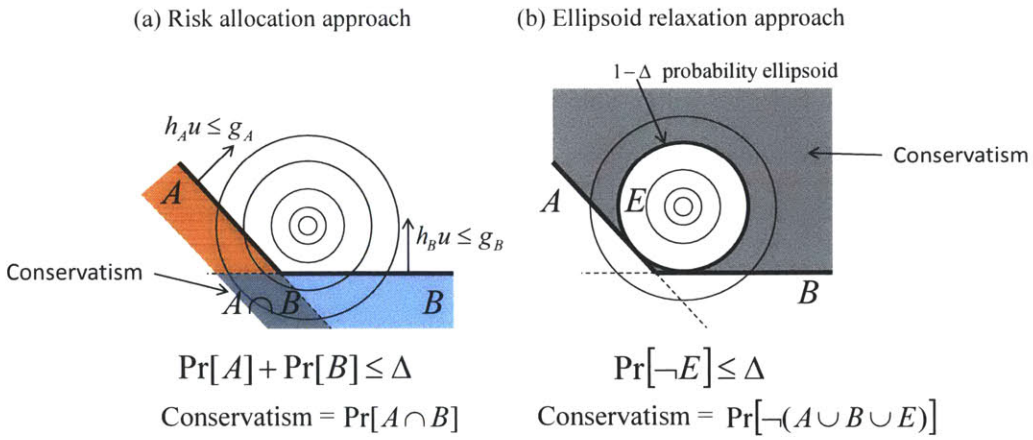


Figure 4-3: Conservatism of the risk allocation approach and the elliptical relaxation approach. Typically, the latter results in a significantly more conservative solution than the former.

4.2 Iterative Risk Allocation Algorithm

In this section we develop the iterative risk allocation (IRA) algorithm, which can solve Problem 7 in Section 4.1.4 by iteratively solving the corresponding deterministic problem (Problem 5) while improving the risk allocation at each iterative. Hence, the IRA algorithm can be implemented on top of any existing deterministic solvers and turn them into a joint chance-constrained programming solvers. This feature is important since Problem 7 is often hard to solve directly due to the nonlinearity of $F_{c,i}^{-1}(\cdot)$ (often it cannot be evaluated in a closed form). Although the IRA algorithm does not provide a guarantee of the optimality of the solution, the iterations typically converges quickly.

The IRA algorithm requires the following three assumptions:

- **Assumption 4-1**: The corresponding deterministic optimization problem (Problem 5), which shares the same function $p(u)$ and $\bar{q}_{c,i}(u)$, can be solved optimally
- **Assumption 4-2** $F_{c,i}^{-1}(\cdot)$ can be evaluated
- **Assumption 4-3** A feasible initial risk allocation δ^1 is provided

Assumption 4-2 does not require that a closed-form expression is available for $F_{c,i}^{-1}(\cdot)$. Numerical evaluation using a look-up table is enough to satisfy the assumption. The algorithm starts from the initial risk allocation, and improves it over iterations.

The algorithms presented in Chapters 5-7 and 9 give direct solutions for specific types of problems. Such algorithms give a strictly optimal solution to Problem 7, but often have slower convergence. Therefore, IRA is suitable for real-time execution or receding horizon control, where computation time is more critical than optimality.

4.2.1 Race Car Example

The IRA algorithm starts from an arbitrary feasible risk allocation, such as the one in Figure 4-1-(a), and improves the risk allocation through iterations until the path coincides the boundary of the safety margin as in Figure 4-1-(b). We first offer readers the intuition behind the algorithm using the race car example, as shown in Figure 4-4.

First, IRA sets the safety margins according to the current risk allocation. (Recall that the width of the margin for each constraint is given by $F_{c,i}^{-1}(\delta_{c,i})$.) A deterministic solver is used to obtain the optimal nominal path that does not violate the safety margin, as shown in Figure 4-4-(a). Observe that the boundary of the safety margin touches the nominal path only at the corner. In other words, the constraint at the corner is active while all other constraints are inactive. (Precisely speaking, the constraint is active at the time step when the vehicle passes at the corner, and it is inactive at all other time steps.) This means that risk is redundant at the inactive constraints, while the path may be improved by allocating more risk for the active constraint.

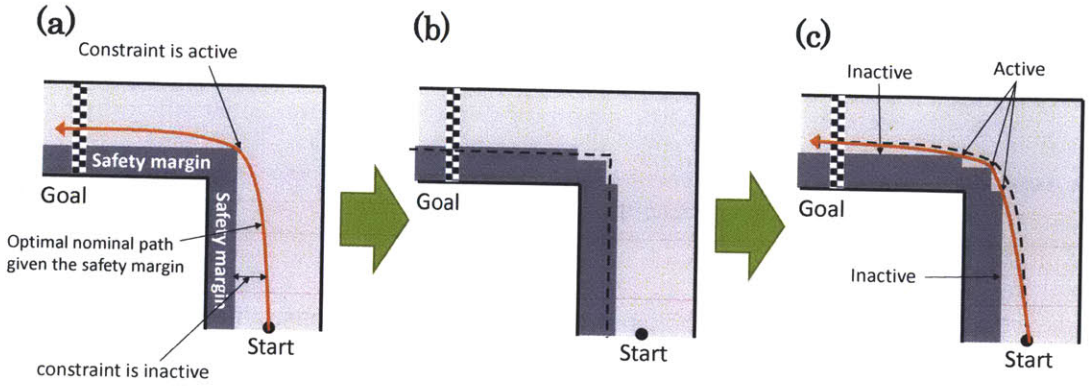


Figure 4-4: Intuitive explanation of the iterative risk allocation (IRA) algorithm. Risk is reallocated from inactive constraints to active constraints at each iteration.

To improve cost, IRA removes the risk from the inactive constraints, and reallocates it to the active constraint. Note that reducing risk allocation results in a wider safety margin, while increasing risk allocation results in the opposite. Thus, the new risk allocation results in the safety margin shown in Figure 4-4-(b). The algorithm then calls the deterministic solver again to obtain the optimal feasible path that does not violate the new safety margin, as shown in Figure 4-4-(c). Note that the new nominal path is shorter than the previous path. With the updated path and the safety margin, three constraints are active. In the next iteration, the algorithm reallocates the risk again from the inactive constraints to the three active constraints. It repeats this process until all constraints become active. It also terminates if all constraints are inactive. Note that the path length monotonically decreases through the iterations.

4.2.2 Algorithm

Algorithm 1 presents the IRA algorithm. It is essentially a descent algorithm that updates the risk allocation, $\delta := \left[\delta_{1,1}, \delta_{1,2}, \cdots \delta_{|\mathcal{C}|,|\mathcal{I}_{|\mathcal{C}|}|}\right]$, through iterations. The algorithm requires solving an inner-loop optimization problem, denoted by $\mathcal{P}_8(\delta^k)$. The formulation of $\mathcal{P}_8(\delta^k)$ is given in Problem 8.

In Line 1, the algorithm is initialized with a known feasible risk allocation δ^1 provided by Assumption 4-3. At each iteration, the following Problem 8 is solved with a *fixed* risk

Algorithm 1 Iterative Risk Allocation (IRA)

1: Set feasible risk allocation $\delta_{c,i}^1$ so that $\forall_{c\in\mathcal{C}} \sum_{i\in\mathcal{I}_c} \delta_{c,i}^1 = \Delta_c$
2: $p^0 \leftarrow \infty$
3: $k \leftarrow 1$
4: **repeat**
5: $u^k \leftarrow$ optimal solution of $\mathcal{P}_8(\delta^k)$
6: $p^k \leftarrow p(u^k)$
7: **for all** $c \in \mathcal{C}$ **do**
8: $N_c \leftarrow$ number of active constraints in the cth chance constraint
9: **if** $0 < N_c < |\mathcal{I}_c|$ **then**
10: $\delta_{residual} \leftarrow 0$
11: **for all** i such that the constraint with index (c,i) is inactive **do**
12: $\delta_{c,i}^{k+1} \leftarrow \alpha\delta_{c,i}^k + (1-\alpha)\left\{1 - F_{c,i}(-\bar{q}_{c,i}(u^k))\right\}$
13: $\delta_{residual} \leftarrow \delta_{residual} + (\delta_{c,i}^k - \delta_{c,i}^{k+1})$
14: **end for**
15: **for all** i such that the constraint with index (c,i) is active **do**
16: $\delta_{c,i}^{k+1} \leftarrow \delta_{c,i}^k + \delta_{residual}/N_c$
17: **end for**
18: **end if**
19: **end for**
20: $k \leftarrow k+1$
21: **until** $|p^k - p^{k-1}| < \epsilon$ **or** $\forall_{c\in\mathcal{C}}\ N_c = 0$ **or** $\forall_{c\in\mathcal{C}}\ N_c = |\mathcal{I}_c|$

allocation δ^k (Line 5).

Problem 8: $\mathcal{P}_8(\delta)$ - Inner-loop optimization

$$\begin{aligned} \min_{u\in\mathbb{U}} \quad & p(u) \\ \text{s.t.} \quad & \bigwedge_{c\in\mathcal{C}} \bigwedge_{i\in\mathcal{I}_c} \bar{q}_{c,i}(u) \leq -F_{c,i}^{-1}(1-\delta_{c,i}) \end{aligned}$$

Problem 8 is essentially the same as Problem 5 since $F_{c,i}^{-1}(1-\delta_{c,i})$ is a constant with a fixed $\delta_{c,i}$. The constant right hand side can be incorporated in $\bar{q}_{c,i}(u)$ without increasing the complexity of the optimization problem. Since we assume that there is a deterministic solver that can optimally solve Problem 5, Problem 8 can also be solved optimally by the same solver.

The optimal objective function value is stored in p^k in Line 6. Then, the algorithm

reallocates risk from inactive constraints to active constraints in Lines 7-19. A constraint with index (c, i) is considered as active if the following condition holds for a small constant ϵ:

$$\left|\bar{q}_{c,i}(u) + F_{c,i}^{-1}(1 - \delta_{c,i})\right| \leq \epsilon.$$

The algorithm first reduces the risk allocation to the inactive constraints. Note that the current risk allocation to the constraint (4.10) with the index (c, i) is $\delta_{c,i}^k$, while the actual probability that u^k violates the constraint is $1 - F_{c,i}(-\bar{q}_{c,i}(u^k))$, which represents the amount of risk actually needed. In Line 12, the algorithm chooses the new risk allocation $\delta_{c,i}^{k+1}$ so that

$$1 - F_{c,i}(-\bar{q}_{c,i}(u^k)) < \delta_{c,i}^{k+1} < \delta_{c,i}^k,$$

with an interpolation coefficient $0 < \alpha < 1$. In Line 13, the algorithm deposits the amount of risk removed from the inactive constraints in $\delta_{residual}$. Then algorithm reallocates the amount of risk saved in $\delta_{residual}$ to the active constraints. It splits the deposit of risk equally to the N_c active constraints in Line 16. By going through one iteration, the risk allocation is updated from δ^k to δ^{k+1}.

Since the IRA algorithm improves risk allocation by reallocating risk from inactive constraints to active constraints, it cannot work if all the constraints are active, or all the constraints are inactive. The algorithm is terminated at Line 21 in such cases. If all the constraints are inactive, the solution is in fact an optimal solution. Having all constraints active is a necessary condition for optimality, although not sufficient.

4.2.3 Recursive Feasibility and Monotonicity

The algorithm has two issues to be addressed. First, it is not obvious that $\mathcal{P}_8(\delta^k)$ has a feasible solution for all iterations. Second, we need a guarantee that this iterative process improves the risk allocation. The next lemma is the key to address these two issues.

Lemma 3. *If $\mathcal{P}_8(\delta^k)$ is feasible, then $\mathcal{P}_8(\delta^{k+1})$ is also feasible.*

Intuitively, we prove the lemma by showing that the nominal path in Figure 4-4-(a) does not violate the updated safety margin in Figure 4-4-(b).

Proof: Let $u^\star$ be a feasible solution for $\mathcal{P}_8(\delta^k)$. We prove that $u^\star$ is also a feasible solution to $\mathcal{P}_8(\delta^{k+1})$.

Let $\mathcal{I}_A$ be the set of index (c,i) where the constraint (4.14) is active in $\mathcal{P}_8(\delta^k)$, and $\mathcal{I}_I$ be a set of index (c,i) where the constraint is inactive in $\mathcal{P}_8(\delta^k)$.

As for $(c,i) \in \mathcal{I}_A$, Line 16 guarantees that $\delta_{c,i}^{k+1} \geq \delta_{c,i}^k$. Hence, $u^\star$ satisfies the constraint (4.14) of $\mathcal{P}_8(\delta^{k+1})$ as well because

$$\bar{q}_{c,i}(u^k) \leq -F_{c,i}^{-1}(1-\delta_{c,i}^k) \leq -F_{c,i}^{-1}(1-\delta_{c,i}^{k+1}).$$

Note that the inverse cumulative distribution function $F_{c,i}^{-1}$ is a monotonically increasing function.

As for $(c,i) \in \mathcal{I}_I$, since $u^\star$ satisfies the constraint (4.14) in $\mathcal{P}_8(\delta_{c,i}^k)$,

$$\bar{q}_{c,i}(u) \leq -F_{c,i}^{-1}(1-\delta_{c,i}^k) \Longleftrightarrow \delta_{c,i}^k \geq 1 - F_{c,i}(-\bar{q}_{c,i}(u^k)).$$

Since $\delta_{c,i}^{k+1}$ is an interpolation of $\delta_{c,i}^k$ and $1 - F_{c,i}(-\bar{q}_{c,i}(u^k))$,

$$\begin{aligned}
& \delta_{c,i}^{k+1} \geq 1 - F_{c,i}(-\bar{q}_{c,i}(u^k)) \\
\Longleftrightarrow \quad & F_{c,i}(-\bar{q}_{c,i}(u^k)) \geq 1 - \delta_{c,i}^{k+1} \\
\Longleftrightarrow \quad & F_{c,i}^{-1}\left(F_{c,i}(-\bar{q}_{c,i}(u^k))\right) \geq F_{c,i}^{-1}(1-\delta_{c,i}^{k+1}) \\
\Longleftrightarrow \quad & \bar{q}_{c,i}(u^k) \leq -F_{c,i}^{-1}(1-\delta_{c,i}^{k+1}),
\end{aligned}$$

hence the $u^\star$ satisfies the constraint (4.14) of $\mathcal{P}_8(\delta^{k+1})$ as well.

Therefore, $u^\star$ satisfies all constraints, and hence is a feasible solution to $\mathcal{P}_8(\delta^{k+1})$. ■

It immediately follows from the lemma that $\mathcal{P}_8(\delta^k)$ has a feasible solution for all iterations, since we assume that the initial risk allocation δ^1 is known to be feasible (Assumption 4-3). The following theorem, which guarantees that the risk allocation is improved by each iteration of the IRA algorithm, is also derived from Lemma 3.

Theorem 1. *$p^k \leq p^{k-1}$ for all $k \geq 1$*

Proof: Let u^k and u^{k+1} be optimal solutions to $\mathcal{P}_8(\delta^k)$ and $\mathcal{P}_8(\delta^{k+1})$, respectively.

Lemma 3 guarantees that u^k is a feasible solution to $\mathcal{P}_8(\delta^{k+1})$. Since u^{k+1} is an optimal solution to the same problem, its objective function value with u^{k+1} is less than or equal to the objective function value with u^k. Hence,

$$p^{k+1} = p(u^{k+1}) \leq p(u^k) = p^k$$

■

In sum, given a feasible initial risk allocation, the IRA algorithm generates a sequence of risk allocations that monotonically decreases the objective function value. In other words, IRA is an anytime algorithm.

4.3 Conclusion

We presented the risk allocation approach, which enables an efficient solution of general joint chance-constrained optimization problems. The key idea was to distribute risk among individual constraints in order to obtain a tractable deterministic approximation of the original problem. All planners and executives presented in this thesis are built upon the risk allocation approach. We also developed the IRA algorithm, a general algorithm that optimizes risk allocation. IRA can be implemented on top of any existing deterministic solvers and turns them into a joint chance-constrained programming solvers. IRA is a key building component of the p-Sulu and dp-Sulu plan executives, presented in Chapters 8 and 10. The simulation results of IRA are presented in Sections 11.1.5 and 11.3.1.

In the next chapter, we deploy the risk allocation approach on a CCQSP planning problem with fixed schedule and convex state constraints.

Chapter 5

CCQSP Planning with a Convex State Space and a Fixed Schedule

This chapter presents the convex risk allocation (CRA) algorithm, which solves the CCQSP planning problem with a convex state space and a fixed schedule (Problem 4). When there are no obstacles in the environment and the execution time steps to achieve temporally extended goals are fixed, the CCQSP planning problem is reduced to a convex chance-constrained finite-horizon optimal control problem. The application of the risk allocation approach to such a convex chance-constrained finite-horizon optimal control problem results in a deterministic convex optimization problem. The CRA algorithm solves the convex optimization problem using a off-the-shelf solver, such as SNOPT. The advantage of this algorithm compared to the IRA algorithm is that it has a theoretical guarantee of optimality.

5.1 Convex Deterministic Approximation of Problem 4

The CRA algorithm decomposes the joint chance-constraint (3.15) by using the risk allocation approach introduced in Section 4.1.2. The following result immediately follows from Lemma 1 by substituting a linear constraint $\boldsymbol{h}_{c,i}^T \boldsymbol{x}_{t_i} - g_{c,i} \leq 0$ for C_i for each chance constraint c.

Corollary 2. *The following set of constraints is a sufficient condition of the joint chance*

constraint (3.15) in Problem 4:

$$\exists \delta_{c,i} \geq 0 \quad \bigwedge_{c \in \mathcal{C}} \left\{ \bigwedge_{i \in \mathcal{I}_c(s)} \Pr\left[\boldsymbol{h}_{c,i}^T \boldsymbol{x}_{t_i} - g_{c,i} \leq 0 \right] \geq 1 - \delta_{c,i} \wedge \sum_{i \in \mathcal{I}_c(s)} \delta_{c,i} \leq \Delta_c \right\} \tag{5.1}$$

Recall that the newly added variable $0 \leq \delta_{c,i} \leq 1$ represents the risk allocated to the state constraint specified by the index (c, i). Also recall that the vector of risk allocations is denoted by δ:

$$\delta := \left[\delta_{1,1}, \delta_{1,2}, \cdots \delta_{|\mathcal{C}|,|\mathcal{I}_c(s)|} \right]$$

The set of constraints (5.1) is a sufficient condition for the original chance constraint (3.15). Therefore, any solution that satisfies (5.1) is guaranteed to satisfy (3.15).

5.1.1 Conversion to Deterministic Constraints

The CRA algorithm applies Lemma 2 to convert the individual chance constraints in (5.1) to deterministic constraints. Since Problem 4 assumes linear state constraints and Gaussian-distributed uncertainty, the specific form of the deterministic approximation can be obtained.

The application of Lemma 2 to the the individual chance constraints in (5.1) immediately results in the following Corollary:

Corollary 3.

$$\Pr\left[\boldsymbol{h}_{c,i}^T \boldsymbol{x}_{t_i} - g_{c,i} \leq 0 \right] \geq 1 - \delta_{c,i} \quad \Leftrightarrow \quad \boldsymbol{h}_{c,i}^T \bar{\boldsymbol{x}}_{t_i} - g_{c,i} \leq -m_{c,i}(\delta_{c,i}) \tag{5.2}$$

where

$$m_{c,i}(\delta_{c,i}) = F_{c,i}^{-1}(1 - \delta_{c,i}) = -\sqrt{2 \boldsymbol{h}_{c,i}^T \Sigma_{x,t_i} \boldsymbol{h}_{c,i}}\, \mathrm{erf}^{-1}(2\delta_{c,i} - 1). \tag{5.3}$$

Note that erf^{-1} is the inverse of the Gauss error function and Σ_{x,t_i} is the covariance matrix of $\boldsymbol{x}_{t_i}$. Intuitively, $m_{c,i}(\delta_{c,i})$ represents the width of the safety margin in the Race Car Example in Figure 4-1.

5.2 Convex Programming Solution to Problem 4

Using Corollaries 2 and 3, the CRA algorithm replaces the stochastic optimization problem, Problem 4, with the deterministic convex optimization problem as follows:

Problem 9: Deterministic Approximation of Problem 4

$$\min_{\boldsymbol{u}_{1:N}\in\mathbb{U}^N,\delta\succ 0} \quad J'(\boldsymbol{u}_{1:N},\bar{\boldsymbol{x}}_{1:N}) \tag{5.4}$$

$$\text{s.t.} \quad \forall t\in\mathbb{T}^-,\quad \bar{\boldsymbol{x}}_{t+1}=\boldsymbol{A}_t\bar{\boldsymbol{x}}_t+\boldsymbol{B}_t\boldsymbol{u}_t \tag{5.5}$$

$$\bigwedge_{c\in\mathcal{C}}\left\{\bigwedge_{i\in\mathcal{I}_c(s)}\boldsymbol{h}_{c,i}^T\bar{\boldsymbol{x}}_{t_i}-g_{c,i}\le -m_{c,i}(\delta_{c,i})\ \wedge\ \sum_{i\in\mathcal{I}_c(s)}\delta_{c,i}\le\Delta_c\right\}. \tag{5.6}$$

It follows immediately from Corollaries 2 and 3 that an optimal solution to Problem 9 is always a feasible solution to Problem 4. Furthermore, [19] showed that an optimal solution to Problem 9 is a near-optimal solution to Problem 4. The following lemma guarantees the tractability of Problem 9.

Lemma 4. *Problem 9 is a convex optimization problem.*

Proof: The inverse error function $\text{erf}^{-1}(x)$ is concave for x. Since we assume in Section 2.4.3 that $\Delta\le 0.5$, the feasible range of δ is upperbounded by 0.5 ($\delta_{c,i}\le 0.5$). Since the safety margin function $m_{c,i}(\delta_{c,i})$ is convex for $0<\delta_{c,i}\le 0.5$ as shown in Figure 5-1, the constraints (5.6) are convex within the feasible region. All other constraints are also convex since they are linear. The objective function is convex by assumption (Section 2.4.4). Therefore, Problem 9 is a convex optimization problem. ■

Also note that $m_{c,i}(\cdot)$ is a *nonlinear* constraint. Therefore, Problem 9 is a nonlinear convex optimization problem. It does not belong to more specific form than the convex optimization, such as the semidefinite programming or the second-order cone programming. Hence, the CRA algorithm solves Problem 9 using a general convex programming solver, such as SNOPT.

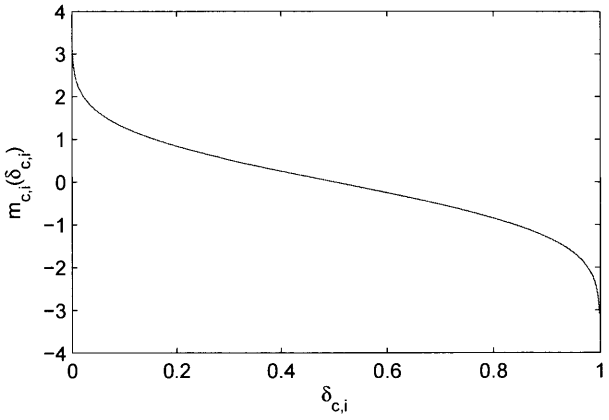


Figure 5-1: The plot of the safety margin function $m_{c,i}(\delta_{c,i})$. It is a convex function for $0 < \delta_{c,i} \leq 0.5$.

5.3 Conclusion

We obtained the CRA algorithm by deploying the risk allocation approach on the fixed-schedule CCQSP planning problem with *convex* state constraints. We proved that the resulting deterministic approximation of the problem is also a convex optimization problem. In the next chapter we extend CRA to non-convex state constraints. The CRA algorithm is empirically validated in Section 11.1.6.

Chapter 6

CCQSP Planning with a Non-convex State Space

Next, we consider the second spiral, comprised of Problem 3 in Section 3.2, a variant of the CCQSP planning problem involving a fixed schedule and non-convex constraints, such as obstacles, as shown in Figure 1-6-(b). Once again, this is encoded as a chance-constrained optimization problem, but the addition of the obstacle avoidance constraints requires disjunctive state constraints. Hence, the problem results in a non-convex, chance-constrained optimization. This chapter introduces a novel algorithm, called Non-convex Iterative Risk Allocation (NIRA), that optimally solves a deterministic approximation of Problem 3. We also present an extension of NIRA, called NIRA+BoostLP, which significantly reduces computation time without making any compromise in the optimality of the solution by incorporating with a regression-based LP solver, called BoostLP [10].

The solution to a CCQSP planning problem with a non-convex state space is two-fold. In the first step, we obtain a deterministic approximation of Problem 3. Recall that a convex, fixed-schedule CCQSP planning problem (Problem 4 in Section 3.3) is mapped to a deterministic convex programming problem (Problem 9 in Section 5.2) through risk allocation. Likewise, we map a *non-convex*, fixed-schedule CCQSP planning problem (Problem 3 in Section 3.2) to a deterministic disjunctive convex programming problem (Problem 10 in Section 6.1.3). The challenge is that the risk allocation approach presented in Section 4.1 cannot decompose the chance constraints (3.11), because it involves a disjunction of con-

straints. In order to handle such disjunctive chance constraints, we incorporate an additional decomposition approach called *risk selection*, which reformulates each chance constraint over a disjunction of constraints into a disjunction of individual chance constraints. Once the chance constraints in (3.11) are decomposed into a set of individual chance constraints through risk allocation and risk selection, the same technique as in Section 4.1.3 is used to obtain equivalent deterministic constraints. As a result, we obtain a disjunctive convex program problem (Problem 10 in Section 6.1.3) that approximates the CCQSP planning problem with obstacles and a fixed schedule (Problem 3). An optimal solution to Problem 10 is guaranteed to be a feasible solution to the original problem with regarding to satisfying the chance constraints (Problem 3). Furthermore, we empirically demonstrate in Section 11 that it is a near-optimal solution to Problem 3 in our applications.

In the second step, we solve the deterministic disjunctive convex programming problem through a branch-and-bound algorithm, which requires computing bounds on its convex subproblems. The subproblems correspond to the *convex*, fixed schedule CCQSP planning problem (Problem 4), which was solved in the previous section. However, the computation time of the naive branch-and-bound approach is not sufficiently fast for our applications, such as robust path planning with obstacles. We introduces the NIRA algorithm (Algorithm 2) that significantly reduces the computation time without making any compromise in the optimality of the solution. The reduction in computation time is enabled by our new bounding approach, Fixed Risk Relaxation (FRR). FRR is a linear relaxation of nonlinear constraints in the subproblems of the branch-and-bound algorithm with linear constraints. In many cases, FRR of the nonlinear subproblems is formulated as a linear program (LP) or approximated by an LP. NIRA obtains a strictly optimal solution of Problem 10 by solving the subproblems *exactly* without FRR at unpruned leaf nodes of the search tree.

We also present the NIRA+BoostLP algorithm, which achieves additional speed-up. It solves FRRs *approximately* by using a regression-based LP solver called BoostLP [10]. Again, NIRA+BoostLP obtains a strictly optimal solution of Problem 10 by solving the subproblems *exactly* without FRR and BoostLP at unpruned leaf nodes of the search tree, while other subproblems are solved approximately with FRR and BoostLP in order to reduce the computation time.

6.1 Deterministic Approximation

6.1.1 Risk Selection Approach

As in Section 5, we first obtain a deterministic approximation of Problem 3 by decomposing the non-convex joint chance constraint (3.11) into a set of individual chance constraints, through risk allocation and *risk selection*. We revisit the race car example to explain the concept of risk selection intuitively.

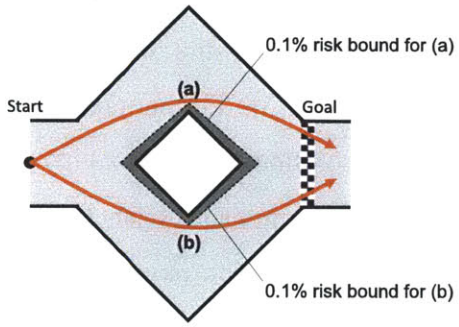


Figure 6-1: In the racing car example, the risk selection approach guarantees the 0.1% risk bound for both paths, and lets the vehicle choose the better one.

Racing Car Example We consider the example shown in Figure 6-1, where a vehicle with uncertain dynamics plans a path that minimizes the time to reach the goal. The vehicle is allowed to choose one of the two routes shown in Figure 6-1. We impose a chance constraint that limits the probability of crashing into a wall during the mission to 0.1%.

The satisfaction of the chance constraint can be guaranteed by the following process. First, for each of the routes, we find a safety margin that limits the probability of crash throughout the route to 0.1% from the start to the goal. Then, we let the vehicle plan a nominal path that operates within the safety margins. Since both routes have a 0.1% safety margin, the chance constraint is satisfied no matter which route the vehicle chooses. Therefore, the vehicle can optimize the path by choosing the route that results in a smaller cost.

The optimization process can be considered as a selection of risk; the vehicle is given two options as in Figure 6-1, routes (a) and (b), both of which involve the same level of risk;

then the vehicle selects the one that results in less cost. Hence, we name this decomposition approach as the risk selection.

6.1.2 Decomposition of Conjunctive Joint Chance Constraint through Risk Selection

In this subsection, we derive the mathematical representation of risk selection by reformulating each chance constraint over a disjunction of constraints into a disjunction of individual chance constraints. Let C_i be a proposition that is either true or false. Then the following lemma holds:

Lemma 5.

$$\Pr\left[\bigvee_{i=1}^{N} C_i\right] \geq 1-\Delta \quad \Leftarrow \quad \bigvee_{i=1}^{N} \Pr\left[C_i\right] 1-\Delta$$

Proof: The following inequality always holds:

$$\forall_i \quad \Pr\left[\bigvee_{i=1}^{N} C_i\right] \geq \Pr\left[C_i\right]. \tag{6.1}$$

Hence,

$$\Pr\left[\bigvee_{i=1}^{N} C_i\right] \geq 1-\Delta \Leftarrow \exists i \ \Pr\left[C_i\right] \geq 1-\Delta \Leftrightarrow \bigvee_{i=1}^{N} \Pr\left[C_i\right] \geq 1-\Delta. \tag{6.2}$$

■

The following corollary follows immediately from Lemmas 1 and 5.

Corollary 4. *The following set of constraints is a sufficient condition of the disjunctive joint chance constraint (3.11) in Problem 3:*

$$\exists \delta_{c,i} \geq 0 \quad \bigwedge_{c\in\mathcal{C}} \left\{ \bigwedge_{i\in\mathcal{I}_c(s)} \bigvee_{j\in\mathcal{J}_{c,i}} \Pr\left[\boldsymbol{h}_{c,i,j}^T \boldsymbol{x}_{t_i} - g_{c,i,j} \leq 0\right] \geq 1-\delta_{c,i} \ \wedge \ \sum_{i=1}^{N} \delta_{c,i} \leq \Delta_c \right\} \tag{6.3}$$

Note that the resulting set of constraints (6.3) is a *sufficient* condition for the original chance constraint (3.11). Therefore, any solution that satisfies (6.3) is guaranteed to satisfy (3.11).

Furthermore, although (6.3) is a conservative approximation of (3.11), the conservatism introduced by risk selection is generally small in many practical applications. This claim is empirically validated in Section 11.

6.1.3 Deterministic Approximation of Problem 4

The individual chance constraints in (6.3) can be transformed into equivalent deterministic convex nonlinear constraints by applying Lemma 2 in Section 4.1.3. As a result, the non-convex fixed-schedule CCQSP planning problem (Problem 3) is approximated by the following deterministic convex optimization problem. For later convenience, we label each part of the optimization problem as O (objective function), M (plant model), C (chance constraints), and R (risk allocation constraint).

Problem 10: Deterministic Approximation of Problem 3

$$\min_{\boldsymbol{u}_{1:N}\in\mathbb{U}^N,\delta_{c,i}>0} \quad (O:) \quad J'(\boldsymbol{u}_{1:N},\bar{\boldsymbol{x}}_{1:N}) \tag{6.4}$$

$$\text{s.t.} \quad (M:) \quad \forall t\in\mathbb{T}^-, \quad \bar{\boldsymbol{x}}_{t+1}=\boldsymbol{A}_t\bar{\boldsymbol{x}}_t+\boldsymbol{B}_t\boldsymbol{u}_t \tag{6.5}$$

$$(C:) \quad \bigwedge_{c\in\mathcal{C}}\bigwedge_{i\in\mathcal{I}_c(s)}\bigvee_{j\in\mathcal{J}_{c,i}} \boldsymbol{h}_{c,i,j}^T\bar{\boldsymbol{x}}_{t_i}-g_{c,i,j}\le -m_{c,i,j}(\delta_{c,i}) \tag{6.6}$$

$$(R:) \quad \bigwedge_{c\in\mathcal{C}}\sum_{i=1}^{N}\delta_{c,i}\le\Delta_c, \tag{6.7}$$

It follows immediately from Corollary 1, Corollary 4, and Lemma 2 that a feasible solution to Problem 10 is always a feasible solution to Problem 4.

6.2 NIRA: Branch and Bound-Based Solution to Problem 10

We next present the Non-convex Iterative Risk Allocation (NIRA) algorithm. Recall that NIRA optimally solves Problem 10 by a branch and bound algorithm. The standard branch-and-bound solution to Problem 10 uses a bounding approach, whereby the nonlinear convex relaxed subproblems are constructed by removing all non-convex constraints below the corresponding disjunction. This approach was used by [9] and [65] for a different problem known as disjunctive linear program, whose subproblems are LPs instead of convex programs. However, although the standard branch-and-bound algorithm is guaranteed to find a globally optimal solution to Problem 10, its computation time is slow because the algorithm needs to solve numerous nonlinear subproblems.

Our new bounding approach, Fixed Risk Relaxation (FRR), addresses this issue by computing lower bounds more efficiently. The use of FRR results in significant reduction of the computation time of the branch-and-bound algorithm without any loss in optimality. We observe that the relaxed subproblems are nonlinear convex optimization problems. FRR relaxes the nonlinear constraints to linear constraints. Particularly, when the objective function is linear, an FRR of a subproblem is an LP, which can be very efficiently solved. The optimal objective value of an FRR of a subproblem is a lower bound of the optimal objective value of the original subproblem.

NIRA solves the FRRs of the subproblems in order to efficiently obtain the lower bounds, while solving the original subproblems *exactly* without relaxation at unpruned leaf nodes, in order to obtain an exact optimal solution. As a result, NIRA achieves a significant speed-up without any compromise in optimality.

6.2.1 The NIRA Algorithm Overview

Algorithm 2 shows the pseudocode of the NIRA algorithm. The input to the algorithm is the deterministic approximation of a non-convex chance-constrained optimal control problem (Problem 10), which is a four-tuple $\langle O, M, C, R \rangle$, as well as a fixed schedule s. Its output

Algorithm 2 Non-convex Iterative Risk Allocation (NIRA) algorithm

function NIRA($problem$) **returns** optimal solution

1: Set up $queue$ as a FILO queue
2: $Incumbent \leftarrow \infty$
3: $\epsilon \leftarrow$ BoostLP error bound
4: $rootSubproblem \leftarrow$ obtainRootSubproblem($problem$)
5: $queue \leftarrow rootSubproblem$
6: **while** $queue$ is not empty **do**
7: $subproblem \leftarrow queue$.removeLastEntry()
8: $lb \leftarrow$ obtainLowerBound($subproblem$)
9: **if** $lb \leq Incumbent + \epsilon$ **then**
10: **if** $subproblem$ is a leaf node **then**
11: $(J, \boldsymbol{U}) \leftarrow$ Solve($subproblem$)
12: **if** $J^\star < Incumbent$ **then**
13: $Incumbent \leftarrow J$; $\boldsymbol{U}^\star \leftarrow \boldsymbol{U}$ //Update the optimal solution
14: **end if**
15: **else**
16: **for** $j \in \mathcal{J}_{c,i}$ **do**
17: $queue$.add(Expand($subproblem$,$problem$))
18: **end for**
19: **end if**
20: **end if**
21: **end while**
22: **return** $\boldsymbol{U}^\star$

is an optimal control sequence $\boldsymbol{U}^\star$.

Overall, Algorithm 2 is a standard branch-and-bound algorithm. Our main contributions are the new branching and bounding methods, presented in following sections. Each node of the branch-and-bound search tree corresponds to a subproblem that is a convex chance-constrained optimization problem (Problem 11 in Section 6.3.3). We use a FILO queue to store subproblems so that the search is conducted in a depth-first manner (Line 1). At each node, the corresponding subproblem is solved to obtain a lower bound of the objective value of all subsequent subproblems (Line 8). The details of the bounding approaches are explained in Subsection 6.4. If the lower bound is larger than the incumbent by the tolerance ϵ, the algorithm prunes the branch. Otherwise, the branch is expanded (Line 17). If a branch is expanded to the leaf without being pruned, subproblems are solved exactly (Line 11). Subsection 6.3 explains our expansion procedure in detail.

NIRA exploit the structure of the branch-and-bound algorithm so that an *exact* optimal

solution to Problem 10 is obtained with significantly less computation time. Specifically, at non-leaf nodes, it solves FRRs of the subproblems to obtain lower bounds efficiently, while solving the subproblems without relaxations at leaf nodes in order to ensure that the final solution of the branch-and-bound algorithm is exact.

The parameter ϵ (Lines 3 and 9) is set to zero in NIRA. In NIRA+BoostLP, which is presented in Section 6.5, ϵ is set to a positive value in order to accommodate the estimation error of BoostLP.

6.3 Branching

This subsection explains how NIRA constructs the root subproblem (Line 3 of Algorithm 2), as well as how it expands nodes (Line 17 of Algorithm 2). The root subproblem is a convex CCQSP planning problem without any chance constraints. When a node is expanded, the subproblems of its children nodes are constructed by adding one constraint in a disjunction to the subproblem of the parent node. In order to simplify notations, we let $C_{c,i,j}$ represent each individual chance constraint (6.6) in Problem 10:

$$C_{c,i,j} := \begin{cases} True & (\text{if } \boldsymbol{h}_{c,i,j}^T - g_{c,i,j}\bar{\boldsymbol{x}}_{t_i} \leq -m_{c,i,j}(\delta_{c,i})) \\ False & (\text{otherwise}). \end{cases}$$

6.3.1 Walk-through Example

We first present a walk-through example to intuitively explain the branching procedure. The example is an instance of Problem 10, which involves four individual chance constraints:

$$\bigwedge_{i\in\{1,2\}} \bigvee_{j\in\{1,2\}} \boldsymbol{h}_{1,i,j}^T \bar{\boldsymbol{x}}_{t_i} - g_{1,i,j} \leq -m_{1,i,j}(\delta_{1,i}) \tag{6.8}$$

Using this notation defined above, the set of individual chance constraints (6.6) is represented as follows:

$$(C_{1,1,1} \vee C_{1,1,2}) \wedge (C_{1,2,1} \vee C_{1,2,2}) \tag{6.9}$$

Figure 6-2-(a) shows a tree obtained by dividing the original problem into subproblems sequentially. The subproblems corresponding to the tree's four leaf nodes (Nodes 4-7 in Figure 6-2-(a)) exhaust all conjunctive (i.e., *convex*) combinations among the chance constraints (6.9). On the other hand, the subproblems corresponding to the three branch nodes (Nodes 1-3 in Figure 6-2-(a)) involve disjunctive (i.e., *nonconvex*) clauses of chance constraints. We relax such non-convex subproblems to convex subproblems by removing all clauses that contain disjunctions in order to obtain the search tree shown in Figure 6-2-(b).

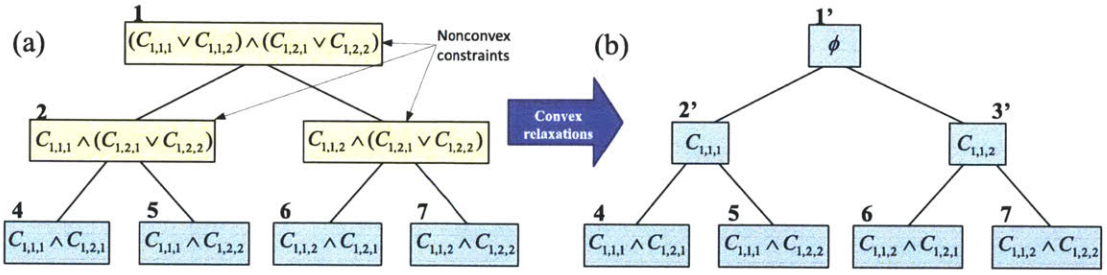


Figure 6-2: Branch-and-bound search tree for a sample disjunctive convex programming problem (Problem 10) with constraints (6.8). (a) Tree of non-convex subproblems, (b) Tree of relaxed, convex subproblems.

Note that all the subproblems in Figure 6-2-(b) are *convex* programming problems, which are solved in Chapter 5. Hence, the non-convex problem (Problem 10) can be optimally solved by repeatedly solving the convex subproblems using the CRA algorithm presented in Chapter 5. The following subsections introduce the algorithms that construct a search tree with *convex* subproblems, such as the one in Figure 6-2-(b).

6.3.2 Construction of Root Subproblem

The function presented in Algorithm 3 is used in Line 3 of the NIRA algorithm (Algorithm 2) to construct the root subproblem of the branch-and-bound tree. The root subproblem has the same objective function O and plant model M as the input non-convex chance-constrained optimization problem (Problem 10), but has no chance constraints. The resulting root subproblem is as follows:

Algorithm 3 Construction of the root subproblem of NIRA

function obtainRootSubproblem($problem$) **returns** root subproblem

1: $rootSubproblem.O \leftarrow problem.O$
2: $rootSubproblem.M \leftarrow problem.M$
3: $rootSubproblem.C \leftarrow \phi$
4: **for** $c \in |\mathcal{C}|$ **do**
5: $\quad rootSubproblem.R_c.lhs \leftarrow 0$
6: $\quad rootSubproblem.R_c.rhs \leftarrow problem.R_c.rhs$
7: $\quad rootSubproblem.c \leftarrow 1, rootSubproblem.i \leftarrow 1$
8: **end for**
9: **return** $rootSubproblem$

$$\begin{aligned} \min_{\boldsymbol{u}_{1:N} \in \mathbb{U}^N, \delta_{c,i} > 0} \quad & (O:) \quad J'(\boldsymbol{u}_{1:N}, \bar{\boldsymbol{x}}_{1:N}) \\ \text{s.t.} \quad & (M:) \quad \forall t \in \mathbb{T}^-, \quad \bar{\boldsymbol{x}}_{t+1} = \boldsymbol{A}_t \bar{\boldsymbol{x}}_t + \boldsymbol{B}_t \boldsymbol{u}_t \\ & (R:) \quad 0 \leq \Delta_c, \end{aligned}$$

6.3.3 Expansion of subproblems

In order to create a child subproblem of a subproblem, the function described in Algorithm 4 is used in Line 17 of the NIRA algorithm (Algorithm 2). It increments the indices (c, i) (Lines 1-4), and adds the individual chance constraint specified by the indices (c, i, j) as a conjunct (Lines 5-6). Note that the resulting child subproblem is still a convex optimization because the individual chance constraint is added conjunctively. The NIRA algorithm (Algorithm 2) enumerates children nodes for all disjuncts in $\mathcal{J}_{c,i}$ (Lines 16-18).

The formulation of the convex relaxed subproblems is given in Problem 11. We denote the index j as $j(c, i)$ since the convex relaxation chooses only one disjunct for each disjunction specified by (c, i). We denote by $J^\star_{SP}$ the optimal objective value of the relaxed subproblem for later convenience.

Problem 11: Convex Relaxed Subproblem of NIRA

Algorithm 4 Expansion of a subproblem of NIRA

function Expand($subproblem, problem, j$) **returns** a child subproblem

1: $c \leftarrow subproblem.c$; $i \leftarrow subproblem.i + 1$
2: **if** $i > |\mathcal{I}_c(s)|$ **then**
3: $\quad c \leftarrow c + 1; i = 1$
4: **end if**
5: $subproblem.C \leftarrow subproblem.C \wedge problem.C_{c,i,j}$
6: $subproblem.R_c.lhs \leftarrow subproblem.R_c.lhs + \delta_{c,i}$
7: $subproblem.c \leftarrow c$; $subproblem.i \leftarrow i$
8: **return** $subproblem$

$$
\begin{aligned}
J^{\star}_{SP} = \min_{\boldsymbol{u}_{1:N} \in \mathbb{U}^N, \delta_{c,i} \geq 0} \quad & (O:) \quad J'(\boldsymbol{u}_{1:N}, \bar{\boldsymbol{x}}_{1:N}) \\
\text{s.t.} \quad & (M:) \quad \forall t \in \mathbb{T}^-, \quad \bar{\boldsymbol{x}}_{t+1} = \boldsymbol{A}_t \bar{\boldsymbol{x}}_t + \boldsymbol{B}_t \boldsymbol{u}_t \\
& (C:) \quad \bigwedge_{c \in \mathcal{C}} \bigwedge_{i \in \mathcal{I}_c(s)} \boldsymbol{h}^T_{c,i,j(c,i)} \bar{\boldsymbol{x}}_{t_i} - g_{c,i,j(c,i)} \leq -m_{c,i,j(c,i)}(\delta_{c,i}) \qquad (6.10) \\
& (R:) \quad \bigwedge_{c \in \mathcal{C}} \sum_{i \in \mathcal{I}_c(s)} \delta_{c,i} \leq \Delta_c. \qquad (6.11)
\end{aligned}
$$

Note that Problem 11 is identical to Problem 9. Hence, the algorithms introduced in Section 5 can be used to solve the relaxed subproblems.

6.4 Bounding

In this subsection, we present two implementations of the *obtainLowerBound* function in Line 8 of Algorithm 2. The first one uses the optimal solution of the convex subproblems (Problem 11) as lower bounds.

This approach typically results in extensive computation time. The second one solves an LP relaxation of the convex subproblems, called fixed risk relaxation (FRR). FRR dramatically reduces the computation time compared to the first implementation. The NIRA algorithm employs the second implementation.

Algorithm 5 A simple implementation of the obtainLowerBound function in Line 8 of Algorithm 2

function obtainLowerBound-Naive(*subproblem*) **returns** a lower bound

1: Solve *subproblem* using algorithms presented in Section 5.2
2: **return** the optimal objective value

6.4.1 Simple Bounding

Algorithm 5 shows the most straight-forward way to obtain lower bounds. It simply solves the convex relaxed subproblems (Problem 11) using the methods presented in Section 5.2. The optimal objective value of a relaxed subproblem gives a lower bound of the optimal objective value of all the subproblems below it. For example, the optimal solution of the relaxed subproblem at Node $2'$ in Figure 6-2-(b) gives a lower bound of the objective value of the subproblems at Nodes 4 and 5. This is because the constraints of the relaxed subproblems are always a subset of the constraints of the subproblems below. Note that optimization problems are formulated as minimizations.

However, despite the simplicity of this approach, its computation time is slow because the algorithm needs to solve a myriad of subproblems. For example, a simple path planning problem with ten time steps and one rectangular obstacle requires the solution of $4^{10} = 1,048,576$ in the worst case, although the branch-and-bound process often significantly reduces the number of subproblems to be solved. Moreover, the subproblems (Problem 11) are *nonlinear* convex optimization problems, because the constraints (6.10) are nonlinear due to the nonlinearity of the inverse of a Gaussian cumulative distribution function $m_{c,i,j}(\delta_{c,i})$. A general nonlinear optimization problem requires significantly more solving time than more specific classes of optimization problems, such as linear programmings (LPs) and quadratic programmings (QPs).

6.4.2 Fixed Risk Relaxation

Our new relaxation approach, fixed risk relaxation (FRR), addresses this issue. FRR linearizes the nonlinear constraint (6.10) of the branch-and-bound subproblems (Problem 11) by fixing all the individual risk allocations $\delta_{c,i}$ to its upper bound Δ. When the objective

function is linear, an FRR is an LP. An FRR with a convex piecewise linear objective function can also be reformulated as an LP by introducing slack variables (see Section 11.1.4 for an example). A general convex objective function can be approximated by a convex piecewise linear function. Hence, in many cases, the FRRs of subproblems result in LPs, which can be solved very efficiently.

The fixed risk relaxation of a convex relaxed subproblem (Problem 11) is as follows:

Problem 12: Fixed Risk Relaxation of Problem 11

$$
\begin{aligned}
J^{\star}_{FRR} = \min_{\boldsymbol{u}_{1:N} \in \mathbb{U}^N, \delta_{c,i} \geq 0} \quad & J'(\boldsymbol{u}_{1:N}, \bar{\boldsymbol{x}}_{1:N}) \\
\text{s.t.} \quad & \forall t \in \mathbb{T}^-, \quad \bar{\boldsymbol{x}}_{t+1} = \boldsymbol{A}_t \bar{\boldsymbol{x}}_t + \boldsymbol{B}_t \boldsymbol{u}_t \\
& \bigwedge_{c \in \mathcal{C}} \bigwedge_{i \in \mathcal{I}_c(s)} h_{c,i}^T \bar{\boldsymbol{x}}_{t_i} - g_{c,i} \leq -m_{c,i}(\Delta_c) \qquad (6.12)
\end{aligned}
$$

Note that the nonlinear constraint (6.10) is turned into a linear constraint (6.12) since the nonlinear term $m'_{c,i,j}$ becomes constant by fixing $\delta_{c,i}$ to Δ_c, which is a constant.

The optimal objective value of the FRR of a subproblem provides a tightest lower bound in the branch-and-bound search among the linear relaxations of constraints (6.10). The following lemmas hold:

Lemma 6. *Problem 12 gives a lower bound to the optimal objective value of Problem 11:*

$$J^{\star}_{FRR} \leq J^{\star}_{SP}$$

Proof: $m_{c,i,j}(\cdot)$ is a monotonically decreasing function. Since $\delta_{c,i} \leq \Delta_c$, all individual chance constraints (6.12) of the Fixed Risk Relaxation are less stricter than the first conjunct of (6.10). Therefore, the cost of the optimal solution of the Fixed Risk Relaxation is less than or equal to the original subproblem. ■

Lemma 7. *FRR gives the tightest lower bound among the linear relaxations of constraints (6.10).*

Algorithm 6 An FRR implementation of the obtainLowerBound function in Line 8 of Algorithm 2

function obtainLowreBound-FRR($subproblem$) **returns** lower bound

1: **for** $\forall(c,i,j)$ in $subproblem.C$ **do**
2: $\quad subproblem.C_{c,i,j}.rhs \leftarrow -m_{c,i,j}(\Delta_c)$ //Apply fixed risk relaxation
3: **end for**
4: Remove $subproblem.R$
5: Solve $subproblem$ using an LP solver
6: **return** the optimal objective value

Proof: The linear relaxation of (6.10) becomes tighter by fixing $\delta_{c,i}$ to a less value. However, setting $\delta_{c,i}$ to the value less than Δ_c make the constraint tighter than the original constraint since there can be a solution that sets $\delta_{c,i} = \Delta_c$ for some (c,i). Hence, FRR is the tightest linear relaxation of (6.10), resulting in the tightest lower bound. ■

Note that the optimal solution of Fixed Risk Relaxation (Problem 12) is an infeasible solution to Problem 11 in general, since setting $\delta_{c,i} = \Delta_c$ violates the constraint (6.11). In a special case where $|\mathcal{I}_c(s)| = 1$, the Fixed Risk Relaxation is equivalent to the original problem.

Algorithm 6 implements the fixed risk relaxation. It constructs an LP relaxation by substituting a constant Δ_c for a variable $\delta_{c,i}$, and thus replacing the nonlinear right-hand side of (6.10) with a constant value $-m_{c,i,j}(\Delta_c)$. The LP relaxation is solved by an LP solver, and its optimal objective value is returned. We use $\epsilon = 0$ in Line 9 of Algorithm 2.

6.5 NIRA+BoostLP Algorithm

When the objective function $J'(\boldsymbol{u}_{1:N}, \bar{\boldsymbol{x}}_{1:N})$ is linear, additional reduction in computation time can be achieved by the NIRA+BoostLP algorithm, an extension of NIRA. NIRA+BoostLP incorporates the regression-based approximate LP solver, BoostLP, developed by [10]. We observe that the NIRA algorithm (Algorithm 2) repeatedly solves FRRs, and the repeated solution to the FRRs dominates the computation time. We also observe that FRR problems in a particular branch-and-bound tree often share multiple common constraints and always contain the same objective function and number of decision variables.

Algorithm 7 An implementation of the obtainLowerBound function in Line 8 of Algorithm 2 with FRR and BoostLP

function obtainLowreBound-FRR-BoostLP($subproblem$) **returns** approximate lower bound

1: **for** $\forall(c, i, j)$ in $subproblem.C$ **do**
2: $\quad subproblem.C_{c,i,j}.rhs \leftarrow -m_{c,i,j}(\Delta_c)$ //Apply fixed risk relaxation
3: **end for**
4: Remove $subproblem.R$
5: Solve $subproblem$ approximately using Boost LP
6: **return** the estimated optimal objective value

These observations justifies the use of a regression-based approximate LP solver called BoostLP [10]. BoostLP takes an LP problem as an input, and outputs an approximate solution with a known error bound on the objective value. In order to construct a regression model, BoostLP uses a set of similar LP problems and their solutions as learning data. The concept of "similarity" between LP problems is defined in [10].

The NIRA+BoostLP algorithm is obtained through two modifications on NIRA. Firstly, the obtainLowerBound function in Line 8 of Algorithm 2 is replaced by the obtainLowreBound-FRR-BoostLP function shown in Algorithm 7. Lines 1-4 of Algorithm 7, which are identical to Algorithm 6, construct an FRR of the input subproblem. Line 5 solves the FRR approximately using the BoostLP algorithm. The estimated object value obtained from the BoostLP is returned.

The second modification is on ϵ in Line 9 of Algorithm 2, which is set to zero in NIRA. NIRA-Boost LP sets this parameter to the worst-case error bound of BoostLP (Line 3 of Algorithm 2). BoostLP provides a *fixed* worst-case error bound for a given system of similar LP problems. The error bound is constant for a given training data set and regression model and is applicable for any test FRR that belongs to the same space of similar LP problems. In other words, there exists a positive finite real number ϵ such that:

$$\left|\hat{J}_{FRR} - J^{\star}_{FRR}\right| \leq \epsilon,$$

where $\hat{J}_{FRR}$ is the estimated optimal cost of the FRR. The existence of a hard error bound allows the branch-and-bound algorithm to solve the problem optimally, since there is no danger of pruning the branch that includes the optimal solution. Hence, NIRA-BoostLP use

$\hat{J}_{FRR} + \epsilon$ as the lower bound in the branch-and-bound optimization (Line 9 of Algorithm 2).

Although FRRs are solved approximately by BoostLP, the subproblems at unpruned leaf nodes of the search tree are solved *exactly* without FRR and BoostLP. Hence, the solution of NIRA+BoostLP is a strictly optimal solution to Problem 10. Empirical Results presented in Section 11.1.8 shows that the computation time of NIRA-BoostLP is less than NIRA by 10-25 times.

A limitation of NIRA+BoostLP is that it can only solve problems with the same number of non-convex constraints as the problems used for learning. As a result, problems with different numbers of obstacles or with different number of edges in each obstacle cannot be solved by the same regression model. This is because BoostLP can only solve problems with the same number of decision variables as the learning problems. Since non-convex constraints are convexified by introducing slack variables, different numbers of non-convex constraints for NIRA+BoostLP results in different numbers of decision variables for BoostLP. Extending NIRA+BoostLP for a flexible number of non-convex constraints is our future work.

6.6 Conclusion

In this chapter we developed the non-convex iterative risk allocation (NIRA) algorithm, which can plan in a *non-convex* state space with a fixed schedule. NIRA employs a branch-and-bound algorithm to solve the disjunctive convex program. Its subproblems are fixed-schedule CCQSP problems with a *convex* state space, which can be solved by the CRA algorithm introduced in Chapter 5. We developed a novel relaxation method called fixed risk relaxation (FRR), which provides the tightest linear relaxation of the nonlinear constraints in the convex subproblems. We also developed the NIRA+BoostLP algorithm by integrating a regression-based LP solver, BoostLP, into NIRA. Compared to NIRA, NIRA+BoostLP achieved 10- to 25-fold reduction in computation time without any compromise in solution optimality. Simulation results of NIRA and NIRA+BoostLP are presented in Sections 11.1.7 and 11.1.8, respectively.

In the next chapter, we extend NIRA to a problem with a *flexible* schedule.

Chapter 7

CCQSP Planning with a Flexible Schedule

This chapter presents the full-horizon probabilistic Sulu or *p-Sulu FH* algorithm, which efficiently solves the general CCQSP planning problem with a *flexible* schedule and a non-convex state space (Problem 2 in Section 3.1.2). The problem is to find a schedule of events s (defined in Section 2.3.1) that satisfies simple temporal constraints, as well as a control sequence $\boldsymbol{u}_{0:N-1}$ that satisfies the chance constraints and minimizes cost. Our approach is to first generate a feasible schedule and then to extend the schedule to an optimal control sequence for that scheduling, while iteratively improving the candidate schedules using branch and bound.

We build our CCQSP planner, p-Sulu FH, upon the NIRA algorithm presented in the previous section. Recall that NIRA optimizes the control sequence $\boldsymbol{u}_{0:N-1}$ and computes the optimal objective function value, given a *fixed* schedule s. p-Sulu FH uses NIRA as a subroutine that takes a schedule s as an input, and outputs the optimal objective value as well as the optimal control sequence. We denote the optimal objective value for a given schedule s as $J^\star(s)$, highlighting that it is a function of s. Using this notation, the CC-QSP planning problem with a *flexible* schedule (Problem 2) can be rewritten as a schedule optimization problem as follows:

$$\min_{s \in \mathcal{S}_F} J^\star(s). \tag{7.1}$$

Recall that the domain of feasible schedules S_F (defined by (2.7) in Section 2.4.3) is a finite set, since we consider a discretized, finite set of time steps $\mathbb{T}$ (see Section 2.1). Hence, the schedule optimization problem (7.1) is a combinatorial constraint optimization problem (COP), where the constraints are given in the form of simple temporal constraints.

7.1 Algorithm Overview

Our idea is again to use a branch-and-bound approach to solve the schedule optimization problem (7.1). In the branch-and-bound search, p-Sulu FH incrementally assigns an execution time step to each event in order to find the schedule that minimizes $J^\star(s)$ in (7.1). The objective function value is evaluated by solving the fixed schedule CCQSP planning problems using the NIRA algorithm, presented in Chapter 6. Hence, p-Sulu FH uses NIRA as a function that takes a schedule as an input and returns an optimal control sequence and the optimal objective value. Although the combination of the two branch-and-bound searches in p-Sulu FH and NIRA are equivalent to one unified branch-and-bound search in practice, we treat them separately for ease of explanation.

As shown in Figure 7-3, the branch-and-bound algorithm searches for an optimal schedule by incrementally assigning execution time steps to each event in a depth-first manner. Each node of the search tree corresponds to a partial schedule (Definition 3), which assigns execution time steps to a subset of the events included in the CCQSP. The partial schedule at the root node only involves an assignment to the start node e_0. The tree is expanded by assigning an execution time step to one new event at a time. For example, the node $\sigma(e_1) = 2$ in Figure 7-3-(a) represents a partial schedule that assigns the execution time step $t = 0$ to the event e_0 and $t = 2$ to e_1, while leaving e_E unassigned.

p-Sulu FH obtains the lower bound of the objective function value $J^\star(s)$ by solving a CCQSP planning problem with a *partial* schedule that can be extended to s. p-Sulu FH minimizes the search space by dynamically pruning the domain through forward-checking. More specifically, after an execution time is assigned to an event at each iteration of the branch-and-bound search, p-Sulu FH runs a shortest-path algorithm to tighten the real-valued upper and lower bounds of the execution time step of unassigned events according

Algorithm 8 p-Sulu FH

function pSulu($ccqsp$) **returns** optimal schedule and control sequence

1: $Incumbent = \infty$
2: Set up $queue$ as a FILO queue
3: $\mathcal{E}_{\sigma_0} = \{e_0\}, \sigma_0(e_0) = 0$ //initialize the partial schedule
4: $queue \leftarrow \langle \mathcal{E}_{\sigma_0}, \sigma_0 \rangle$
5: **while** $queue$ is not empty **do**
6: $\langle \mathcal{E}_\sigma, \sigma \rangle \leftarrow queue$.removeLastEntry();
7: $[J^\star, \boldsymbol{u}_{0:N-1}] \leftarrow$ obtainLowerBound($ccqsp, \mathcal{E}_\sigma, \sigma$);
8: **if** $J^\star < Incumbent$ **then**
9: **if** $\mathcal{E}_\sigma = \mathcal{E}$ **then**
10: $Incumbent \leftarrow J^\star$; $OptCtlSequence \leftarrow \boldsymbol{u}_{0:N-1}$; $OptSchedule \leftarrow \sigma$
11: **else**
12: expand($ccqsp, queue, \mathcal{E}_\sigma, \sigma$)
13: **end if**
14: **end if**
15: **end while**
16: **return** $OptCtlSequence, OptSchedule$

to the newly assigned execution time step.

Algorithm 8 shows the pseudocode of the algorithm. At each node of the search tree, a fixed-schedule CCQSP planning problem is solved with the given partial schedule. If the node is at the leaf of the tree and the optimal objective value is less than the incumbent, the optimal solution is updated (Line 10). If the node is not at the leaf, the optimal objective value of the corresponding subproblem is a lower bound for the optimal objective value of subsequent nodes. If the lower bound is less than the incumbent, the node is expanded by enumerating the feasible execution time assignments to an unassigned event (Line 12). Otherwise, the node is not expanded, and hence pruned. Details of this branch and bound process are described in later subsections.

Walk-through example We present a walk-through example to give readers insight into the solution process. We consider a CCQSP shown in Figure 7-1-(a). The CCQSP specifies a mission to go through a waypoint A and get to the goal region B while avoiding the obstacle C, as shown in Figure 7-1-(b). We assume that the time interval is $\Delta T = 1$.

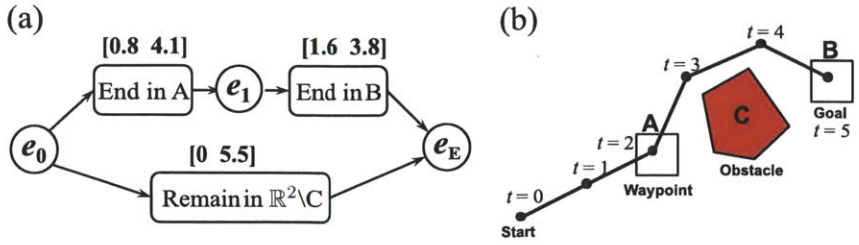


Figure 7-1: **(a)** An example of CCQSP; **(b)** a plan that satisfies the CCQSP in (a)

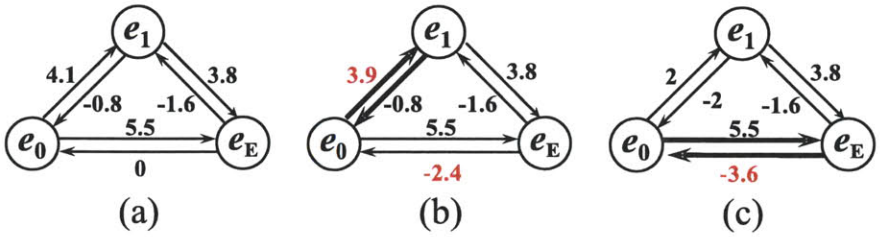


Figure 7-2: **(a)** The directed distance graph representation of the CCQSP in Figure 7-1-(a); **(b)** the d-graph of (a), which shows the shortest distances between nodes; **(c)** the updated d-graph after the execution time $t = 2$ is assigned to the event e_1.

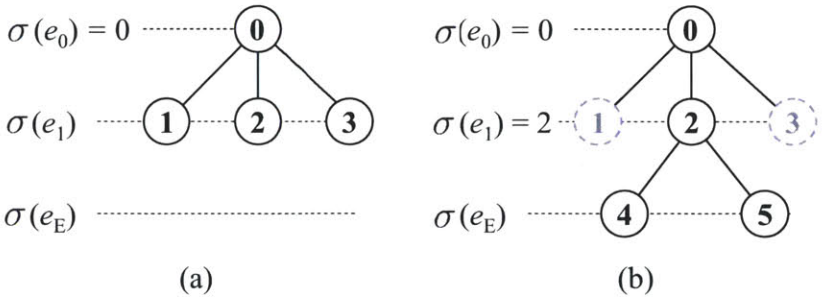


Figure 7-3: Branch-and-bound search over a schedule s. We assume that the time interval is $\Delta T = 1$. **(a)** The node $\sigma(e_0) = 0$ is expanded; $D_{e_1}(\sigma) = \{1, 2, 3\}$ given $\sigma(e_0) = 0$, since $\left[d_e^{\max}(\sigma), d_e^{\min}(\sigma)\right] = [0.8, 3.9]$ from Figure 7-2-(b); **(b)** the node $\sigma(e_1) = 2$ is expanded; $D_{e_E}(\sigma) = \{4, 5\}$ given $\sigma(e_0) = 0$ and $\sigma(e_1) = 2$, since $\left[d_e^{\max}(\sigma), d_e^{\min}(\sigma)\right] = [3.6, 5.5]$ from Figure 7-2-(c).

Figures 7-2 and 7-3 illustrate the solution process. The p-Sulu FH algorithm is initialized by assigning the execution time 0 to the start event e_0. Figure 7-2-(a) is the distance graph representation of simple temporal constraints of the CCQSP. Note that a simple chance constraint is equivalently represented as a pair of inequality constraints as follows:

$$s(e) - s(e') \in [l, u] \iff s(e) - s(e') \leq u \ \wedge \ s(e') - s(e) \leq -l.$$

The two inequality constraints are represented by two directional edges between each two nodes in the distance graph. p-Sulu FH runs an all-pair shortest-path algorithm on the distance graph to obtain the d-graph shown in Figure 7-2-(b). The d-graph represents the tightest temporal constraints. Then the algorithm enumerates the feasible execution-time assignments for the event e_1 using the d-graph. According to the d-graph, the execution time for the event e_1 must be between 0.8 and 3.9. Since we consider discrete time steps with the time interval $\Delta T = 1$, the feasible execution time steps for e_1 are $\{1, 2, 3\}$, as shown in Figure 7-3-(a). The idea behind enumerating all feasible execution time steps is to assign an event, and thus to tighten the bounds of all unassigned events in order to ensure feasibility.

At the node $\sigma(e_1) = 1$ in Figure 7-3-(a), p-Sulu FH solves the FRR of the fixed-schedule CCQSP planning problem *only* with the "End in A" episode and the execution schedule $\sigma(e_1) = 1$. In other words, it tries to find the optimal path that goes through A at $t = 1$, but neglects the goal B and obstacle C. If a solution exists, its optimal cost gives a lower bound on the objective value of all feasible paths that go through A at $t = 1$. Assume here that such a solution does not exist. Then, p-Sulu FH prunes the node $\sigma(e_1) = 1$, and goes to the next node $\sigma(e_1) = 2$. It solves the FRR of the corresponding fixed-schedule subproblem to find the best path that goes through A at $t = 2$. Assume that p-Sulu FH finds a solution.

Then, p-Sulu FH expands the node in the following process. First, it fixes the execution time $\sigma(e_1) = 2$ in the d-graph, and runs a shortest-path algorithm in order to tighten the temporal constraints (see Figure 7-2-(c)). Then p-Sulu FH uses the updated d-graph to enumerate the feasible execution-time assignments for the event e_E, which are $\{4, 5\}$, as

shown in Figure 7-3-(b). It visits both nodes and solves the fixed-schedule subproblems exactly with all episodes and a fully assigned schedule. For example, at the node $\sigma(e_E) = 5$, it computes the best path that goes through A at $t = 2$ and reaches B at $t = 5$ while avoiding the obstacle C, as shown in Figure 7-1-(b). Assume that the optimal objective values of the subproblems are 10.0 for $\sigma(e_E) = 4$ and 8.0 for $\sigma(e_E) = 5$. The algorithm records the solution with $\sigma(e_E) = 5$ and its cost 8.0 as the incumbent.

The p-Sulu FH algorithm then backs up and visits the node $\sigma(e_1) = 3$, where a relaxed subproblem with only the "End in A" episode is solved to obtain the lower bound of the objective value of subsequent nodes. The lower bound turns out to be 9.0, which exceeds the incumbent. Therefore, the branch is pruned. Since there are no more nodes to expand, the algorithm is terminated, and the incumbent solution is returned.

The order of node expansion is very important. Note that we visited the event e_1 before the event e_E in this example. This is because the "End in A" episode only involves a convex state constraint, while "Remain in $\mathbb{R}^2 \backslash C$" (2D plane minus the obstacle C) is non-convex. Therefore, the subproblems at nodes $\sigma(e_1) = 1, 2, 3$ that impose only the "End in A" episode are convex optimization problems, which can be solved much more efficiently than the non-convex relaxed subproblems with only the "Remain in $\mathbb{R}^2 \backslash C$" episode. Furthermore, the fixed risk relaxation (Section 6.4.2) can be applied to obtain a lower bound even more efficiently when the problem is convex. Therefore, we can enhance the speed of the branch-and-bound search by sorting the events so that the episodes with a convex feasible region are always considered before the episodes with a non-convex feasible region.

7.2 Branching with d-graph

This subsection presents the implementation of the expand(...) function in Line 12 of Algorithm 8. Recall that each non-leaf node of the search tree represents a partial schedule (Definition 3), where some of the events are not assigned an execution time step.

Algorithm 9 outlines the implementation of the expand(...) function in Algorithm 8. It takes a partial schedule σ as an input, and adds to the queue a set of schedules that assign an execution time step to an additional event e'. In other words, the domain of the newly added

Algorithm 9 Implementation of expand function in Line 12 of Algorithm 8

function expand($ccqsp, queue, \mathcal{E}_\sigma, \sigma$)

1: **for** $e \in \mathcal{E}_\sigma$ **do**
2: Fix the distance between e_0 and e to $\sigma(e)\Delta T$ on the d-graph of $ccqsp$;
3: **end for**
4: Update the d-graph by running a shortest-path algorithm;
5: Choose e' from $\mathcal{E}\backslash\mathcal{E}_\sigma$ //choose an unassigned event
6: $\mathcal{E}_{\sigma'} := \mathcal{E}_\sigma \cup e'$
7: $D_{e'}(\sigma) := \{\, t \in \mathbb{T} \mid d_{e'}^{\min}(\sigma) \leq t\Delta T \leq d_{e'}^{\max}(\sigma)\}$
8: **for** t in $D_{e'}(\sigma)$ **do**
9: $\sigma'(e) := \begin{cases} \sigma(e) & (e \in \mathcal{E}_\sigma) \\ t & (e = e') \end{cases}$ //update the partial schedule
10: $queue \leftarrow \langle \mathcal{E}_{\sigma'}, \sigma' \rangle$
11: **end for**

schedules $\mathcal{E}_{\sigma'}$ has one more assigned event than the domain of the input partial schedule $\mathcal{E}_\sigma$. The details of Algorithm 9 are explained in the following parts of this subsection.

7.2.1 Enumeration of Feasible Time Step Assignments using d-graph

When enumerating all feasible time steps, the simple temporal constraints must be respected. To accomplish this, we use a d-graph to translate the bounds on the durations between two events into the bounds on the execution time step of each event. It is shown by [33] that the set of feasible execution times for an event e is bounded by the distance between e and e_0 on the d-graph. A d-graph is a directed graph, where the weights of the edges represent the *shortest* distances between nodes, as in Figure 7-2-(b). In order to obtain the d-graph representation, we first translate the simple temporal constraints into a directed distance graph, as in Figure 7-2-(a). The weight of an edge between two nodes (events) corresponds to the minimum duration of time from the origin node to the destination node, as specified by the corresponding simple temporal constraint. The distance takes a negative value to represent lower bounds. The d-graph (Figure 7-2-(b)) is obtained from the distance graph (Figure 7-2-(a)) by running an all-pair shortest-path algorithm [33].

Forward-checking with d-graph The p-Sulu FH algorithm incrementally assigns an execution time step to each event, as explained in the walk-through example. p-Sulu FH

minimizes the search space through forward-checking using the d-graph. What is different here from normal forward checking is that no back tracking is performed, due to decomposability of d-graph. The forward-checking is conducted in the following process.

Once an execution time step t is assigned to an event e (i.e., $\sigma(e) = t$), the distance from e_0 to e is fixed to $t\Delta T$, and the distance from e to e_0 is fixed to $-t\Delta T$ on the distance graph (Line 2 of Algorithm 9). Recall that t is an index of discretized time steps with a fixed interval ΔT, while the temporal bounds are given as real-valued times (see Section 2.1). We then run a shortest-path algorithm to update the d-graph (Line 4). Given a partial schedule σ, we denote the updated shortest distance from the start event e_0 to e' on the d-graph by $d^{\max}_{e'}(\sigma)$, and the distance from e' to e_0 by $d^{\min}_{e'}(\sigma)$.

For example, the execution time 2 is assigned to the event e_1 in Figure 7-2-(c) (i.e., $\sigma(e_1) = 2$), so the distance between e_0 and e_1 is fixed to 2 and the distance in the opposite direction is fixed to -2. Then we run a shortest-path algorithm again to update the d-graph. As a result, we obtain updated distances $d^{\max}_{e_E}(\sigma) = 5.5$ and $d^{\min}_{e_E}(\sigma) = -3.6$.

[33] showed that $d^{\max}_{e'}(\sigma)$ corresponds to the upper bound of the feasible execution time for an unassigned event e', while $d^{\min}_{e_E}(\sigma)$ corresponds to the negative of the lower bound. Hence, after a partial schedule σ is assigned to events $e \in \mathcal{E}_\sigma$, the updated domain for an unassigned event $e' \notin \mathcal{E}_\sigma$ is bounded by $d^{\min}_{e'}(\sigma)$ and $d^{\max}_{e'}(\sigma)$. Note that the domain of the execution time steps e' is included in, but not equal to $[d^{\min}_{e'}(\sigma), d^{\max}_{e'}(\sigma)]$, because we only consider discrete execution time steps in a finite set $\mathbb{T}$. In the forward-checking, p-Sulu FH only computes the real-valued bounds $[d^{\min}_{e'}(\sigma), d^{\max}_{e'}(\sigma)]$. The feasible values of an unassigned variable e' are not enumerated until the search tree is expanded to e'.

Enumerating the domain of execution time steps for an unassigned event We can readily extract the feasible execution time steps for any unassigned event $e' \notin \mathcal{E}_\sigma$ from the updated d-graph with a partial schedule σ. Let $D_{e'}(\sigma)$ be the domain of execution time steps for an unassigned event $e' \notin \mathcal{E}_\sigma$, given a partial schedule σ. The finite domain $D_{e'}(\sigma)$ is obtained as follows:

$$D_{e'}(\sigma) := \{\, t \in \mathbb{T} \mid d^{\min}_{e'}(\sigma) \leq t\Delta T \leq d^{\max}_{e'}(\sigma)\},$$

where ΔT is the fixed time interval of the discretized time steps (Line 7). Note that $D_e(\sigma)$ may be empty when the temporal constraints are tight, even though they are feasible. The user of p-Sulu FH must make ΔT small enough so that D_e is not empty.

For example, Figure 7-2-(b) is the d-graph given the partial schedule $\{\sigma(e_0) = 0\}$. According to the d-graph, e_1 must be executed between 0.8 and 3.9. Assuming that $\Delta T = 1$, the set of feasible execution time steps for e_1 is $D_{e_1}(\sigma) = \{1, 2, 3\}$, as shown in Figure 7-3-(a). Likewise, Figure 7-2-(c) is the d-graph given the partial schedule $\{\sigma(e_0) = 0, \sigma(e_1) = 2\}$; the feasible execution time of e_E is between 3.6 and 5.5. Hence, the set of feasible execution time steps for e_E is $D_{e_E}(\sigma) = \{4, 5\}$, as shown in Figure 7-3-(b).

The enumeration is conducted in Line 7 of Algorithm 9. Then the algorithm creates extensions of the input partial schedule by assigning each of the time steps to e' (Line 9), and puts the extended partial schedules in the queue (Line 10).

7.2.2 Efficient Variable Ordering of Branch and Bound Search

When choosing the next event to assign a time step in Line 5 of Algorithm 9, two variable ordering heuristics are found to be effective in order to reduce computation time.

The first heuristic is our new *convex-episode-first* (CEF) heuristic, which prioritizes events that are not associated with non-convex constraints. The idea of the CEF heuristic is based on the observation that subproblems of the branch-and-bound algorithm are particularly difficult to solve when the episodes in $\mathcal{A}(\mathcal{E}_\sigma)$ involve non-convex state constraints. The "Remain in $\mathbb{R}^2 \backslash C$" episode in the walk-through example in Figures 7-1 is an example of such non-convex episodes. Therefore, an effective approach to reduce the computation time of p-Sulu FH is to minimize the number of non-convex subproblems solved in the branch-and-bound process. This idea can be realized by sorting the events so that the episodes with a convex feasible region are always examined in the branch-and-bound process before the episodes with a non-convex feasible region.

The second one is the well-known most constrained variable heuristic. When p-Sulu FH expands a node, it counts the number of feasible time steps in $D_{e'}(\sigma)$ for all unassigned events $e' \in \mathcal{E} \backslash \mathcal{E}_\sigma$, and chooses the one with the least number of feasible time steps in Line 5 of Algorithm 9.

7.3 Bounding with Partial Schedule and FRR

This subsection presents the implementation of the obtainLowerBound() function in Line 7 of Algorithm 8. The Branch-and-bound process depends on the ability to compute a bound on the optimal objective value of those subproblems that lie below each node in the search tree. The p-Sulu FH algorithm obtains the lower bound by solving a relaxed CCQSP planning problem with a fixed partial schedule.

Algorithm 10 outlines the implementation of the obtainLowerBound() function. It takes a partial schedule σ as an input, and outputs the lower bound of the objective function, as well as the optimal control sequence given the partial schedule σ. It constructs a relaxed optimization problem, which only involves episodes whose start and end events are both assigned execution time steps (Line 1). If the optimization problem involves non-convex constraints, the NIRA algorithm is used to obtain the solution to the problem (Line 3). Otherwise we solve the FRR of the convex optimization problem to obtain the lower bound efficiently (Line 5). If the input is a fully assigned schedule ($\mathcal{E}_\sigma = \mathcal{E}$), the corresponding node is a leaf node. In such case we obtain an *exact* solution to the CCQSP planning problem with the fixed schedule σ by running the NIRA algorithm (Line 3). The details of Algorithm 10 are explained in the subsequent part of this section.

Algorithm 10 Implementation of obtainLowerBound function in Line 7 of Algorithm 8

function obtainLowerBound($ccqsp, \mathcal{E}_\sigma, \sigma$) **returns** optimal objective value and control sequence

1: $subprblem \leftarrow$ Problem 13 with σ given $ccqsp$;
2: **if** $\mathcal{E}_\sigma = \mathcal{E}$ **or** $\mathcal{A}(\sigma)$ has episodes with non-convex state regions, **then**
3: $[J^\star, \boldsymbol{u}_{0:N-1}] \leftarrow$ NIRA($subprblem$) //Algorithm 2
4: **else**
5: $J^\star \leftarrow$obtainLowreBound-FRR($subprblem$) - ϵ //Algorithm 7
6: $\boldsymbol{u}_{0:N-1} \leftarrow \Phi$
7: **end if**
8: **return** $[J^\star, \boldsymbol{u}_{0:N-1}]$

7.3.1 Relaxed Optimization Problem with Partial Schedule

In Line 1 of Algorithm 10, we construct a relaxed optimization problem that only involves episodes whose start and end events have both been assigned execution time steps:

Problem 13: Relaxed Optimization Problem for a Partial Schedule σ

$$J^\star(\sigma) = \min_{\boldsymbol{u}_{0:N-1} \in \mathbb{U}^N} \quad J(\boldsymbol{u}_{0:N-1}, \bar{\boldsymbol{x}}_{1:N}, \sigma) \tag{7.2}$$

$$\text{s.t.} \quad \forall t \in \mathbb{T}^-, \quad \boldsymbol{x}_{t+1} = \boldsymbol{A}_t \bar{\boldsymbol{x}}_t + \boldsymbol{B}_t \boldsymbol{u}_t \tag{7.3}$$

$$\bigwedge_{c \in \mathcal{C}} \bigwedge_{a \in (\Psi_c \cap \mathcal{A}(\sigma))} \bigwedge_{t \in \Pi_a(\sigma)} \bigwedge_{k \in \mathcal{K}_a} \bigvee_{j \in \mathcal{J}_{a,k}} \boldsymbol{h}_{c,a,k,j}^T \boldsymbol{x}_t - g_{c,a,k,j} \leq -m_{c,a,k,j}(\delta_{c,a,k}) \tag{7.4}$$

$$\sum_{k \in \mathcal{K}_a, a \in (\Psi_c \cap \mathcal{A}(\sigma))} \delta_{c,a,k} \geq 1 - \Delta_c, \tag{7.5}$$

where $J^\star(\sigma)$ is the optimal objective value of the relaxed subproblem with a partial schedule σ. Recall that $\mathcal{A}(\sigma)$ is the partial episode set of σ, which only involves the episodes whose start and end nodes are both assigned execution time steps by the partial schedule σ (Definition 12). In Section 3.1, we obtained a simplified notation of a chance constraint (3.3) from (3.2) by merging indices. Likewise, we merge the three conjunctions of (7.4) and obtain the following:

$$\bigwedge_{c \in \mathcal{C}} \bigwedge_{i \in \mathcal{I}_c(\sigma)} \bigvee_{j \in \mathcal{J}_{c,i}} \boldsymbol{h}_{c,i,j}^T \bar{\boldsymbol{x}}_{\boldsymbol{t_i}} - g_{c,i,j} \leq -m_{c,i,j}(\delta_{c,i}).$$

Note that this chance constraint is exactly the same as (6.6), except that a partial schedule σ is specified instead of a fully assigned schedule s. Hence, Problem 13 is an instance of a non-convex CCQSP planning problem with a fixed schedule (Problem 10), and can be optimally solved by the NIRA algorithm. Also note that σ is a fully assigned schedule at the leaf node of the branch-and-bound search tree.

The optimal objective value of Problem 13 gives a lower bound of the optimal objective value of all the subsequent subproblems in the branch-and-bound tree. This property is

formally stated in Lemma 8 below. In order to prove this feature, we first define the concept of an *extension* of a partial schedule as follows:

Definition 17. *A schedule $s : \mathcal{E} \mapsto \mathbb{T}$ is an **extension** of a partial schedule $\sigma : \mathcal{E}_\sigma \mapsto \mathbb{T}$ if and only if both assign the same time steps to all the events in the domain of σ:*

$$\sigma(e) \;=\; s(e) \quad \forall e \in \mathcal{E}_\sigma.$$

For example, in Figure 7-3-(b), a fully assigned schedule $\{s(e_0) = 0, s(e_1) = 2, s(e_E) = 4\}$ and $\{s(e_0) = 0, s(e_1) = 2, s(e_E) = 5\}$ is an extension of a partial schedule $\{\sigma(e_0) = 0, \sigma(e_1) = 2\}$.

The following lemma always holds:

Lemma 8. *If a schedule s is an extension of a partial schedule σ, then the optimal objective value of Problem 13 with σ is a lower bound of the optimal objective value with s:*

$$J^\star(\sigma) \leq J^\star(s).$$

Proof: Since σ is a partial schedule, $\mathcal{E}_\sigma \subset \mathcal{E}$, and hence $\mathcal{A}(\sigma) \subseteq \mathcal{A}$. Also, since $\sigma(e) = s(e)$ for all $e \in \mathcal{E}_\sigma$, all the state constraints in the chance constraint (7.4) of Problem 13 with a partial schedule σ are included in the problem with a full schedule s. This means that the feasible state space of the problem with s is a subset of the one with σ. Hence, if the chance constraint (3.7) of the problem with s is satisfied, the chance constraint (7.4) of the problem with σ is also satisfied. Therefore, the problem with σ always results in a better (less) or equal cost than the problem with σ', because the former has looser constraints. ■

For example, in Figure 7-3-(b), e_1 has been assigned an execution time step but e_E has not. Therefore, at node $\sigma(e_1) = 2$, the chance-constrained optimization problem with only the "End in A" episode is solved with the partial schedule $\{\sigma(e_0) = 0, \sigma(e_1) = 2\}$ (see Figure 7-1-(a)). It gives a lower bound of the cost of the problems with the fully assigned schedules $\{s(e_0) = 0, s(e_1) = 2, s(e_E) = 4\}$ and $\{s(e_0) = 0, s(e_1) = 2, s(e_E) = 5\}$.

Algorithm 10 obtains a lower bound by solving Problem 13 exactly using the NIRA algorithm, if it involves episodes with non-convex state regions (Line 3). If the function

is called on a leaf node, Problem 13 is also solved exactly by NIRA even if it does not involve non-convex state constraints. This is because the solutions of leaf subproblems are candidate solutions of an optimal solution of the overall problem. Hence, by solving them exactly, we can ensure the optimality of the branch-and-bound search.

7.3.2 Further Bounding with FRR and BoostLP

If the relaxed subproblem (Problem 13) is convex, then p-Sulus solves the FRR of the subproblem approximately, instead of solving it exactly with NIRA, in order to obtain a lower bound more efficiently (Line 5 of Algorithm 10). Furthermore, if the objective function is linear, BoostLP (see Section 6.5) can be used to solve FRR even more efficiently. In such case, the worst-case estimation error of Boost-LP, ϵ, is subtracted from the estimated lower bound in order to accommodate the estimation error of BoostLP.

Many practical CCQSP execution problems have only one episode that has a non-convex feasible region. For example, in the CCQSP planning problem shown in Figures 1-2 and 1-4, only the "safe region" ($\mathbb{R}^2$ minus the obstacles) is non-convex, while "Provincetown" (start region), "Scenic region," and "Bedford" (goal region) are convex. In such a case subproblems are solved exactly only at the leaf nodes, and their lower bounds are always evaluated by approximate solutions of FRRs of the subproblems at the non-leaf nodes.

7.4 Conclusion

In this chapter, we developed p-Sulu FH, which can solve a CCQSP planning problem with a *flexible* schedule. The scheduling problem was formulated as a combinatorial constrained optimization problem, which is solved by a branch-and-bound algorithm. Each subproblem of the branch-and-bound search is a CCQSP planning problem with a *fixed* schedule, which is solved by NIRA, introduced in Chapter 6. The domain of the feasible schedule is pruned by running a shortest-path algorithm on the d-graph representation of the given temporal constraints. The lower bounds of the optimal objective value of the subproblems are obtained by solving fixed-schedule CCQSP planning problems where a subset of the

state constraints are imposed. We proposed an efficient variable ordering heuristic called the convex-episode-first (CEF) heuristic, which prioritizes convex subproblems over non-convex ones. The simulation results of p-Sulu FH are presented in Sections 11.1.9, 11.2.2, and 11.3.2.

In the next chapter we develop a *real-time* CCQSP executive, p-Sulu, which employs the receding-horizon planning approach.

Chapter 8

Receding Horizon Execution of CCQSP

This chapter presents a novel online plan executive, probabilistic Sulu or *p-Sulu*, which executes a CCQSP in real time with a non-convex state space and a flexible schedule.

The purpose of p-Sulu is to support real-time execution of CCQSPs. Our approach is to employ the receding horizon planning approach [41]: at each planning cycle, the executive computes a control sequence over a short, fixed duration, which is called the *planning horizon*; the vehicle applies the control inputs of the first few steps in the planning horizon, which we refer to as the *execution horizon*; and the executive computes the control sequence for the next planning horizon while the control sequence of the current execution horizon is being executed. At each planning cycle, p-Sulu repeatedly solves a *finite-horizon CCQSP planning problem* by the IRA algorithm, presented in Section 4.2. A finite-horizon CCQSP planning problem is essentially the same as a full-horizon CCQSP planning problem, which is solved by p-Sulu FH, except for several modifications. Such repeated replanning allows the executive to adapt to environmental changes in real time. Moreover, unlike full-horizon planning, receding horizon planning allows the executive to use the posterior distribution of the future state based on a current state estimate. This is particularly important for controlling stochastic systems, since uncertainty accumulates over time. Although the receding horizon planning approach is not new, it has not been used for chance-constrained planning problems. The main technical challenge in receding horizon CCQSP execution is how to find feasible risk allocation without detailed analysis of the planning problem beyond the current planning horizon.

We address this issue by the newly developed *risk budgeting* approach. In this approach, we view the risk bound as a "budget" that can be spent over the plan duration to achieve the given temporally extended goals. At every planning cycle, the executive spends risk out of the budget. The executive keeps track of the availability of risk so that it does not overspend the budget. Such a risk budgeting approach guarantees the satisfaction of chance constraints.

Since p-Sulu does not plan beyond the current planning horizon, the amount of risk spent at the current horizon must be decided heuristically. We present a risk spending heuristic that allocates the risk approximately uniformly over planning horizons while guaranteeing that the executive does not run out of budget during the execution. The risk spending heuristic is particularly suitable for CCQSP planning problems that involve path planning, but can also be used for other domains.

This chapter is organized as follows. Section 8.1 presents the risk budgeting approach, and develops the p-Sulu algorithm. Section 8.2 formulates the finite-horizon CCQSP planning problem, which is solved by p-Sulu at every planning cycle. Section 8.3 presents the solution method of the finite-horizon CCQSP planning problem using IRA. Finally, Section 8.4 develops heuristics for risk budgeting, as well as for guidance. The appendix of this chapter includes technical notes on the implementation of p-Sulu.

8.1 p-Sulu with the Risk Budgeting Approach

As introduced above, the key enabler of p-Sulu is the risk budgeting approach. We first present a brief review of the receding horizon approach in Section 8.1.1. Section 8.1.2 presents the key insights behind risk budgeting, and Section 8.1.3 presents the risk budgeting approach. Using this approach, we develop the p-Sulu algorithm in Section 8.1.4.

8.1.1 Review of the Receding Horizon Approach

A receding horizon planner plans for a fixed number of time steps at every planning cycle, and sequentially extends the plan until all temporally extended goals are achieved, as shown in Figure 8-1. The receding horizon approach has two time horizons: *planning*

horizon and *execution horizon*. A planning horizon means the length of time for which the executive computes a sequence of control inputs at each planning cycle; an execution horizon means the length of time for which the planned control inputs are executed. The control inputs within a planning horizon but beyond the execution horizon are not applied. The computation for the next planning horizon must be finished while executing the control inputs of the current execution horizon. For example, in Figure 8-1-(b), the control inputs for $t = 2 \cdots 5$ must have been computed between $0 \leq t < 2$. Therefore, the length of the execution horizon imposes an upper bound on the computation time. The notion of the receding horizon approach was originally developed for the model predictive control (MPC) [41], where the execution horizon length is typically set to one. Sulu [61] adopts this approach for model-based plan execution, with an extended execution horizon length.

The receding horizon approach allows the executive to use the up-to-date measurement of the system's state. This is a particularly important feature for plan execution under uncertainty since the system under control may reach a different state than predicted due to uncertainty.

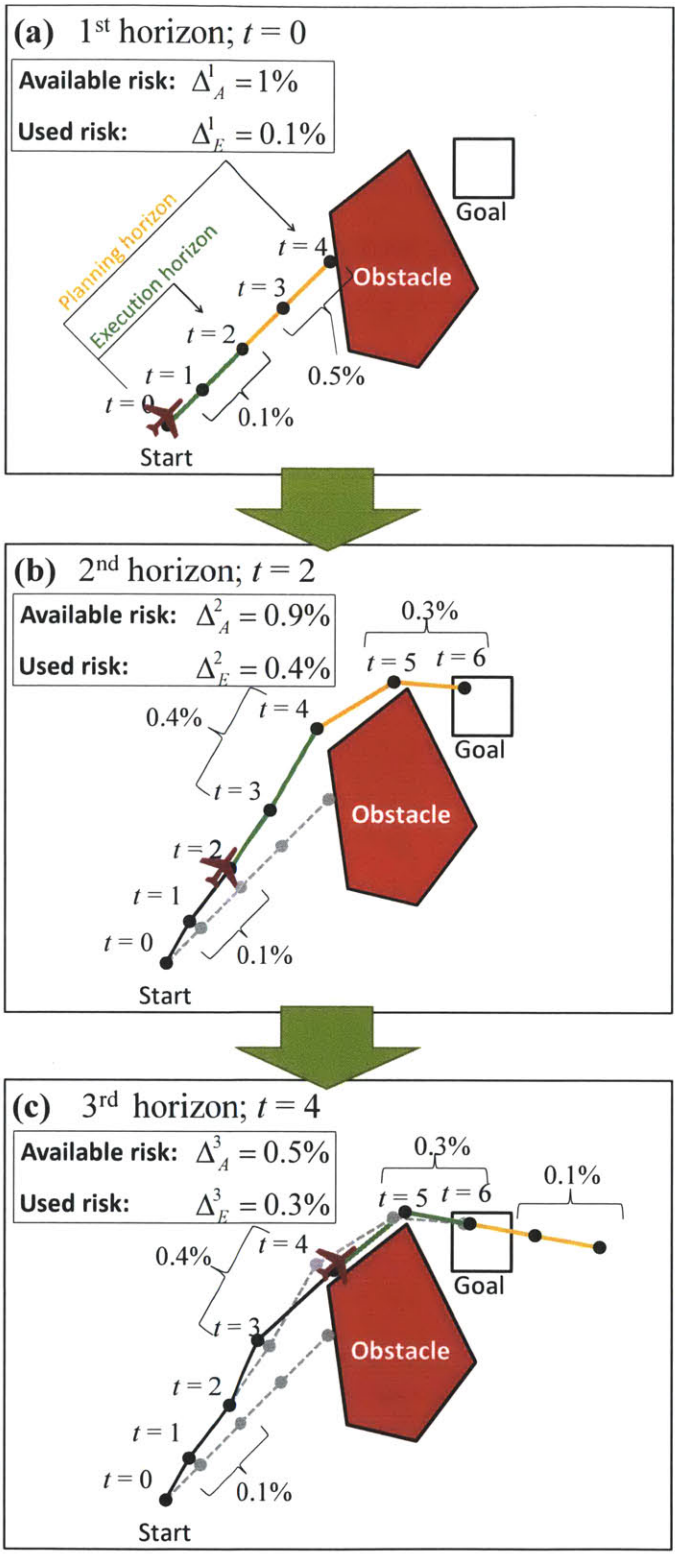


Figure 8-1: Overview of the receding horizon approach with risk budgeting.

8.1.2 Risk Allocation at Replanning

How should we allocate risk to each time step when the receding horizon planning approach is employed?

Consider an example where the user requires the plan executive to reach a goal safely with 99% of probability (hence, the risk bound is $\Delta = 1\%$), as in Figure 8-2. Assume the executive allocated 0.3% of risk to the first execution horizon, and the plan in the first execution horizon is successfully executed. The executive then replans for the second horizon. However, what risk bound should be used for replanning is not straightforward. There are two possible policies, both of which are seemingly valid:

(a) Since the executive has already "consumed" 0.3% of risk, it leaves the remaining 0.7% of risk bound for the rest of the plan.

(b) Since the executive already knows that the execution of the first horizon was successful and the 0.3% risk of failure in the first execution horizon was not realized, it uses the 1% risk bound again for the rest of the plan.

The two policies above is generally stated as follows:

(a) The prior probability of failure at the beginning of the plan should be bound by Δ.

(b) The posterior probability of failure at each replanning cycle should be bound by Δ.

We show that only (a) is valid for receding horizon planning with chance constraints. This conclusion is derived from the fact that the users of the plan executive can *know* in advance about which policy the executive uses for replanning. A receding horizon planner, such as p-Sulu, is an algorithm that repeats replanning in a scheduled manner with a predetermined policy. Therefore, the replanning policy programmed in the algorithm can be known to the users before the beginning of execution. For example, assume that the executive allocates the 0.3% of risk for the first horizon, as in Figure 8-2. Also assume that the user of the executive knows, at the beginning of the execution of the first horizon, that the executive employs the (b) replanning policy, and hence it takes the 1% of risk after the second horizon. Then the user can reason that, at the beginning of the initial execution

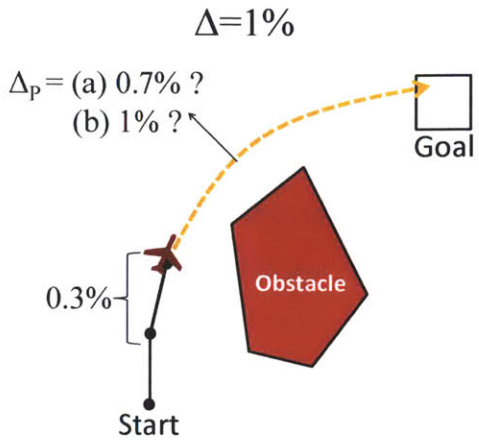


Figure 8-2: Two policies of risk allocation in the replanning. (a) considers that the 0.3% of risk has been "consumed," so the rest of the plan can spend at most the remaining 0.7% of risk. On the other hand, (b) puts the 0.3% of risk back in and uses the 1% risk bound for replanning since the executive knows that the past control sequence has been successfully executed, and the 0.3% risk was not realized. We conclude that the (a) policy is appropriate for p-Sulu.

horizon, the probability of failure throughout the plan is:

$$0.003 + (1 - 0.003) \times 0.01 = 0.01297,$$

which exceeds the specified risk bound, 1%. On the other hand, if the user knows that the executive employs the (a) policy, then the user can reason that the probability of failure throughout the plan is:

$$0.003 + (1 - 0.003) \times 0.007 = 0.009979,$$

which is below the specified risk bound.

A formal discussion follows shortly in the next subsection.

8.1.3 Risk Budgeting

In order to simplify the notation, we consider a problem with only one chance constraint in the following discussion, and hence omit the subscript c from the risk bound Δ_c (Definition

13). It is straightforward to apply the following discussion to the problem with multiple chance constraints. We use the following notations:

- Δ_E^n - The amount of risk allocated to the nth execution horizon, and
- Δ_A^n - The amount of risk that is available at the nth planning cycle.

The (a) policy described in the previous subsection is formally stated by the following procedure, which we call *risk budgeting*:

Algorithm 11 Risk Budgeting

1: $\Delta_A^1 \leftarrow \Delta, n \leftarrow 1$
2: **loop**
3: Choose Δ_E^n so that $0 < \Delta_E^n < \Delta_A^n$.
4: $\Delta_A^{n+1} \leftarrow \Delta_A^n - \Delta_E^n$
5: $n \leftarrow n + 1$
6: **end loop**

Before starting a formal discussion, let us intuitively explain the risk budgeting approach using an example.

Example

We consider the example shown in Figure 8-1 with the risk bound $\Delta = 1\%$. Consider that Δ is the *budget of risk* that can be spent over the plan duration to achieve the specified goal. At the beginning, the available amount of risk is equal to the budget. Hence, $\Delta_A^1 = \Delta = 1\%$ (Line 1 of Algorithm 11). At the first planning cycle, the receding horizon executive allocates $\Delta_E^1 = 0.1\%$ to the first execution horizon (Line 3). Since we "used" the 0.1% of risk, the available risk at the second planning cycle is $\Delta_A^2 = 0.9\%$ (Line 4). Then, at the second planning cycle, the executive allocates $\Delta_E^1 = 0.4\%$ to the first execution horizon (Line 3). Since we "used" the 0.4% of risk out of $\Delta_A^2 = 0.9\%$, the available risk at the third planning cycle is $\Delta_A^3 = 0.5\%$ (Line 4). In this way, the risk budgeting approach keeps track of the availability of risk so that the executive does not overspend the budget.

Justification

We next formally justify the risk budgeting approach.

Assume that the given CCQSP can be executed within a finite number of planning cycles, N. We do not have to know the value of N; it suffices if we know that such N exists. Δ_E^n is a lower bound of the conditional probability that the plan execution fails at the nth horizon, given a successful execution before the nth horizon. We denote by S_n an event where the plan is successful at the nth horizon. Then, at the beginning of the plan execution, the probability that the entire plan is successfully executed, denoted by $P^0_{success}$, is given as follows:

$$\begin{aligned} P^0_{success} &= \Pr[S_N \mid S_{N-1} \wedge \cdots S_1] \cdot \Pr[S_{N-1} \mid S_{N-2} \wedge \cdots S_1] \cdots \Pr[S_1] \\ &\geq \prod_{n=1}^{N} (1 - \Delta_E^n). \end{aligned} \tag{8.1}$$

We prove the following lemma:

Lemma 9. *The risk budgeting approach (Algorithm 11) guarantees that* $P^0_{success} \geq 1 - \Delta$.

Proof: We first prove that the following inequality holds:

$$\prod_{i=1}^{N} (1 - \alpha_i) \geq 1 - \sum_{i=1}^{N} \alpha_i \quad (0 \leq \alpha_i \leq 1, \forall i) \tag{8.2}$$

It is trivial when $N = 1$. When (8.2) holds for $N = k$, it also holds for $N = k+1$ because:

$$\begin{aligned} \prod_{i=1}^{k+1} (1 - \alpha_i) &\geq \left(1 - \sum_{i=1}^{k} \alpha_i\right)(1 - \alpha_{k+1}) \\ &= 1 - \sum_{i=1}^{k+1} \alpha_i + \left(\sum_{i=1}^{k} \alpha_i\right)(\alpha_{k+1}) \geq 1 - \sum_{i=1}^{k+1} \alpha_i. \end{aligned}$$

Therefore, by recursion, (8.2) holds for $N = 1, 2, \cdots$.

At the nth planning cycle, the risk budgeting approach guarantees the following inequality:

$$0 \leq \Delta_E^n \leq \Delta - \sum_{k=1}^{n-1} \Delta_E^k. \tag{8.3}$$

It follows from (8.3) with $n = N$ that:

$$0 \leq \sum_{n=1}^{N} \Delta_E^n \leq \Delta.$$

Furthermore, the existence of such $\Delta_E^1 \cdots \Delta_E^N$ is recursively implied by (8.3) with $n = 1 \cdots N-1$. Hence, using (8.1) and (8.2),

$$P_{success}^0 \geq \prod_{n=1}^{N}(1-\Delta_E^n) \geq 1 - \sum_{n=1}^{N} \Delta_E^n \geq 1 - \Delta.$$

■

On the other hand, the (b) policy in Section 8.1.2 only requires that $\Delta_E^n \leq \Delta$. This is not sufficient to guarantee $P_{success} \geq 1 - \Delta$.

8.1.4 p-Sulu Algorithm

We build p-Sulu algorithm upon the risk budgeting approach introduced in the previous subsection. Recall that a deterministic receding horizon controller obtains the control sequence for the next execution horizon by solving a finite-horizon optimal control problem. p-Sulu obtains the control sequence as well as the risk allocation, Δ_E^n, by solving a finite-horizon CCQSP planning problem. We denote by $\mathcal{P}^n(\Delta_A^n)$ the finite-horizon CCQSP planning problem solved at the nth planning cycle, highlighting that it takes the currently available risk Δ_A^n as a parameter. We view $\mathcal{P}^n(\Delta_A^n)$ as a function that returns the control sequence and the risk allocation at the nth planning cycle:

$$[\boldsymbol{u}^n, \Delta_E^n] = \mathcal{P}^n(\Delta_A^n),$$

where

$$\boldsymbol{u}^n := \boldsymbol{u}_{(n-1)N_E} \cdots \boldsymbol{u}_{(n-1)N_E+N_P-1}.$$

Furthermore, we require that Δ_E^n returned by $\mathcal{P}^n(\Delta_A^n)$ satisfies the condition in Line 3 of the risk budgeting approach (Algorithm 11). The formulation of $\mathcal{P}^n(\Delta_A^n)$ is presented in Section 8.2.

The p-Sulu algorithm is outlined in Algorithm 12 as below:

Algorithm 12 p-Sulu algorithm

function pSuluRH($ccqsp$)

1: $\Delta_A^1 \leftarrow \Delta, n \leftarrow 1$

2: **while** e_E in $ccqsp$ is not executed **do**

3: Wait until $t = (n-1)N_E + 1$

4: $[\boldsymbol{u}^n, \Delta_E^n] \leftarrow \mathcal{P}^n(\Delta_A^n)$

5: Put $\boldsymbol{u}^n$ in the execution queue.

6: $\Delta_A^{n+1} \leftarrow \Delta_A^n - \Delta_E^n$

7: $n \leftarrow n + 1$

8: **end while**

9: **return** $\boldsymbol{U}^\star$

At the beginning of the p-Sulu algorithm, the available amount of risk for the first planning cycle Δ_A^1 is equal to the risk bound specified by the user (i.e., risk budget). At each planning cycle, the algorithm solves the CCQSP planning problem, $\mathcal{P}^n(\Delta_A^n)$ (Line 4). As a result, the algorithm obtains the optimal control inputs, $\boldsymbol{u}^n$, as well as the risk allocation for the next execution horizon, Δ_E^n. The control inputs are put in the execution queue (Line 5). The control inputs in the execution queue are executed by a different thread at the specified time. Following the risk budgeting approach, the algorithm considers that the risk allocated to the current execution horizon, Δ_E^n, is used. Hence, we deduct Δ_E^n from the risk availability, Δ_A^n (Line 6). The algorithm is terminated if all the events in the given CCQSP are executed (Line 2).

In the next section we present the formulation of the finite-horizon CCQSP planning problem, $\mathcal{P}^n(\Delta_A^n)$.

8.2 Receding Horizon Modifications to the CCQSP Planning Problem

Although the finite-horizon CCQSP planning problem solved at the nth planning cycle, $\mathcal{P}^n(\Delta_A^n)$, is essentially the same as the full-horizon CCQSP planning problem (Problem 2 in Section 3.1), it requires four modifications in order to solve the problem in a receding horizon manner. The four modifications are summarized below. They are described in detail shortly.

1. The fixed time horizon $\mathbb{T}$ in Problem 2 is replaced with a receding time horizon.
2. The risk bound Δ is replaced by Δ_P^n, a risk bound for the nth planning horizon.
3. The constraints (3.7), which require executing all the episodes within the current horizon, is relaxed so that the executive can postpone the execution of episodes for later time horizons.
4. A cost-to-go function is added to the objective function as a guidance heuristic.

The first and the fourth are the standard methods for receding horizon control. The third modifications is used by Sulu [61]. We newly added the second modification in order to respect the chance constraints. The four items will be formally discussed in the following four subsections.

8.2.1 Receding Time Horizon

In this subsection we present the first modification, which introduces a receding time horizon.

We denote by N_E and N_P the number of time steps in an execution horizon and a planning horizon, respectively. In $\mathcal{P}^n(\Delta_A^n)$, the receding time horizon,

$$\mathbb{T}^n := \{(n-1)N_E, (n-1)N_E + 1, (n-1)N_E + 2, \cdots (n-1)N_E + N_P\}, \qquad (8.4)$$

is used instead of the fixed time horizon $\mathbb{T}$. For later convenience, we denote by T_0^n and T_F^n the first and the last time steps in the nth planning horizon $\mathbb{T}^n$:

$$\begin{aligned} T_0^n &:= (n-1)N_E \\ T_F^n &:= (n-1)N_E + N_P. \end{aligned}$$

8.2.2 Risk Bound for Planning Horizons

Next, we explain the second modification, which replaces the risk bound Δ by the risk bound for each planning horizon.

If you spend all the money you have today, you will be in trouble tomorrow. Likewise, when using the risk budgeting approach, the executive must leave sufficient risk budget for future planning horizons.

More concretely, the risk bound Δ in the finite-horizon CCQSP planning problem (Problem 2 in Section 3.1) is replaced by Δ_P^n in $\mathcal{P}^n(\Delta_A^n)$, where Δ_P^n is a constant that satisfies:

$$0 < \Delta_P^n < \Delta_A^n. \tag{8.5}$$

The constant Δ_P^n represents the risk bound for the nth *planning* horizon. Note that it is different from Δ_E^n, which is the amount of risk allocated to the nth *execution* horizon. In the nth planning cycle, p-Sulu solves a finite-horizon CCQSP planning problem, $\mathcal{P}^n(\Delta_A^n)$, and obtains the risk allocation within the planning horizon. Let δ_t^n be the risk allocated to the tth time step by solving $\mathcal{P}^n(\Delta_A^n)$. Then, δ_t^n must satisfy the following constraint:

$$\sum_{t=T_0^n}^{T_F^n} \delta_t^n \leq \Delta_P^n.$$

Then, let $\delta_t^{n\star}$ be the optimal solution obtained by solving $\mathcal{P}^n(\Delta_A^n)$. The amount of risk used in the n th horizon, Δ_E^n, is given as follows:

$$\Delta_E^n = \sum_{t=T_0^n}^{T_0^n+N_E-1} \delta_t^{n\star}.$$

This Δ_E^n obtained as above is returned in Line 4 of p-Sulu (Algorithm 12).

There is one remaining question: how to choose Δ_P^n?

Theoretically, it can be any value that satisfies (8.5). However, if Δ_P^n is too close to Δ_A^n, little risk is left for future planning cycles, and in the worst case, the future CCQSP planning problems may become infeasible. On the other hand, if Δ_P^n is too close to zero, the CCQSP planning problem at the current planning cycle may become infeasible, or at best, the resulting plan for the next execution would be overly risk-averse. It is very hard to optimize the choice of Δ_P^n for a general problem. Hence, it must be chosen heuristically for each specific problem. We pick a heuristic function $0 < \alpha(\boldsymbol{x}_{T_0^n}) < 1$, and let

$$\Delta_P^n = \alpha(\boldsymbol{x}_{T_0^n})\Delta_A^n. \tag{8.6}$$

In Section 8.4, we present such a heuristic function $\alpha(\boldsymbol{x}_{T_0^n})$ for path planning problems.

Example

Consider again the example shown in Figure 8-1, where $N_E = 2$, $N_P = 4$, and $\Delta = 1\%$. At the first planning cycle, p-Sulu solves $\mathcal{P}_1(1\%)$ with $\Delta_P^1 = 0.6\%$ (Line 4 in Algorithm 12). As a result, the algorithm obtains the plan shown in Figure 8-1-(a), which allocates $\Delta_E^1 = 0.1\%$ of risk to the first execution horizon.

While the control inputs for the first execution horizon are being executed, p-Sulu goes through the second planning cycle. Since 0.1% of risk has been consumed in the first execution horizon, the remaining risk is 0.9%. p-Sulu solves $\mathcal{P}_2(0.9\%)$ with $\Delta_P^2 = 0.7\%$, and obtains the solution shown in Figure 8-1-(b). This time, $\Delta_E^1 = 0.4\%$ of risk is allocated to the second execution horizon.

During the execution of the control inputs in the second execution horizon, p-Sulu plans for the third planning cycle. Since 0.4% of risk has been consumed in the previous execution horizon, the remaining risk is 0.5%. p-Sulu solves $\mathcal{P}_3(0.5\%)$ with $\Delta_P^2 = 0.4\%$, and obtains the solution shown in Figure 8-1-(c). Since the goal is reached within the execution horizon, the algorithm is terminated after the execution of the control inputs in the third execution horizon.

8.2.3 Deferment of Episode Executions

Next, we present the third modification, which allows deferments of episode executions.

Recall that the chance constraints (3.7) in Problem 2 require that all the episodes in a CCQSP must be executed within a time horizon $\mathbb{T}$ with the probability Δ_c. However, in a receding horizon approach, the failure to execute an episode within the current time horizon does not mean the failure of plan execution, since it can be executed in later time horizons. Therefore, the constrains must allow executives to postpone the execution of an episode if it cannot be executed in the current time horizon. Such a constraint is called a soft constraint since it allows deferments of episode execution. At the same time, whenever an episode can be executed at the current horizon, the executive must not postpone its execution since there is no guarantee that the episode is still feasible with regard to state and chance constraints at future time steps. Therefore, we penalize deferments of episode execution. By setting the penalty sufficiently large, we can assure that the executive defers episode executions only when it is impossible. Recall that the formulation of each chance constraint in (3.7) is obtained by simplifying the notation of (3.2). In $\mathcal{P}^n$, (3.2) is replaced with the following soft constraint:

$$\Pr\left[\bigwedge_{a\in\Psi_c}\left\{\left(\bigwedge_{t\in\Pi_a(s)}\bigwedge_{k\in\mathcal{K}_a}\bigvee_{j\in\mathcal{J}_{a,k}}\boldsymbol{h}_{c,a,k,j}^T\boldsymbol{x}_t - g_{c,a,k,j}\le 0\right)\vee\left(s(e_a^E) > T_F^n \wedge p_a = M\right)\right\}\right] \ge 1-\Delta_P^n,$$

where the variable $p_a \ge 0$ represents the penalty of postponing the execution of the episode a, and M is a large positive constant number. This constraint requires that, for each episode a, its state constraints must be satisfied, or the execution time of its end event $s(e_a^E)$ must be after the current planning horizon with an penalty M, with the probability of $1-\Delta_c$.

The penalty, $p_a \ge 0$, must be added to the objective function (3.4) as follows:

$$\min_{\boldsymbol{u}_{0:N-1}\in\mathbb{U}^N, s\in\mathcal{S}_F, p_a\ge 0} J(\boldsymbol{u}_{0:N-1}, \bar{\boldsymbol{x}}_{1:N}, s) + \sum_{a\in\mathcal{A}} p_a$$

Note that $p_a \ge 0$ is included in the decision variables.

8.2.4 Guidance Heuristic

Finally, we turn to the fourth modification, which introduces a guidance heuristic.

Consider that a goal $R_{a_{next}}$ should be achieved next, but it is unreachable in the current planning horizon. In such case, the soft constraint explained above allows the executive to postpone the execution of a_{next}. The issue here is that the soft constraint does not guarantee that the plant state evolves towards the goal region $R_{a_{next}}$; it might go in the wrong direction away from $R_{a_{next}}$.

In order to guide the plant state towards the next goal $R_{a_{next}}$, we introduce to the objective function a cost-to-go function $d(\bar{\boldsymbol{x}}_{T_F^n}, R_{a_{next}})$, which approximates the required cost-to-go from the final state of the current planning horizon $\bar{\boldsymbol{x}}_{T_F^n}$ to the next goal region $R_{a_{next}}$, as follows:

$$\min_{\boldsymbol{u}_{0:N-1}\in\mathbb{U}^N, s\in\mathcal{S}_F, p_a\geq 0} J(\boldsymbol{u}_{0:N-1}, \bar{\boldsymbol{x}}_{1:N}, s) + \sum_{a\in\mathcal{A}} p_a + d(\bar{\boldsymbol{x}}_{T_F^n}, R_{a_{next}})$$

The next episode a_{next} is chosen in the following process. As explained in 8.2.2, each episode must be executed at the earliest possible time step since its feasibility with regard to state and chance constraints at future time steps is not guaranteed. Hence, the next episode to be executed is likely to be the one that has the lowest lower bound of the execution time. p-Sulu updates the temporal bounds at each planning cycle by running a shortest-path algorithm, as described in Section 7.2.1. Then it sorts all unexecuted episodes by the lower bound of the execution time of their start events in ascending order. Among them, p-Sulu chooses the first episode whose goal region $R_{a_{next}}$ cannot be reached in the current planning horizon as a_{next}. For example, in Figure 8-3, there are two unachieved (i.e., unexecuted), unreachable goals. p-Sulu chooses the one that is allowed to be executed earlier as the next goal.

Then, what function should we use to approximate the cost-to-go? There are special cases where the exact cost-to-go can be obtained. An example is an optimal control problem with a quadratic cost and no constraints on state or control. In that case, the cost-to-go is obtained by solving the algebraic Riccati equation. However, when there are state constraints, the cost-to-go cannot be obtained in a closed form in general. The temporal and

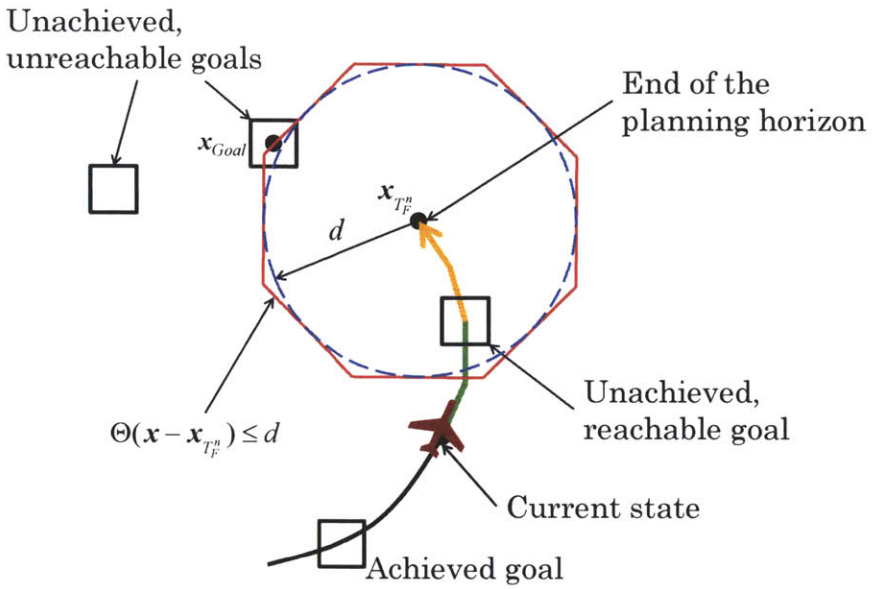


Figure 8-3: Guidance heuristic for receding horizon path planning problem, which guides the vehicle to the next goal.

chance constraints in CCQSP further complicate the problem. Hence, a guidance heuristic, $d(\bar{\boldsymbol{x}}_{T_F^n}, R_{a_{next}})$, must be used. Guidance heuristics is a well studied subject in receding horizon control literatures. We present the heuristic used for p-Sulu in Appendix A of this chapter.

8.3 Solving Finite-horizon CCQSP Problems with IRA

Recall that p-Sulu solves a finite-horizon CCQSP planning problem, $\mathcal{P}^n$, at every planning cycle (Line 4 of Algorithm 12). Such a problem can be solved in two ways. The first one is to use p-Sulu FH (Chapter 7), and the second one is to use the IRA algorithm (Chapter 4) on top of Sulu [61]. The former has an advantage with regard to optimality, while the advantage of the latter is the fast rate of convergence. We choose to use IRA since computation time is more important than optimality in the real-time execution, as we discussed in the introduction to this chapter. Moreover, we cannot expect optimality even if we use p-Sulu FH at each iteration, since the risk allocation between planning horizons cannot be optimized, as explained in Section 8.4. IRA fits to the receding horizon planning since

it is an anytime algorithm. Therefore, p-Sulu runs as many iterations of IRA as possible within each planning horizon to obtain the best available solution for a given replanning frequency.

Recall that IRA can be implemented on top of any existing deterministic solvers to turn them into joint chance-constrained programming solvers (See Section 4.2). The deterministic version of the CCQSP planning problem $\mathcal{P}^n$ has been solved by the QSP executive Sulu by encoding the problem into a mixed integer linear programming (MILP) [61], assuming that the objective function is linear. Hence, the CCQSP planning problem $\mathcal{P}^n$ can be solved by the IRA algorithm by replacing Problem 5 with the MILP encoding of the deterministic QSP planning problem. See Section 5 of [61] for detailed description of the MILP encoding.

8.4 Heuristic Risk Spending Approach

Although p-Sulu can solve general CCQSP planning problems, there are two components in the algorithm that must be tuned for each specific problem. One is the heuristic function $\alpha(\boldsymbol{x}_{T_0^n})$ to choose the risk bound of planning horizons Δ_P^n (see (8.6) in Section 8.2.2), and the other one is the guidance heuristic $d(\bar{\boldsymbol{x}}_{T_F^n}, R_{a_{next}})$ (see Section 8.2.4). Although the latter has been studied extensively in the receding horizon control literatures (e.g., [57, 61]), there is no prior work on the former topic to the best of our knowledge. This section proposes a risk spending heuristic that allocates the risk approximately uniformly over planning horizons while guaranteeing that the executive does not run out of budget during the execution. Since our main application of p-Sulu is a CCQSP execution problems that involves path planning, such as the planning for PTS, we focus on path planning problem in this section. The guidance heuristic used for p-Sulu is explained in Appendix A of this chapter.

In order to find a suitable heuristic, we take an approach to combine two elementary heuristics that have complementary features: one that allocates a constant amount of risk, and another one that allocates a constant proportion of the remaining risk. We first present the two elementary heuristics, and then combine the two.

8.4.1 Uniform-Amount Heuristic

If you have a $100 budget for a 10-day long project, a plausible spending schedule is to use $10 each day. Likewise, if we know the number of planning cycles required to complete the given plan *a priori*, then a plausible heuristic is to allocate risk uniformly among the planning cycles so that

$$\Delta_P^n = \Delta/N,$$

or equivalently, at the nth planning cycle,

$$\Delta_P^n = \frac{1}{N-n+1}\Delta_A^n,$$

where N is the number of the planning cycles. Recall that Δ_A^n is the remaining risk at the nth planning cycle (see Algorithm 11 in Section 8.1.3). The reason for this strategy is that, without additional analysis of the planning problem beyond the current planning horizon, there is no basis for allocating more risk to one horizon over another.

However, N is usually not known *a priori*. Assuming that the path is close to a straight line and a vehicle travels at its maximum speed in order to minimize the flight duration, the fraction $\frac{1}{N-n+1}$ can be approximated by the ratio of the maximum length of the path within the current planning horizon, $d_{horizon}$, to the distance to the final destination, d_{Goal}, as shown in Figure 8-4. Hence, we approximate the above heuristic by the following:

$$\Delta_P^n = \frac{d_{horizon}}{d_{Goal}}\Delta_A^n.$$

The maximum traveling distance in a planning horizon can be obtained as follows:

$$d_{horizon} = v_{max} N_P \Delta t, \tag{8.7}$$

where v_{max} is the maximum speed of the vehicle.

Since this heuristic allocates risk approximately uniformly, the vehicle does not become too risk-averse or too risk-taking at each planning cycle. A drawback of this heuristic is that Δ_P^n can exceed Δ_A^n; another drawback is that, since a vehicle does not always fly straight

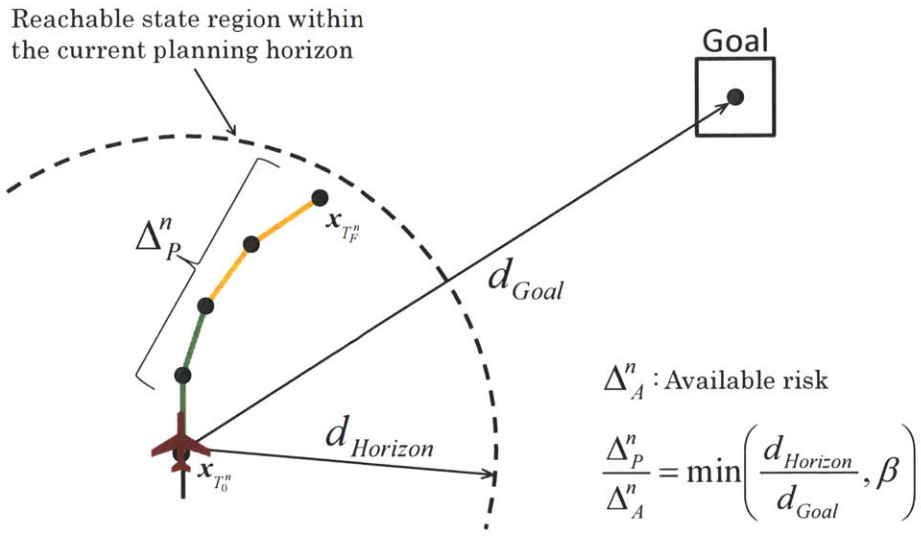


Figure 8-4: Heuristic to determine the risk allocation to the current planning horizon.

at its maximum speed, it often overestimates the uniform amount of risk, $\Delta_P^n = \Delta/N$, particularly near the end of the plan.

8.4.2 Uniform-Proportion Heuristic

If you spend half of the remaining budget every day (i.e., \$50 on the first day, \$25 on the second day...), you never run out of money even if you don't know the duration of the project, although the amount you can spend decreases over time. Likewise, the second elementary heuristic allocates the uniform proportion of the remaining risk at each planning cycle as follows:

$$\Delta_P^n = \beta \Delta_A^n,$$

where β is a constant in$(0, 1)$. This heuristic does not require any knowledge about the plan duration. Moreover, Δ_P^n never exceeds Δ_A^n by definition. A drawback of this heuristic is that, since it allocates decreasing amount of risk towards the end of the plan, the vehicle is risk-taking at the beginning but becomes increasingly risk-averse.

8.4.3 Combined Heuristic

We can have the best of the two worlds by combining the two elementary heuristics as follows:

$$\Delta_P^n = \min\left(\frac{d_{horizon}}{d_{Goal}}, \beta\right)\Delta_A^n.$$

We still need to tune the constant β. Empirically, $\beta = 0.5$ works well for most path planning problems demonstrated in Chapter 11.

In the path planning domain, we use the Euclidean distance for $d_{horizon}$ and d_{Goal}. This heuristic would be applicable to other domains by using an appropriate definition of distance.

8.5 Conclusion

This chapter introduced a CCQSP executive, p-Sulu, which can solve a flexible-schedule CCQSP planning problem with non-convex constraints and a receding planning horizon. The key enabler for p-Sulu was the risk budgeting approach, which guarantees the satisfaction of chance constraints with the receding planning approach. The simulation results of p-Sulu are presented in Sections 11.1.10 and 11.2.3. This is the last chapter that deals with centralized algorithms. Starting from the next chapter, we present decentralized algorithms for multi-agent problems.

Appendix

A. Guidance Heuristic for Path Planning Problems

A standard choice of the guidance heuristic $d(\bar{\boldsymbol{x}}_{T_F^n}, R_{a_{next}})$ in vehicle path planning problems is the distance between the final state of the current planning horizon $\bar{\boldsymbol{x}}_{T_F^n}$ and the center of the next goal region $R_{a_{next}}$ (e.g. [57, 61]). More specifically, let $\boldsymbol{x}_{Goal}$ be the geometric center of $R_{a_{next}}$ and $||\cdot||$ be the Euclidean norm; then the distance-based guidance

heuristic is given as follows:

$$d(\bar{\boldsymbol{x}}_{T_F^n}, R_{a_{next}}) = \gamma ||\boldsymbol{x}_{Goal} - \bar{\boldsymbol{x}}_{T_F^n}||, \tag{8.8}$$

where a constant γ is the weight, which tunes the balance between the cost within the current planning horizon J' and the heuristic cost. The weight must be set appropriately for each application.

Piecewise linear approximation As is explained in the next section, p-Sulu reformulates $\mathcal{P}^n$ into a mixed integer linear programming (MILP). Hence, we need to obtain a piecewise linear approximation of the non-linear guidance heuristic function (8.8). A piecewise linear objective function can be equivalently replaced with a linear objective function, with an additional set of linear constraints. As a result, we use the following objective function and the additional constraints, with a slack variable d:

$$\min_{\boldsymbol{u}_{0:N-1} \in \mathbb{U}^N, s \in \mathcal{S}_F, p_a \geq 0, d} \quad J(\boldsymbol{u}_{0:N-1}, \bar{\boldsymbol{x}}_{1:N}, s) + \sum_{a \in \mathcal{A}} p_a + d \tag{8.9}$$

$$\text{s.t.} \quad \Theta(\boldsymbol{x}_{Goal} - \bar{\boldsymbol{x}}_{T_F^n}) \leq d, \tag{8.10}$$

where Θ is an m-by-two matrix:

$$\Theta = \begin{bmatrix} \cos(0) & \sin(0) \\ \cos(\frac{2\pi}{m}) & \sin(\frac{2\pi}{m}) \\ \cos(\frac{2\cdot 2\pi}{m}) & \sin(\frac{2\cdot 2\pi}{m}) \\ \vdots & \vdots \\ \cos\{\frac{(m-1)\cdot 2\pi}{m}\} & \sin\{\frac{(m-1)\cdot 2\pi}{m}\} \end{bmatrix}.$$

Intuitively, as shown in Figure 8-3, $\Theta(\boldsymbol{x} - \bar{\boldsymbol{x}}_{T_F^n}) \leq d$ represents a regular m-sided polygon centered at $\bar{\boldsymbol{x}}_{T_F^n}$ whose circumscribed circle has the radius d. The constraint (8.10) requires that $\boldsymbol{x}_{Goal}$ is inside of the polygon. Since the objective function includes d, the optimizer places $\bar{\boldsymbol{x}}_{T_F^n}$ so that the radius of the circle can be small. Observe that d is an approximation of the distance between $\bar{\boldsymbol{x}}_{T_F^n}$ and $\boldsymbol{x}_{Goal}$. Hence, $\bar{\boldsymbol{x}}_{T_F^n}$ navigates towards the goal region in

order to reduce d.

Again, we use the Euclidean distance in the path planning domain. This heuristic is applicable to other domains by using an appropriate definition of distance.

8.5.1 B. Guidance with obstacle avoidance

The distance-based guidance heuristic may not work when there are obstacles in the environment. Consider a situation in Figure 8-5-(a), where there is a large obstacle between the current position and the next goal. In such case, the vehicle may get stuck in front of the obstacle as shown in Figure 8-5-(a), since it simply tries to minimize the distance between the tip of the planning horizon and the next goal. We avoid this problem by running a global path planner, such as Dijkstra's algorithm with a visibility graph, to place waypoints[1] at the corners of the obstacles, as shown in Figure 8-5-(b). In the demonstration on the PTS scenario in Chapter 11, we use the Kirk planner [51] to generate such waypoints.

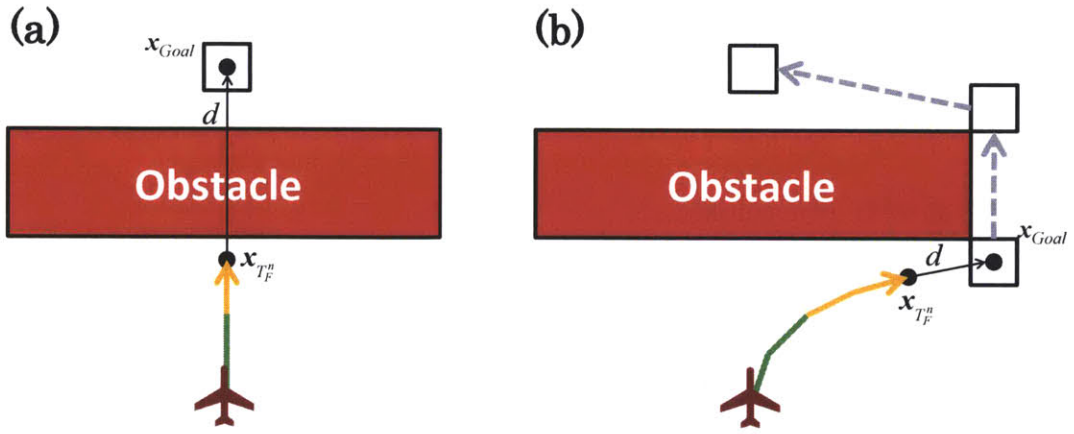


Figure 8-5: (a) The vehicle may get stuck in front of an obstacle when using the distance-based heuristic. (b) This issue can be addressed by placing waypoints at the corners of the obstacle.

Alternatively, more complicated guidance heuristic can achieve obstacle avoidance without the help of a global path planner. Examples of such cost-to-go functions include

[1]Precisely speaking, we convert the way*points* generated by a global path planner into intermediate goal *regions*.

the distance map [61] and the potential field [88]. However, application of such a complex guidance heuristic is beyond the scope of this thesis.

C. Exclusion of Unreachable Obstacles and Goals

We introducing a technique to reduce the computational burden of p-Sulu by excluding unreachable obstacles and goals. This technique is discussed extensively in Chapter 5 of [61]. Here we present a brief overview of the technique.

Obstacle reachability Recall that an obstacle is represented by a disjunctive set of linear constraints as follows (see Figure 3-1):

$$\bigvee_{j\in\mathcal{J}} \boldsymbol{h}_j^T \boldsymbol{x} - g_j \leq 0.$$

Without loss of generality, we can assume that $||\boldsymbol{h}_j|| = 1$. Let $\boldsymbol{x}_{current}$ be the current state, and v_{max} be the maximum speed. The obstacle is unreachable if:

$$\bigvee_{j\in\mathcal{J}} \boldsymbol{h}_j^T \boldsymbol{x}_{current} - g_j \leq -v_{max} \cdot N_P \Delta t.$$

Note that $N_P \Delta t$ is the duration of the planning horizon. The above condition means that an obstacle is unreachable if at least one of the boundary conditions of the obstacle cannot be violated within a planning horizon.

Goal region reachability Recall that a goal region is represented by a conjunctive set of linear constraints as follows:

$$\bigwedge_{j\in\mathcal{J}} \boldsymbol{h}_j^T \boldsymbol{x} - g_j \leq 0.$$

Again, we assume that $||\boldsymbol{h}_j|| = 1$. Then, the goal region is unreachable if:

$$\bigvee_{j\in\mathcal{J}} \boldsymbol{h}_j^T \boldsymbol{x}_{current} - g_j \geq v_{max} \cdot N_P \Delta t.$$

The above condition means that a goal region is unreachable if at least one of the boundary conditions of the region cannot be satisfied within a planning horizon.

At each iteration, p-Sulu checks the reachability of all obstacles and goal regions, and includes only the reachable ones in the optimization problem $\mathcal{P}^n$.

Chapter 9

Distributed CCQSP Planning with a Convex State Space and a Fixed Schedule

This chapter presents the market-based iterative risk allocation (MIRA) algorithm, which solves the full-horizon CCQSP planning problem with multiple agents in a *distributed* manner.

Multi-agent CCQSP planning has a wide range of applications, in addition to our main motivating example, PTS. Consider as an example the problem of planning and control of a power grid [12]. A power grid consists of a number of generators and electric transformers whose control should be carefully planned in order to maximize efficiency. A significant issue that arises for power grid planning is the uncertainty in demand for energy by consumers. As the use of renewable energy, such as solar and wind power, becomes more popular, uncertainty in supply increases due to weather conditions. Another example is the Autonomous Ocean Sampling Network (AOSN) [93], which consists of multiple automated underwater vehicles (AUVs), robotic buoys, and aerial vehicles. AOSN should maximize science gain while avoiding risks of losing vehicles that are exposed to external disturbances, such as tides and currents.

The chance-constrained optimization problem is particularly made challenging when multiple agents are coupled through a joint chance constraint, which limits the combined

probability of constraint violation by any of the agents in the system. In order to address this issue, we extend the concept of risk allocation to multi-agent systems by using the metaphor of risk as a resource that is traded in a computational market. The equilibrium price of risk that balances the supply and demand is found by an iterative price adjustment process called *tâtonnement* (also known as a *Walrasian auction*).

Such a market process can be mathematically framed as a dual decomposition, which is an extension of the method of Lagrange multipliers. In the dual decomposition framework, the price of risk is represented by the dual value, or equivalently the Lagrange multiplier. A central module of MIRA announces a price to the agents. Each agent computes its optimal demand for risk at the price, and conveys the demand to the central module. Then, the central module updates the price based on the balance of the supply and the aggregate demand. The problem of finding the equilibrium price is mathematically framed as a root-finding problem on the aggregate demand function, which also corresponds to the dual optimization problem. The control sequence is obtained by solving an optimal control problem with a price as a parameter. Such a control sequence is optimal at the equilibrium price.

MIRA gives exactly the same optimal solution as the centralized optimization approach since it reproduces the KKT conditions[1] of the centralized approach. Although the algorithm has a centralized part, it typically uses less than 0.1% of the total computation time. We give a proof of the existence and optimality of the solution of our decentralized problem formulation, as well as a theoretical guarantee that MIRA can find the solution. We also provide a complexity analysis of MIRA to demonstrate its advantage in scalability over a centralized approach.

[1]Although KKT conditions are necessary conditions for optimality in general, in our case they are necessary and sufficient conditions for optimality since our optimization problem is convex.

9.1 Overview of Market-based Iterative Risk Allocation

9.1.1 Risk allocation for multi-agent system

We first present the concept of risk allocation for multi-agent systems intuitively, using an example shown in Figure 9-1. The example involves a multi-agent system with two unmanned air vehicles (UAVs), whose mission is to extinguish a forest fire. A water tanker drops water while a reconnaissance vehicle monitors the fire with its sensors. The loss of either vehicle results in a failure of the mission. Two vehicles are required to extinguish the fire as efficiently as possible, while limiting the probability of mission failure to a given risk bound, say, 0.1%. The water tanker can improve efficiency by flying at a lower altitude, but it involves risk. The reconnaissance vehicle can also improve the data resolution by flying low, but the improvement of efficiency is not as great as the water tanker. In such a case a plausible plan is to allow the water tanker to take a large portion of risk by flying low, while keeping the reconnaissance vehicle at a high altitude to avoid risk. This is because the utility of taking risk (i.e. flying low) is greater for the water tanker than for the reconnaissance vehicle.

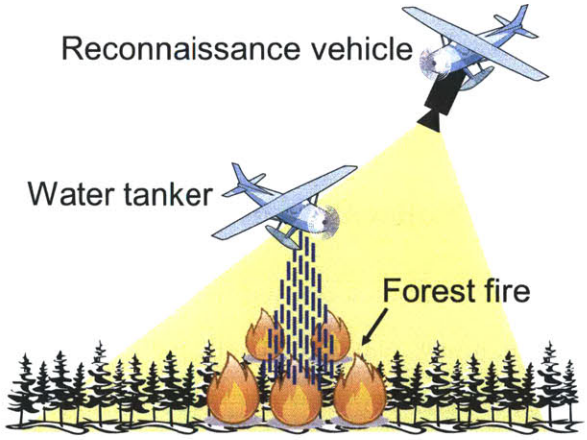


Figure 9-1: Risk allocation for multi-UAV fire-fighting system. The water tanker is allowed to fly low since it is allocated larger risk than the reconnaissance vehicle.

As shown in Figure 9-2, the risk allocated to each agent is reallocated internally to its constraints. The control sequence is optimized so that it is consistent with the internal risk allocation.

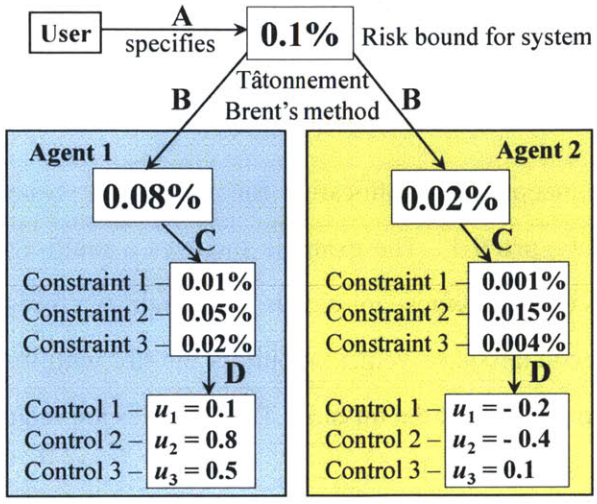


Figure 9-2: Distribution of risk in MIRA. **A**: The user specifies the risk bound for the multi-agent system. **B**: The specified risk bound is distributed among agents. **C**: Each agent optimizes internal risk allocation. **D**: Each agent optimizes the control sequence so that it is consistent with the internal risk allocation.

The optimal risk allocation for a multi-agent system can be found in a *centralized* manner by applying the same algorithm as the single agent problems, such as IRA (Chapter 4), CRA (Chapter 5), or NIRA (Chapter 6). The objective of this chapter is to develop a *decentralized* algorithm that finds the globally optimal risk allocation among multiple agents.

9.1.2 Market-based risk allocation using tâtonnement

Our dual decomposition-based approach is interpreted intuitively as a market-based algorithm. We assume a computational market that sells the risk specified in the joint chance constrains; in the market, each agent demands risk in order to improve its own performance. However, an agent cannot take risk for free; it has to purchase it from the market at a given price.

Agents are price takers, meaning that each agent takes a price as an input, and computes the optimal amount of risk to take (i.e., *demand for risk*) by solving the optimization problem, where the objective function is the agent's utility minus the payment for the risk. The optimal action sequence and the internal risk allocation are also determined by solving the

optimization problem, just as in the single-agent case described before. The demand from each agent can be seen as a function of the price of risk (*demand curve*). Typically, the higher the price is, the less each agent demands. Each agent has a different demand curve according to its sensitivity to risk. The supplier of the risk is the user; she supplies the fixed amount of risk by specifying the upper bound of risk that the multiple agent system can take.

The objective of the market is to find an equilibrium price. At the equilibrium price, the system utility, which is the sum of the utilities of all agents, is maximized. To this end, the market iteratively adjusts the price so that the total demand (*aggregate demand*) becomes equal to the supply. The equilibrium price is found by an iterative process called *tâtonnement* or a *Walrasian auction* [92] as follows:

1) Increase the price if aggregate demand exceeds supply, or

   decrease the price if supply exceeds aggregate demand.

2) Repeat until supply and demand are balanced.

In the classical tâtonnement approach, the price increment is obtained by simply multiplying the excess aggregate demand by a constant. However, an upper bound value on this constant that guarantees convergence is problem specific and hard to determine. The slow convergence rate of the approach is also an issue.

We observe that finding an equilibrium price is a root-finding problem; the problem is to find a price that brings the excess demand, which is a function of the price, to zero. We solve the problem by Brent's method, a commonly-used root-finding algorithm with a fast and guaranteed convergence [6]. At each iteration, the price increment is obtained by computing one step of Brent's method.

Figure 9-3 gives the graphical interpretation of the market-based risk allocation for a system with two agents. Assume that Agents 1 and 2 have different utilities associated with the same risk, and hence have different demand curves. The aggregate demand curve is obtained by adding the two demand curves horizontally. The supply curve is a vertical line since it is constant. The equilibrium price $p^\star$ lies at the intersection of the aggregate demand curve and the supply curve. The optimal risk allocation for the two agents corresponds to their demands at the equilibrium price ($\Delta_1^\star$ and $\Delta_2^\star$ in Figure 9-3).

In our approach, each agent is self-interested, meaning that it considers to minimize its own cost. Nonetheless, it is proven in Section 9.3.2 that the cost of the entire system is minimized at the equilibrium price, even though all the agents behave in a self-interested way. The only information that the central module and the agents need to communicate in each iteration is price and demand.

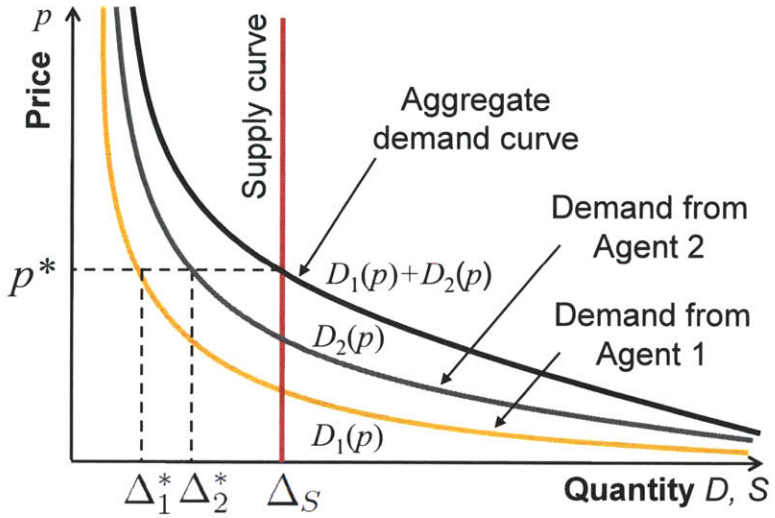


Figure 9-3: Market-based risk allocation in a system with two agents. Note that we followed the economics convention of placing the price on the vertical axis. The equilibrium price is $p^\star$, and the optimal risk allocation is $\Delta_1^\star = D_1(p^\star)$ for Agent 1 and $\Delta_2^\star = D_2(p^\star)$ for Agent 2.

9.1.3 MIRA - Decentralized optimization of risk allocation

Our proposed algorithm, MIRA (Market-based Iterative Risk Allocation), concurrently optimizes risk allocation between agents, internal risk allocation of each agent, and action sequences of each agent.

Figure 9-4 illustrates the Market-based Iterative Risk Allocation (MIRA) algorithm. The tâtonnement process is repeated until it converges to the equilibrium price. Risk is not allocated until the algorithm converges. At each iteration, the optimal demand for risk is computed by solving an optimization problem. The optimal action sequence and the internal risk allocation for each agent are also obtained by solving the same optimization

problem (Step 2 in the Figure 9-4).

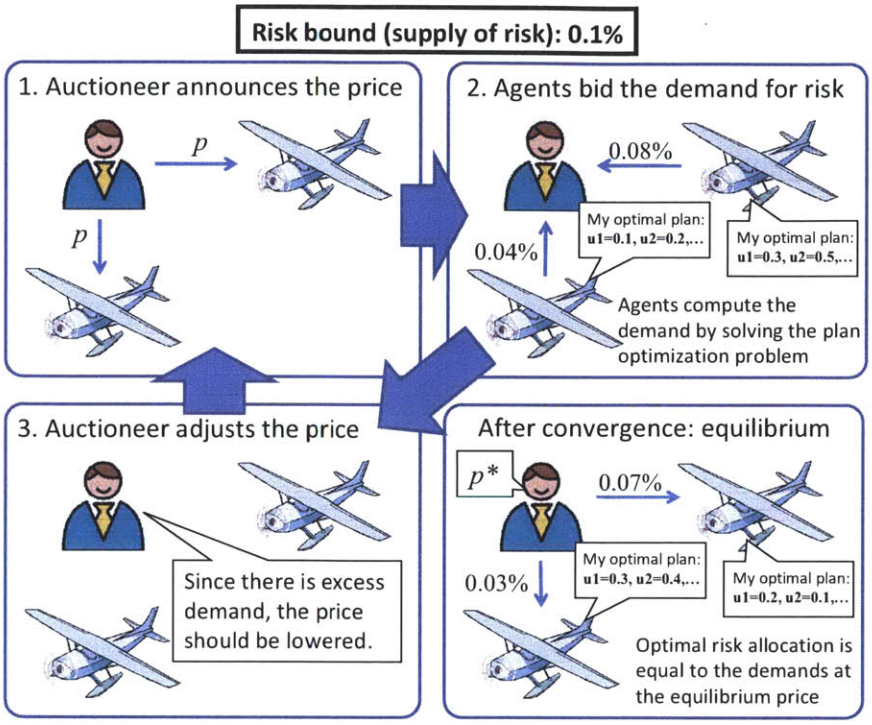


Figure 9-4: Illustration of MIRA algorithm. Risk is allocated to agents through tâtonnement; their continuous action sequences are also optimized in the loop when computing the demand at the given price.

9.2 Problem Formulation

MIRA solves in a distributed manner a CCQSP planning problem with convex state constraints and a fixed schedule. As described in Section 3.4, such a multi-agent problem is formulated as Problem 4. In this chapter, we consider a special case where there is no coupling between agents through state constraints; state-constraint coupling is addressed in Chapter 10. In order to simplify the discussion, we assume for now that there is only one chance constraint. We present the extension to multiple chance constraints in Section 9.6.

In order to define the multi-agent problem, we let

$$\mathcal{L} := \{1, 2, \cdots M\}$$

be the set of indices of M agents in the system. We use a superscript l to specify an agent. We also let $\mathcal{I}^l$ be the set of state constraints imposed on the lth agent. In this chapter we denote the overall risk bound by Δ^S, instead of Δ, in order to highlight that it corresponds to the *supply* of risk. The multi-agent CCQSP planning problem discussed in this chapter is formulated as follows:

Problem 14: Multi-agent CCQSP Planning Problem with a Fixed Schedule and a Convex State Space

$$\min_{\boldsymbol{u}_{0:N-1}^{1:M} \in \mathbb{U}^{N \cdot M}} \quad \sum_{l \in \mathcal{L}} J^l(\boldsymbol{u}_{0:N-1}^l, \bar{\boldsymbol{x}}_{l:N}^l) \tag{9.1}$$

$$\text{s.t.} \quad \forall t \in \mathbb{T}^-, l \in \mathcal{L}, \quad \boldsymbol{x}_{t+1}^l = \boldsymbol{A}_t^l \boldsymbol{x}_t^l + \boldsymbol{B}_t^l \boldsymbol{u}_t^l + \boldsymbol{w}_t^l \tag{9.2}$$

$$\Pr\left[\bigwedge_{l \in \mathcal{L}} \bigwedge_{i \in \mathcal{I}^l} \boldsymbol{h}_i^{lT} \boldsymbol{x}_{t_i}^l - g_i^l \leq 0\right] \geq 1 - \Delta^S \tag{9.3}$$

This problem formulation requires the minimization of the *system cost* (9.1), defined as the total cost of individual agents J^l. Note that the joint chance constraint (9.3) requires that the probability that all state constraints of *all* agents are satisfied must be at least $1 - \Delta^S$.

We use the risk allocation approach to solve Problem 14. Recall that, in Section 5.1, we obtain Problem 9, the deterministic approximation of Problem 4, by using Corollaries 2 and 3. Likewise, we apply Corollaries 2 and 3 to Problem 14 in order to obtain the following approximated deterministic optimization problem, defined on the mean state $\bar{\boldsymbol{x}}_t^l$:

Problem 15: Centralized deterministic optimization with decomposed chance constraints

$$\min_{\boldsymbol{u}_{0:N-1}^{1:M} \in \mathbb{U}^{N \cdot M}, \delta^{1:M} \succ 0} \quad \sum_{l \in \mathcal{L}} J^l(\boldsymbol{u}_{0:N-1}^l, \bar{\boldsymbol{x}}_{l:N}^l) \tag{9.4}$$

$$\text{s.t.} \quad \forall t \in \mathbb{T}^-, l \in \boldsymbol{\mathcal{L}}, \quad \bar{\boldsymbol{x}}_{t+1}^l = \boldsymbol{A}_t^l \bar{\boldsymbol{x}}_t^l + \boldsymbol{B}_t^l \boldsymbol{u}_t^l \tag{9.5}$$

$$\bigwedge_{l \in \mathcal{L}} \bigwedge_{i \in \mathcal{I}^l} \boldsymbol{h}_i^{lT} \bar{\boldsymbol{x}}_{t_i} - g_i^l \leq -m_i^l(\delta_i^l) \tag{9.6}$$

$$\sum_{l \in \mathcal{L}} \sum_{i \in \mathcal{I}^l} \delta_i^l \leq \Delta^S. \tag{9.7}$$

As we discussed in Section 5.1, the optimal solution to Problem 15 is a feasible and near-optimal solution to Problem 14. It is not an exactly optimal solution since (9.7) is a conservative approximation of (9.3). Problem 15 is tractable since it is a deterministic convex optimization problem.

The new variable $\delta_i^l \geq 0$ is the risk bound imposed on the ith constraint of the lth agent. We denote by δ^l the risk vector of the lth agent:

$$\delta^l := \left[\delta_1^l, \delta_2^l, \cdots \delta_N^l\right]$$

Similar to before, $m_i^l(\cdot)$ represents the width of the safety margin for the ith constraint of the lth agents:

$$m_i^l(\delta_i^l) = -\sqrt{2\boldsymbol{h}_i^{lT} \Sigma_{x,t_i}^l \boldsymbol{h}_i^l}\, \text{erf}^{-1}(2\delta_i^l - 1) \tag{9.8}$$

where erf^{-1} is the inverse of the Gauss error function. Recall that $m_i^l(\delta_i^l)$ is a convex, monotonically decreasing, and non-negative function for $\delta_i^l \in (0, 0.5]$. Also note that it is *strictly* convex for $\delta_i^l \in (0, 0.5)$. Our assumption that $\Delta^S \leq 0.5$ guarantees $\delta_i^l \leq 0.5$.

Problem 15 can be solved in a *centralized* manner by the CRA algorithm, presented in Chapter 5. In such a solution method, the control sequences and risk allocations of all agents are optimized by a single solver. In this chapter, we seek a *decentralized* solution method for Problem 15. For that purpose, we reformulate Problem 15 in the next section so that optimization can be conducted in a decentralized manner.

9.3 Decentralization

We propose a decentralized method for solving Problem 15, based on a dual decomposition. This method has each agent solve a decomposed convex optimization problem while a central module solves a root-finding problem.

9.3.1 The Decentralized Optimization Approach

Each individual agent solves the following problem, Problem 16, which involves a convex optimization:

Problem 16: Decomposed optimization problem for l'th agent

$$\min_{\boldsymbol{u}^l_{0:N-1}\in\mathbb{U}^N,\delta^l\succ 0} \quad J^l(\boldsymbol{u}^l_{0:N-1},\bar{\boldsymbol{x}}^l_{l:N}) + p\sum_{i\in\mathcal{I}^l}\delta^l_i \tag{9.9}$$

$$\text{s.t.} \quad \forall t\in\mathbb{T}^-, \quad \bar{\boldsymbol{x}}^l_{t+1} = \boldsymbol{A}^l_t\bar{\boldsymbol{x}}^l_t + \boldsymbol{B}^l_t\boldsymbol{u}^l_t \tag{9.10}$$

$$\bigwedge_{i\in\mathcal{I}^l} \boldsymbol{h}^{lT}_i\bar{\boldsymbol{x}}_{t_i} - g^l_i \le -m^l_i(\delta^l_i) \tag{9.11}$$

where $p \ge 0$ is a constant given by the central module and shared by all agents. This constant p can be interpreted as the *price of risk*, as discussed in Section 9.1.2. Problem 16 is completely decoupled from other agents, since it does not include the coupling constraint, chance constraint (9.7).

Note that $\sum_{i\in\mathcal{I}^l}\delta^l_i$, the total amount of risk the lth agent takes, is not bounded in Problem 16. Instead, the agent's objective function is penalized by the risk it takes with a factor p in (9.9). Given p, the agent finds $\delta^{l\star}_i(p)$ that minimizes the penalized cost. Since the total amount of risk that each agent takes is also a function of p, we denote it by $D^l(p) := \sum_{i\in\mathcal{I}^l}\delta^{l\star}_i(p)$. In Section 9.3.3, we will give a more formal definition of $D^l(p)$, and prove that it is a single-valued, continuous, monotonically decreasing function. $D^l(p)$ can be interpreted as the lth agent's *demand for risk* given the price p.

The central module finds an optimal price $p^\star$ that minimizes the system cost $\sum_{l\in\mathcal{L}} J^l$

in (9.4). It turns out that the optimal price corresponds exactly to the equilibrium price, which equalizes the aggregate demand and supply. In other words, the optimal price is the solution to the following root-finding problem:

Problem 17: Root-finding problem for the central module

$$\text{Find} \quad p \geq 0 \quad \text{where} \quad \sum_{l \in \mathcal{L}} D^l(p) = \Delta^S, \tag{9.12}$$

where $D^l(p)$ is the demand of the lth agent at the price p, as defined above. We note that, in a special case where $p = 0$, the optimal price does *not* correspond to the equilibrium price[2]. We treat the special case separately.

9.3.2 Existence and Optimality of Decentralized Solution

The following theorem holds:

Theorem 2. ***(a)*** *If Problem 16 has an optimal solution* $(U^{i\star}, \delta^{l\star})$ *for all* $l = 1 \cdots M$ *with* $p^\star > 0$ *that is a root of Problem 17, then* $(U^{1:I\star}, \delta^{1:M\star})$ *is a globally optimal solution for Problem 15.*

(b) *If Problem 16 has an optimal solution* $(U^{i\star}, \delta^{l\star})$ *with* $p = 0$ *for all* $l = 1 \cdots M$ *that satisfies* $\Sigma_{l \in \mathcal{L}} D^l(0) \leq \Delta^S$*, then* $(U^{1:I\star}, \delta^{1:M\star})$ *is a globally optimal solution for Problem 15.*

In other words, this theorem states that *if* a decentralized solution (i.e., solution for Problems 15 and 17) exists, then it is an optimal solution for the centralized problem (Problem 15).

Proof: We prove by showing that $(U^{1:I\star}, \delta^{1:M\star})$ and $p^\star$ satisfy the KKT conditions of Problem 15. Although Problem 16 shares most of the constraints of Problem 15, it misses (9.7). Therefore we need to pay attention to the KKT conditions that are related to (9.7)

[2]This is because $p = 0$ satisfies the complementary slackness condition: $p\left(D^l(p) - \Delta^S\right) = 0$

and δ_i^l:

$$\mu_i^l \frac{dm_i^l}{d\delta_i^l} + p = 0 \tag{9.13}$$

$$\sum_{l\in\mathcal{L}} \sum_{i\in\mathcal{I}^l} \delta_i^l \leq \Delta^S \tag{9.14}$$

$$p\left(\sum_{l\in\mathcal{L}} \sum_{i\in\mathcal{I}^l} \delta_i^l - \Delta^S\right) = 0 \tag{9.15}$$

$$p \geq 0 \tag{9.16}$$

where μ_i^l and p are the dual variables corresponding to (9.6) and (9.7), respectively. The KKT conditions are the necessary and *sufficient* conditions for global optimality of Problem 15 since J^l and m_i^l are convex functions, and the equality constraint (9.5) is linear. The optimal solution of Problem 16 $(U^{i\star}, \delta_i^{l\star})$ satisfies (9.13) because it is also a part of the KKT conditions of Problem 16. In the case of (a), $\delta^{1:M\star}$ satisfies (9.14) and (9.15) since $p^\star$ is a root of (9.12). In the case of (b), (9.14) and (9.15) are satisfied since $\Sigma_{l\in\mathcal{L}}^I D^l(0) \leq \Delta^S$ and $p = 0$. Since the cost function of the centralized optimization (9.4) is a sum of the individual cost functions (9.9) and the constraints (9.5)-(9.6) are the same as (9.10)-(9.11), the partial derivatives of the Lagrangians of Problem 15 and 16 respect to $\boldsymbol{u}_{0:N-1}^l$ and $\boldsymbol{x}_{t_i}^l$ are the same. Therefore, their stationary constraints regarding to $\boldsymbol{u}_{0:N-1}^l$ and $\boldsymbol{x}_{t_i}^l$ are the same. Problem 15 and 3 also share the same primary feasibility, dual feasibility, and the complementary slackness conditions regarding to $\boldsymbol{u}_{0:N-1}^l$ and $\boldsymbol{x}_{t_i}^l$ since (9.5)-(9.6) and (9.10)-(9.11) are the same.

Since all KKT conditions of Problem 15 are satisfied by $(\boldsymbol{U}^{1:I\star}, \delta^{1:M\star})$, which satisfies the KKT conditions of Problem 16 together with $p^\star$, $(\boldsymbol{U}^{1:I\star}, \delta^{1:M\star})$ is an optimal solution of Problem 15. ■

Recall that Theorem 2 guarantees the optimality of a decentralized solution. The following Theorem 3 guarantees the existence of a decentralized solution if there is an optimal solution for the centralized problem.

Theorem 3. *If Problem 15 has an optimal solution* $(U^{1:I\star}, \delta^{1:M\star})$,
(a) $(U^{i\star}, \delta^{l\star})$ *is an optimal solution of Problem 16 for all* $l = 1 \cdots M$ *given* $p > 0$, *which*

is a root of Problem 17, ***or***

(b) *$(U^{i\star}, \delta^{l\star})$ is an optimal solution of Problem 16 for all $l = 1 \cdots M$ with $p = 0$ and $\Sigma_{l \in \mathcal{L}} D^l(0) \leq \Delta^S$.*

Proof: The KKT conditions of Problem 16 are the necessary and *sufficient* conditions for its optimality since J'^l and m_i^l are convex functions, and the equality constraint (9.10) is linear. Since the KKT conditions of Problem 16 are the subset of the KKT conditions of Problem 15, the optimal solution of Problem 15 always satisfies all KKT conditions of Problem 16 for all $l = 1 \cdots M$; hence it is an optimal solution of Problem 16. When $p > 0$, it is a root of Problem 17 since the second term of (9.15) must be zero. When $p = 0$, $\Sigma_{l \in \mathcal{L}} D^l(0) \leq \Delta^S$ because (9.14) is satisfied. ■

Although the following Lemma 10 is just a contraposition of Theorem 3, it is useful when checking the feasibility of Problem 15.

Lemma 10. *If both Theorem 3(a) and (b) do not apply, Problem 15 does not have an optimal solution.*

9.3.3 Convergence to the Optimal Solution

Although the existence of a decentralized solution is established by Theorem 3, it does not specify how to find it. The objective of this subsection is to guarantee that the MIRA algorithm converges to the solution. Recall that we use Brent's method to solve the root-finding problem. The convergence of Brent's method is guaranteed for a continuous function. Hence, it suffices to prove the continuity of the demand function $D^l(p)$ in order to guarantee the convergence of MIRA. Additionally, we also prove in this subsection that $D^l(p)$ is a monotonically decreasing function. This feature is important since it allows us to find the absence of a root by checking the feasibility conditions at the boundaries.

We first derive the optimal cost as a function of a risk bound. Observe that the following

optimization problem gives the same solution as Problem 16:

$$\min_{\Delta^l \le 0.5} \min_{\boldsymbol{u}^l_{0:N-1} \in \mathbb{U}^N, \delta^l \succ 0} \quad J^l(\boldsymbol{u}^l_{0:N-1}, \bar{\boldsymbol{x}}^l_{l:N}) + p\Delta^l \tag{9.17}$$

$$\text{s.t.} \quad (9.10) - (9.11)$$

$$\sum_{i \in \mathcal{I}^l} \delta^l_i \le \Delta^l \tag{9.18}$$

Therefore, Problem 16 is equivalent to solving the following:

$$\min_{\Delta^l \le 0.5} J^{l\star}(\Delta^l) + p\Delta^l \tag{9.19}$$

where

$$J^{l\star}(\Delta^l) = \min_{\boldsymbol{u}^l_{0:N-1} \in \mathbb{U}^N, \delta^l \succ 0} J^l(\boldsymbol{u}^l_{0:N-1}, \bar{\boldsymbol{x}}^l_{l:N}) \tag{9.20}$$

$$\text{s.t.} \quad (9.10) - (9.11), (9.18)$$

The conditions (9.10)-(9.11), (9.18) define a compact space. If it is non-empty, (9.20) has a minimum since $J^l(\boldsymbol{u}^l_{0:N-1}, \bar{\boldsymbol{x}}^l_{l:N})$ is a proper convex function by assumption. We denote by $\Delta^l_{\min}$ the smallest Δ^l that makes (9.10)-(9.11),(9.18) non-empty. Then it is non-empty for all $\Delta^l \ge \Delta^l_{\min}$ since $m^l_i(\delta^l_i)$ is a monotonically decreasing function. Therefore $J^{l\star}(\Delta^l)$ is a single-valued function for all $\Delta^l \ge \Delta^l_{\min}$. Since $\Delta^S \le 0.5$, any feasible solution of Problem 15 has $\Delta^l \le 0.5$. Therefore we limit the domain of $J^{l\star}(\Delta^l)$ to $[\Delta^l_{\min}, 0.5]$ without the loss of generality. The convexity of $m^l_i(\delta^l_i)$ is implied by $\Delta^l \le 0.5$.

Lemma 11. *$J^{l\star}(\Delta^l)$ is a convex, monotonically decreasing function for its entire domain.*

Proof: Let Δ^l_1 and Δ^l_2 be real numbers that satisfy $\Delta^l_{\min} \le \Delta^l_1 < \Delta^l_2 \le 0.5$. Since larger Δ^l loosens the constraint (9.18) by allowing larger δ^l_i, $J^{l\star}(\Delta^l_1) \ge J^{l\star}(\Delta^l_2)$. Therefore $J^{l\star}$ is monotonically decreasing.

Let λ be a real scalar in $[0, 1]$. Let also $\boldsymbol{U}^{l\star}_1$ and $\boldsymbol{U}^{l\star}_2$ be optimal solutions of (9.20) with

Δ_1^l and Δ_2^l, respectively. Note that we use an abbreviated notations:

$$\boldsymbol{U}_1^{l\star} := \boldsymbol{u}_{0:N-1,1}^{l\star} = \left[\boldsymbol{u}_{0,1}^{l\star T} \cdots \boldsymbol{u}_{N-1,1}^{l\star T}\right]^T$$
$$\boldsymbol{U}_2^{l\star} := \boldsymbol{u}_{0:N-1,2}^{l\star} = \left[\boldsymbol{u}_{0,2}^{l\star T} \cdots \boldsymbol{u}_{N-1,2}^{l\star T}\right]^T$$

Since the feasible space defined by (9.10)-(9.11), (9.18) is convex, $\lambda \boldsymbol{U}_1^{l\star} + (1-\lambda)\boldsymbol{U}_2^{l\star}$ is a feasible (but not necessarily optimal) solution of (9.20) with $\lambda\Delta_1^l + (1-\lambda)\Delta_2^l$. Therefore,

$$\begin{aligned} J^{l\star}(\lambda\Delta_1^l + (1-\lambda)\Delta_2^l) &\leq J^l(\lambda \boldsymbol{U}_1^{l\star} + (1-\lambda)\boldsymbol{U}_2^{l\star}) \\ &\leq \lambda J^l(\boldsymbol{U}_1^\star) + (1-\lambda)J^l(\boldsymbol{U}_2^\star) \\ &= \lambda J^{l\star}(\Delta_1) + (1-\lambda)J^{l\star}(\Delta_2). \end{aligned}$$

The second inequality holds since $J^l(\boldsymbol{u}_{0:N-1}^l, \bar{\boldsymbol{x}}_{l:N}^l)$ is a convex function of $\boldsymbol{u}_{0:N-1}^l$. ■

It immediately follows from Lemma 10 that $J^{l\star}(\Delta^l)$ is continuous, and differentiable at all but countably many points.

We then prove the *strict* convexity for a portion of the domain of $J^{l\star}$ where it is strictly decreasing. We define $\Delta_{\max}^l$ as follows:

$$\Delta_{\max}^l = \min\left[0.5, \sup\left\{\Delta^l \mid \partial J^{l\star}(\Delta^l) < 0.\right\}\right]$$

where $\partial J^{l\star}$ is the subdifferencial of $J^{l\star}$. The inequality means that all subgradients are less than zero. See Fig. 9-5 for graphical interpretation.

Lemma 12. *$J^{l\star}(\Delta^l)$ is strictly convex for all $\Delta_{\min}^l \leq \Delta^l \leq \Delta_{\max}^l$.*

Proof: Fix Δ_1^l, Δ_2^l, and λ such that $\Delta_{\min}^l \leq \Delta_1^l < \Delta_2^l \leq \Delta_{\max}^l$, $0 < \lambda < 1$. Let $(\boldsymbol{U}_1^{l\star}, \bar{\boldsymbol{X}}_1^{l\star}, \delta_1^{l\star})$ and $(\boldsymbol{U}_2^{l\star}, \bar{\boldsymbol{X}}_2^{l\star}, \delta_2^{l\star})$ be optimal solutions of (9.20) with Δ_1^l and Δ_2^l, respectively. Note that we use an abbreviated notations:

$$\bar{\boldsymbol{X}}_1^{l\star} := \boldsymbol{x}_{1:N,1}^{l\star} = \left[\boldsymbol{x}_{1,1}^{l\star T} \cdots \boldsymbol{x}_{N,1}^{l\star T}\right]^T$$
$$\bar{\boldsymbol{X}}_2^{l\star} := \boldsymbol{x}_{1:N,2}^{l\star} = \left[\boldsymbol{x}_{1,2}^{l\star T} \cdots \boldsymbol{x}_{N,2}^{l\star T}\right]^T$$

Also note that $\bar{\boldsymbol{X}}_j^{l\star}$ is linearly tied to $\boldsymbol{U}_1^{l\star}$ by (9.10). Since $m_i^l(\delta_i^l)$ in (9.11) is strictly convex,

$$\forall i \in \mathcal{I}^l h_n^{iT}\left(\lambda \bar{\boldsymbol{X}}_1^i + (1-\lambda)\bar{\boldsymbol{X}}_2^i\right) < g_i^l - m_i^l\left(\lambda\delta_{i,1}^{l\star} + (1-\lambda)\delta_{i,2}^{l\star}\right).$$

In other words, the constraints (9.11) are inactive for all $i \in \mathcal{I}^l$ at $\lambda\bar{\boldsymbol{X}}_1^i + (1-\lambda)\bar{\boldsymbol{X}}_2^i$ (hence, at $\lambda\boldsymbol{U}_1^{l\star} + (1-\lambda)\boldsymbol{U}_2^{l\star}$) and $\lambda\delta_{i,1}^{l\star} + (1-\lambda)\delta_{i,2}^{l\star}$.

Also, the following inequality holds:

$$J^l(\lambda\boldsymbol{U}_1^{l\star} + (1-\lambda)\boldsymbol{U}_2^{l\star}) \geq J^{l\star}(\lambda\Delta_1^i + (1-\lambda)\Delta_2^i) > J^{l\star}(\Delta_2^i) = J^l(\boldsymbol{U}_2^{l\star}).$$

The second inequality follows from the mean-value theorem and the definition of $\Delta_{\max}^l$. Note that by assumption, $\lambda > 0$ (hence, $\lambda\Delta_1^i + (1-\lambda)\Delta_2^i < \Delta_2^l$) and $\Delta_2^l \leq \Delta_{\max}^l$. As for the first inequality, refer to the proof of Lemma 11. It is implied by (9.3.3) that $\lambda\boldsymbol{U}_1^{l\star} + (1-\lambda)\boldsymbol{U}_2^{l\star}$ is not a globally optimal solution; hence, it is not a local optimal solution either.

Therefore, there exists a non-zero perturbation $\delta\boldsymbol{U}^l$ to $\lambda\boldsymbol{U}_1^{l\star} + (1-\lambda)\boldsymbol{U}_2^{l\star}$ that satisfies the constraints (9.11) with $\lambda\delta_1^{l\star} + (1-\lambda)\delta_2^{l\star}$, and results in strictly less cost. Hence, we have:

$$\begin{aligned} J^{l\star}(\lambda\Delta_1 + (1-\lambda)\Delta_2) &\leq J^l(\lambda\boldsymbol{U}_1^{l\star} + (1-\lambda)\boldsymbol{U}_2^{l\star} + \delta\boldsymbol{U}^l) \\ &< J^l(\lambda\boldsymbol{U}_1^{l\star} + (1-\lambda)\boldsymbol{U}_2^{l\star}) \leq \lambda J^l(\boldsymbol{U}_1^\star) + (1-\lambda)J^l(\boldsymbol{U}_2^\star) \\ &= \lambda J^{l\star}(\Delta_1) + (1-\lambda)J^{l\star}(\Delta_2). \quad (9.21) \end{aligned}$$

Therefore, $J^{l\star}(\Delta^l)$ is strictly convex for all $\Delta_{\min}^l \leq \Delta^l < \Delta_{\max}^l$. ■

Since $J^{l\star}(\Delta^l)$ is strictly convex and monotonically decreasing, there is a unique minimizer of (9.19) for $p > 0$. When $p = 0$, the optimal solution of (9.19) may not be unique.

Then, we define the demand function as follows so that $D^l(p)$ is a single-valued function for all $p \geq 0$:

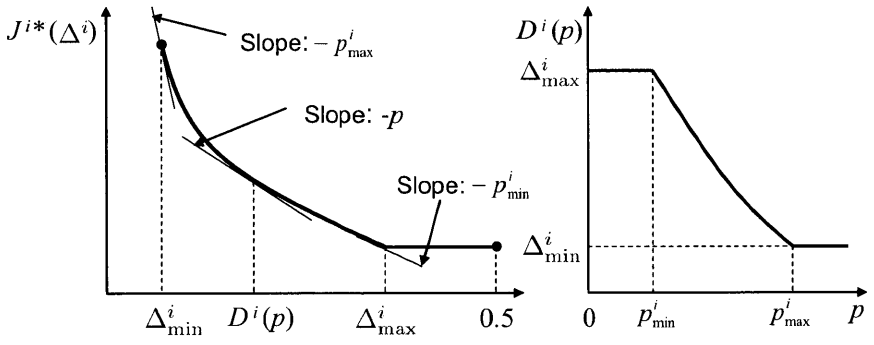


Figure 9-5: Sketch of the functions $J^{l\star}(\Delta^l)$ and $D^l(p)$ (demand function of lth agent). Note that in many practical cases $\Delta^l_{\max} = 0.5$.

Definition: Demand function

$$D^l(p) := \begin{cases} \arg\min_{\Delta^l \le 0.5} J^{l\star}(\Delta^l) + p\Delta^l & (p > 0) \\ \Delta^l_{\max} & (p = 0) \end{cases}$$

This definition is natural since $\Delta^l_{\max}$ is an optimal solution for $p = 0$ if it exists. Moreover, in such a case, $\Delta^l_{\max}$ is the smallest optimal solution. This feature is important since we need to check the condition (9.14), which is equivalent to $\sum_{l\in\mathcal{L}} D^l(0) \le \Delta^S$, against the smallest optimal solution in order to tell if Theorem 2(b) applies.

Proposition 3: Continuity and Monotonicity of Demand Function

(a) $D^l(p)$ is a continuous, monotonically decreasing function for $p \ge 0$.

(b) $D^l(p) = \Delta^l_{\max}$ for $0 \le p \le p^l_{\min}$, and $D^l(p) = \Delta^l_{\min}$ for $p \ge p^l_{\max}$.

Proof: We define $p^l_{\min}$ and $p^l_{\max}$ as follows:

$$\begin{aligned} p^l_{\max} &= \sup_{\Delta^l \in (\Delta^l_{\min}, \Delta^l_{\max})} -\partial J^{l\star}(\Delta^l) \\ p^l_{\min} &= \inf_{\Delta^l \in (\Delta^l_{\min}, \Delta^l_{\max})} -\partial J^{l\star}(\Delta^l), \end{aligned} \tag{9.22}$$

where $\partial J^{l\star}(\Delta^l)$ is the subdifferential of $J^{l\star}$.

We first prove the continuity and monotonicity in $(p^l_{\min}, p^l_{\max})$. Since $D^l(p)$ is the opti-

mal solution for (9.19), the following optimality condition is satisfied:

$$-p \in \partial J^{l\star}\left(D^l(p)\right). \tag{9.23}$$

It follows from the Conjugate Subgradient Theorem (Proposition 5.4.3 of [15]) that

$$D^l(p) \in \partial (J^{l\star})^\star(-p)$$

where $(J^{l\star})^\star$ is the conjugate function of $J^{l\star}$. Since the minimum of $J^{l\star}(\Delta^l) + p\Delta^l$ is uniquely attained, $(J^{l\star})^\star$ is differentiable everywhere, and hence continuously differentiable, in $(p^l_{\min}, p^l_{\max})$ (Proposition 5.4.4 of [15]). Therefore $D^l(p)$ is continuous in $(p^l_{\min}, p^l_{\max})$. Also, since $(J^{l\star})^\star$ is a convex function (Ch. 1.6 of [15]), $D^l(p)$ is monotonically decreasing in $(p^l_{\min}, p^l_{\max})$.

Next, we show that $D^l(p) = \Delta^l_{\max}$ for $0 \leq p \leq p^l_{\min}$. When $\Delta^l_{\max} < 0.5$, it follows from the definition of $\Delta^l_{\max}$ and $p^l_{\min}$ that $\partial J^{l\star}(\Delta^l_{\max}) = [0, -p^l_{\min}]$. Therefore, $\Delta^l_{\max}$ is an optimal solution of (9.19) for $0 \leq p \leq p^l_{\min}$, since the optimality condition (9.23) is satisfied. This result agrees with the definition of $D^l(p)$ at $p = 0$. When $\Delta^l_{\max} = 0.5$, it follows from (9.22) that $p^l_{\min} < -\partial J^{l\star}(\Delta^l)$ in $(\Delta^l_{\min}, 0.5)$. Therefore, the minimum of $J^{l\star}(\Delta^l) + p\Delta^l$ is attained at the upper bound of Δ^l, which is $\Delta^l = 0.5 = \Delta^l_{\max}$, and hence $D^l(p) = \Delta^l_{\max}$. By definition, $D^l(0) = \Delta^l_{\max}$. The continuity at $p = 0$ is obvious.

Then we show that $D^l(p) = \Delta^l_{\min}$ for $p \geq p^l_{\max}$. It follows from (9.22) that $p^l_{\max} > -\partial J^{l\star}(\Delta^l)$ in $(\Delta^l_{\min}, \Delta^l_{\max})$. Therefore, the minimum of $J^{l\star}(\Delta^l) + p\Delta^l$ is attained at the lower bound of the domain of $J^{l\star}$, which is $\Delta^l = \Delta^l_{\min}$. Therefore, $D^l(p) = \Delta^l_{\min}$.

Finally, we prove that $D^l(p)$ is continuous at $p^l_{\min}$ and $p^l_{\max}$. Since $D^l(p)$ is constant for $0 \leq p \leq p^l_{\min}$ and $p \geq p^l_{\max}$, we only have to show that it is upper semi-continuous at $p^l_{\min}$ and lower semi-continuous at $p^l_{\max}$. Consider a sequence $p_k \in (p^l_{\min}, p^l_{\max})$ with $p_k \to p^l_{\min}$. It follows from the definition of $p^l_{\min}$ (9.22) and the convexity of $J^{l\star}$ that we can find a sequence D^i_k such that

$$p_k \in -\partial J^{l\star}\left(D^i_k\right), \quad D^i_k \to \Delta^l_{\max}.$$

Since D_k^i satisfies the condition for optimality for p_k, $D^l(p_k) = D_k^i$. Therefore,

$$\lim_{p_k \to p_{\min}^l} D^l(p_k) \to \Delta_{\max}^l.$$

Hence, $D(p)$ is continuous at $p_{\min}^l$. In the same way, it is lower semi-continuous at $p_{\max}^l$.

Therefore, $D^l(p)$ is a continuous, monotonically decreasing function for $p \geq 0$. ■

See Fig. 9-5 for the sketch of $D^l(p)$.

9.4 The Algorithm

Now we present the Market-based Iterative Risk Allocation (MIRA), that finds the solution for Problems 3 and 4. Algorithm 13 (see the box below) shows the entire flow of the MIRA algorithm. Exploiting the fact that risk is a scalar, we use Brent's method to efficiently find the root of Problem 17.

Algorithm 13 Market-based Iterative Risk Allocation

1: Each agent computes $\Delta_{\max}^l$ and $\Delta_{\min}^l$;
2: **if** $\sum_{l \in \mathcal{L}} \Delta_{\max}^l \leq \Delta^S$ **then**
3: $p = 0$ gives the optimal solution; terminate;
4: **else if** $\sum_{l \in \mathcal{L}} \Delta_{\min}^l > \Delta^S$ **then**
5: There is no feasible solution; terminate;
6: **else**
7: **while** $|\sum_{l \in \mathcal{L}} D^l(p) - \Delta^S| > \epsilon$ **do**
8: The central module announces p to agents;
9: Each agent computes $D^l(p)$ by solving Problem 16;
10: Each agent submits $D^l(p)$ to the central module;
11: The central module updates p by computing one step of Brent's method;
12: **end while**
13: **end if**

9.4.1 Obtaining $D^l(0)(=\Delta^l_{\max})$ (Algorithm 13, Line 1-3)

As we explained earlier, we need to treat the spacial case, $p = 0$, separately. To this end, the algorithm first computes $D^l(0)(=\Delta^l_{\max})$ in order to find if Theorem 2(b) applies. If so, $p = 0$ is the optimal price. $D^l(0)$ is obtained through the following process:

1. Relax Problem 16 by fixing all risk bounds at $\delta^l_i = 0.5$. This relaxation makes $m^l_i(\delta^l_i) = 0$. The relaxed problem is a convex optimization problem with only linear constraints, which can be solved efficiently. If it does not have a feasible solution, Problem 16 is infeasible, and Problem 15 is also infeasible (Lemma 10). The algorithm terminates in this case.

2. Compute $\delta^{l\star}_i$ using the following equation with the optimal solution $\bar{\boldsymbol{X}}^{l\star}$ obtained from the relaxed problem:

$$\delta^{l\star}_i = \mathrm{cdf}^l_i(h^{iT}_n \bar{\boldsymbol{x}}^{l\star}_{t_i} - g^l_i)$$

   where $\mathrm{cdf}^l_i(\cdot)$ is a cumulative distribution function of univariate Gaussian distribution with variance $\boldsymbol{h}^{lT}_i \Sigma_{X^l} \boldsymbol{h}^l_i$, or in other words, the inverse of $-m^l_i(\cdot)$.

3. Obtain $D^l(0)$ by:

$$D^l(0) = \min\left[0.5, \sum_{i\in\mathcal{I}^l} \delta^{l\star}_i\right]$$

Each agent computes $D^l(0)$ and sends it to the central module. The central module checks if $\sum_{l\in\mathcal{L}} D^l(0) \leq \Delta^S$ holds. If so, the optimal solution of the relaxed problem is the solution of Problem 15 by Theorem 2(b). Proposition 3 guarantees that there is no positive root of Problem 17, and hence, the algorithm terminates. Otherwise the algorithm looks for a solution with $p > 0$.

9.4.2 Obtaining $\Delta^l_{\min}$ (Algorithm 13, Line 1, 4, and 5)

Each agent also computes $\Delta^l_{\min}$ by solving the following convex optimization problem:

$$\begin{aligned} \Delta^l_{\min} \quad &= \quad \min_{\Delta^l \le 0.5, \boldsymbol{u}^l_{0:N-1}, \delta^l \succ 0} \Delta^l \\ \text{s.t.} \quad &\quad (9.10) - (9.11), (9.18) \end{aligned} \tag{9.24}$$

If $\sum_{l\in\mathcal{L}} \Delta^l_{\min} > \Delta^S$, then it follows from Proposition 3 that $\sum_{l\in\mathcal{L}} D^l(p) > \Delta^S$ for all $p \ge 0$. In this case, since both Theorem 2(a) and (b) do not apply, there is no feasible solution (Lemma 10), and the algorithm terminates. Otherwise, Proposition 3 guarantees that a solution exists in $(0, \max_l p^l_{\max}]$. $p^l_{\max}$ can be computed from the solution of the optimization problem (9.24).

9.4.3 Finding a root for Problem 17 (Algorithm 13, Line 7-12)

If the algorithm has not terminated in the previous steps, we have $\sum_{l\in\mathcal{L}} D^l(0) > \Delta^S$ and $\sum_{l\in\mathcal{L}} D^l(p^l_{\max}) \le \Delta^S$. Therefore, the continuity of $D^l(p)$ (Proposition 3) guarantees that Brent's method can find a root between $(0, \max_l p^l_{\max}]$. Brent's method provides superliner rate of convergence [6]. It is suitable for our application since it does not require the derivative of $D^l(p)$, which is generally hard to obtain. In many practical cases, it is more efficient to incrementally search p that is large enough to make $\sum_{l\in\mathcal{L}} D^l(p^l_{\max}) \le \Delta^S$, rather than initializing Brent's method with $[0, \max_l p^l_{\max}]$.

The algorithm updates $(U^{l\star}, \delta^{l\star})$ (hence, $D^l(p)$) and p alternatively and iteratively. In each iteration, each agent computes $D^l(p)$ (Line 9) in a distributed manner by solving Problem 16 with p, which is given by the central module (Line 8). Since Problem 16 is a convex optimization problem, it can be solved efficiently using interior-point methods. The central module collects $D^l(p)$ from all agents (Line 10) and updates p by computing one step of Brent's method (Line 11).

The communication requirements between agents are small: in each iteration, each agent receives p (Line 8) and transmits $D^l(p)$ (Line 10), both of which are scalars.

The central module can be removed by letting all individual agents solve Problem 17

to update p simultaneously. However, since the computation of p is duplicated among the agents, there is no advantage of doing so in terms of computation time.

9.5 Complexity Analysis

The purpose of this section is to demonstrate the advantage of MIRA over the decentralized approach by comparing computational complexity. We show that the worst-case total computational complexity of MIRA grows only linearly with the number of agents. Moreover, when computation is distributed over all the agents, the worst-case parallel computational complexity is constant. This is a clear advantage over the centralized approach, whose worst-case complexity grows at least cubically with the number of agents.

We consider two kinds of complexity concepts in this subsection: *total computation complexity* and *parallel computation complexity*. Roughly speaking, the total computation complexity represents the computation time of the algorithm if all computation is conducted by a single processor. The parallel computation complexity represents the computation time when the computation is conducted in parallel using the processors of all agents. For ease of analysis, we assume that every agent owns the same number N_V of decision variables and is subject to the same number N_C of constraints.

The following theorem presents the main result of this section:

Theorem 4. *The worst-case total computational complexity of MIRA is $O(|\mathcal{L}|)$, where $|\mathcal{L}|$ is the number of agents. Moreover, the worst-case parallel computational complexity is constant with $|\mathcal{L}|$.*

Compare this result with the following:

Remark 1. *The worst-case computational complexity of the centralized optimization of Problem 15 is $O(|\mathcal{L}|^{3.307} \log_2 |\mathcal{L}|)$, assuming that matrix inversion is computed by using the Strassen algorithm.*

Proof: MIRA has three computational layers. On the top is Brent's method, which is run by the central module to solve the root-finding problem (Problem 17) to optimize the price of risk (dual variable). The middle layer is the outer loop of an interior point method,

which is run by each agent to solve the decomposed convex optimization problem (Problem 16) to obtain an optimal control sequence. The bottom layer is the inner loop of the interior point method, typically Newton's method, which is called by the outer loop (See Chapter 11 of [23] for the details of interior point methods). We evaluate the computational complexity of MIRA by combining the complexity of the three layers. The computational complexity of the centralized approach is obtained by considering the middle and the bottom layer.

Top layer: Brent's Method The complexity of the top layer of MIRA is constant because the root-finding problem (Problem 17) involves only one variable (the price of risk), regardless of the number of agents. In fact, the number of iterations required for Brent's method to solve Problem 17 depends only on the initial error and the numerical tolerance [6], and is independent of the number of agents. This fact is empirically validated in Section 11.1.11: as shown in Figure 11-21, the number of iterations of Brent's method is almost constant at 40. Although the computation time per each iteration step grows with the number of agents, the computation time of the top layer (< 50 milliseconds; see Figure 11-21) is negligible compared with the computation time of the middle and the third layers (10-100 seconds; see Figure 11-20).

Middle layer: Outer loop of an interior point method Here we assume the use of a specific form of iterator point methods, the barrier method, with a self-concordant cost function. This assumption allows us to analytically obtain the following worst-case bound on the total number of Newton steps:

$$\text{Number of Newton steps} \leq N = O\left(\sqrt{m}\log_2\left(\frac{m/t^{(0)}}{\epsilon}\right)\right),$$

where m is the number of constraints, $t^{(0)}$ is the initial coefficient of the logarithmic barrier, and ϵ is the tolerance on duality gap. See Section 11.5 of [23] for the derivation.

Bottom layer: Inner loop of an interior point method In each Newton step, the computation of the inverse of the KKT matrix dominates the computation time. KKT matrix is a $(m + n)$-by-$(m + n)$ matrix, where n is a number of variables. The complexity of

inverting an N-by-N matrix is $O(N^{2.807})$ when using the Strassen algorithm, which is a typical choice for practical computations. Hence, the complexity of the bottom layer is $O\left((m+n)^{2.807}\right)$.

Complexity of the Centralized Approach We first prove Remark 1. As mentioned previously, the centralized approach corresponds to solving Problem 15 by the middle and the bottom layers of MIRA in a single process. Since every agent has N_V variables and N_C constraints by assumption, $n = N_V|\mathcal{L}|$ and $m = N_C|\mathcal{L}|$. Therefore, the worst-case computational complexity is:

$$O\left(\left\{(N_V+N_C)|\mathcal{L}|\right\}^{2.807}\sqrt{N_C|\mathcal{L}|}\log_2\left(\frac{N_C|\mathcal{L}|/t^{(0)}}{\epsilon}\right)\right).$$

Hence, the complexity regarding the number of agents $|\mathcal{L}|$ is:

$$O\left(|\mathcal{L}|^{3.307}\log_2\left(|\mathcal{L}|\right)\right).$$

This result agrees with the empirical result in Section 11.1.11. In Figure 11-20, the computation time of the centralized approach appears to grow about cubically with the number of agents.

Complexity of MIRA We then prove Theorem 4 by combining the complexity of the three layers. Recall that, in MIRA, every agent runs an interior point method with only its own variables and constraints. Therefore, $n = N_V$ and $m = N_C$ for each agent. Hence, the worst-case computational complexity of the middle and bottom layers of each agent is:

$$O\left((N_V+N_C)^{2.807}\sqrt{N_C}\log_2\left(\frac{N_C/t^{(0)}}{\epsilon}\right)\right),$$

which does not involve the number of agents $|\mathcal{L}|$. As mentioned before, the number of iterations in the top layer is constant. Therefore, the worst-case *parallel* computational complexity of MIRA is constant regarding $|\mathcal{L}|$.

In MIRA, the parallel computation is conducted by $|\mathcal{L}|$ processors. Hence, the worst-

case *total* computational complexity of MIRA grows linearly with $|\mathcal{L}|$.

■

Note that Theorem 4 is concerned with the *worst-case* bound. Practically, parallel computation time typically grows with $|\mathcal{L}|$, although significantly more slowly than the centralized approach. In fact, in Figure 11-20, the parallel computation time of MIRA appears to grow about linearly with the number of agents. This is because iterations must be synchronized among all agents. When each agent computes its demand for risk by solving the non-linear optimization problem, the computation time diverges from agent to agent. In each iteration, all agents must wait until the slowest agent finishes computing its demand. As a result, MIRA slows down for large problems, as the expected computation time of the slowest agent grows.

9.6 Extension to Multiple Chance Constraints

This section presents an extension of MIRA that can solve a problem with multiple chance constraints.

In such a problem, we consider multiple markets, each of which deals with each chance constraint. Risks associated with different chance constraints are treated as different goods, which are traded in the corresponding markets. Each risk has its price; each agent takes as an input a set of the prices of all risks. At each iteration, each agent computes the demands for every risk given the set of prices. The central module solves a multiple root-finding problem to find equilibrium prices that equalize the aggregate demands and supplies at all markets.

Let $\mathcal{C}$ be the set of chance constraints. A multi-agent CCQSP planning problem with multiple chance constraints is formulated as follows.

Problem 18: Centralized deterministic optimization with multiple chance constraints

$$\begin{aligned}
\min_{\boldsymbol{u}_{0:N-1}^{1:M} \in \mathbb{U}^{N \cdot M}, \delta^{1:M} \succ 0} \quad & \sum_{l \in \mathcal{L}} J^l(\boldsymbol{u}_{0:N-1}^l, \bar{\boldsymbol{x}}_{l:N}^l) \\
\text{s.t.} \quad & \forall t \in \mathbb{T}^-, l \in \mathcal{L}, \quad \bar{\boldsymbol{x}}_{t+1}^l = \boldsymbol{A}_t^l \bar{\boldsymbol{x}}_t^l + \boldsymbol{B}_t^l \boldsymbol{u}_t^l \\
& \bigwedge_{c \in \mathcal{C}} \bigwedge_{l \in \mathcal{L}} \bigwedge_{i \in \mathcal{I}_c^l} \boldsymbol{h}_{i,c}^{lT} \bar{\boldsymbol{x}}_{t_i} - g_{i,c}^l \leq -m_{i,c}^l(\delta_{i,c}^l) \\
& \bigwedge_{c \in \mathcal{C}} \sum_{l \in \mathcal{L}} \sum_{i \in \mathcal{I}_c^l} \delta_{i,c}^l \leq \Delta_c^S,
\end{aligned}$$

where Δ_c^S is the risk bound for the chance constraint c.

We obtain a decentralized optimization method for Problem 18 in the same manner as in Section 9.3. Let p_c be the price of risk with regard to the chance constraint c. At each iteration, agents receive the prices p_c for all $c \in \mathcal{C}$, and solve the following optimization problem:

Problem 19: Decomposed optimization problem for lth agent with multiple chance constraints

$$\begin{aligned}
\min_{\boldsymbol{u}_{0:N-1}^l \in \mathbb{U}^N, \delta^l \succ 0} \quad & J^l(\boldsymbol{u}_{0:N-1}^l, \bar{\boldsymbol{x}}_{l:N}^l) + \sum_{c \in \mathcal{C}} \left(p_c \sum_{i \in \mathcal{I}_c^l} \delta_{i,c}^l \right) \qquad (9.25) \\
\text{s.t.} \quad & \forall t \in \mathbb{T}^-, \quad \bar{\boldsymbol{x}}_{t+1}^l = \boldsymbol{A}_t^l \bar{\boldsymbol{x}}_t^l + \boldsymbol{B}_t^l \boldsymbol{u}_t^l \\
& \bigwedge_{i \in \mathcal{I}^l} \boldsymbol{h}_{i,c}^{lT} \bar{\boldsymbol{x}}_{t_i} - g_{i,c}^l \leq -m_{i,c}^l(\delta_{i,c}^l)
\end{aligned}$$

The main difference from the problem with a single chance constraint (Problem 16) is the penalty term in the objective function (9.25). Each risk with regard to c penalizes the objective function with a factor p_c; the objective function adds up the penalties regarding all chance constraints.

Let $\boldsymbol{p} := \{p_c \mid c \in \mathcal{C}\}$ be the price vector, which includes all prices. Given $\boldsymbol{p}$, the agent

finds the optimal risk allocation $\delta_{i,c}^{l\star}(\boldsymbol{p})$ that minimizes the penalized cost. We denote it by $D_c^l(\boldsymbol{p}) := \sum_{i\in\mathcal{I}_c^l} \delta_{i,c}^{l\star}(p)$ the lth agent's demand for risk regarding the chance constraint c. In general, the prices are coupled in Problem 19, meaning that the price of one risk affects the demand for other risks[3]. Hence, $D_c^l(\boldsymbol{p})$ is a function of a price *vector*.

The central module finds an optimal price vector $\boldsymbol{p}^\star$ that minimizes the system cost $\sum_{l\in\mathcal{L}} J^l$ in (9.25). As in MIRA with a single chance constraint, the optimal price vector corresponds exactly to the equilibrium price vector, which equalizes the aggregate demands and supplies at *all* markets. In other words, the optimal price is the solution to the following root-finding problem:

Problem 20: Root-finding problem for the central module with multiple chance constraints

$$\text{Find} \quad \boldsymbol{p} \succeq 0 \quad \text{where} \quad \sum_{l\in\mathcal{L}} D_c^l(\boldsymbol{p}) = \Delta_c^S, \ \forall c \in \mathcal{C}.$$

As in the single chance-constraint case, an optimal solution to Problems 19 and 20 is guaranteed to be an optimal solution to Problem 18. Furthermore, if a solution to Problem 18 exists, then Problems 19 and 20 also have solutions.

However, the convergence guarantee in the single chance-constraint case does not apply to the multiple chance-constraint case. This is because the root-finding problem (Problem 20) involves a vector $\boldsymbol{p}$. Brent's method, the root finding algorithm used for the single chance-constraint case, can only take a scalar variable. Instead, other root-finding methods, such as Newton's method or the subgradient method, are used to solve Problem 20. Convergence of these root-finding methods can also be guaranteed, but with stronger conditions than Brent's method in general. See [6] for further discussions on convergence.

[3]In economics terminology, the market has non-zero cross-elasticity of demand.

9.7 Conclusion

This chapter presented the MIRA algorithm, which can solve a full-horizon CCQSP planner with convex state constraints in a decentralized manner. We proved that MIRA gives exactly the same optimal solution as the centralized optimization approach. We also proved that MIRA converges to the optimal solution, if one exists. These theoretical results are validated by simulations presented in Section 11.1.11. In the next chapter, we present a receding horizon CCQSP executive, dp-Sulu, which can handle couplings through chance, state, and temporal constraints in a decentralized manner.

Chapter 10

Distributed, Receding Horizon CCQSP Execution

In this chapter we develop dp-Sulu, a distributed model-based executive that solves a multi-agent plan (CCQSP) execution problem with a receding-horizon planning approach. We consider the general form of the CCQSP execution problem (Problem 2), which includes a non-convex state space and a flexible schedule. The three main challenges in distributed CCQSP planning are how to decompose coupled temporal constraints, chance constraints, and state constraints. These three challenges are overcome by three innovations: Multi-agent Heuristic Risk Allocation, Bi-stage Robust Collision Avoidance, and Robust Feasible Temporal Constraint Decomposition. Using the three methods, the central module of dp-Sulu decomposes a coupled CCQSP into decoupled CCQSPs for each agent. Then each agent executes its own CCQSP using p-Sulu, presented in Chapter 8.

For example, consider the multi-vehicle CCQSP planning scenario shown in Figure 10-1-(a), where two aerial vehicles are going to land at the same airport. We assume that the control tower allows Vehicle 1 to land first. See the temporal constraints in the CCQSP in Figure 10-1-(b). Both vehicles take at least 10 time units to get to the airport, and must land within 20 time units due to fuel constraints. In order for Vehicle 2 to land safely, the two landings must be separated by an interval of at least four time units. Such a temporal constraint introduces coupling since it involves the two vehicles. The CCQSP specifies four chance constraints. The first two, c_1 and c_2, require each vehicle to stay away from

the no-fly zone with the probability of Δ_S^1 and Δ_S^2, respectively. The third one, c_3, requires that the two vehicles arrive at the airport within the temporal bounds with the probability of Δ_D. Such a chance constraint introduces coupling since the risk is defined over the two vehicles. See the fire-fighting scenario in Section 9.1 for another example of coupled chance constraints. The fourth chance constraint, c_4, requires that the two vehicles do not collide with each other with the probability of Δ_C. Such a chance constraint involves a coupled state constraint, since collision avoidance requires the two vehicle positions to be separated from each other by a certain margin.

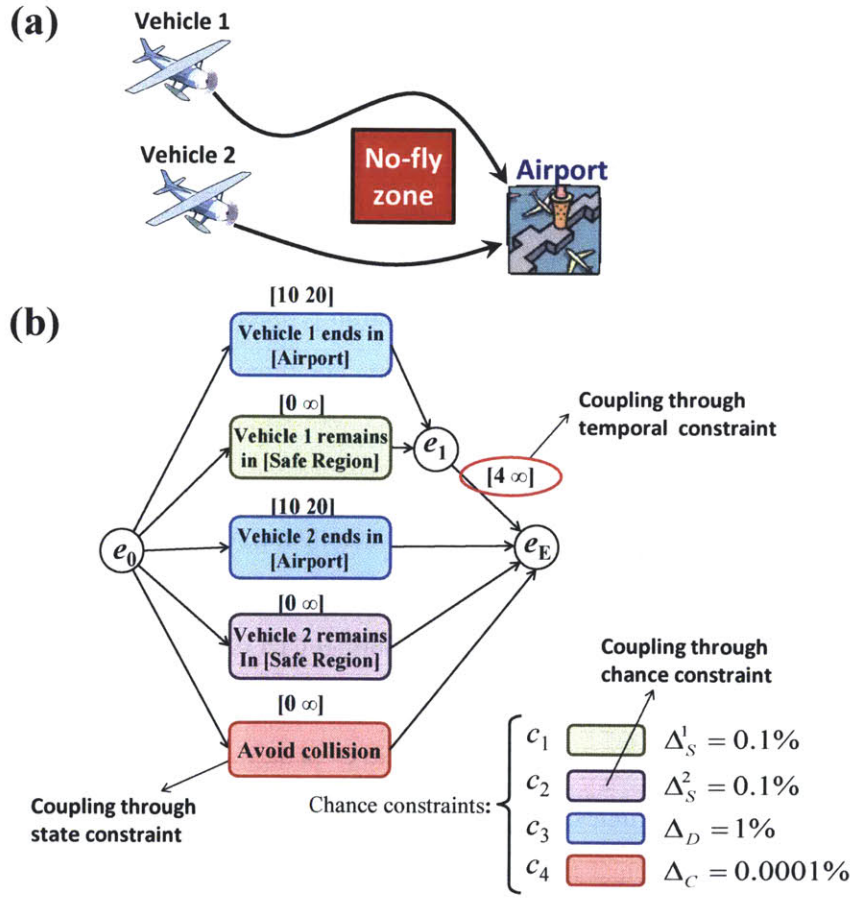


Figure 10-1: A sample multi-vehicle CCQSP planning scenario where two vehicles must land at the same airport with at least four minutes intervals. Two vehicles are coupled through state, temporal, and chance constraints in this scenario. "Safe Region" means the entire space except the no-fly zone.

Figure 10-2 shows the overall architecture of dp-Sulu. Like the Market-based Itera-

tive Risk Allocation (MIRA) algorithm presented in the previous chapter, dp-Sulu consists of a centralized and decentralized part. A central module, which executes the centralized portion of dp-Sulu, takes a CCQSP with coupled constraints as an input, and outputs CCQSPs with decoupled constraints for each agent. Each agent takes the sub-CCQSP as an input, and outputs its own control sequence and schedule by using p-Sulu. The decoupled CCQSPs are updated by the central module at every planning cycle. Since the most computationally intensive part of dp-Sulu (MILP optimization) is distributed among the agents, its scalability is significantly improved compared to the centralized CCQSP execution.

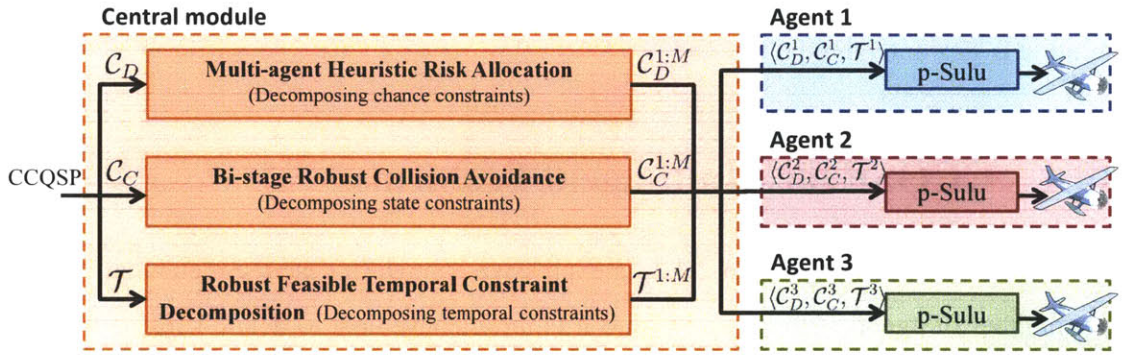


Figure 10-2: Structure of the dp-Sulu algorithm

The central module of dp-Sulu decomposes a coupled CCQSP by using three innovative methods. See Figure 10-2 for the three types of couplings and the three decoupling methods. Firstly, coupled chance constraints are decomposed by the *Multi-agent Heuristic Risk Allocation* approach, which is an extension of the Heuristic Risk Spending approach presented in Sections 8.2.2 and 8.4. Secondly, coupled state constraints (collision avoidance constraints) are decomposed by the *Bi-stage Robust Collision Avoidance (BRCA)* method. In BRCA, each agent solves the path planning problem without the collision avoidance constraints in the first iteration. If the resulting solution satisfies the collision avoidance constraints, the agents executes the control sequence of the solution. Otherwise, the central module constructs decomposed chance constraints using the solution of the first iteration, and each agent plans the path again with the decomposed constraints. Finally, coupled temporal constraints are decomposed by the *Robust Feasible Temporal Constraint Decomposition* approach. This approach takes a set of simple temporal constraints as an input,

and outputs a set of absolute temporal constraints for every agent. The absolute temporal constraints impose temporal bounds on the execution time of each atomic event, instead of the duration between two events. Therefore, each agent can execute its own events independently from the execution schedules of other agents.

As with p-Sulu, computation time and solution feasibility is more important than optimality for dp-Sulu, since it runs in real time. The computation time at every planning cycle must be within the planning horizon, and the obtained solution must be feasible. For this purpose, we design dp-Sulu so that it does not require agents to iteratively solve planning problems. In dp-Sulu, each vehicle has to solve p-Sulu at most twice, and the resulting solution is guaranteed to be feasible.

10.1 CCQSP for Multi-agent Problems

The CCQSP, defined in Section 2.4.3, is expressive enough to describe plans that involve any number of agents. However, we find it is more convenient to define substructures of CCQSP when expressing a multi-agent plan. More specifically, these substructures divide episodes and chance constraints that are specific to an individual agent, and those that couple to multiple agents. Note that the introduction of the CCQSP substructures does not extend or reduce the expressiveness of a CCQSP.

We consider a multi-agent problem with M agents. The state vector $\boldsymbol{x}_t \in \mathbb{R}^N$ of the multi-agent problem consists of the state vectors of all agents:

$$\boldsymbol{x}_t := \begin{bmatrix} \boldsymbol{x}_t^1 \\ \boldsymbol{x}_t^2 \\ \vdots \\ \boldsymbol{x}_t^M \end{bmatrix}.$$

The superscripts represent the indices of the agents. We denote by $\mathcal{X}^l$ the subspace in $\mathbb{R}^N$ that involves the components of $\boldsymbol{x}_t^l$. Hence, $\boldsymbol{x}_t^l \in \mathcal{X}^l$ and $\bigcup_{i=1}^M \mathcal{X}^l = \mathbb{R}^N$.

Recall that a CCQSP is a four-tuple $P = \langle \mathcal{E}, \mathcal{A}, \mathcal{T}, \mathcal{C} \rangle$, where $\mathcal{E}$ is a set of discrete events, $\mathcal{A}$ is a set of *episodes*, $\mathcal{T}$ is a set of *simple temporal constraints*, and $\mathcal{C}$ is a set of

chance constraints. Below, we define substructures of $\mathcal{A}$ and $\mathcal{C}$.

10.1.1 Categorizing Episodes

We consider the following categories of episodes.

- $\mathcal{A}_D^l \subseteq \mathcal{A}$: A set of episodes that only involve the lth agent.
- $\mathcal{A}_C \subseteq \mathcal{A}$: A set of episodes that involve multiple agents.

In other words, episodes in $\mathcal{A}_D^l$ impose constraints that are decoupled from other agents, while episodes in $\mathcal{A}_C$ impose state constraints that are coupled. We refer to the chance constraints in $\mathcal{A}_D^l$ and $\mathcal{A}_C$ as *decoupled episodes* and *coupled episodes*, respectively.

There are four decoupled episodes in Figure 10-1: "Vehicle 1 ends in [Airport]" and "Vehicle 1 remains in [Safe Region]", which are categorized to $\mathcal{A}_D^1$, and "Vehicle 2 ends in [Airport]" and "Vehicle 2 remains in [Safe Region]," which are categorized to $\mathcal{A}_D^2$. The "Avoid collision" episode is an example of a coupled episode that is categorized into $\mathcal{A}_C$.

More formally,

$$\begin{aligned} \mathcal{A}_D^l &:= \{a \in \mathcal{A} \mid \forall_{k \neq l}\, R_a \cap \mathcal{X}^k = \mathcal{X}^k\}. \\ \mathcal{A}_C &:= \{a \in \mathcal{A} \mid \forall_l\, a \notin \mathcal{A}_D^l\}. \end{aligned}$$

In other words, an episode a is categorized into $\mathcal{A}_D^l$ if and only if its state constraint does not constrain the values of the states of other agents than l. An episode a is categorized into $\mathcal{A}_C$ if it does not belong to any of $\mathcal{A}_D^l$.

Among various types of coupled episodes $\mathcal{A}_C$, our particular interest in this chapter is collision avoidance. A collision avoidance is a special case of the remain-in episode, where the feasible region is the entire space except a hyperdimensional obstacle that contains the origin point.

10.1.2 Categorizing Chance Constraints

In this section we classify the chance constraints in $\mathcal{C}$ into the following four categories:

- $\mathcal{C}_S^l \subseteq \mathcal{C}$: Chance constraints that involve only decoupled episodes of the lth agent.
- $\mathcal{C}_D \subseteq \mathcal{C}$: Chance constraints that involve only decoupled episodes but multiple agents.
- $\mathcal{C}_C \subseteq \mathcal{C}$: Chance constraints that involve only coupled episodes between multiple agents.
- $\mathcal{C}_M \subseteq \mathcal{C}$: Chance constraints that involve both coupled and decoupled episodes.

In Figure 10-1, $c_1 \in \mathcal{C}_S^1$ and $c_2 \in \mathcal{C}_S^2$, since they only involve a single vehicle. c_3 is an example of a chance constraint categorized into $\mathcal{C}_D$. Such a chance constraint introduces a coupling between the two vehicles, although its episodes (state constraints) are decoupled. When we say "a coupling through chance constraints," we refer to a coupling between agents caused by the chance constraints in this category. This type of chance constraints is also considered in Section 9. The collision avoidance requirement c_4 is classified into $\mathcal{C}_C$. Such a chance constraint involves coupled episodes, which introduces a coupling through state constraints. We don't consider chance constraints in $\mathcal{C}_M$ in this chapter, in order to simplify the discussion. It is straightforward to extend the Bi-stage Robust Collision Avoidance approach, explained in Section 10.4, to such chance constraints.

Recall that a chance constraint c is a pair $\langle \Psi_c, \Delta_c \rangle$, where $\Psi_c \subseteq \mathcal{A}$ is a set of episodes associated with the chance constraint c, and Δ_c is a user-specified risk bound (Definition 13 in Section 2.4.3). The four categories are formally defined as follows:

$$
\begin{aligned}
\mathcal{C}_S^l &:= \{c \in \mathcal{C} \mid \Psi_c \subseteq \mathcal{A}_D^l\} \\
\mathcal{C}_D &:= \{c \in \mathcal{C} \mid \forall_l\, c \notin \mathcal{C}_S^l, \Psi_c \subseteq \bigcup_{i=1}^{M} \mathcal{A}_D^l\} \\
\mathcal{C}_C &:= \{c \in \mathcal{C} \mid \Psi_c \subseteq \mathcal{A}_C\} \\
\mathcal{C}_M &:= \{c \in \mathcal{C} \mid c \notin \bigcup_{l=1}^{M} \mathcal{C}_S^l \cup \mathcal{C}_D \cup \mathcal{C}_C\}.
\end{aligned}
$$

A single-agent chance constraint in $\mathcal{C}_S^l$ only involves decoupled episodes of a specific agent. A chance constraint in $\mathcal{C}_D$ involves decoupled episodes of multiple agents. A chance con-

straint in $\mathcal{C}_C$ involves coupled episodes. Any chance constraints that do not belong to the above three categories fall into $\mathcal{C}_M$, which involves both coupled and decoupled episodes.

10.2 dp-Sulu Algorithm Overview

We outline the dp-Sulu algorithm in Algorithm 14.

As discussed in the introduction, agents can be coupled in three ways: by sharing chance constraints in $\mathcal{C}_D$, by sharing episodes (state constraints) in $\mathcal{A}_C$ (hence, sharing chance constraints in $\mathcal{C}_C$), or by sharing simple temporal constraints in $\mathcal{T}$. These couplings are decomposed by the three new methods proposed in this chapter, as shown in Figure 10-2.

The coupled chance constraints, $\mathcal{C}_D$, are handled by the Multi-agent Heuristic Risk Allocation approach. Multi-agent Heuristic Risk Allocation takes a set of coupled chance constraints as an input, and outputs a set of decomposed chance constraints for every agent, $\mathcal{C}_D^{1:M}$, with a feasible risk allocation among agents (Line 4 in Algorithm 14). The Multi-agent Heuristic Risk Allocation approach is explained in detail in Section 10.3.

Bi-stage Robust Collision Avoidance (BRCA) addresses the coupling through state constraints (collision avoidance constraints). In BRCA, each agent solves the path planning problem without the collision avoidance constraints in the first iteration (Line 8). We call the resulting solution $(\tilde{\boldsymbol{x}}_t^{1:M}, \tilde{\boldsymbol{u}}_t^{1:M})$ the candidate solution. If the candidate solution satisfies the collision avoidance constraints, the executive executes the control sequence of the candidate solution (Line 12). Otherwise, the central module constructs decomposed chance constraints $\mathcal{C}_C^l$, which involves decomposed episodes $\mathcal{C}_D^l$, using the candidate solution. Each agent plans the path again with the decomposed constraints (Line 18). We call the new solution the adjusted solution. The adjusted solution is guaranteed to satisfy the collision avoidance constraints. Hence, each agent executes the control sequence of the adjusted solution (Line 12). The BRCA approach is explained in detail in Section 10.4.

Simple temporal constraints, $\mathcal{T}$, couple agents essentially because they are relative constraints, meaning that they impose upper and lower bounds on the *duration* between the execution times of two events. The Robust Feasible Temporal Constraint Decomposition

approach derives a set of absolute temporal constraints for each agent, denoted by $\mathcal{T}^l$ in Figure 10-2, by solving a linear programming. The set of absolute temporal constraints is a sufficient condition of the original simple temporal constraints. This approach takes a set of simple temporal constraints $\mathcal{T}$ and a partially assigned schedule σ as an input, and outputs a set of absolute temporal constraints for every agent, $\mathcal{T}^{1:M}$ (Line 5). The absolute temporal constraints impose temporal bounds on the execution time of each atomic event. The linear programming problem is guaranteed to have a feasible solution if the given simple temporal constraints are feasible. Our approach is suitable for real-time execution because it does not involve iteration or backtracking. The Robust Feasible Temporal Constraint Decomposition approach is explained in detail in Section 10.5.

10.3 Multi-agent Heuristic Risk Allocation: Decomposition of Chance Constraints

In this section, we present the Multi-agent Heuristic Risk Allocation approach, which decomposes coupled chance constraints with decoupled episodes $\mathcal{C}_D$. The implementation of the MultiAgentHeuristicRiskAllocation function in Line 4 in Algorithm 14 is shown in Algorithm 15. The algorithm is explained in detail later in this section.

Recall that we developed a heuristic risk spending approach in Section 8.4, which allows an agent to take risk at an approximately uniform rate over time. We extend this approach to a multi-agent problem. The amount of risk the multi-agent system can take at each planning cycle is determined by the heuristic risk spending approach. Then, the agents divide that portion of risk in proportion to the remaining distances to the goal. For example, in Figure 10-3, Vehicle 2 is allocated a larger portion of risk since it has a longer distance to the goal.

More formally, a coupled chance constraint with decoupled episodes, $c \in \mathcal{C}_D$, is described as follows:

$$\Pr\left[\bigwedge_{l=1}^{M}\bigwedge_{i\in\mathcal{I}_c(s)} \boldsymbol{h}_i^{lT}\boldsymbol{x}_{t_i}^l - g_i^l \leq 0\right] \geq 1-\Delta_D,$$

Algorithm 14 dp-Sulu algorithm

function dp-Sulu($ccqsp = \langle \mathcal{E}, \mathcal{A}, \mathcal{T}, \mathcal{C} \rangle$)

1: $\Delta_A \leftarrow \Delta_D$;
2: $\sigma^{1:M}(e_0) = 0$ //Initialize the schedule
3: **while** e_E in $ccqsp$ is not executed **do**
4: $\mathcal{C}_D^{1:M} \leftarrow$ MultiAgentHeuristicRiskAllocation($\mathcal{C}_D$); //Decompose coupling chance constraints
5: $\mathcal{T}^{1:M} \leftarrow$ RobustFeasibleTemporalConstraintDecomposition($\mathcal{T}, \sigma^{1:M}$); //Decompose coupling temporal constraints
6: **for** $l = 1 \cdots M$ **do**
7: $ccqsp^l \leftarrow \langle \mathcal{E}, \mathcal{A}_D^l, \mathcal{T}^l, \mathcal{C}_S^l \cup \mathcal{C}_D^l \rangle$
8: $(\tilde{\boldsymbol{x}}_t^l, \tilde{\boldsymbol{u}}_t^l, \sigma^l, \Delta_{E,c}^l) \leftarrow agent[l]$.runPSulu($ccqsp^l$); //Obtain a candidate solution
9: **end for**
10: **if** $\tilde{\boldsymbol{x}}_t^{1:M}$ satisfies all chance constraints in $\mathcal{C}_C$ **then**
11: **for** $l = 1 \cdots M$ **do**
12: $agent[l]$.execute($\tilde{\boldsymbol{u}}_t^l$);
13: **end for**
14: **else**
15: $(\mathcal{C}_C^{1:M}, \mathcal{A}_C^{1:M}) \leftarrow$ DecomposeCollisionAvoidanceConstraint($\mathcal{C}_C, \tilde{\boldsymbol{x}}_t^{1:M}$); //Decompose coupling state constraints
16: **for** $l = 1 \cdots M$ **do**
17: $ccqsp^l \leftarrow \langle \mathcal{E}, \mathcal{A}_D^l \cup \mathcal{A}_C^l, \mathcal{T}^l, \mathcal{C}_S^l \cup \mathcal{C}_D^l \cup \mathcal{C}_C^l \rangle$
18: $(\boldsymbol{x}_t^l, \boldsymbol{u}_t^l, \sigma^l, \Delta_{E,c}^l) \leftarrow agent[l]$.runPSulu($ccqsp^l$); //Obtain an adjusted solution
19: $agent[l]$.execute($\boldsymbol{u}_t^l$);
20: **end for**
21: **end if**
22: **for** $c \in \mathcal{C}_D$ **do**
23: $\Delta_c \leftarrow \Delta_c - \sum_{l=1}^M \Delta_{E,c}^l$;
24: **end for**
25: **end while**

where l is the index of agents and Δ_D is the risk bound. At the beginning of the algorithm, we set the available risk Δ_A equal to Δ_D (Line 1 in Algorithm 14). At the nth planning cycle, dp-Sulu allocates the $\Delta_P^{n,l}$ of risk for the current planning horizon using the following rule:

$$\Delta_P^{n,l} = \min\left(\frac{d_{horizon}^l}{\sum_{l=1}^M d_{Goal}^l}, \beta\right) \Delta_A^n \tag{10.1}$$

where $0 < \beta < 1$ is a constant, M is the number of agents, d_{Goal} is the distance from the current position to the final destination, and $d_{horizon}$ is the maximum distance the vehicle can travel in the nth planning horizon. Refer to Equation (8.7) for the derivation of $d_{horizon}$.

Algorithm 15 Implementation of MultiAgentHeuristicRiskAllocation function in Line 4 of Algorithm 14

function MultiAgentHeuristicRiskAllocation($\mathcal{C}_D$)

1: **for** $c \in \mathcal{C}_D$ **do**
2: **for** $l = 1 \cdots M$ **do**
3: $\Delta_P^{n,l} \leftarrow \min\left(\frac{d^l_{horizon}}{\sum_{l=1}^{M} d^l_{Goal}}, \beta\right) \Delta_c$;
4: $\Phi^l \leftarrow \left\{a \in \Phi_c \mid a \in \mathcal{A}^l_D\right\}$;
5: $\mathcal{C}^l_D \leftarrow \mathcal{C}^l_D \cup \langle \Phi^l, \Delta_P^{n,l} \rangle$;
6: **end for**
7: **end for**
8: **return** $\mathcal{C}_D^{1:M}$;

Equation (10.1) corresponds to Line 3 in Algorithm 15.

The rationale behind the heuristic (10.1) is as follows. The available risk Δ_A should be distributed among the agents according to the remaining distance to the final destinations. Hence, let Δ^l be the risk that should be allocated to the lth vehicle, and

$$\Delta^l = \frac{d^l_{Goal}}{\sum_{l=1}^{M} d^l_{Goal}} \Delta_A. \tag{10.2}$$

Then, a portion of Δ^l should be allocated to the current planning horizon in proportion to the ratio of the maximum length of the path within the current planning horizon to the distance to the lth agent's final destination:

$$\Delta_P^{n,l} = \frac{d^l_{horizon}}{d^l_{Goal}} \Delta^l. \tag{10.3}$$

By substituting (10.2) into (10.3), we have:

$$\Delta_P^{n,l} = \frac{d^l_{horizon}}{\sum_{l=1}^{M} d^l_{Goal}} \Delta_A^n$$

The constant β in (10.1) imposes an upper bound on the risk allocation at each planning horizon.

The episodes involved in the chance constraints in $\mathcal{C}_D$ are decoupled from the beginning. For each agent, dp-Sulu constructs decomposed chance constraints by combining the decoupled episodes and the risk allocation $\Delta_P^{n,l}$ (Lines 4 and 5). The function returns the

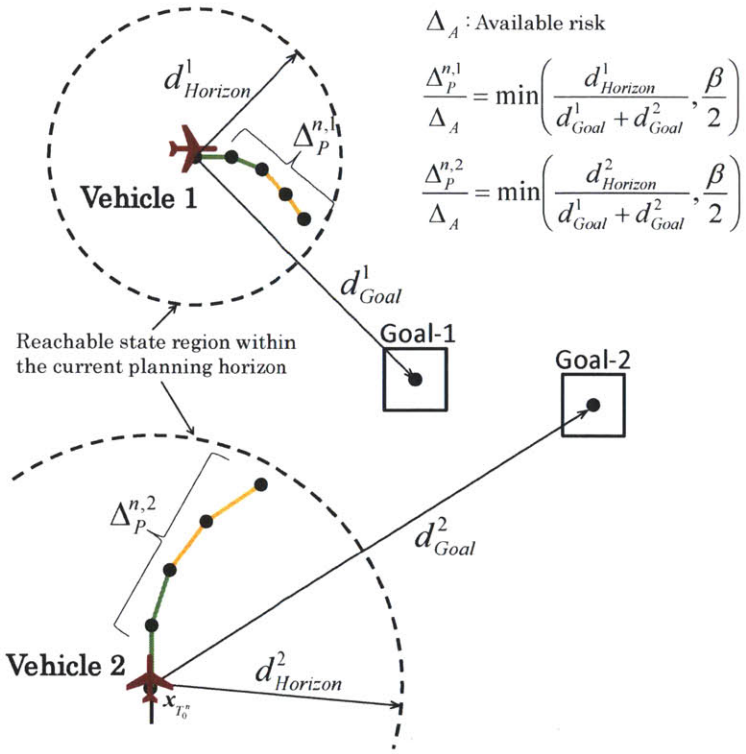


Figure 10-3: Multi-agent Heuristic Risk Allocation

set of decomposed chance constraints (Line 8). As with p-Sulu, the available risk bound is reduced by the amount of risk allocated to the execution horizon at every planning cycle. This operation is implemented in Line 23 of Algorithm 14.

Next, we consider the coupling state constraints.

10.4 Bi-stage Robust Collision Avoidance: Decomposition of State Constraints

In this section, we present the Bi-stage Robust Collision Avoidance (BRCA) approach that decomposes a chance constraint in $\mathcal{C}_C$, which is defined over coupled state constraints $\mathcal{A}_C$. In order to simplify the mathematical discussion, we assume that there is only one chance constraint in $\mathcal{C}_C$, with a risk bound Δ_C. The full algorithm that can handle multiple chance constraints in $\mathcal{C}_C$ is shown in Algorithms 14 and 15. Since our main application is vehicle

path planning, we particularly consider robust collision avoidance constraints.

The outline of the BRCA approach is shown in the flow chart in Figure 10-4. At the first iteration, each agent solves the path planning problem without the collision avoidance constraints in order to obtain a candidate solution. If the candidate solution satisfies the collision avoidance constraints, the executive executes the control sequence of the candidate solution. Otherwise, the central module constructs decomposed collision avoidance constraints C_C^l. Each agent plans the path again with the decomposed constraints in order to obtain an adjusted solution. The control sequence of the adjusted solution is executed by each agent.

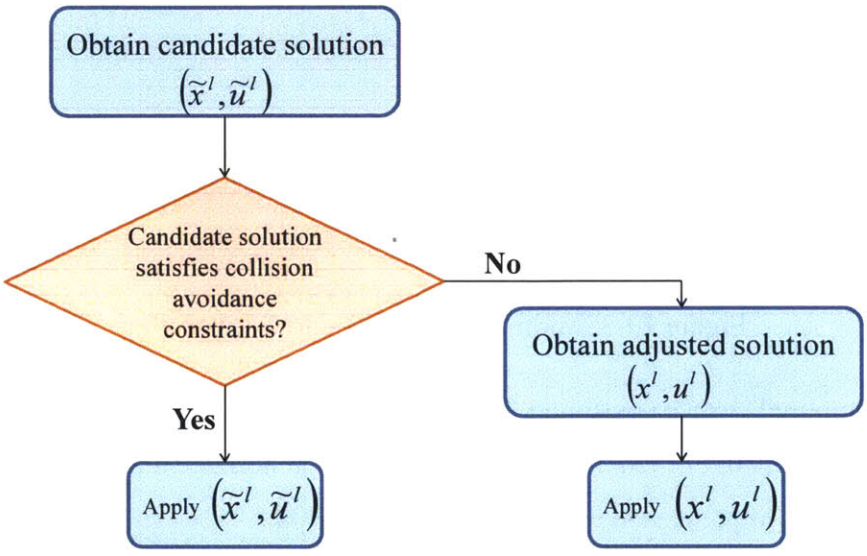


Figure 10-4: Multi-agent Heuristic Risk Allocation

The BRCA approach is implemented in Lines 6 - 21 of Algorithm 14. The implementation of the DecomposeCollisionAvoidanceConstraint function in Line 15 is shown in Section 10.4.3.

Suppose there are M agents, and superscripts k and l represent the indices of the agents. The constraint we consider in this section is represented as follows:

$$\Pr\left[\bigwedge_{1\leq k\leq M}\bigwedge_{1\leq l<k}\bigwedge_{t\in\mathbb{T}^n}\bigvee_{i} h_i^T(\boldsymbol{x}_t^k - \boldsymbol{x}_t^l) \geq g_i\right] \geq 1 - \Delta_C^n, \tag{10.4}$$

where Δ_C^n is the risk allocated to the current planning horizon. As before, Δ_C^n is found heuristically as follows:

$$\Delta_C^n = \min\left(\frac{\sum_{l=1}^{M} d_{horizon}^l}{\sum_{l=1}^{M} d_{Goal}^l}, \beta\right)\Delta_C. \tag{10.5}$$

Note that the numerator of (10.5) includes the summation of $d_{horizon}^l$ of all agents. This is because the collision avoidance constraint (10.4) involves the state variable of all agents.

The chance constraint (10.4) requires that any two vehicles in the system satisfy the constraint

$$\bigvee_i h_i^T(\boldsymbol{x}_t^k - \boldsymbol{x}_t^l) \geq g_i \tag{10.6}$$

for all time steps within the planning horizon with the probability of $1 - \Delta_C^n$. Figure 10-5 illustrates the constraint (10.6). $h_1 \cdots h_4$ are four direction vectors. The state constraint $h_i^T(\boldsymbol{x}_t^k - \boldsymbol{x}_t^l) \geq g_i$ requires that the distance in the directions h_i is more than g_i. We have a disjunction in (10.6) since at least one of these constraints must be satisfied in order to avoid collision.

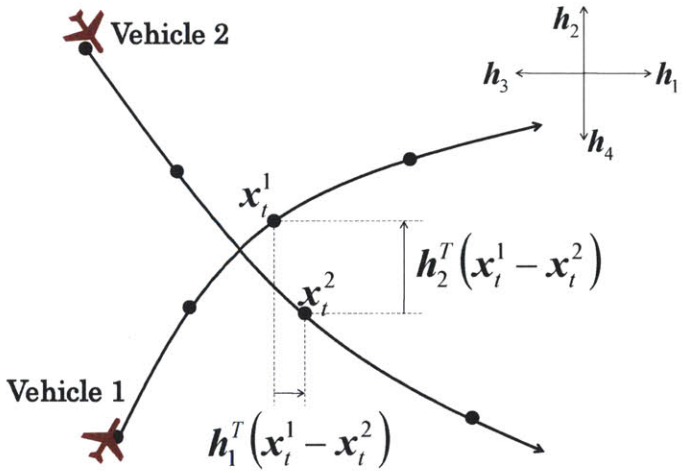


Figure 10-5: Robust collision avoidance constraint

Note that the state constraint $h_i^T(\boldsymbol{x}_t^k - \boldsymbol{x}_t^l) \geq g_i$ couples two agents since it includes the state variables of the two agents.

10.4.1 Risk allocation approach

As always, we apply the risk allocation approach to (10.4) to obtain:

$$\bigwedge_{1\le k\le M}\ \bigwedge_{1\le l<k}\ \bigwedge_{t\in\mathbb{T}^n}\ \bigvee_{i} h_i^T(\bar{\boldsymbol{x}}_t^k - \bar{\boldsymbol{x}}_t^l) \ge g_i + m_{t,i}^{k,l}(\delta_t^{k,l}) \tag{10.7}$$

$$\sum_{1\le k\le M}\ \sum_{1\le l<k}\ \sum_{t\in\mathbb{T}^n} \delta_t^{k,l} \le \Delta_C^n \tag{10.8}$$

where

$$m_{t,i}^{k,l}(\delta_t^{k,l}) = -\sqrt{2\boldsymbol{h}_i^T(\Sigma_{x_t^k} + \Sigma_{x_t^l})\boldsymbol{h}_i}\ \mathrm{erf}^{-1}(2\delta_t^{k,l} - 1). \tag{10.9}$$

Note that the covariance of $\boldsymbol{x}_t^k - \boldsymbol{x}_t^l$ is $\Sigma_{x_t^k} + \Sigma_{x_t^l}$. Corollary 4 in Chapter 6 guarantees that the solution that satisfies 10.8 always satisfies 10.4. Hence, the constraint (10.4) can be safely replaced with (10.8).

In the newly proposed Bi-stage Robust Collision Avoidance (BRCA) approach, each agent finds its own paths that satisfy the constraint (10.4) by solving p-Sulu problem at most twice, as illustrated in Figure 10-6.

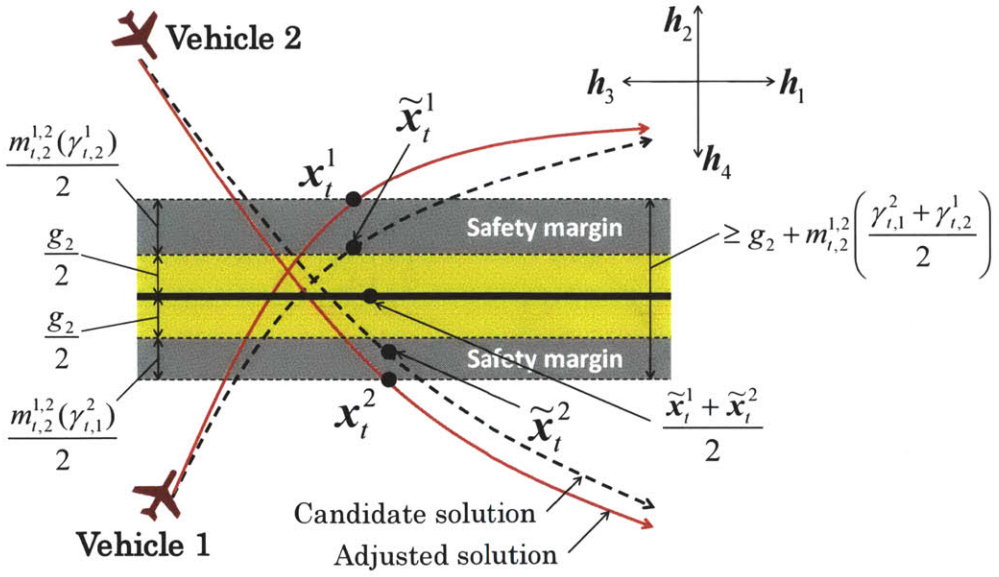


Figure 10-6: Overview of the Bi-stage Robust Collision Avoidance (BRCA) approach. If the candidate solutions do not satisfy the collision avoidance constraint, it deviates the solutions to create enough safety margin.

10.4.2 First Iteration

In the first iteration, each vehicle solves its own problem *without* imposing the collision avoidance constraint (10.7) and (10.8) (See Algorithm 14, Line 8). As a result, each agent obtains a *candidate solution* $(\tilde{\boldsymbol{x}}_t^{1:M}, \tilde{\boldsymbol{u}}_t^{1:M})$, which is reported to the central module. The dashed lines in Figure 10-6 represent the candidate solution.

Then, the central module evaluates whether the candidate solution satisfies the collision avoidance constraint (10.4). There are two evaluation methods:

1. Numerical evaluation of (10.4) using a Monte-Carlo simulation
2. Analytical evaluation of the approximated constraints (10.7) and (10.8)

The analytical evaluation is more conservative but computationally efficient than the numerical evaluation.

If the candidate solution satisfies the collision avoidance constraint, each agent executes its own control sequence of the candidate solution $\tilde{\boldsymbol{u}}_t^l$ (Line 12).

10.4.3 Second Iteration

Computation of minimum conflict direction If the candidate solution does not satisfy the collision avoidance constraint, the central module decomposes the collision avoidance constraint, and each agent solves a p-Sulu problem again with the decomposed constraint. Roughly speaking, the decomposed collision avoidance constraints can be considered as an imaginary "wall" between the two agents. The thick horizontal line in Figure 10-6 represents the wall. It goes through the midpoint of the two vehicles' location in the candidate solution at each time step. The "wall" is perpendicular to the direction vector $h_t^{k,l\star}$ that maximizes the probability of satisfying the individual chance constraint (10.7). More specifically,

$$h_t^{k,l\star} := h_{i^\star(t,k,l)},$$

where the index $i^\star(t,k,l)$ is chosen for each t,k,l such that

$$\begin{aligned} i^\star(t,k,l) &:= \arg\max_i \Pr\left[h_i^T(\tilde{\boldsymbol{x}}_t^k - \tilde{\boldsymbol{x}}_t^l) \geq g_i\right] \\ &= \arg\max_i \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{h_i^T(\bar{\tilde{\boldsymbol{x}}}_t^k - \bar{\tilde{\boldsymbol{x}}}_t^l) - g_i}{\sqrt{2\boldsymbol{h}_i^T(\Sigma_{\boldsymbol{x}_t^k} + \Sigma_{\boldsymbol{x}_t^l})\boldsymbol{h}_i}}\right)\right]. \end{aligned} \tag{10.10}$$

In the example of Figure 10-6, $h_t^{1,2\star} = h_2$. Intuitively, $h_t^{k,l\star}$ is the direction in which the minimum deviation is required for the candidate solution, in order to satisfy the collision avoidance constraint. Hence, we refer to $h_t^{k,l\star}$ as the *minimum conflict direction*. We denote by $m_t^{k,l\star}(\cdot)$ the safety margin function in the minimum conflict direction:

$$m_t^{k,l\star}(\gamma_{t,k}^l) := m_{t,i^\star(t,k,l)}^{k,l}(\gamma_{t,k}^l).$$

Decomposed collision avoidance constraint For every (t,k,l) such that $k > l$, the central module imposes the following constraint on the vehicle k:

$$h_t^{k,l\star T}\left(\bar{\boldsymbol{x}}_t^k - \frac{\tilde{\boldsymbol{x}}_t^k + \tilde{\boldsymbol{x}}_t^l}{2}\right) \geq \frac{g_i}{2} + \frac{1}{2}m_t^{k,l\star}(\gamma_{t,l}^k), \tag{10.11}$$

while it imposes the following on the vehicle l:

$$h_t^{k,l\star T}\left(\frac{\tilde{\boldsymbol{x}}_t^k + \tilde{\boldsymbol{x}}_t^l}{2} - \bar{\boldsymbol{x}}_t^l\right) \geq \frac{g_i}{2} + \frac{1}{2}m_t^{k,l\star}(\gamma_{t,k}^l). \tag{10.12}$$

In the example in Figure 10-6 where $k = 1$ and $l = 2$, (10.11) requires Vehicle 1 to stay above the "wall" with the margin $\frac{g_i}{2} + \frac{1}{2}m_t^{k,l\star}(\gamma_{t,l}^k)$ at the time step t, while (10.12) requires Vehicle 2 to stay below the wall with the margin $\frac{g_i}{2} + \frac{1}{2}m_t^{k,l\star}(\gamma_{t,k}^l)$ at the same time step. Note that the midpoint of the candidate solution $\frac{\tilde{\boldsymbol{x}}_t^k + \tilde{\boldsymbol{x}}_t^l}{2}$ is a constant. Therefore, (10.11) only involves variables associated to the kth agent, while (10.12) only involves variables associated to the lth agent. Hence, these constraints are handled by each agent in a distributed manner.

It is very important to note that both (10.11) and (10.12) share the same safety margin function $\frac{1}{2}m_t^{k,l\star}(\cdot)$, instead of using the safety margin based on their own covariance. In

other words, although $\boldsymbol{x}_t^k$ has the covariance $\Sigma_{\boldsymbol{x}_t^k}$, the safety margin is computed using a different covariance $(\Sigma_{\boldsymbol{x}_t^k} + \Sigma_{\boldsymbol{x}_t^l})/4$. This covariance is typically smaller than the real covariance. This mathematical trick plays an important role in proving later that the decomposed collision avoidance constraints (10.11) and (10.12) are the sufficient condition of the original collision avoidance constraint.

The newly introduced variable $\gamma_{t,l}^k$ is a *pseudo risk bound* of collision. Although the pseudo risk bounds are in a very similar form to the risk bounds, they do not have physical meaning since each vehicle assumes a different covariance than the real one. Note that $\gamma_{t,l}^k$, which belongs to the kth vehicle, is a different variable than $\gamma_{t,k}^l$, which belongs to the lth vehicle. The two pseudo risk bounds $\gamma_{t,l}^k$, $\gamma_{t,k}^l$ are independent from each other, hence allowing the two agents to optimize pseudo risk allocation in a distributed manner.

Each vehicle k is allowed to optimize its pseudo risk allocation $\gamma_{t,l}^k$ with the following constraint:

$$\sum_{l \in \mathcal{L}^{-k}} \sum_{t \in \mathbb{T}^n} \gamma_{t,l}^k \leq \Gamma_C^{n,k}, \tag{10.13}$$

where the set $\mathcal{L}^{-k}$ is defined as follows:

$$\mathcal{L}^{-k} := \{1 \cdots k-1, k+1 \cdots M\}.$$

The constant $\Gamma_C^{n,k}$ is the pseudo risk allocated to the kth vehicle for the nth planning horizon. The central module is responsible for allocating the pseudo risk among the agents so that

$$\sum_{k=1}^{M} \Gamma_C^{n,k} \leq 2\Delta_C^n. \tag{10.14}$$

Note that the summation of the pseudo risk over all agents is bounded by the *double* of the overall risk bound. This factor is essential in the feasibility proof (10.19).

Then the second iteration replans the path with the decomposed collision avoidance constraints $\mathcal{C}_C^l$ that require the two vehicles to stay on the different sides of the "wall" with sufficient safety margins (Line 18). We call the new solution the adjusted solution, which is represented by the solid curves in Figure 10-6. The second iteration moves the path from the candidate solution so that the two vehicles stay away from each other with a necessary

margin in between. The adjusted solution is guaranteed to satisfy (10.4). Hence, each agent executes the control sequence of the adjusted solution (Line 12).

The following equations summarize all the constraints imposed on the kth agent:

$$\bigwedge_{1\leq l\leq k-1}\bigwedge_{t\in\mathbb{T}^n} h_t^{k,l\star T}\left(\bar{\boldsymbol{x}}_t^k - \frac{\tilde{\boldsymbol{x}}_t^k + \tilde{\boldsymbol{x}}_t^l}{2}\right) \geq \frac{g_i}{2} + \frac{1}{2}m_t^{k,l\star}(\gamma_{t,l}^k) \tag{10.15}$$

$$\bigwedge_{k+1\leq l\leq M}\bigwedge_{t\in\mathbb{T}^n} h_t^{k,l\star T}\left(\frac{\tilde{\boldsymbol{x}}_t^k + \tilde{\boldsymbol{x}}_t^l}{2} - \bar{\boldsymbol{x}}_t^k\right) \geq \frac{g_i}{2} + \frac{1}{2}m_t^{k,l\star}(\gamma_{t,l}^k) \tag{10.16}$$

$$\sum_{l\in\mathcal{L}^{-k}}\sum_{t\in\mathbb{T}^n}\gamma_{t,l}^k \leq \Gamma_C^{n,k} \tag{10.17}$$

Algorithm 16 shows the implementation of DecomposeCollisionAvoidanceConstraint function in Line 15 of Algorithm 14. The coupled collision avoidance episode is decomposed by using (10.15) and (10.16) (Line 3). The risk bound for each vehicle is allocated by Line 4, using the Multi-agent Heuristic Risk Allocation presented in the previous section. The factor of 2 in the left-hand side corresponds to (10.14). A decomposed chance constraint is constructed for each agent in Line 5, and returned in Line 8.

Algorithm 16 Implementation of DecomposeCollisionAvoidanceConstraint function in Line 15 of Algorithm 14

function DecomposeCollisionAvoidanceConstraint($\mathcal{C}_C$)

1: **for** $c \in \mathcal{C}_D$ **do**
2: **for** $l = 1 \cdots M$ **do**
3: $\Phi^l \leftarrow (10.15) \wedge (10.16)$;
4: $\Gamma_C^{n,l} \leftarrow \min\left(\frac{2d_{horizon}^l}{\sum_{l=1}^M d_{Goal}^l}, \beta\right)\Delta_c$;
5: $\mathcal{C}_C^l \leftarrow \mathcal{C}_D^l \cup \langle \Phi^l, \Gamma_C^{n,l}\rangle$;
6: **end for**
7: **end for**
8: **return** $\mathcal{C}_D^{1:M}$;

10.4.4 Proof of BRCA Feasibility

The following theorem holds:

Theorem 5. *If a solution satisfies (10.14), (10.15), (10.16), (10.17) for all agents ($k = 1 \cdots M$), the solution satisfies the collision avoidance constraint (10.4).*

Proof: Since (10.7) $\wedge$ (10.8) is a sufficient condition of (10.4), it suffices to prove that

$$\forall_k (10.14) \wedge (10.15) \wedge (10.16) \wedge (10.17) \Longrightarrow (10.7) \wedge (10.8).$$

Note that satisfying (10.15) and (10.16) for all k is equivalent to satisfying (10.11) and (10.12) for all combinations of (k, l) such that $k < l$. Using (10.10), we have

$$\begin{aligned}
(10.7) &\Longleftrightarrow \bigwedge_{1\le k\le M} \bigwedge_{1\le l<k} \bigwedge_{t\in\mathbb{T}^n} \bigvee_i \Pr\left[h_i^T(\boldsymbol{x}_t^k - \boldsymbol{x}_t^l) \ge g_i\right] \ge 1 - \delta_t^{k,l} \\
&\Longleftrightarrow \bigwedge_{1\le k\le M} \bigwedge_{1\le l<k} \bigwedge_{t\in\mathbb{T}^n} \max_i \Pr\left[h_i^T(\boldsymbol{x}_t^k - \boldsymbol{x}_t^l) \ge g_i\right] \ge 1 - \delta_t^{k,l} \\
&\Longleftrightarrow \bigwedge_{1\le k\le M} \bigwedge_{1\le l<k} \bigwedge_{t\in\mathbb{T}^n} \Pr\left[h_t^{k,l\star T}(\boldsymbol{x}_t^k - \boldsymbol{x}_t^l) \ge g_i\right] \ge 1 - \delta_t^{k,l} \\
&\Longleftrightarrow \bigwedge_{1\le k\le M} \bigwedge_{1\le l<k} \bigwedge_{t\in\mathbb{T}^n} h_t^{k,l\star T}(\bar{\boldsymbol{x}}_t^k - \bar{\boldsymbol{x}}_t^l) - g_i \ge m_t^{k,l\star}(\delta_t^{k,l}) \qquad (10.18)
\end{aligned}$$

The bound on the left-hand side of (10.18) is found by using (10.11) and (10.12) as follows:

$$\begin{aligned}
h_t^{k,l\star T}(\bar{\boldsymbol{x}}_t^k - \bar{\boldsymbol{x}}_t^l) - g_i &= h_i^T\left(\bar{\boldsymbol{x}}_t^k - \frac{\tilde{\boldsymbol{x}}_t^k + \tilde{\boldsymbol{x}}_t^l}{2}\right) - \frac{g_i}{2} + h_i^T\left(\frac{\tilde{\boldsymbol{x}}_t^k + \tilde{\boldsymbol{x}}_t^l}{2} - \bar{\boldsymbol{x}}_t^l\right) - \frac{g_i}{2} \\
&\ge \frac{1}{2}\left(m_t^{k,l\star}(\gamma_{t,l}^k) + m_t^{k,l\star}(\gamma_{t,k}^l)\right) \\
&\ge m_t^{k,l\star}\left(\frac{\gamma_{t,l}^k + \gamma_{t,k}^l}{2}\right)
\end{aligned}$$

The last inequality is derived from the convexity of $m_t^{k,l\star}$. Therefore, (10.18) is satisfied by setting

$$\delta_t^{k,l} = \frac{\gamma_{t,l}^k + \gamma_{t,k}^l}{2},$$

and hence (10.7) is satisfied. Such $\delta_t^{k,l}$ also satisfies (10.8) because

$$\begin{aligned}
\sum_{1\le k\le M}\sum_{1\le l<k}\sum_{t\in\mathbb{T}^n}\delta_t^{k,l} &= \sum_{1\le k\le M}\sum_{1\le l<k}\sum_{t\in\mathbb{T}^n}\frac{\gamma_{t,l}^k+\gamma_{t,k}^l}{2}\\
&= \sum_{1\le k\le M}\sum_{l\in\mathcal{L}^{-k}}\sum_{t\in\mathbb{T}^n}\frac{\gamma_{t,l}^k}{2}\\
&\le \sum_{1\le k\le M}\frac{\Gamma_C^{n,k}}{2}\\
&\le \Delta_C^n
\end{aligned} \tag{10.19}$$

See Figure 10-7 to understand the conversion from the first line to the second line. We use (10.17) to bound the second line by the third line, and (10.14) to bound the third line by the fourth line. ■

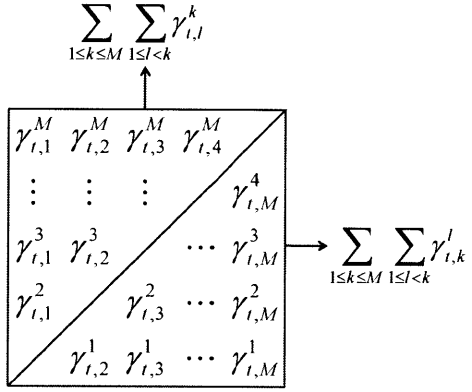


Figure 10-7: Conversion from the first line of (10.19) to the second line

Intuitively, in Figure 10-6, the proof shows that the combined margin of the decomposed collision avoidance constraint

$$\frac{g_i}{2}+\frac{m_{t,i}^{k,l}(\gamma_{t,l}^k)}{2}+\frac{g_i}{2}+\frac{m_{t,i}^{k,l}(\gamma_{t,k}^l)}{2}$$

is thicker than the margin in the original collision avoidance constraint $g_i + m_{t,i}^{k,l}(\delta_t^{k,l})$ by setting

$$\delta_t^{k,l}=\frac{\gamma_{t,l}^k+\gamma_{t,k}^l}{2}.$$

10.5 Robust Feasible Temporal Constraint Decomposition

Recall that the distributed CCQSP execution requires decoupling chance, state, and temporal constraints. We addressed the problem of decoupling chance and state constraints in the previous two sections. In this section we present the Robust Feasible Temporal Constraint Decomposition approach, which decouples temporal constraints.

Recall that simple temporal constraints are relative constraints that impose upper and lower bounds on the *duration* between the execution times of two events. Our approach is to obtain a set of absolute temporal constraints on the execution time of every atomic event, which is a sufficient condition of the original simple temporal constraints.

For example, consider a CCQSP shown in Figure 10-8-(a). The temporal plan involves two aerial vehicles that are going to land at the same airport. Both vehicles take at least 10 time units to get to the airport, and must land within 20 time units due to fuel constraint. The control tower allows Vehicle 1 to land first. In order for Vehicle 2 to land safely, two landings must be separated by an interval of at least four time units. This requirement causes a coupling between the temporal constraints on two vehicles.

The decoupled absolute temporal constraints are shown in Figure 10-8-(b). Constraints are directly imposed on the execution time of each event, instead of on durations. Observe that any execution times of e_1 and e_2 that satisfy the absolute constraints in Figure 10-8-(b) always satisfy all the temporal constraints in Figure 10-8-(a). In other words, Vehicle 1 does not have to consider the temporal constraints on Vehicle 2, as long as it makes sure that it land at the airport within its own temporal constraint, shown in Figure 10-8-(c). Likewise, Vehicle 2 can land anytime within the temporal bounds shown in Figure 10-8-(d), regardless of the landing time of Vehicle 1.

The absolute temporal constraints can be relaxed on the fly. For example, consider that Vehicle 1 landed on the airport at $t = 12$, as in Figure 10-8-(e). Then, the central module fixes the execution time of e_1 to 12 and relaxes the temporal constraints on e_E to $[16, 20]$, as shown in Figure 10-8-(f). As long as Vehicle 2 lands at the airport within the updated time bound, all temporal constraints in Figure 10-8-(a) are satisfied.

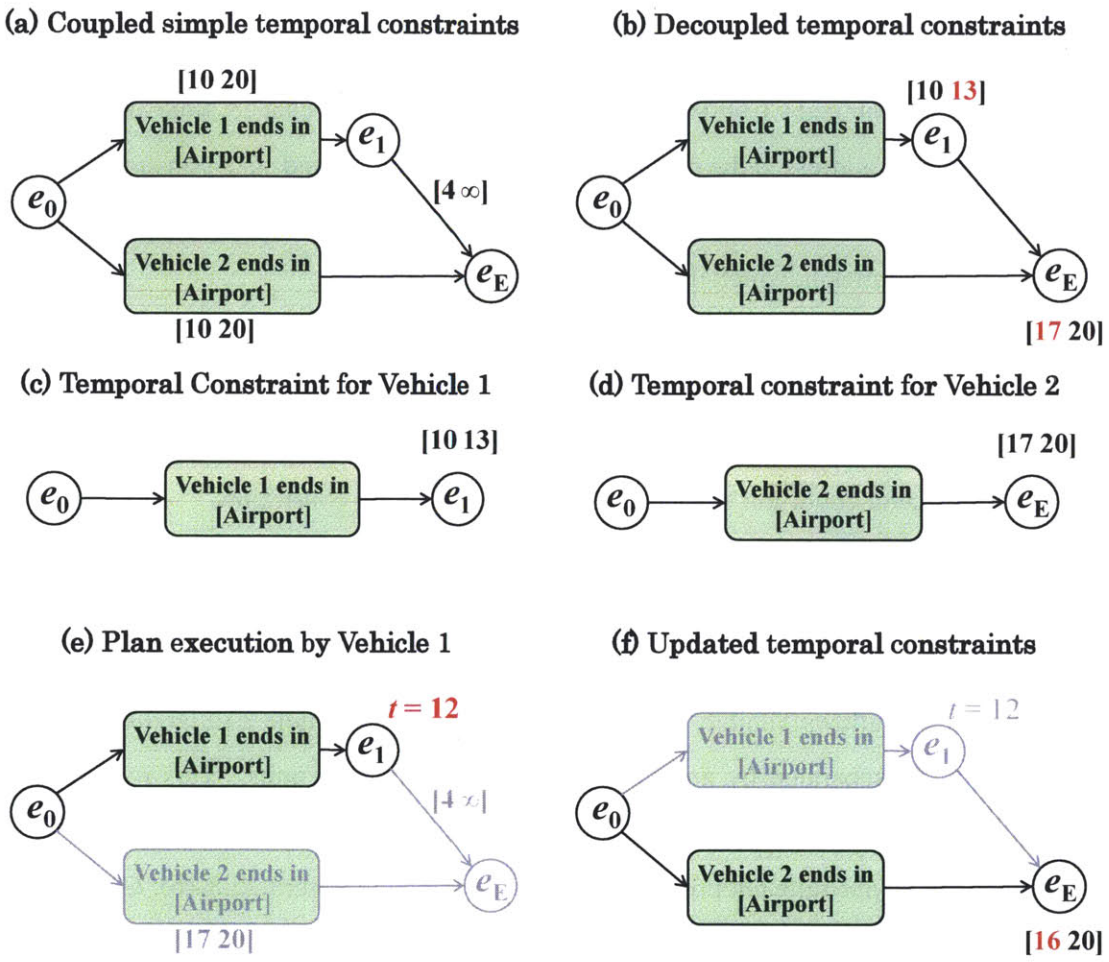


Figure 10-8: Decomposition of simple temporal constraints

10.5.1 Obtaining Distributed Temporal Constraints

In Line 5 of Algorithm 14, the absolute temporal constraints are obtained by running the RobustFeasibleTemporalConstraintDecomposition function. The implementation of the function is described in Algorithm 17 below.

The function takes the simple temporal constraints $\mathcal{T}$ and a partially assigned schedule σ as inputs. $\underline{s}(e)$ and $\bar{s}(e)$ are the lower and upper bounds on $s(e)$, the execution time of an episode e. An absolute temporal constraint on e is expressed as follows:

$$\underline{s}(e) \leq s(e) \leq \bar{s}(e).$$

Algorithm 17 Implementation of RobustFeasibleTemporalConstraintDecomposition function in Line 15 of Algorithm 14

function RobustFeasibleTemporalConstraintDecomposition($\mathcal{T}, \sigma$)

1: Solve Problem 21 with σ;
2: **for** $e \in \mathcal{E}$ **do**
3: $\quad \tau \leftarrow \langle e_0, e, \underline{s}(e), \bar{s}(e) \rangle$;
4: $\quad l \leftarrow$ the index of vehicle that is associated with e;
5: $\quad \mathcal{T}^l \leftarrow \mathcal{T}^l \cup \tau$;
6: **end for**
7: **return** $\mathcal{T}^{1:M}$;

The absolute constraint is equivalently expressed as a simple temporal constraint between e_0, the start event of the CCQSP, and e, as shown in Line 3 (see also Definition 14 in Section 2.4.3). In Line 1, the absolute temporal constraints are obtained by solving the following linear programming.

Problem 21: Obtaining Distributed Temporal Constraints

$$\max_{\underline{s}, \bar{s}, t \geq 0} \quad t \tag{10.20}$$

$$\text{s.t.} \quad \bigwedge_{e \in \mathcal{E} \setminus \mathcal{E}_\sigma} t \leq \bar{s}(e) - \underline{s}(e) \tag{10.21}$$

$$\bigwedge_{e \in \mathcal{E}_\sigma} \bar{s}(e) = \underline{s}(e) = \sigma(e) \tag{10.22}$$

$$\bigwedge_{\tau \in \mathcal{T}} \left(\underline{s}(e_\tau^E) - \bar{s}(e_\tau^S) \geq b_\tau^{\min} \wedge \bar{s}(e_\tau^E) - \underline{s}(e_\tau^S) \leq b_\tau^{\max} \right) \tag{10.23}$$

Recall that $\mathcal{E}_\sigma$ is the set of events that are assigned execution time steps by the partial schedule σ (see Definition 3 in Section 2.3.1). $\mathcal{E} \setminus \mathcal{E}_\sigma$ is the complement $\mathcal{E}$, which is formally defined as follows:

$$\mathcal{E}_\sigma := \{ e \in \mathcal{E} \mid e \notin \mathcal{E}_\sigma \}.$$

Note that sets of absolute constraints that are sufficient conditions of the given simple temporal constraints are not unique. For example, besides the absolute constraints shown in Figure 10-8-(b), [10 11] for e_1 and [15 20] for e_E, or [10 15] for e_1 and [19 20] for e_E,

are also valid. However, we prefer not to have particularly short feasible durations, since dp-Sulu will be hard to find a feasible assignment for very tight constraints. The objective function of Problem 21 is designed to find a "balanced" solution, such as the one in Figure 10-8-(b). The slack variable $t \geq 0$ represents the shortest duration of feasible execution time among all events. Hence, the objective function of Problem 21 maximizes the feasible duration of the tightest constraint.

Lemma 13. *If the set of simple temporal constraints is feasible, Problem 21 has a feasible solution.*

Proof: Since the set of simple temporal constraints $\mathcal{T}$ has a solution, there is a schedule s that satisfies all temporal constraints in $\mathcal{T}$. Then,

$$\forall_{e \in \mathcal{E}} \quad \bar{s}(e) = \underline{s}(e) = s(e)$$

is a feasible solution for Problem 21. ■

The decomposed temporal constraints are grouped by the indices of agents (Line 4) and returned (Line 5).

10.5.2 Updating Distributed Temporal Constraints

Note that Robust Feasible Temporal Constraint Decomposition takes a partially assigned schedule σ as an input. This is to specify the execution time of the events that have already been executed.

For example, in Figure 10-8-(a), the only executed event is e_0, the start event. Hence, at the beginning of the dp-Sulu algorithm, σ has an assignment only to e_0:

$$\sigma(e_0) = 0.$$

In Figure 10-8-(e), the event e_1 has also been assigned an execution time. Hence, when updating the absolute constraints, the input partial schedule has the following two assignments:

$$\sigma(e_0) = 0, \sigma(e_1) = 12.$$

Solution of Problem 21 with this σ results in the updated bounds shown in Figure 10-8-(f).

In Line 2 of Algorithm 14, the partial schedule σ is initialized with a sole assignment to $\sigma(e_0)$. In every planning cycle, each agent returns a schedule of the executed events (Lines 12 and 19 in Algorithm 14). In the next planning cycle, the decomposed temporal constraints are updated by taking in the latest schedule σ (Line 5 in Algorithm 14).

10.5.3 Relation to Strongly Controllable Simple Temporal Network with Uncertainty

The proposed Robust Feasible Temporal Constraint Decomposition has an interesting connection to the controllability theory on a Simple Temporal Network with Uncertainty (STNU) [100]. An STNU is an extension of a simple temporal network that contains *contingent events*, whose execution time is bounded but uncertain. Regular events, whose execution time can be determined by an executive, are called executable events. An STNU is *strongly controllable* if there exists a set of execution times for all executable events that is consistent with *any* realization of the execution times of the contingent events [99].

The decoupled temporal constraints generated by Robust Feasible Temporal Constraint Decomposition can be viewed as a strongly controllable STNU, by considering the events of other agents as contingent events. For example, the decoupled temporal constraints shown in Figure 10-8-(b) can be viewed as an STNU shown in Figure 10-9-(a) from a standpoint of Vehicle 1 by considering the event of Vehicle 2 (e_E) as a contingent event, drawn as a square. Observe that the STNU in Figure 10-9-(a) is strongly controllable because an assignment of an execution time in [10 13] to e_1 is consistent with any realization of the execution time of the contingent event, e_E, within the bound [17 20]. Likewise, the same decoupled temporal constraints can also be viewed as another STNU shown in Figure 10-9-(b) from a standpoint of Vehicle 2 by considering the event of Vehicle 1 (e_1) as a contingent event. Again, the STNU in Figure 10-9-(b) is also strongly controllable because an assignment of an execution time in [17 20] to e_E is consistent with any realization of the execution time of the contingent event, e_1, within the bound [10 13]. Note that, in Figure 10-9, contingent events are represented by squares while executable events are represented

by circles.

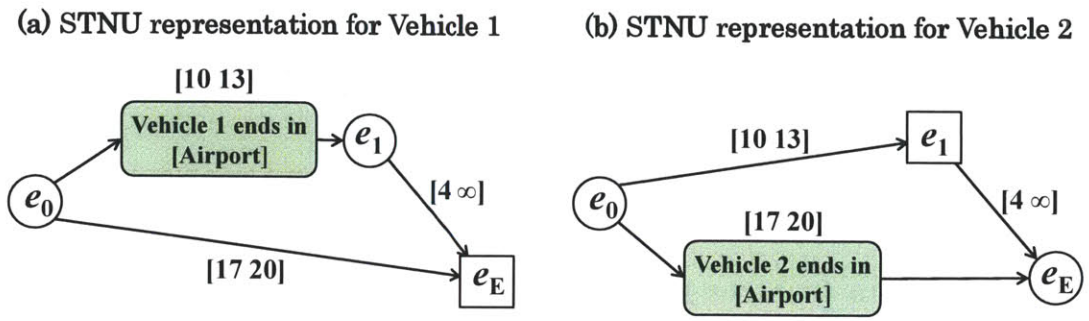


Figure 10-9: STNU representations of the decoupled temporal constraints shown in Figure 10-8-(b), which are generated by Robust Feasible Temporal Constraint Decomposition. The STNU representation for each agent is obtained by considering the events of other agents as contingent events. Note that both STNUs are strongly controllable.

The Robust Feasible Temporal Constraint Decomposition approach is also closely related to the work by Stedl [91] on decoupling distributed systems. He uses strong decomposability to generate a set of absolute temporal constraints from an STNU. There are two differences between our approach and Stedl's. Firstly, Stedl's work is concerned with decoupling of STNU, while our approach focuses on decoupling of an STN, although the resulting decomposed temporal constraints can be viewed as STNUs, as discussed above. Secondly, Stedl builds the decomposition approach upon the strong controllability algorithm introduced by [98], while we obtain the decoupled temporal constraints directly by solving an LP.

10.6 Conclusion

In this section we presented a distributed CCQSP executive, dp-Sulu. dp-Sulu was enabled by the three innovations: Multi-agent Heuristic Risk Allocation, Bi-stage Robust Collision Avoidance, and Robust Feasible Temporal Constraint Decomposition. The three methods decompose coupling chance, state, and temporal constraints. The central module of dp-Sulu provides decomposed CCQSPs to each agent, which solves its own planning problem using p-Sulu. The empirical results of dp-Sulu are presented in Section 11.2.4.

This chapter completes the theoretical part of this thesis. The next chapter presents empirical results.

Chapter 11

Simulation Results

This section presents empirical results. We deploy the proposed planners and executives on aerial, underwater, and space vehicles, as well as various benchmark problems. This chapter consists of three sections. Section 11.1 evaluates the performances of the proposed planners and executives by extensive simulations with randomized conditions on benchmark problems. The focus of this section is to highlight the new capabilities of each algorithm by comparing with prior art methods. Next, in Section 11.2, we deploy the proposed planners and executives on the personal transportation system (PTS), which is introduced in Section 1.1. The focus of this section is to demonstrate that our plan executives can solve real-world scale problems. Finally, in Section 11.3, we apply our planners and executives to the control of underwater and space vehicles, both of which have significantly different plant models than the PTS. The focus of this section is to show that our approach is applicable to a wide range of real-world systems.

11.1 Comparison with Prior Art

In this section we evaluate the performances of the proposed planners and executives (IRA, CRA, NIRA, NIRA+BoostLP, p-Sulu FH, p-Sulu, and MIRA) by comparing with prior art methods. This section is structured as follows. In Section 11.1.1, we introduce the prior art methods against which our planners and executives are compared. Section 11.1.2 presents the performance criterion used to compare the performances of algorithms. Section 11.1.3

presents the general results that are shared among all planners and executives. Section 11.1.4 presents the plant models and other simulation settings. Then, Sections 11.1.5 to 11.1.11 present the simulation results specific to each algorithm.

11.1.1 Prior Arts

Table 11.1 lists the prior arts against which the performance of proposed planners/executives are compared.

Table 11.1: Mapping of proposed planners/executives and corresponding prior arts.

Algorithms	Prior art methods	Chapter/Section
IRA	Subgradient, Elliptical Relaxation, Particle Control	Sec. 4.2
CRA	Elliptical Relaxation, Particle Control	Ch. 5
NIRA	Fixed Risk Allocation, Particle Control	Ch. 6
NIRA+BoostLP	NIRA	Sec. 6.5
p-Sulu FH	Sulu	Ch. 7
p-Sulu	Sulu	Ch. 8
MIRA	CRA	Ch. 9

Recall that NIRA+BoostLP results in exactly the same solution as NIRA, but with less computation time. Hence, NIRA+BoostLP is compared against NIRA. Also recall that MIRA obtains exactly the same solution as CRA, but in a decentralized manner. Therefore, computation time of MIRA is compared against CRA with various numbers of agents.

Below, we briefly introduce the prior arts listed in Table 11.1.

Subgradient Methods

The subgradient method is a standard method used to solve general convex optimization problems [14]. When constraints are convex, it can be used to optimize risk allocation. Hence, we compare IRA with subgradient methods on convex problems. There are two types of subgradient methods that we consider:

- Subgradient method with diminishing step size: step size is reduced over iterations;
- Subgradient method with constant step size: a costant step size is used throughout iterations;

The one with constant step size achieves faster convergence, but only the diminishing step size can guarantee the convergence to the optimum [14, 89]. Since subgradient methods do not monotonically reduce the cost, the stop condition is hard to define; therefore we computed a fixed number (300) of iterations.

Elliptical Relaxation

The elliptical relaxation approach, developed by [97], solves joint chance-constrained optimal control problems. The relaxation approach turns a stochastic problem into a deterministic problem using a very conservative ellipsoidal relaxation. Although this algorithm is computationally efficient, its result is highly suboptimal, since the ellipsoidal relaxation produces a very conservative bound.

Particle Control

Particle control [20], or a scenario approach [25], is a sampling-based method, which uses a finite number of samples to approximate the joint chance constraints. The control sequence is optimized so that the number of samples that violate constraints is less than $N_p \cdot \Delta$, where Δ is the risk bound and N_p is the total number of samples. The optimization problem is reformulated into a mixed-integer linear program (MILP), with an assumption that the cost function is linear.

Fixed Risk Allocation Approach

The fixed risk allocation approach [18] can solve a non-convex chance-constrained optimal control problem. The approach is to *fix* the risk allocation to a uniform value. As a result, with an assumption that the cost function is linear, the non-convex chance-constrained optimal control problem is reformulated into a MILP. The fixed risk allocation approach solves the MILP problem using a MILP solver, such as CPLEX.

Sulu

Sulu [62] is a deterministic counterpart of p-Sulu. It takes as input a deterministic plant model and a qualitative state plan (QSP), which specifies a desired evolution of the plant state as well as flexible temporal constraints, and outputs a continuous control sequence. Its approach is to encode the QSP execution problem into a mixed integer linear program (MILP) and solve it using the CPLEX optimizer.

11.1.2 Performance Criteria

The performances of the algorithms are evaluated by the following three criteria.

Probability of failure

The *probability of failure*, denoted by P_{fail}, means the probability that the resulting solution violates at least one of the given state constraints. We use Monte Carlo simulation to evaluate the probability of failure. Note that the joint chance constraint only requires that the probability of failure is below the risk bound Δ. Due to the conservativeness of the approximation of the joint chance constraint, the probability of failure is typically smaller than Δ. Hence, the difference between the probability of failure and the risk bound, $\Delta - P_{fail}$, is used as the measure of the degree of conservativeness of each algorithm. If $\Delta > P_{fail}$, it indicates that the solution violates the chance constraint.

Cost function value

All of our benchmark problems are framed as minimization problems. Hence, less cost means better solution. Note that a strictly optimal solution to a joint chance-constrained optimization problem is unavailable in general, since there is no algorithm to solve such a problem exactly. Therefore, the optimal cost is not known. Hence, although the cost function value can be used to compare two algorithms, it cannot be used to evaluate the conservativeness of each algorithm. Instead, we evaluate the conservativeness of each algorithm by its probability of failure, as discussed above.

Computation time

It is crucial for planning and execution algorithms to obtain solutions within a reasonable computation time, particularly when applied to real-world problems. In all benchmark results, we present the solution time of each algorithm.

11.1.3 Result Summary

Since the proposed planners and executives solve different problems in different ways, each has different advantages and disadvantages. Nevertheless, since all planners and executives are built upon the risk allocation approach, there are common tendencies in their simulation results. The objective of this subsection is to extract from the simulation results in Sections 11.1.5 - 11.1.11 insights about the common properties shared by all the proposed algorithms. Results specific to each algorithm are presented in Sections 11.1.5 - 11.1.11.

Our empirical results demonstrate that all planners and executives presented in this thesis share the following two properties:

Soundness

The risk allocation-based algorithms are sound with regard to the chance constraints. Theoretically, the solutions of the algorithms built upon the risk allocation approach are guaranteed to satisfy the chance constraints. This is because, as presented in Section 4.1, the risk allocation approach provides *sufficient* conditions of the given chance constraints.

This theoretical claim is empirically validated in all of our simulation results. For each algorithm, we run randomized simulations hundreds of times with various simulation settings. All of them result in solutions with $P_{fail} \leq \Delta$, i.e., satisfaction of the chance constraints. On the other hand, Particle Control and Sulu tend to result in violations of chance constraints. This is because the former employs a sampling-based approximation, which is not a sufficient condition of the chance constraints if the number of samples are finite, and the latter does not explicitly reason about the effects of uncertainty during planning.

Near-optimality

The risk allocation approach provides a tighter bound of the chance constraints than prior art methods, such as the elliptical relaxation and the fixed risk allocation approach. This is because the risk allocation approach allows flexibility of optimally distributing risk among agents, time steps, and constraints. Therefore, it is predicted theoretically that the risk allocation-based algorithms result in a solution that is closer to the strictly optimal solution compared to the prior art methods.

Our empirical results validate this theoretical prediction in two ways. Firstly, the proposed algorithms always result in lower cost function value than the prior art methods. Secondly, recall that the difference between the risk bound and the resulting probability of failure, $\Delta - P_{fail}$, indicates the level of conservativeness; the proposed algorithms always result in significantly lower values of $\Delta - P_{fail}$ than prior arts. These results support our claim that, although the risk allocation is a conservative bound, its conservativeness is less than prior arts.

We note that Particle Control and Sulu can result in lower cost function values than the proposed algorithms. This is because the solutions of Particle Control and Sulu can violate the chance constraints. On the other hand, as we mentioned above, the proposed planners and executives respect the chance constraints. Therefore, the fact that our algorithms result in more cost than Particle Control or Sulu does not mean the inferiority of our algorithms.

11.1.4 Problem Settings

Plant Model

This subsection explains the plant model used throughout this section. The same plant model is also used for the PTS demonstration in Section 11.2. Section 11.3 uses different plant models for underwater and space vehicles.

We consider a point-mass double-integrator plants, shown in (11.1)-(11.2). Parameters, such as u_{max}, v_{max}, σ, and ΔT are set individually for each problem. This plant model is commonly assumed in the literatures on unmanned aerial vehicle (UAV) path planning (e.g., [57, 58, 61, 102]). The state vector $\boldsymbol{x}_t$ of an n-dimensional point-mass

double-integrator plants consists of $2n$ variables; the first n variables represent position, while the other n variables represent velocity. The control vector $\boldsymbol{u}_t$ consists of n variables, which represent the acceleration. Note that the acceleration is proportional to the force applied to the point mass. We consider an additive uncertainty in the position.

The plant model is given as follows:

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{I}_n & \Delta T \cdot \boldsymbol{I}_n \\ \boldsymbol{0}_n & \boldsymbol{I}_n \end{pmatrix}, \quad \boldsymbol{B} = \begin{pmatrix} \Delta T^2/2 \cdot \boldsymbol{I}_n \\ \Delta T \cdot \boldsymbol{I}_n \end{pmatrix}, \quad \Sigma_w = \begin{pmatrix} \sigma^2 \cdot \boldsymbol{I}_n & \boldsymbol{0}_n \\ \boldsymbol{0}_n & \boldsymbol{0}_n \end{pmatrix} \tag{11.1}$$

$$\forall t \in \mathbb{T}, \quad \|\boldsymbol{u}_t\|_p \leq u_{\max}, \quad \|\boldsymbol{C}\boldsymbol{x}_t\|_p \leq v_{\max}, \tag{11.2}$$

where $\boldsymbol{I}_n$ and $\boldsymbol{0}_n$ are the n-dimensional identity matrix and zero matrix, respectively. The matrix $\boldsymbol{C}$ is defined as follows:

$$\boldsymbol{C} = \begin{pmatrix} \boldsymbol{0}_n & \boldsymbol{I}_n \end{pmatrix}.$$

In (11.2), we denote by $\|\cdot\|_p$ a p-norm. The first constraint in (11.2) represents the limit the acceleration, while the second constraint bounds the velocity. When we consider the 1-norm or ∞-norm, these constraints are linear. When we consider the 2-norm (Euclidean norm), they becomes nonlinear constraints. When solving the problems using LP or MILP solvers, such as CPLEX, we approximate the nonlinear constraints in (11.2) by the following set of linear constraints:

$$\forall t \in \mathbb{T}, \quad \boldsymbol{r}_n \cdot \boldsymbol{u}_t \leq u_{\max} \quad (n = 1, 2, \cdots, N_r)$$
$$\boldsymbol{r}_n = \left[\cos\frac{2\pi n}{N_r}, \sin\frac{2\pi n}{N_r}\right]$$

Objective Function

We use two types of cost functions, both of which are commonly used in path planning literatures (e.g. [61, 85, 102]). The first one is the 1-norm (Manhattan norm) of the control inputs over the planning horizon. For example, in a two-dimensional case, the objective

function is:

$$J(\bar{\boldsymbol{x}}_{t_i}, \boldsymbol{U}, s) = \sum_{t=1}^{T} \left(|u_{x,t}| + |u_{y,t}| \right).$$

Note that a minimization problem with the piece-wise linear cost function above can be equivalently replaced by the following minimization problem with a linear cost function and additional linear constraints:

$$\begin{aligned} \min \quad & \sum_{t=1}^{T} (\mu_{x,t} + \mu_{y,t}) \\ \text{s.t.} \quad & \forall t \in \mathbb{T}, \quad \mu_{x,t} \geq u_{x,t} \ \wedge \ \mu_{x,t} \geq -u_{x,t} \ \wedge \ \mu_{y,t} \geq u_{y,t} \ \wedge \ \mu_{y,t} \geq -u_{y,t}, \end{aligned}$$

where $\mu_{x,t}$ and $\mu_{y,t}$ are slack variables.

The second objective function is the quadratic objective function, such as the following:

$$J(\bar{\boldsymbol{x}}_{t_i}, \boldsymbol{U}, s) = \sum_{t=1}^{T} u_{x,t}^2 + u_{y,t}^2.$$

Since this is a convex function, a convex program solver can handle such an objective function.

Implementation

The IRA, CRA, NIRA, and MIRA algorithms are implemented using Matlab and Yalmip [68], while NIRA+BoostLP, p-Sulu FH, p-Sulu, and dp-Sulu are implemented in C/C++. We use CPLEX to solve LPs and MILPs. SNOPT is used to solve general nonlinear convex optimization problems. The computing environments used for the simulations are described at each section.

11.1.5 IRA

This subsection presents the simulation results of the IRA algorithm, introduced in Section 4.2.

Simulation Settings

In this simulation we use the one-dimensional point-mass double-integrator plant presented in 11.1.4, with $\Delta T = 0.5$, $u_{\max} = 0.0066$, and $\sigma^2 = 0.001$. The Manhattan norm objective function is employed, and the planning horizon length is set to $N = 2$. The initial condition is:

$$\bar{x}_0 = \begin{pmatrix} 0.01 \\ 0 \end{pmatrix}, \quad \Sigma_{x_0} = \begin{pmatrix} 0.001 & 0 \\ 0 & 0 \end{pmatrix}$$

We impose the following joint chance constraint:

$$\Pr\left[\bigwedge_{t\in\mathbb{T}} \bigwedge_{i\in\{1,2\}} h_{t,i}^T \boldsymbol{x}_t - g_{t,i} \leq 0\right] \geq 1 - \Delta_c,$$

where $h_{t,1} = (1,0), h_{t,2} = (-1,0)$, and the bounds $g_{t,i}$ are randomly generated. The constant α in Line 12 of Algorithm 1 (Section 4.2.2) is set to $0.7 \cdot 0.98^n$, where n is the iteration index. This means that we reduce α over iterations. The simulation is conducted on a machine with a Pentium 4 processor clocked at 2.80 GHz and 1.00 GB of RAM.

Performance Evaluation

We compare IRA with four prior art methods: the subgradient method with diminishing step size (abbreviated as SM(d) in Table 11.2 below), the subgradient method with constant step size (SM(c)), the elliptical relaxation approach (ER), and Particle Control with 20 samples (PC). Note that IRA, SM(d), and SM(c) optimize risk allocation, while the other two do not. The performance of the five algorithms is compared in Table 11.2. The values in the table are the average of 237 randomly generated problems. The three risk-allocation approaches (IRA, SM(d), and SM(c)) result in P_{fail} that is significantly closer to the risk bound $\Delta = 0.05$ compared to the ellipsoidal relaxation approach, indicating that these algorithms are much less conservative. The three algorithms also achieve a significant speed-up compared to Particle Control. Among the three, IRA outperforms the other two according to all three criteria.

Table 11.2: Performance of two-stage optimization methods with risk allocation and two prior arts; values are the average of 237 randomly generated problems. Names of the algorithms are abbreviated as follows - IRA: Iterative risk allocation, SM(d): subgradient method with diminishing step size, SM(c): subgradient method with constant step size, ER: elliptical relaxation, PC: particle control.

Algorithm	**IRA**	SM (d)	SM (c)	ER	PC
Risk bound Δ	0.05				
Probability of failure P_{fail}	0.0378	0.0183	0.0306	$< 10^{-5}$	0.0281
Cost function value $\bar{J}^\star$	0.0906	0.0978	0.0957	0.3502	0.0959
Computation time [sec]	0.33	26.4	30.7	0.05	212.2

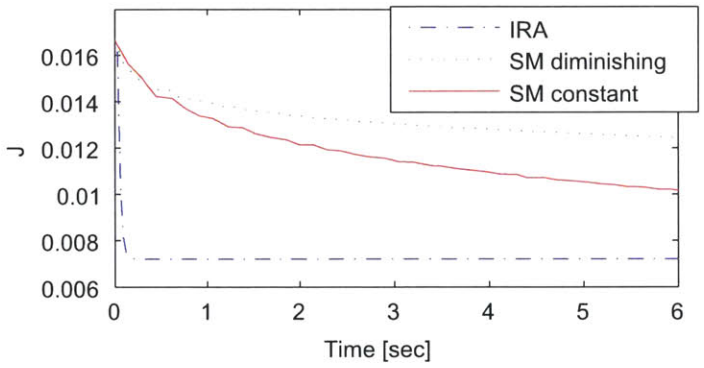


Figure 11-1: Convergence of IRA and the subgradient methods

Figure 11-1 compares the convergence speed of IRA and the subgradient methods on a typical problem. The convergence of IRA is significantly faster than the subgradient methods. The weakness of IRA is a lack of the theoretical guarantee of convergence to the optimal. However, the empirical result shows that the suboptimality is considerably small. Table 11.2 shows that IRA yields even better solutions than the subgradient methods after 300 iterations on average. Figure 11-2 is the histogram of the difference between the cost function values of IRA and the subgradient methods, $\bar{J}^\star_{\mathrm{IRA}} - \min(\bar{J}^\star_{\mathrm{SM(d)}}, \bar{J}^\star_{\mathrm{SM(c)}})$ (note that, among the two subgradient methods, the one with better cost is compared with IRA). The objective function value of IRA is smaller or equal to the objective function value of both subgradient methods in most cases; IRA yields worse solutions in several cases, but the difference is less than 0.01 in those cases; on the other hand, the subgradient methods may

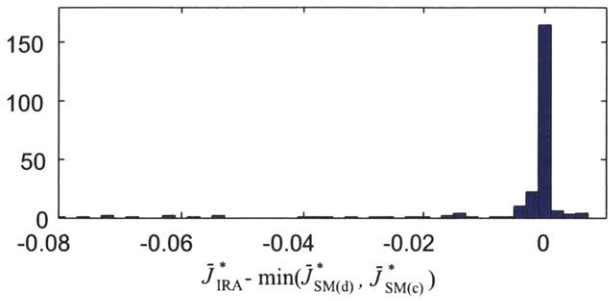


Figure 11-2: Histogram of the difference in the objective function value of IRA and both subgradient methods.

be worse than IRA by up to 0.08.

11.1.6 CRA

We next show simulation results demonstrating the CRA algorithm, developed in Chapter 5.

Simulation Settings

The system to be controlled has state $\boldsymbol{x}_t = [x_t \; y_t]'$ and the system parameters are defined by:

$$\boldsymbol{A} = \begin{pmatrix} \mathbf{1} & 1 \\ -0.5 & 0 \end{pmatrix}, \; \boldsymbol{B} = \begin{pmatrix} 0 \\ 0.0\dot{3} \end{pmatrix}, \; \Sigma_w = \begin{pmatrix} 0.001 & 0 \\ 0 & 0.001 \end{pmatrix} \tag{11.3}$$

The constraints on the state are:

$$-0.25 \leq y_t \leq 0.25 \;\; \forall t.$$

The cost is defined as:

$$f(\bar{\mathbb{X}}, \mathbb{U}) = \sum_{t=0}^{N} (\bar{\mathbf{x}}_t - \mathbf{x}^r)'(\bar{\mathbf{x}}_t - \mathbf{x}^r).$$

In other words, we try to minimize the squared distance of the expected state from some reference state $\mathbf{x}^r$, averaged over the planning horizon. Optimization was performed on a MacBook Pro with a 2.4GHz processor and 4GB RAM.

Example Solutions

Figure 11-3 shows the solution to the finite-horizon optimal control problem using the CRA algorithm. In this case, $\mathbf{x}^r = [1\ 0]'$, $N = 20$ and $\Delta = 0.01$. The CRA algorithm optimizes the allocation of risk at each time step, while ensuring that the probability of failure over the entire horizon is less than Δ. As shown in Figure 11-5, risk allocation values δ_i are tiny ($< 10^{-8}$) for all constraints except for 5 of the 42 constraints. The non-negligible values correspond to the bound $y_t \leq 0.25$ at time steps $t = 1, \ldots, 5$. This implies that optimizing risk allocation can lead to significant gains over an elliptical relaxation approach, which uses an *a priori* fixed backoff from the constraints.

We compare the performance of CRA with the elliptical relaxation approach and Particle Control. In this simulation, we use the implementation of [25] for Particle Control. Figure 11-4 shows a solution to the same problem using the elliptical relaxation approach. Notice that the state means are very far from the constraints compared to the solution in Figure 11-3, indicating a great deal of conservatism. This is because the elliptical relaxation approach assumes a 'worst-case' allocation of risk to each of the constraints over the time horizon, rather than optimizing the risk allocation. Table 11.3 compares the conservatism of the CRA algorithm with that of elliptical relaxation and Particle Control. In this case $\mathbf{x}^r = [1\ 0]'$, $N = 20$ and $\Delta = 0.1$. For elliptical relaxation and Particle Control, optimization was performed using SDPT3[94], which is able to more efficiently exploit the structure of the Second Order Cone Programs that result from the approach. For Particle Control we used $\beta = 0.001$, resulting in 2615 scenarios (samples) being used to ensure that $P(P(\mathbb{X} \notin F_X) > 0.1) \leq 0.001$. Table 11.3 shows that CRA is orders of magnitude less conservative that the elliptical relaxation approach, and as a result the cost of the optimal solution is reduced by 40%. Compared to Particle Control, CRA is both less conservative and far less computationally expensive. It should be noted, however, that Particle Control applies to arbitrary uncertainty distributions, whereas both CRA and the elliptical set

bounding approaches are restricted to Gaussian distributions.

Algorithm	**IRA**	ER	PC
Risk bound (Δ)	0.1		
Probability of failure P_{fail}	0.097	2×10^{-6}	0.0022
Cost function value	3.15	5.26	3.76
Computation Time (s)	3.38	1.76	1.41×10^4

Table 11.3: Comparison of the CRA algorithm with elliptical relaxation for a single example. CRA is orders of magnitude less conservative that the elliptical relaxation approach, and as a result the cost of the optimal solution is reduced by 40%. Also shown are results for Particle Control, which is more conservative than CRA and requires orders of magnitude greater optimization time. Names of the algorithms are abbreviated as follows - IRA: Iterative risk allocation, ER: elliptical relaxation, PC: particle control.

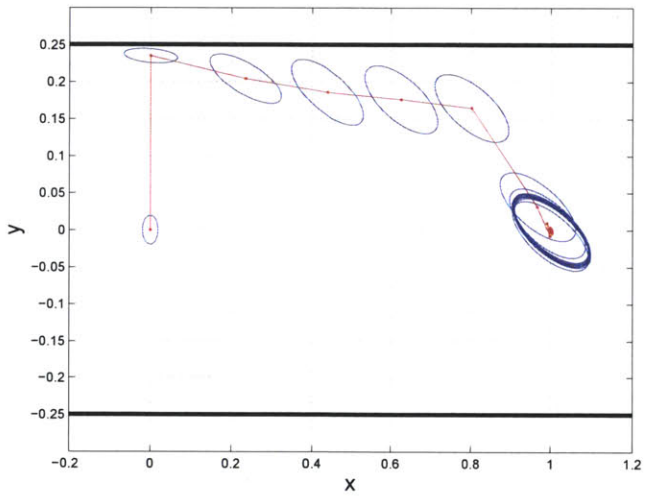


Figure 11-3: Single solution using new convex optimization approach for $\mathbf{x}^r = [1\ 0]'$, $N = 20$ and $\Delta = 0.01$. The red dots show the state mean $\bar{\mathbf{x}}_t$ for $t = 0, \ldots, N$. The blue ellipses show the 3-sigma ellipses for $\mathbf{x}_t$. The state constraints are shown as thick black lines. The new approach optimizes the allocation of risk at each time step, while ensuring that the probability of failure over the entire horizon is less than Δ.

Conservatism Against Δ and Horizon Length

Figure 11-6 shows how the conservatism of CRA depends on the allowable maximum probability of failure, Δ. For this example we used $N = 10$ steps and $\mathbf{x}^r = [1\ 0]'$, and

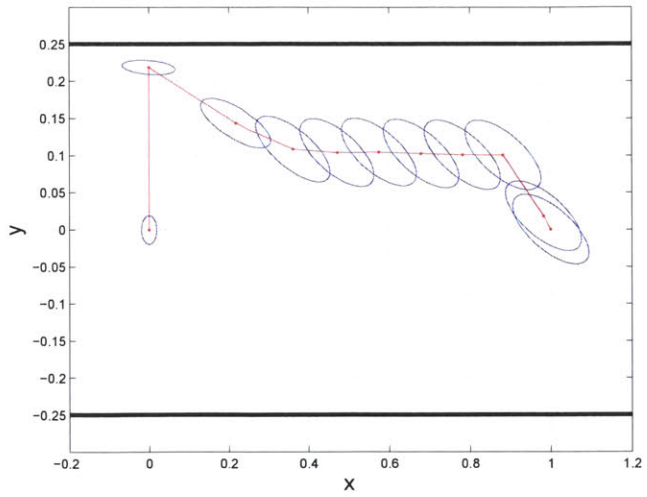


Figure 11-4: Single solution for $\mathbf{x}^r = [1\ 0]'$, $N = 20$ and $\Delta = 0.01$, using elliptical relaxation approach of [97]. The state means are very far from the constraints compared to the solution in Figure 11-3, indicating a great deal of conservatism.

10^7 Monte Carlo simulations were used to estimate the true probability of failure in the returned solution. Risk allocation yields probabilities of failure very close to the allowable value, indicating very little conservatism. Critically, the conservatism does not increase appreciably as Δ decreases. For comparison Figure 11-6 also shows the estimated probability of failure for elliptical set bounding. Note that this approach is highly conservative, and the conservatism increases as Δ decreases. This conservatism prevents elliptical set bounding from finding a feasible solution for $\Delta \leq 0.01$, even though feasible solutions exist for much smaller Δ.

Figure 11-7 shows how the conservatism of CRA depends on the horizon length, N. For this example we used $\Delta = 0.1$ and $\mathbf{x}^r = [1\ 0]'$, and 10^7 Monte Carlo simulations were used to estimate the true probability of failure in the returned solution. Risk allocation gives solutions with very little conservatism, and the conservatism increases only slightly as the horizon length, N, increases; for $N = 5$, the estimated probability of failure is 0.0974, while for $N = 20$ the estimated probability of failure is 0.0970. By contrast the set bounding approach is highly conservative, and the conservatism increases as the horizon length increases. This conservatism prevents set bounding from finding a feasible solution for $N > 10$.

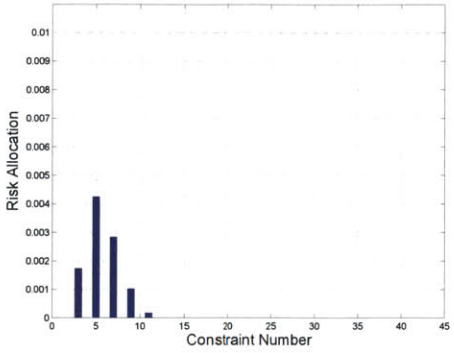


Figure 11-5: Risk allocation in optimal solution of Figure 11-3. The bars show the value δ_i, i.e. the risk allocation, for each constraint in the problem. The allocated risks add to the maximum probability of failure for the entire horizon, Δ, shown as the dashed line. Only a handful of the δ_i are non-negligible.

Average performance

In order to assess the average performance of the new algorithm, we generated random instances of the control problem by setting $\mathbf{x}^r = [n\ 0]'$ with n uniformly distributed in the range $[0, 1]$. Again we used $N = 20$ and $\Delta = 0.01$. The average results for 20 solutions are shown in Table 11.4. Since we are interested in the conservatism of CRA, we have removed instances where the chance constraints were not tight. In the globally optimal solution, we would expect the true probability of failure to be the same as Δ. The results show that the new convex optimization approach is many orders of magnitude less conservative than the elliptical relaxation approach for a small penalty in solution time.

Algorithm	Time (s)	$P(fail)$
Risk Allocation	1.03	0.0079
Elliptical Relaxation	0.22	$< 10^{-6}$

Table 11.4: Optimization time and estimated probability of failure averaged for 20 randomized problem instances with $\Delta = 0.01$. Instances where the chance constraint is not tight have been removed. The convex optimization approach is orders of magnitude less conservative than the elliptical relaxation approach for a small penalty in solution time.

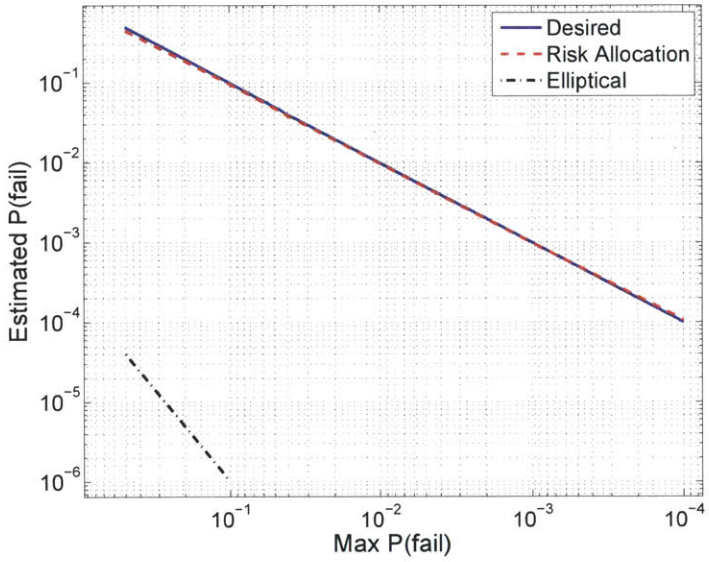


Figure 11-6: Comparison of maximum allowable probability of failure Δ and estimated probability of failure as a function of Δ. Risk allocation yields probabilities of failure very close to the allowable value, indicating very little conservatism. Furthermore this conservatism does not increase as Δ decreases. By contrast the set bounding approach is highly conservative, and the conservatism increases as Δ decreases. In fact set bounding fails to find a solution for $\Delta \leq 0.01$, even though feasible solutions exist for much smaller Δ.

11.1.7 NIRA

This subsection compares the performance of NIRA, introduced in Chapter 6, with prior arts.

Problem Settings

We consider a 2-D path planning problem with a randomized location of an obstacle, as shown in Figure 11-8. A vehicle starts from $[0, 0]$ and heads to the goal at $[1.0, 1.0]$, while avoiding a rectangular obstacle. The obstacle with edge length 0.6 is placed at a random location within the square region with its corners at $[0, 0]$, $[1, 0]$, $[1, 1]$, and $[0, 1]$. We consider ten time steps with the time interval $\Delta T = 1.0$. We require that the mean state at $t = 10$ is at $[1.0, 1.0]$. The risk bound is set to $\Delta = 0.01$. We set the standard deviation of the disturbance as $\sigma = 0.01$. The simulations are conducted on a machine with a quad-core Intel Core i7 CPU clocked at 2.67 GHz, and with 8 GB of RAM.

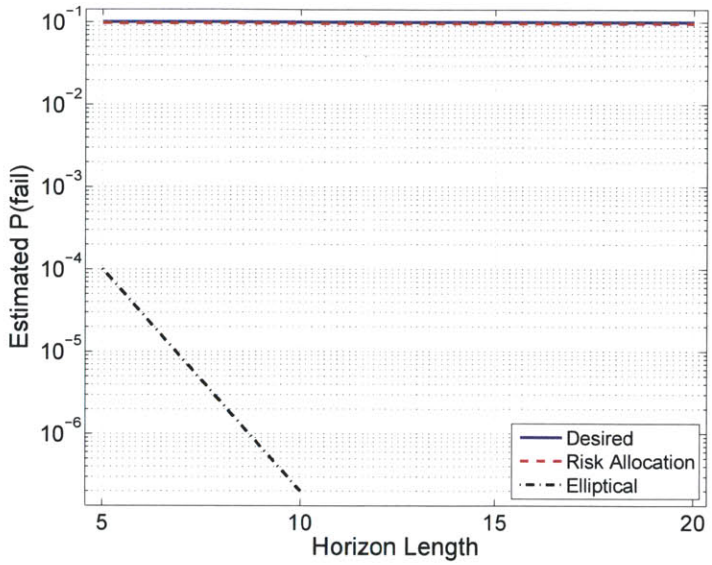


Figure 11-7: Comparison of maximum allowable probability of failure Δ and estimated probability of failure as a function of horizon length. Risk allocation gives solutions with very little conservatism, and the conservatism increases only slightly as the horizon length, N, increases. By contrast the set bounding approach is highly conservative, and the conservatism increases as the horizon length increases. Set bounding fails to find a solution for $N > 10$.

Performance Evaluation

We compare NIRA with the fixed risk allocation approach and Particle Control. Table 11.5 compares the performance of the three algorithms. The values in the table are the averages and the standard deviations of 100 runs with random locations for the obstacle. The probability of constraint violation, P_{fail}, is evaluated by Monte-Carlo simulations with 10^6 samples.

Comparison with the fixed risk allocation approach Recall that the difference between Δ and P_{fail} represents the conservativeness of the algorithm. Table 11.5 shows that NIRA results in the average probability of failure $P_{fail} = 0.00984$, which is smaller than the user-specified risk bound $\Delta = 0.01$ only by 1.6% difference. On the other hand, the fixed risk allocation approach results in $P_{fail} = 0.000228$, which is 98% smaller than Δ. This result indicates that the solution by NIRA is significantly closer to the exactly optimal solution. In fact, the NIRA algorithm results in less cost than the fixed risk allocation approach in all

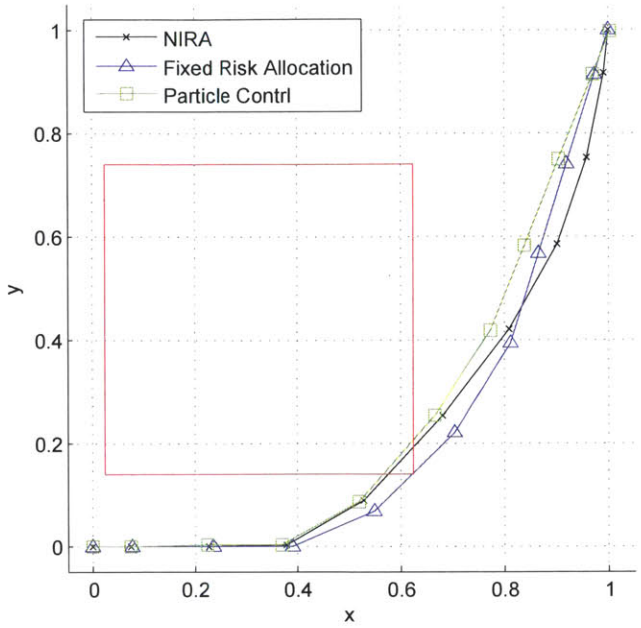


Figure 11-8: An instance of the 2-D path planning problem used in 11.1.7. The obstacle with a fixed size is randomly placed within the unit square for each run.

the 100 runs. This is because it optimizes the risk allocation while the fixed risk allocation approach uses the predetermined risk allocation.

Figure 11-9 plots Δ/P_{fail} with different settings of the risk bound Δ. For all values of Δ, the conservativeness of NIRA is significantly smaller than the fixed risk allocation approach. The graph shows a tendency that the conservativeness of NIRA gets smaller for less Δ, while the conservativeness of the fixed risk allocation approach is approximately constant.

NIRA achieves the improvement in solution optimality with a cost of computation time; Table 11.5 shows that NIRA takes significantly longer computation time than the risk allocation approach. Hence, NIRA and the fixed risk allocation approach provide users with a trade-off between suboptimality and computation time.

Comparison with the Particle Control Table 11.5 shows that the average probability of failure of the Particle Control approach is higher than the risk bound $\Delta = 0.01$, meaning that the approach tends to generate infeasible solutions. In fact, 96 runs out of 100 violate the chance constraint. On the other hand, NIRA guarantees the satisfaction of the chance

Table 11.5: The averages and the standard deviations of the the probability of constraint violation (P_{fail}), the cost function value, and the computation time of the three algorithms. We use 100 particles for the Particle Control. Each algorithms are ran 100 times with random location of an obstacle. The second row shows the resulting probability of constraint violation. The risk bound is set to $\Delta = 0.01$. Note that Particle Control results in less cost than the other two methods because its solutions violate the chance constraint. Names of the algorithms are abbreviated as follows - IRA: Iterative risk allocation, FR: fixed risk allocation, PC: particle control.

Algorithm	**NIRA**	FR	PC
Δ	0.01		
P_{fail}	9.84×10^{-3}	2.28×10^{-4}	3.48×10^{-2}
Cost	0.638	0.664	0.631
Comp. time (s)	43.9 sec	0.938 sec	129.2 sec

constraint since it employs a conservative approximation of the joint chance constraint.

As opposed to NIRA, Particle Control has a guarantee that its solution converges to an exactly optimal solution when increasing the number of samples to infinite. However, using a large number of samples is impractical since computation time and memory usage grow exponentially as the number of samples increases. For example, we used only 100 samples in the analysis in Table 11.5. When using 300 samples, it took 4,596 seconds (about 1.5 hours) to solve the same problem with the obstacle's center at $[0.5, 0.5]$. Computation with 1000 samples could not be conducted because of the shortage of memory. On the other hand, the computation time of NIRA is significantly shorter than PC while guaranteeing the feasibility of the solution.

11.1.8 NIRA+BoostLP

We next demonstrate NIRA+BoostLP, introduced in Section 6.5.[1] Recall that NIRA+BoostLP results in exactly the same solution as NIRA, but with less computation time. Hence, we focus on the comparison of computation time with NIRA.

[1]This research is conducted in collaboration with Dr. Ashis Gopal Banerjee and Prof. Nicholas Roy at Robust Robotics Group, CSAIL, MIT.

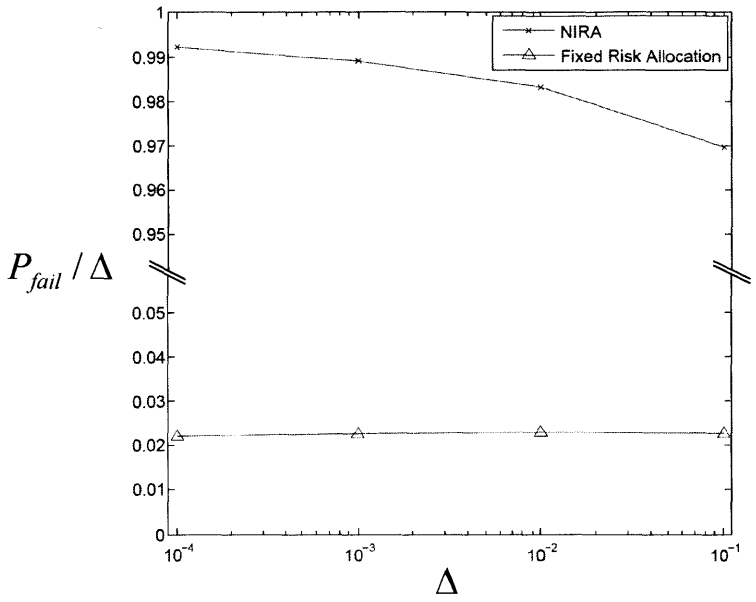


Figure 11-9: Suboptimality of NIRA and the fixed risk allocation approach. Strictly optimal solution has $\Delta/P_{fail} = 1$. A smaller value of Δ/P_{fail} indicates that the solution is suboptimal.

Simulation Settings

We tested our approach on 2D path planning problems under Gaussian uncertainty with a single chance constraint, involving obstacle avoidance and finding paths through waypoints at desired time instants. A vehicle starts from $[1, 1]$ and heads to the goal at $[12, 12]$, while avoiding a rectangular obstacle. The obstacle with edge length 0.6 is placed at a random location within the square region with its corners at $[0, 0]$, $[1, 0]$, $[1, 1]$, and $[0, 1]$. A discrete-time, point-mass plant model with $\Delta T = 0.5$ and $u_{max} = [5, 5]$ is used. The standard deviation of the disturbance is varied in $\sigma_x, \sigma y \in [0.1, 0.001]$. The Manhattan norm of the control inputs is used as the objective function. The risk bound is set to $\Delta = 0.01$. The parameters in BoostLP is set as follows: the total number of regression trees, M, is always chosen as 1000, the number of leaf nodes in any tree, Q, as 16, and the shrinkage factor, ν, as 0.1. The simulations are conducted on a machine with a quad-core Intel Core i7 CPU clocked at 2.67 GHz, and with 8 GB of RAM.

Table 11.6: Performance evaluation on different planning problem scenarios. The proposed algorithm uses regression for approximately solving FRRs whereas the previous algorithm solves FRRs exactly using CPLEX optimizer. All the reported data are for average values; standard deviation values are not presented as they are of the order of 0.1% of the average.

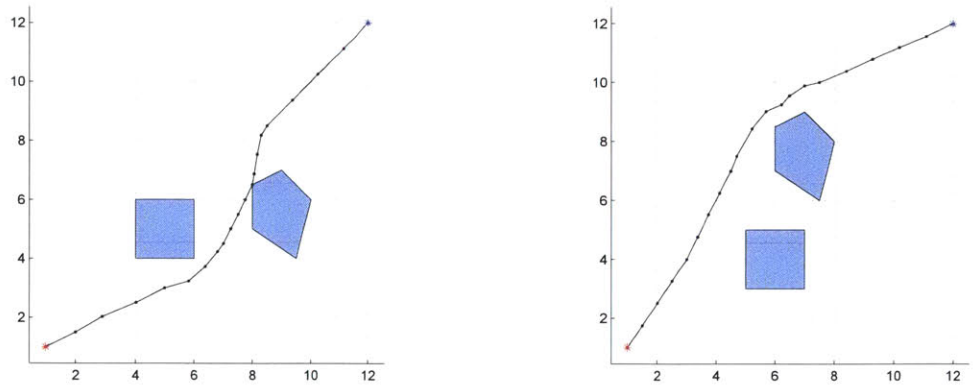
Scenario #	1		2		3		4	
Algorithm	**NIRA+ BoostLP**	NIRA	**NIRA+ BoostLP**	NIRA	**NIRA+ BoostLP**	NIRA	**NIRA+ BoostLP**	NIRA
Comp. time (s)	7.51	135.21	8.14	219.76	6.15	79.99	5.73	80.15
Speed-up	18X		27X		13X		14X	
Cost	4.23	4.23	5.86	5.86	3.45	3.45	3.60	3.60

Performance Evaluation

Table 11.6 enumerates the performance of our algorithm for four different planning scenarios. Scenario 1 deals with planning in an environment consisting of two obstacles, scenario 2 with avoiding two obstacles and passing through two waypoints at specified time instants, scenario 3 with different levels of disturbances (Gaussian distribution variance) in the vehicle location, and scenario 4 with varying maximum limits on the vehicle acceleration respectively. One obstacle and one waypoint are used in the last two scenarios. All the obstacles and waypoints are rectangular except in the case of scenario 1. N is always selected as 20 in all the scenarios; σ_x and σ_y are both chosen to be 0.01 in all the scenarios excepting 3 and u_{min} and u_{max} are taken to be -2.5 and 2.5 respectively in the first three scenarios.

Optimum solutions of feasible FRRs arising in 16 different problem instances are used as the training data set and the FRRs occurring in 4 new instances are utilized for testing purposes in each of the four scenarios. The locations of the obstacles and the waypoints are varied randomly in the first two scenarios, whereas the values of the location disturbance variance and the maximum acceleration are altered randomly in the next two scenarios, keeping the obstacle and waypoint locations fixed. The generated trajectories for one of the training and test problem instances for scenario 1 are shown in Fig. 11-10.

Table 11.6 shows that significant speed-up is obtained by using the proposed algorithm as compared to the previous one for all the different scenarios. At the same time, identical



(a) Sample training problem

(b) Test problem

Figure 11-10: Trajectories generated by NIRA+BoostLP for planning with obstacle avoidance using non-convex chance constrained optimal control algorithm. The obstacles are displaced in the test problem from their locations in the sample training problem, showing that regression models learnt from different but similar problems can be utilized to compute the optimum solutions for new problems.

cost (objective) function values are returned by both the algorithms. This fact clearly indicates that the optimality of the overall branch and bound algorithm is strictly preserved, even though its LP subproblems are solved approximately.

Figures 11-11 and 11-12 show the enhanced effect of computational speed-up for greater number of obstacles and longer planning time horizons respectively. The values of σ_x, σ_y, u_{min}, u_{max}, as well as the number of training and test problem instances are identical to those used for the scenarios in Table 11.6. N is chosen as 20 for all the problem instances in Fig. 11-11, only one obstacle is present for the problem instances in Fig. 11-12, no waypoints are present, and all the obstacles are rectangular.

It may be noted here that although solving FRRs approximately using regression does not prevent the exponential growth in computational time, it does reduce the *rate* of exponential growth. This happens because the regression inference time remains the same (for identical model complexity) at the internal nodes of the branch and bound trees in all the problem instances, unlike the CPLEX optimizer, whose running time increases significantly with the number of constraints and decision variables. The plots are not extended any further as the trends do not change and the computation time of the proposed algorithm

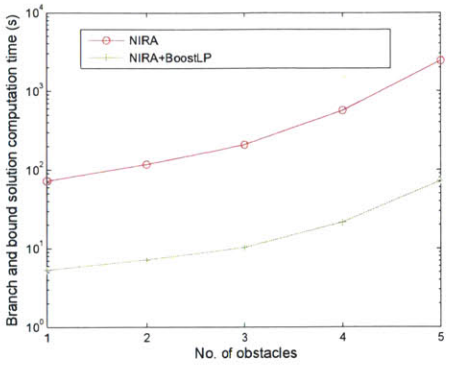


Figure 11-11: Computation time comparison for varying number of obstacles. Note that the plot is in semi-log scale and error bars are not shown as they are negligibly small.

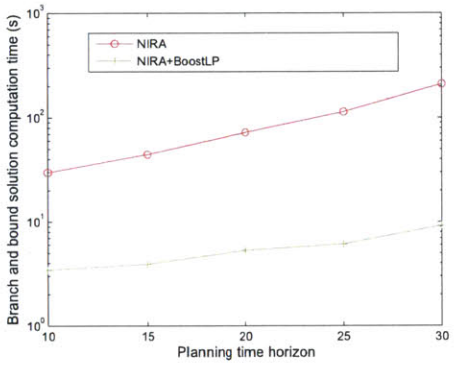


Figure 11-12: Computation time comparison for varying planning time horizons. Note that the plot is in semi-log scale and error bars are not shown as they are negligibly small.

also becomes significantly more than a few seconds, thereby rendering the approach less useful for practical applications. Again, it should be noted here that even though individual FRR problems are solved approximately, we ensure that we obtain an optimal solution to the overall control problem by conservatively pruning branches and invoking the exact convex optimization solver at the leaf nodes.

11.1.9 p-Sulu FH

Next we present the simulation results of p-Sulu FH on two problems. The simulations are conducted on a machine with a quad-core Intel Xeon CPU clocked at 2.40 GHz, and with 16 GB of RAM.

Path Planning with Obstacles

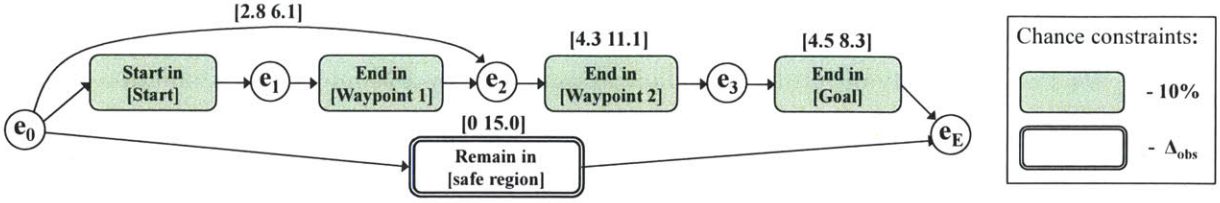


Figure 11-13: A sample CCQSP for a personal aerial vehicle's path planning and scheduling problem.

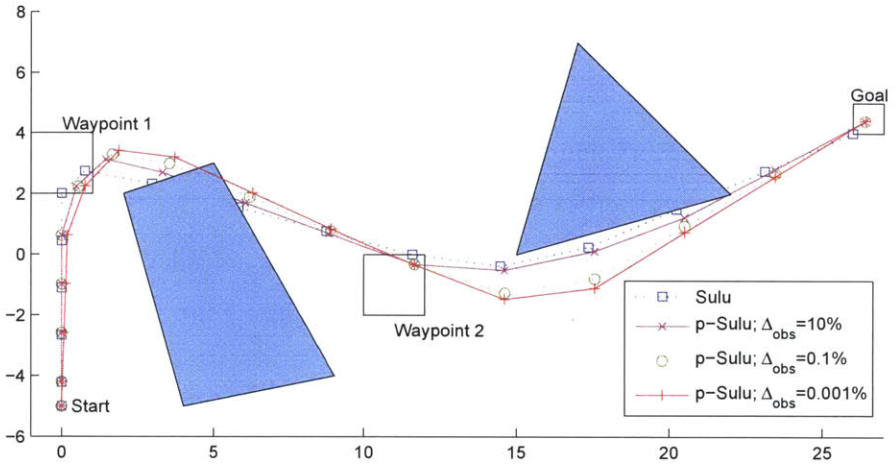


Figure 11-14: Output of p-Sulu FH for the CCQSP in Figure 11-13 with three different settings of the risk bound Δ_{obs}, compared to the path planned by a deterministic planner, Sulu, which does not consider chance constraints.

In this simulation we test p-Sulu FH on a path planning problem in the environment shown in Figure 11-14. The input CCQSP is shown in Figure 11-13. The CCQSP requires a vehicle to arrive at the goal region within 15 minutes, by going through Waypoint 1 and Waypoint 2 with the temporal constraints specified in Figure 11-13. It also imposes two chance constraints: one that requires the vehicle to achieve the temporally extended goals with 90% certainty, and another that requires the vehicle to limit the probability of violating the obstacles to Δ_{obs}. We set $\Delta T = 1$ and $\sigma^2 = 0.0025$.

Figure 11-14 shows the plans generated by p-Sulu FH with three different risk bounds: $\Delta_{obs} = 10\%$, 0.1%, and 0.001%. The computation times were 79.9 seconds, 86.4 seconds,

and 88.1 seconds, respectively. Figure 11-14 also shows the plan generated by Sulu, a deterministic planner that does not explicitly consider uncertainty [62]. Observe that Sulu leaves no margin between the path and obstacles. As a result, the Sulu path results in a 94.1% probability of hitting obstacles, as estimated by a Monte-Carlo simulation with 10^7 samples. On the other hand, p-Sulu FH leaves margins between the path and the obstacles in order to allow for uncertainties, and the margins are larger for the plans with smaller risk bounds. The probabilities of obstacle avoidance failure for the three p-Sulu FH plans, estimated by Monte-Carlo simulations with 10^7 samples, are 9.53%, 0.0964%, and 0.00095%, respectively. Hence the chance constraints are satisfied. The schedule optimized by p-Sulu FH is $\{s(e_0) = 0, s(e_1) = 5, s(e_2) = 10, s(e_E) = 15\}$, which satisfies all the temporal constraints in the CCQSP.

In Figure 11-13, it appears that the path cuts across the obstacle. This is due to the discretization of the time; the optimization problem only requires that the vehicle locations at each discrete time step satisfy the constraints, and does not consider the state in between. This issue can be addressed by a constraint-tightening method [56].

Path Planning in an Indoor Environment

We next give p-Sulu FH the CCQSP shown in Figure 11-15, which simulates a path planning problem in an indoor environment. A vehicle must get to the goal region at the other side of the room in three to five seconds. The "Remain in safe region" episode requires the vehicle to stay within the room and outside of the obstacle during the five-second planning horizon. The CCQSP imposes two chance constraints shown in Figure 11-15. We set $\Delta t = 0.5$ and $\sigma^2 = 5.0 \times 10^{-5}$.

Given this CCQSP, the planner faces a choice: heading straight to the goal by going through the narrow passage between the left wall and the obstacle minimizes the path length, but involves higher risk of constraint violation; making a detour around the right side of the obstacle involves less risk, but results in a longer path.

Figure 11-16 shows p-Sulu FH's outputs with $\Delta_{obs} = 10\%$, 1%, and 0.1%. The computation times were 35.1 seconds, 84.5 seconds, and 13.3 seconds, respectively. The result is consistent with our intuition. When p-Sulu FH is allowed a 10% risk, the planner chooses

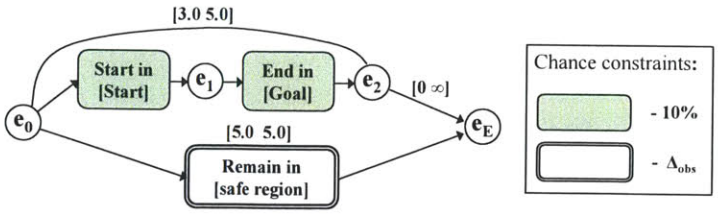


Figure 11-15: A sample CCQSP for a path planning problem in an indoor environment.

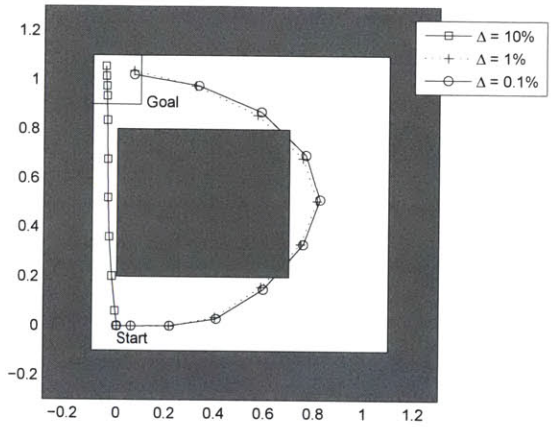


Figure 11-16: Output of p-Sulu FH for the CCQSP in Figure 11-13 with three different settings of the risk bound Δ_{obs}.

to go straight to the goal, resulting in the cost function value of 1.21; when the user gives a 1% or 0.1% risk bound, it chooses the risk-averse path, resulting in the cost function values of 3.64 and 3.84, respectively. This example demonstrates p-Sulu FH's capability to make an intelligent choice in order to minimize the cost, while limiting the risks to user-specified levels.

11.1.10 p-Sulu

We demonstrate p-Sulu on a path planning problem shown in Figure 11-17. The input CCQSP is shown in Figure 11-18. The simulations are conducted on a machine with a quad-core Intel Xeon CPU clocked at 2.40 GHz, and with 16 GB of RAM.

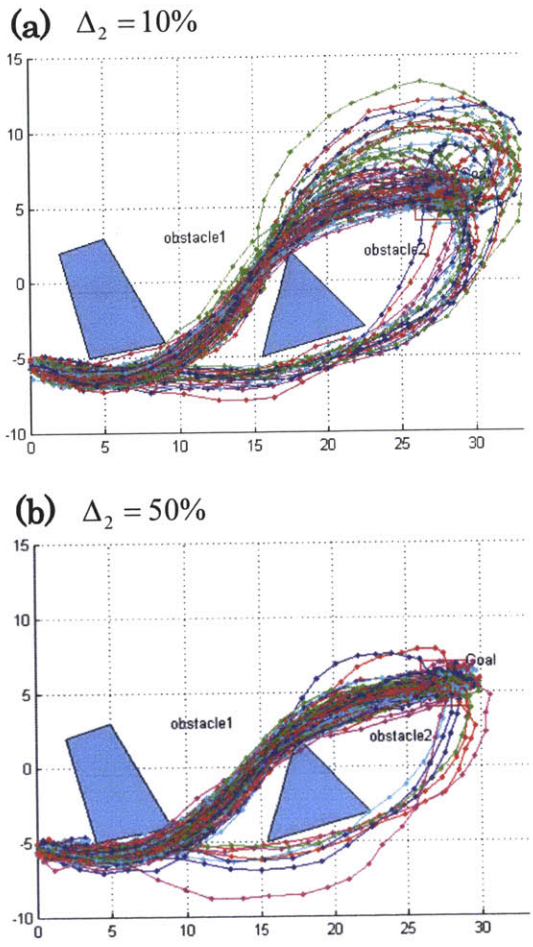


Figure 11-17: Simulation results of p-Sulu with **(a)**$\Delta = 0.1$ and **(b)** $\Delta = 0.5$. Each graph shows the paths of 100 runs with randomly generated disturbance.

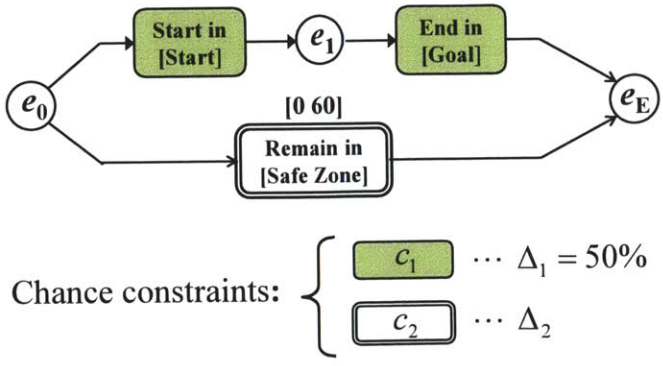


Figure 11-18: CCQSP for the path planning problem shown in Figure 11-17.

Each graph in Figure 11-17 shows the paths of 100 runs. Note that, although the paths go between the two obstacles most of the times, they sometimes go below the triangular obstacle. This is because p-Sulu can plan optimally based on the latest observation. When it finds that it is too risky to go between the two obstacles due to the disturbance, it guides the vehicle to the safer but longer path . Hence, when it is allowed to take higher risk, it is more likely to choose the riskier path, as shown in Figure 11-17-(b). In fact, it can be observed in Figure 11-17-(b) that some of the paths penetrate the obstacles.

Table 11.7 shows the resulting probabilities of failure with (a) $\Delta_2 = 0.1$ and (b)$\Delta_2 = 0.5$, evaluated by Monte-Carlo simulations with 1000 runs each. In both cases, the probabilities of failure are below the given risk bounds. This demonstrates the satisfaction of the chance constraints.

Table 11.7: Risk bounds and the probability of failure of p-Sulu. The probabilities of failure are evaluated by Monte-Carlo simulations with 1000 runs.

Scenario		(a)	(b)
c_1 (Goal achievement)	Δ_1	0.5	
	$P_{fail,1}$	0.058	0.119
c_2 (Obstacle avoidance)	Δ_2	0.1	0.5
	$P_{fail,2}$	0.007	0.055

Note that the probabilities of failure are significantly less than the risk bounds. In other words, solutions of p-Sulu have substantial suboptimality. The suboptimality is mainly resulted from the fact that p-Sulu executes only the first few steps in a planning horizon and abandons the rest. Since uncertainty accumulates over time, the state vectors at the abandoned time steps have more uncertainty than the executed ones. Since p-Sulu need to guarantee constraint satisfaction at all time steps in a planning horizon, resulting plans tend to be overly conservative. On the other hand, in off-line planning, the gaps between the probability of failure and the risk bound are comparably small (See Tables 11.3 and 11.5). Hence, it appears that the off-line planning is less conservative than the on-line planning. However, recall that off-line planners do not use observations on the fly, like walking with eyes closed. Therefore, these simulation results do not imply that on-line planning is less efficient than off-line planning.

We choose 10% and 50% risk bounds, although in practical problems the risk bound should be much smaller, such as 10^{-6}. This is not because p-Sulu cannot take a realistically small risk bound, but because otherwise statistical evaluation of the probability of failure becomes very difficult due to time limitations. Recall that p-Sulu is a receding horizon planner that runs in real time. Since it uses observations obtained on the fly, each run of the algorithm results in different control inputs. Hence, it must be run numerous times to evaluate the probability of failure; the probability cannot be evaluated by the ex-post Monte-Carlo simulation conducted for off-line planners, which runs a planner just once to obtain a nominal path and then enumerates samples around the nominal. If the risk bound is 10^{-6}, we must run the simulation at least 10^6 times to evaluate the probability. Since an execution of the CCQSP in Figure 11-18 takes up to 1 minute, 10^6 runs takes about 2 years. Simulation results with more realistic settings (i.e., more complex CCQSPs with smaller risk bounds) are presented in the next section.

11.1.11 MIRA

Next we demonstrate the MIRA algorithm, presented in Chapter 9, on two problems. One is the UAV altitude control problem in the fire-fighting scenario introduced in Section 9.1, and the other is a general altitude control problem with randomized constraints (we refer to this problem as the randomized problem).

Simulation Settings

In this simulation we consider the one-dimensional point-mass double-integrator plant presented in 11.1.4, with $\Delta T = 1$, $u_{\max} = 0.2$, and $\sigma^2 = 0.001$. We impose state constraints $h_t^l x_t^l \leq g_t^l$, where $h_i^l = -[1\ \ 0]$ and the superscript l is the index of agents. The bounds of the state constraints g_t^l are set to zero for the fire-fighting problem. For the randomized problem, g_t^l are generated by a random walk starting from $g_0^l = 0$, and $g_{t+1}^l - g_t^l$ is sampled from a uniform distribution in $[-0.3, 0.3]$ for the randomized problem. This problem can be considered as an altitude planning problem of aerial vehicles. The first component of the state vector x_t^l represents the altitude of the lth vehicle at time t. Assuming that the vehicles are flying at a constant horizontal speed, t is proportional to the horizontal distance traveled

by the vehicles. The ground level is represented by g^l_{t+1}. Hence, the state constraint above means that the altitude must be above the ground level.

The cost functions of the fire-fighting problem are:

$$\begin{aligned} J_W &= E\left[[100\ \ 0]\,(\boldsymbol{x}_{6,W} + \boldsymbol{x}_{7,W})\right] \\ J_R &= E\left[[1\ \ 0]\,(\boldsymbol{x}_{6,R} + \boldsymbol{x}_{7,R})\right], \end{aligned}$$

where subscript W and R indicate the water tanker and the reconnaissance vehicle respectively. The cont functions of the randomized problem are:

$$J^l = E\left[\begin{bmatrix} 1 & 0 \end{bmatrix} (\sum_{t=1}^{10} \boldsymbol{x}^l_t)\right].$$

Note that the expectation of $\boldsymbol{x}_t$ is a function of $\boldsymbol{u}_{1:t-1}$. Therefore J is a function of $\boldsymbol{u}_{1:N}$.

The simulations are conducted on a machine with a quad-core Intel Core i7 CPU clocked at 2.67 GHz, and with 8 GB of RAM.

UAV Fire-fighting Example

We first test MIRA on the multi-UAV altitude planning problem for the fire-fighting scenario (Figure 9-1). Figure 11-19 shows the simulation result. Two vehicles fly at the constant horizontal speed, starting from $t = 0$ at altitude 0.5. The mission is to extinguish the fire at $t = 6, 7$. Both vehicles minimize the flight altitude above the fire, although the water tanker is given 100 times more penalty (cost) of flying at high altitude than the reconnaissance vehicle. Both have uncertainty in altitude, so flying at lower altitude involves more risk. The total risk must be less than 0.1%.

The optimal plan allocates 99.2% of the total risk to the water tanker, while only 0.8% to the reconnaissance vehicle. This is because the utility of taking risk (i.e. flying low) is larger for the water tanker than for the reconnaissance vehicle. As a result, the water tanker flies at a lower altitude.

Both vehicles optimize the internal risk allocation as well. For example, the water tanker takes 99.9% of the allocated risk above the fire, at $t = 6$ and 7 (the middle graph in

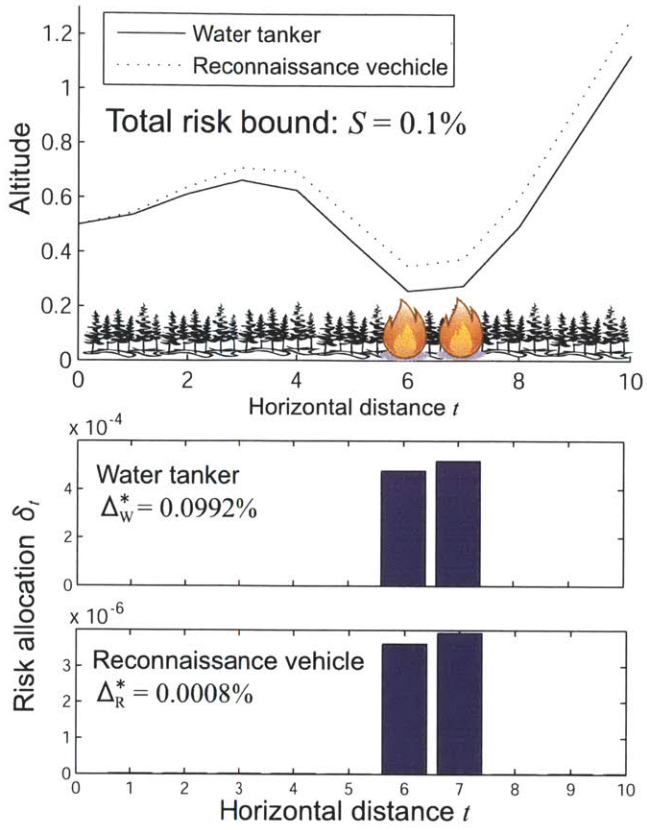


Figure 11-19: Simulation result of flight altitude planning problem for multi-UAV fire-fighting scenario (see Figure 9-1). Upper graph: Optimized altitude profile; Lower graphs: Internal risk allocation of each vehicle.

Figure 11-19).

The optimal action sequence is planned according to this risk allocation; both vehicles dive before the fire, and climb as fast as possible, after they pass the fire (the top graph in Figure 11-19). This is because there is no benefit of conducting risky low-altitude flight before and after the fire.

These results conform with intuition. The optimality of MIRA is validated by the result that the difference in the optimized cost between MIRA and the centralized algorithm is less than 0.01%, which is accounted by numerical error.

Performance Evaluation

We then compare the performance of MIRA with the centralized optimization approach, on the problems with randomly generated constraints. Fig. 11-20 shows the average computation times of 100 runs of the MIRA algorithm with 2 to 128 agents, compared against the centralized approach that directly solves Problem 15. Demands for risk are computed in parallel by each agent.

The computation time of the centralized algorithm quickly grows as the problem size increases. Although MIRA, the proposed algorithm, is slower for the problems with less than eight agents, it outperforms the centralized algorithm when the number of agents is more than eight. The exponential fits to the average computation time of MIRA and the centralized approach are $15.2e^{0.0160n}$ and $0.886e^{0.328n}$, respectively, where n is the number of agents. MIRA has a 20 times smaller exponent than the centralized approach, which means a significant improvement in scalability.

A counterintuitive phenomenon observed in the result is that MIRA also slows down for large problems, although not as significantly as the centralized algorithm. This is because iterations must be synchronized among all agents. When each agent computes its demand for risk by solving the non-linear optimization problem, the computation time diverges from agent to agent. In each iteration, all agents must wait until the slowest agent finishes computing its demand. As a result, MIRA slows down for large problems, as the expected computation time of the slowest agent grows. Our future work is to develop an asynchronous algorithm to improve scalability.

The computation time of the central module (CM), which is shown in Figure 11-21, is at most 0.1% of the total computation time of MIRA. Figure 11-21 also shows that the number of iterations are almost constant. Moreover, the computational complexity of the root finding algorithm used for the CM does not increase with the number of agents. As a result, the computation time of the CM (Figure 11-21) grows less significantly than the computation time of the entire algorithm (Figure 11-21). Therefore, the existence of the central module does not harm the scalability of MIRA. The growth in the computation time of the CM is mainly due to computational overhead of handling the data from multiple agents.

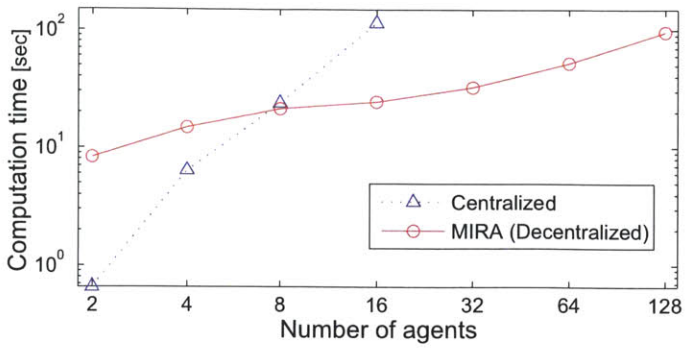


Figure 11-20: Computation time of MIRA compared to the centralized method. Plotted values are the average of 100 runs with randomly generated constraints. Note that plot is in log-log scale.

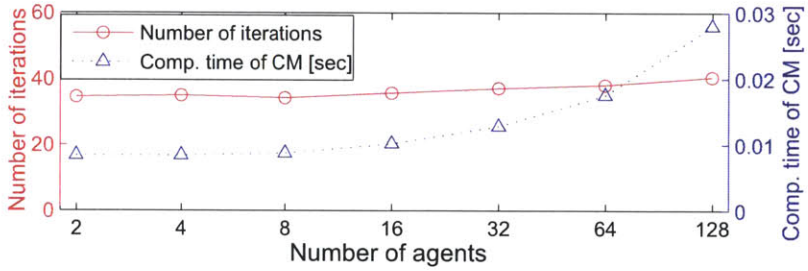


Figure 11-21: Number of iterations and the computation time of the central module (CM) of MIRA. Plotted values are the average of 100 runs with randomly generated constraints.

Additional Notes on Simulation Settings

In the decentralized approach (MIRA), a convex optimization solver SNOPT is used to solve Problem 16 (computation of $D^l(p)$), and the Matlab implementation of Brent's method (fzero) is used to find the root $p^\star$ of Problem 17. In the centralized approach, Problem 15 is solved by SNOPT. Since it is hard to set exactly the same optimality tolerances for centralized and decentralized approaches, we set a *stricter* tolerance for the decentralized approach than the centralized approach. Specifically, the optimality tolerance of SNOPT is defined in terms of the complementary slackness normalized by dual variables $\frac{p}{\|\pi\|}\left|\sum_{l\in\mathcal{L}} D^l(p) - S\right| < \epsilon$, where π is the vector of all dual variables. In the decentralized approach, we set the tolerance of fzero as $\left|\sum_{l\in\mathcal{L}} D^l(p) - S\right| < \epsilon$. This tolerance is stricter than the previous one since $p/\|\pi\| \leq 1$. We set $\epsilon = 10^{-6}$. MIRA is simulated by a single

processor; however, we counted the computation time of the agent that is slowest to compute the demand in each iteration, so that the result shown in Fig. 11-20 corresponds to the computation time when running MIRA with parallel computing. Communication delay is not simulated in our result.

11.2 PTS Simulation

Next, we demonstrate the proposed plan executives on the PTS scenarios.[2] Our demonstration is developed in three spirals.

In Spiral 1, we use p-Sulu FH as a stand-alone controller. The goal of Spiral 1 is to show that our approch can scale to real-world problems.

In Spiral 2, we integrate p-Sulu with other planners to create Integrated Plan Executive (IPE), in order to augment the capabilities of high-level decision making and natural language processing. Besides p-Sulu, IPE include Kirk (a temporally-flexible contingent planner), Collaborative Diagnosis System (a plan conflict fixer), Dialog Management System (a natural language interpreter), and Coordinator (a GUI interface). The goal of Spiral 2 is demonstrate p-Sulu's capability of adapting to environmental changes, such as the change in the location of storms. The simulation result of Spirals 2 is visualized by a flight simulator called X-Plane.

In Spiral 3, we use dp-Sulu to control three PAVs. The goal of Spiral 3 is to demonstrate dp-Sulu's capability of controlling a multi-agent system in a distributed manner. We also demonstrate its capability of avoiding inter-vehicle collisions through Bi-stage Robust Collision Avoidance.

11.2.1 Plant Parameters

The 2-D point-mass double-integrator plant, introduced in Section 11.1.4, is used in all spirals of the PTS simulations. The plant parameters are chosen to approximate a real aircraft. We set the maximum speed, $v_{\max}$, to 250 m/s, which approximates the maximum

[2]We acknowledge Michael Kerstetter, Scott Smith, Ronald Provine, Hui Li, and the Boeing Company for their support and collaboration.

cruise speed of private jet airplanes, such as Gulfstream V. The maximum acceleration is determined from the maximum bank angle. Assuming that an aircraft is flying at a constant speed, the lateral acceleration a is given as a function of the bank angle ϕ as follows:

$$a = g \cdot \tan \phi,$$

where g is the acceleration of gravity. Typically passenger aircrafts limit the bank angle to 25 degrees for passenger comfort, even though the aircrafts are capable of truing with a larger bank angle. Hence, we use:

$$u_{\max} = 9.8 \text{ m/s}^2 \cdot \tan(25^\circ) = 4.6 \text{ m/s}^2.$$

The standard deviation of disturbance is set to $\sigma = 100$ m and the time interval is set to $\Delta T = 60$ seconds. We set such an unrealistically high level of uncertainty in order to make the uncertain dynamics visible (Observe in Figure 11-30, for example, that paths are disturbed.) In order to allow such large uncertainties, we set relatively high risk bounds.

In Spiral 1 and 2 the simulations are conducted on a machine with a quad-core Intel Xeon CPU clocked at 2.40 GHz, and with 16 GB of RAM. In Spiral 3, we deploy dp-Sulu on a PTS with three vehicles. The central module of dp-Sulu are run on a machine with a quad-core Intel Core i7 CPU clocked at 2.67 GHz and with 8 GB of RAM, while the distributed component (i.e., p-Sulu) of the three vehicles are run on three machines: the first one with a quad-core Intel Xeon CPU clocked at 2.40 GHz and with 16 GB of RAM, the second one with a dual-core Intel Xeon CPU clocked at 1.69 GHz and with 512 MB of RAM, and the third one with a quad-core Intel Core i7 CPU clocked at 1.60 GHz and with 8 GB of RAM.

11.2.2 Spiral 1 : Single-agent, Stand-alone

In Spiral 1, we deploy p-Sulu FH on PTS without integrating other planners. We manually generate input CCQSPs to represent the scenarios described below.

Scenarios

We consider three scenarios, specified by the CCQSPs shown in Figure 11-22. Scenarios 1 and 2 are similar to the scenic flight scenario introduced at the beginning of this thesis (see Figure 1-1). In Scenario 1, a PAV takes off from Runway 7 of Provincetown Municipal Airport (KPVC) in Provincetown, Massachusetts[3], flies over a scenic region, and lands on Runway 23 of Hanscom Field (KBED) in Bedford, Massachusetts. The vehicle is required to stay within the scenic region at least for 2 minutes and at most for 10 minutes. The entire flight must take more than 13 minutes and less than 15 minutes. Scenario 2 is the same as Scenario 1, except for the runways used for take-off and landing.

Scenario 3 simulates a leisure flight off the coast of Massachusetts. A PAV takes off Runway 7 of Provincetown Municipal Airport, and flies over two regions where whales are often seen. Then the vehicle lands on Runway 11 of Hanscom Field.

We place three no-fly zones, as shown in Figure 11-23. The entire flight must take more than 13 minutes and less than 15 minutes. Each scenario has three chance constraints, $\{c_1, c_2, c_3\}$, as shown in Figure 11-22. The first one, c_1, is concerned with the vehicle's operation; it requires the vehicle to take off from and land on the right runways at the right airports with less than 10 % probability of failure. The second chance constraint, c_2, is concerned with the leisure activities; it requires the vehicle to fly over the scenic regions with less than 10 % probability of failure. Finally, c_3 is concerned with the passenger's safety; it requires the vehicle to limit the risk of penetrating the no-fly zones to 0.01 %.

Simulation Results

Figure 11-23 shows the paths planned by p-Sulu FH for the three scenarios. In all the scenarios, all the episode requirements in the CCQSPs in Figure 11-22 are met within the specified temporal constraints.

Table 11.8 compares the performance of Sulu and p-Sulu FH. As expected, Sulu's plans result in excessive probabilities of failure in all scenarios. This is because Sulu does not consider uncertainty in the planning, although the PAV is subject to disturbance in real-

[3] A runway of an airport is specified by a number, which represents the clockwise angle from the north. For example, Runway 7 points 70 degrees away from the north.

ity. On the other hand, p-Sulu FH successfully limits the probability of failure within the user-specified risk bounds for all the three scenarios. Furthermore, although p-Sulu FH significantly reduces the risk of failure, its cost function value is higher than that of Sulu only by 9.5 - 12.8 %. Such a capability of limiting the risk and maximizing the efficiency at the same time is a desirable feature for PTS, which transport passengers.

Such robust planning capability of p-Sulu FH is obtained with a cost of computation time; as shown in Table 11.8, p-Sulu FH typically takes several minutes to compute the plan. This length of computation time would be allowed for PTS applications, since we assume that p-Sulu FH is used for preplanning in Spiral 1.

In Spiral 2, the real-time plan executive, p-Sulu, is deployed.

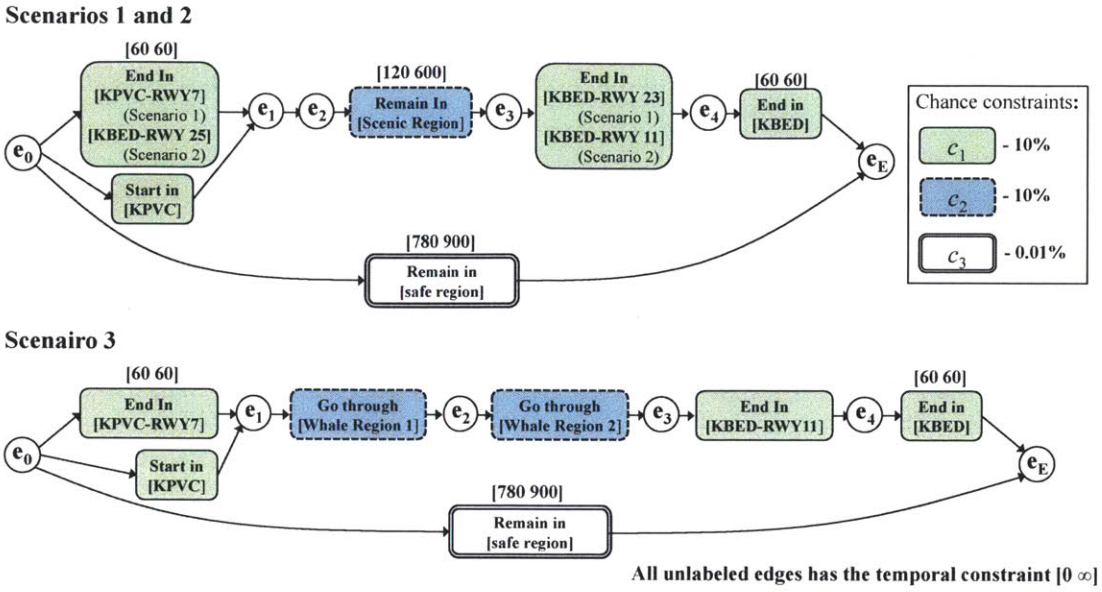


Figure 11-22: The CCQSPs for the PTS scenarios in Spiral 1.

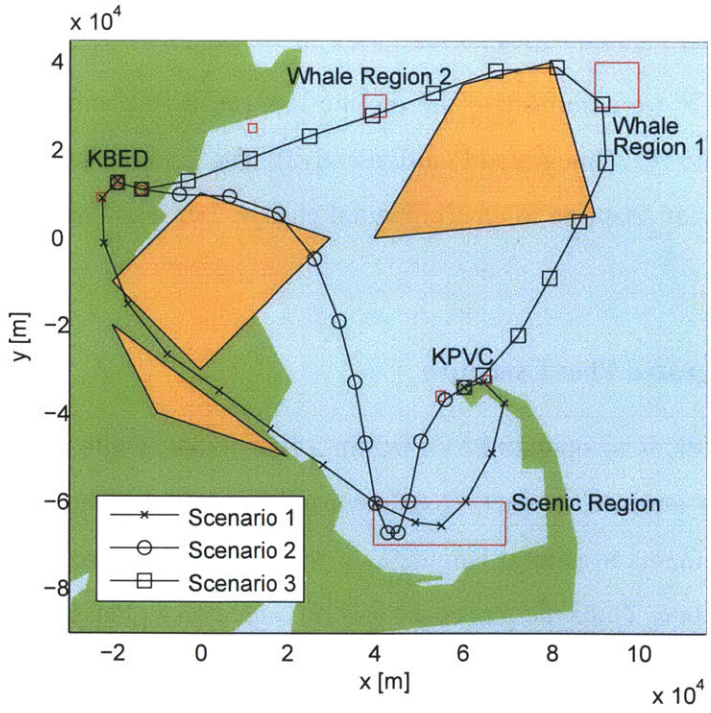


Figure 11-23: The paths planned by p-Sulu FH.

Table 11.8: Performance Comparison of the prior art, Sulu, and the proposed planner, p-Sulu FH. Risk bounds Δ_c and $P_{fail,c}$ of three chance constraints are presented. The three chance constraints are concerned with the PAV's operation (c_1), leisure activities (c_2), and safety (c_3).

Scenario #		1		2		3	
Algorithm		Sulu	**p-Sulu FH**	Sulu	**p-Sulu FH**	Sulu	**p-Sulu FH**
c_1	Δ_1	0.1					
	$P_{fail,1}$	0.999	9.12×10^{-2}	0.996	9.14×10^{-2}	0.999	9.23×10^{-2}
c_2	Δ_2	0.1					
	$P_{fail,2}$	0.807	8.46×10^{-2}	0.813	8.59×10^{-2}	0.603	7.65×10^{-2}
c_3	Δ_3	10^{-4}					
	$P_{fail,3}$	0.373	2.74×10^{-5}	0.227	2.62×10^{-5}	0.372	2.81×10^{-5}
Cost function value $J^\star$		24.2	27.5	21.0	23.7	20.0	22.3
Computation time (s)		2.58	60.2	2.00	390	5.17	198

11.2.3 Spiral 2 : Single-agent, Integrated

Turning to Spiral 2, we demonstrate the on-line risk-sensitive planning capability of p-Sulu, integrated into IPE (Integrated Plan Executive). The goal of Spiral 2 is demonstrate p-Sulu's capability of adapting to environmental changes, such as changes in the location of storms.

Overview of Integrated Plan Executive

The objective of IPE is to operate PAVs within a risk bound, while allowing passengers to command in an intuitive manner. As shown in Figure 11-24, the passenger communicates his requirements in natural language. If the requirements are infeasible, IPE proposes alternative plans. Once the passenger and IPE agree on a plan, IPE starts controlling the PAV. The robust control commands are generated by p-Sulu. Recall that p-Sulu is a receding-horizon executive, meaning that it adapts to changes in environment (e.g., moving obstacles) through continuous replanning. Hence, IPE continuously receives updates on the environment state, as well as the plant state.

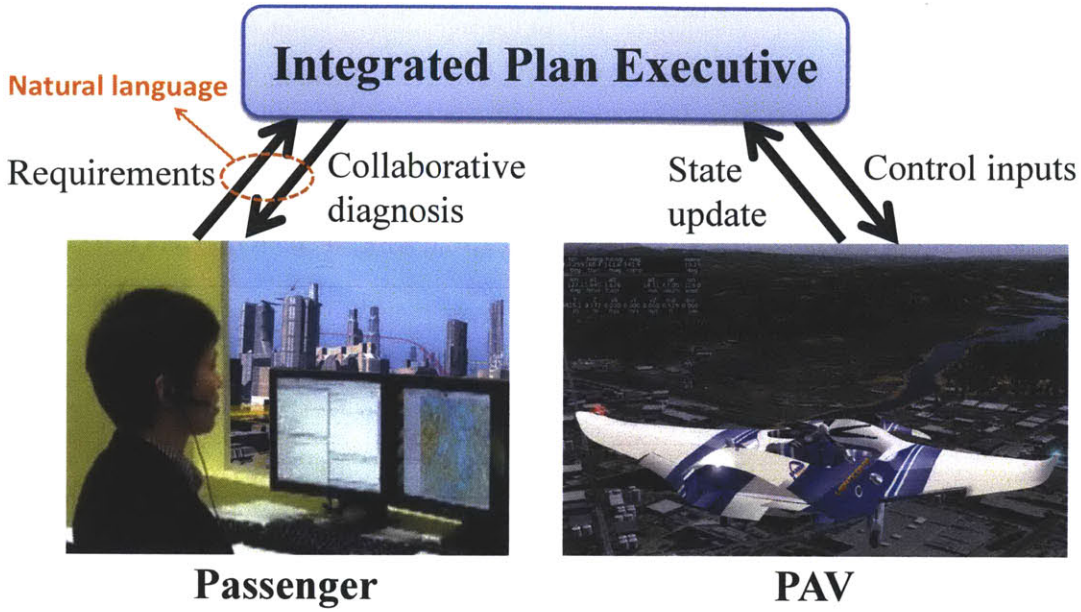


Figure 11-24: Integrated Plan Executive.

Integration

As shown in Figure 11-25, IPE consists of p-Sulu and four other components: Dialog Management System (DMS), Collaborative Diagnosis System (CDS), Kirk, and Coordinator. The functions of the four components are summarized below.

- **Dialog Management System (DMS)** [63, 64], developed by Stanford and Boeing Research & Technology, provides the capability of communicating with users in natural language. Through this software, the passenger communicates to PTS his requirements on the arrival time, preferred route, and acceptable level of risk.

- **Collaborative Diagnosis System (CDS)** [107] detects infeasibilities in the passenger's requirements, and if there are infeasibilities it proposes alternative feasible plans. For example, if the passenger requires the PAV to fly above a scenic region and arrive at a destination within three minutes but such a plan is infeasible, CDS proposes to allow a longer flight time, or to go to the destination directly.

- **Kirk** [37, 44, 50] is a temporally-flexible contingent plan executive, which is capable of making high-level choices. For example, when the destination airport becomes

unavailable unexpectedly, it can choose the best alternative landing site among numerous options.

- **Coordinator** serves as a graphical interface that visualizes the user's requirements, the plan generated by Kirk, and the path generated by p-Sulu. It also serves as a communication hub. It receives a flight plan from Kirk, encodes the plan in a CCQSP, and sends it to p-Sulu.

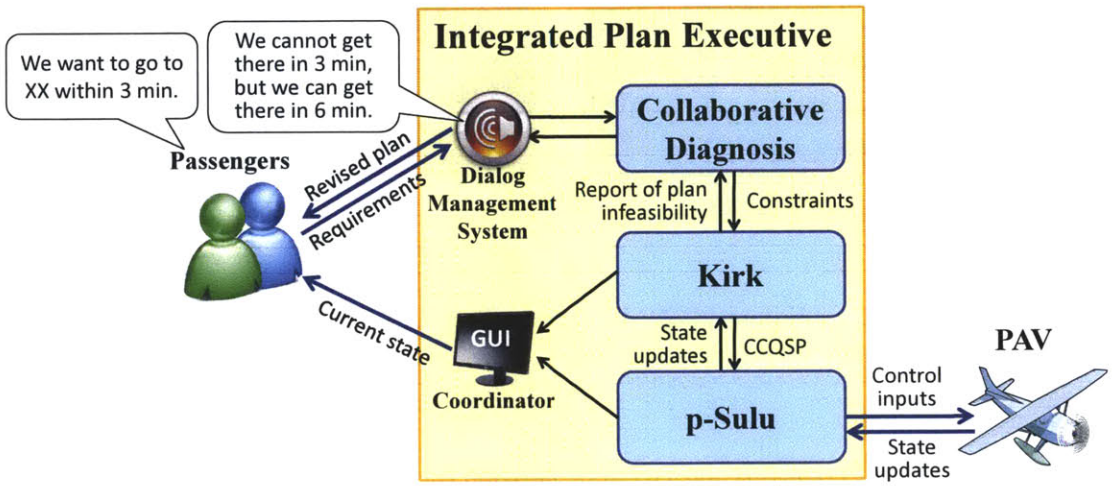


Figure 11-25: Integrated Plan Executive.

Scenario

We set our scene in the Greater Seattle area, as shown in Figure 11-27. The passenger of the PAV is an employee of the Boeing Company, who lives near Sanderson Field. His plan is to start from Sanderson Field, stop by Leisureland Airpark to pick up his colleague, and go to his office, within six minutes.

Simulation Settings

We consider the real no-fly zones in the Greater Seattle area. The blue and pink dashed lines in Figure 11-27 represent the boundary of the no-fly zones around airports. Additionally, there is a military-related no-fly zone near the top of the map, shown by the pink solid

lines. We also consider a moving obstacle that simulates a storm, as shown in Figure 11-28. The storm moves from left to right during the simulation. In order to test the executive's capability of adapting to contingency, we make the destination airport unavailable during the flight, forcing the IPE to choose an alternative landing site.

A CCQSP is automatically generated by IPE. Kirk creates a high-level plan, and Coordinator encodes the plan in a CCQSP. When Kirk changes the flight plan, the CCQSP is updated accordingly.

We impose three chance constraints, as shown in the CCQSPs in Figures 11-27 to 11-29. The first chance constraint requires that the intermediate goals set by Kirk are achieved with 90% certainty. The second chance constraint requires that the PAV reaches the final destination on time with 99% certainty. Finally, the third chance constraint requires the PAV to limit the risk of penetrating into the no-fly zones and the storm region to 0.01%.

The same plant model is used as in Spiral 1. We set $\Delta T = 1.0$ second. Execution horizon is set to 5.0 seconds, and the planning horizon is set to 10.0 seconds. The simulations are conducted on a machine with a quad-core Intel Xeon CPU clocked at 2.40 GHz, and with 16 GB of RAM.

Simulation Result

Figure 11-26 shows the natural-language input to IPE, as well as IPE's response to the passenger. IPE found that the passenger's request to reach the destination in six minutes was infeasible. Hence, it proposed an alternative to allow a nine-minute flight time, to which the passenger agreed.

Figure 11-27 shows the flight plan generated by Kirk at the beginning of the flight, as well as its CCQSP representation. The original plan was to go to the Boeing Company within 9 minutes with a stop at Leisureland Airpark. The initial CCQSP is shown in Figures 11-27.

During the execution of this CCQSP by p-Sulu, a storm cut across the planned path. As a result, Kirk changed the flight plan as shown in Figures 11-28, and updated the CCQSP. p-Sulu responded to this plan change, and planned a dynamically feasible path that avoids the storm with a sufficient margin in order to guarantee the satisfaction of the chance

constraints. This replanning process was repeated five times during the flight as the storm moved. Since the storm moved away eventually, p-Sulu reoptimized the path to reach the goals on time, resulting in the path shown in Figure 11-29.

Figures 11-27 to 11-29 shows the execution result. After successfully avoiding the storm and stopping by Leisureland Airpark, the PAV found that the destination airport was closed. As a result, Kirk made a decision to land on the back-up landing site, Harvey Field, and updated the CCQSP again, as shown in Figure 11-29. p-Sulu adapted to the plan change, and generated a path that avoided the no-fly zone around Snohomish County Airport and guided the PAV to Harvey Field.

Passenger:	I want to go to the Boeing Company.
PTS:	There are four back-up landing sites: Firstair Field, Heineck Farm, Ed Large Field, and Harvey Field. Which is your preferred alternate?
Passenger:	Harvey Field.
PTS:	Any other requirements? There is a small storm between Sanderson Field and the Boeing Company.
Passenger:	I want to be there in 6 minutes.
PTS:	Any other requirements?
Passenger:	I want to stop at Leisureland.
PTS:	Any other requirements?
Passenger:	No.
PTS:	So you want to get to the Boeing Company in 6 minutes, with a stop at Leisureland. (PTS checks plan feasibility)
PTS:	We cannot get there in 6 minutes, but we can get there in 9 minutes.
Passenger:	That's fine.
PTS:	OK, I have a valid plan.
Passenger:	Execute the plan.

Figure 11-26: The natural-language inputs to Integrated Plan Executive and its responses.

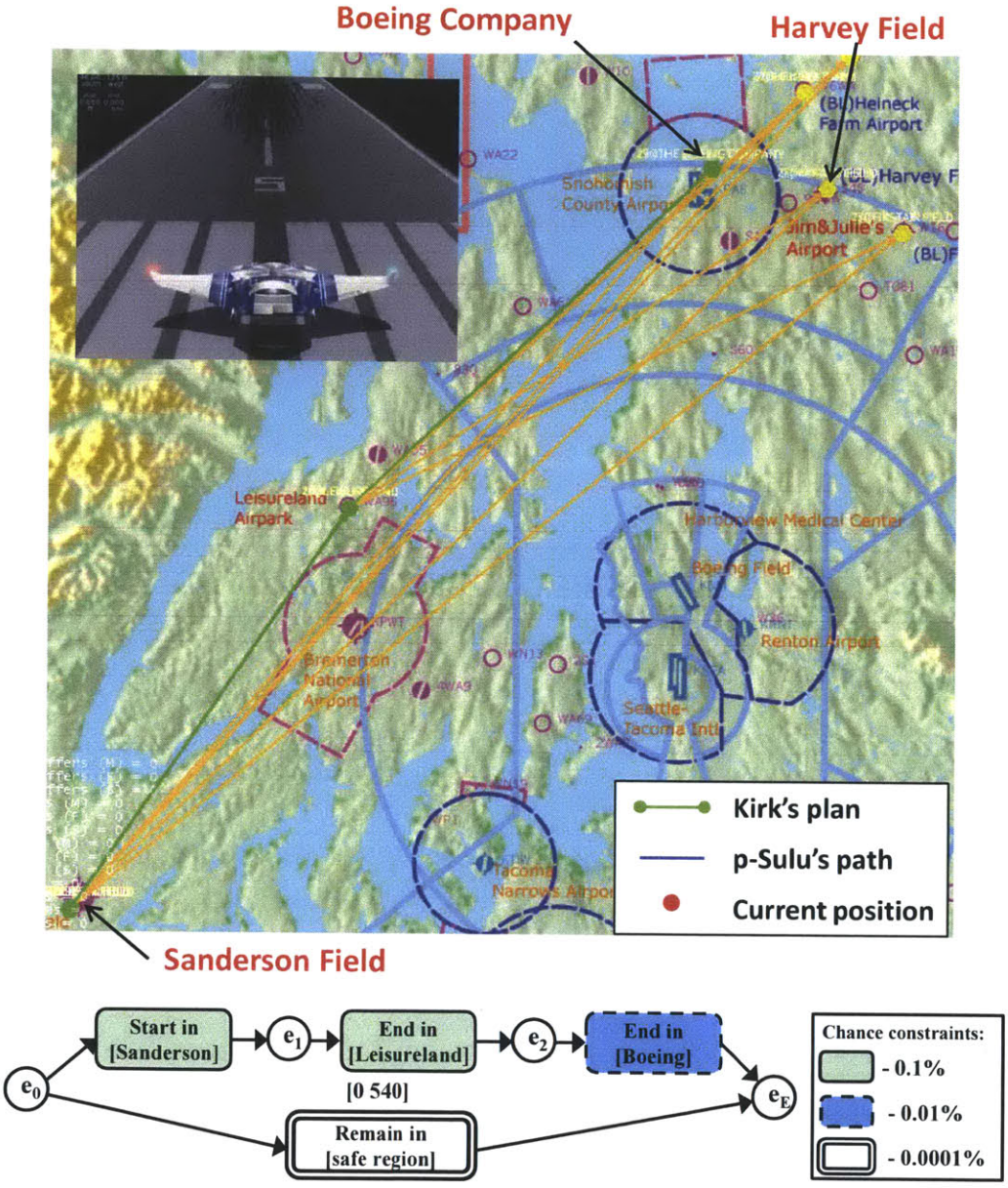


Figure 11-27: The flight plan generated by Kirk before the take-off, shown as the green solid line, and its CCQSP representation. The blue and pink dashed lines represent the boundaries of no-fly zones.

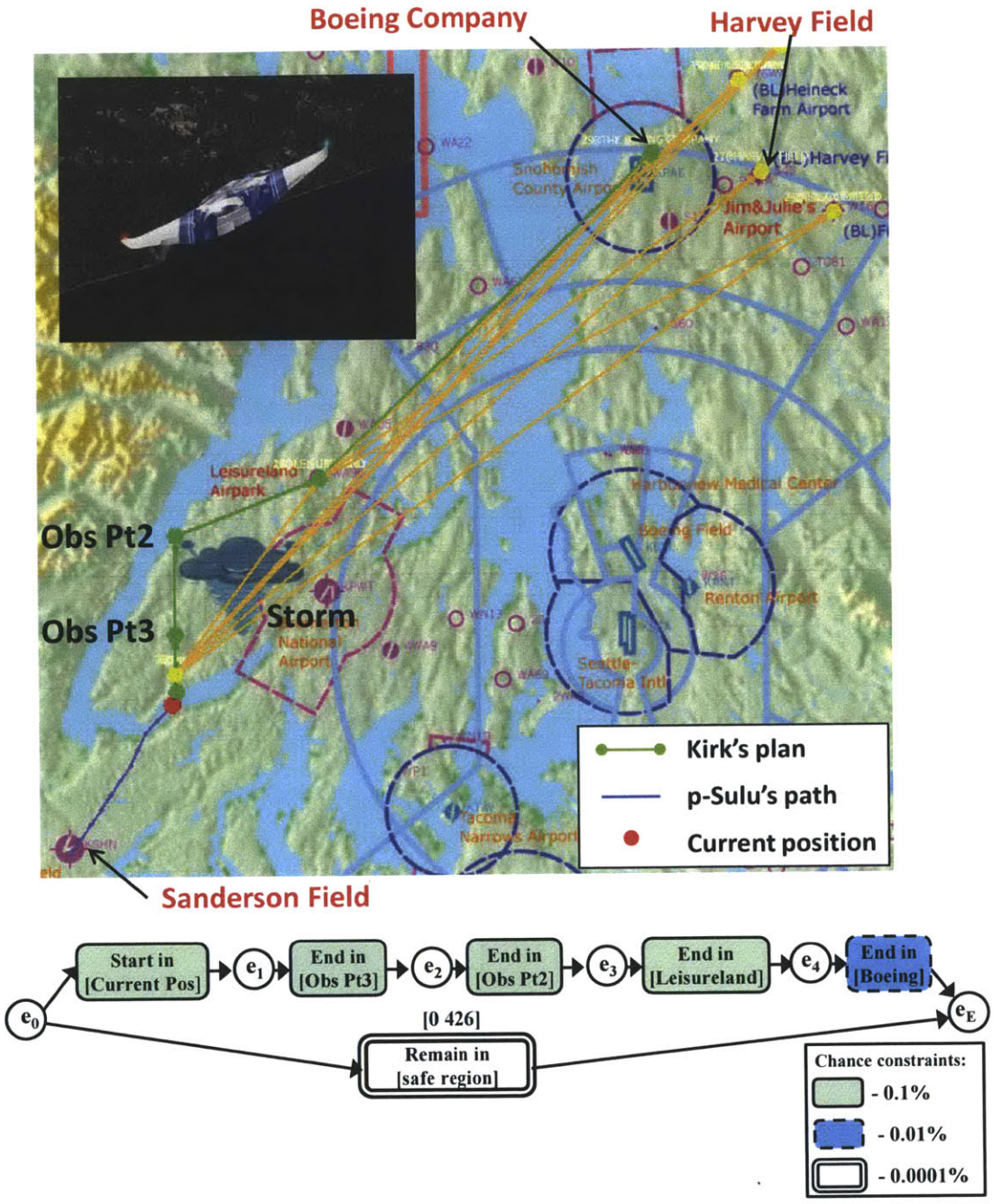


Figure 11-28: The updated plan after the storm's approach, and its CCQSP representation. The storm moves from left to right. Kirk and p-Sulu react to the changes in the storm's position.

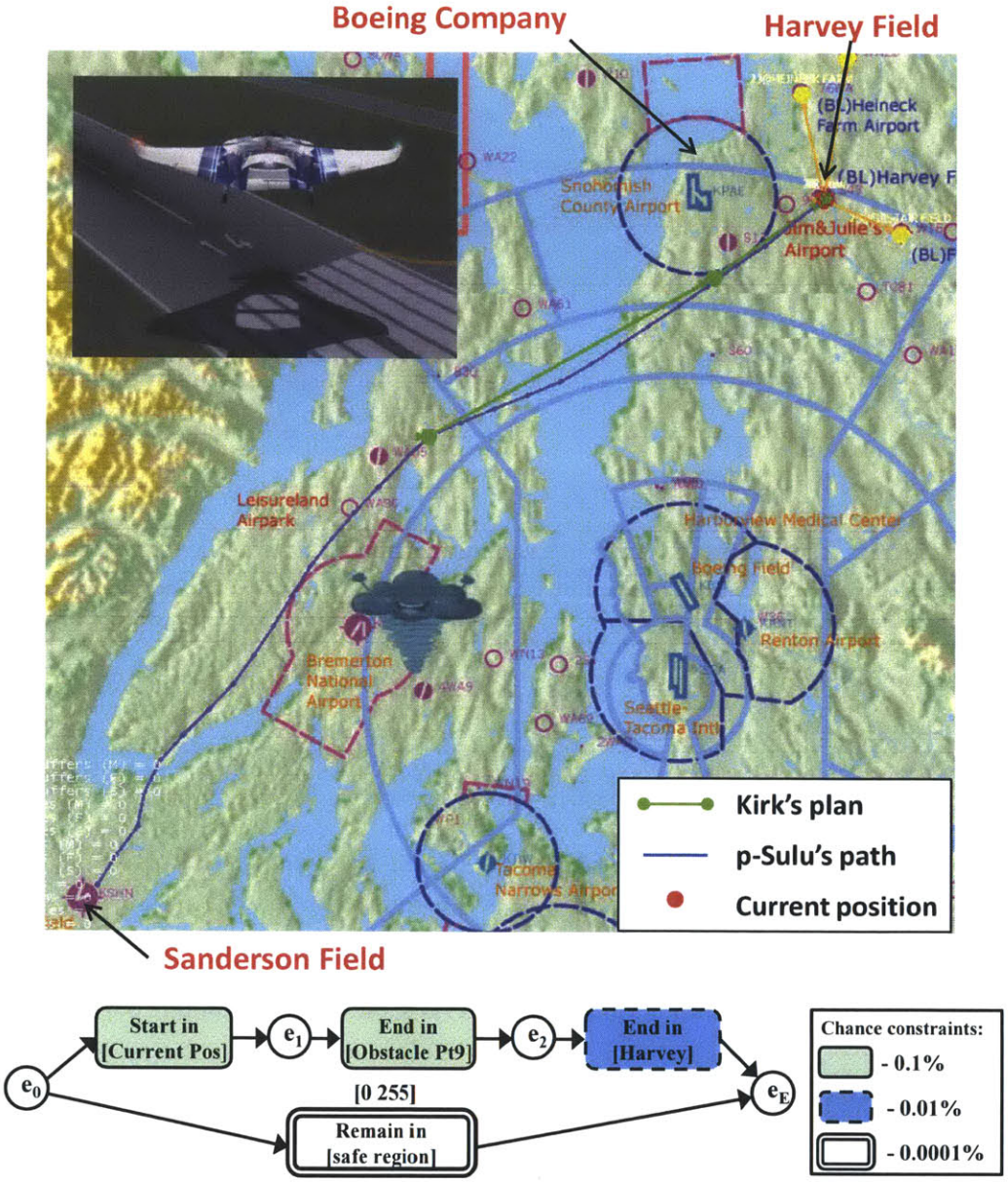


Figure 11-29: Completed path after landing. Since the original destination become unavailable, the PAV landed at an alternative landing site.

In the next spiral, we present a PTS demonstration with multiple PAVs, using dp-Sulu.

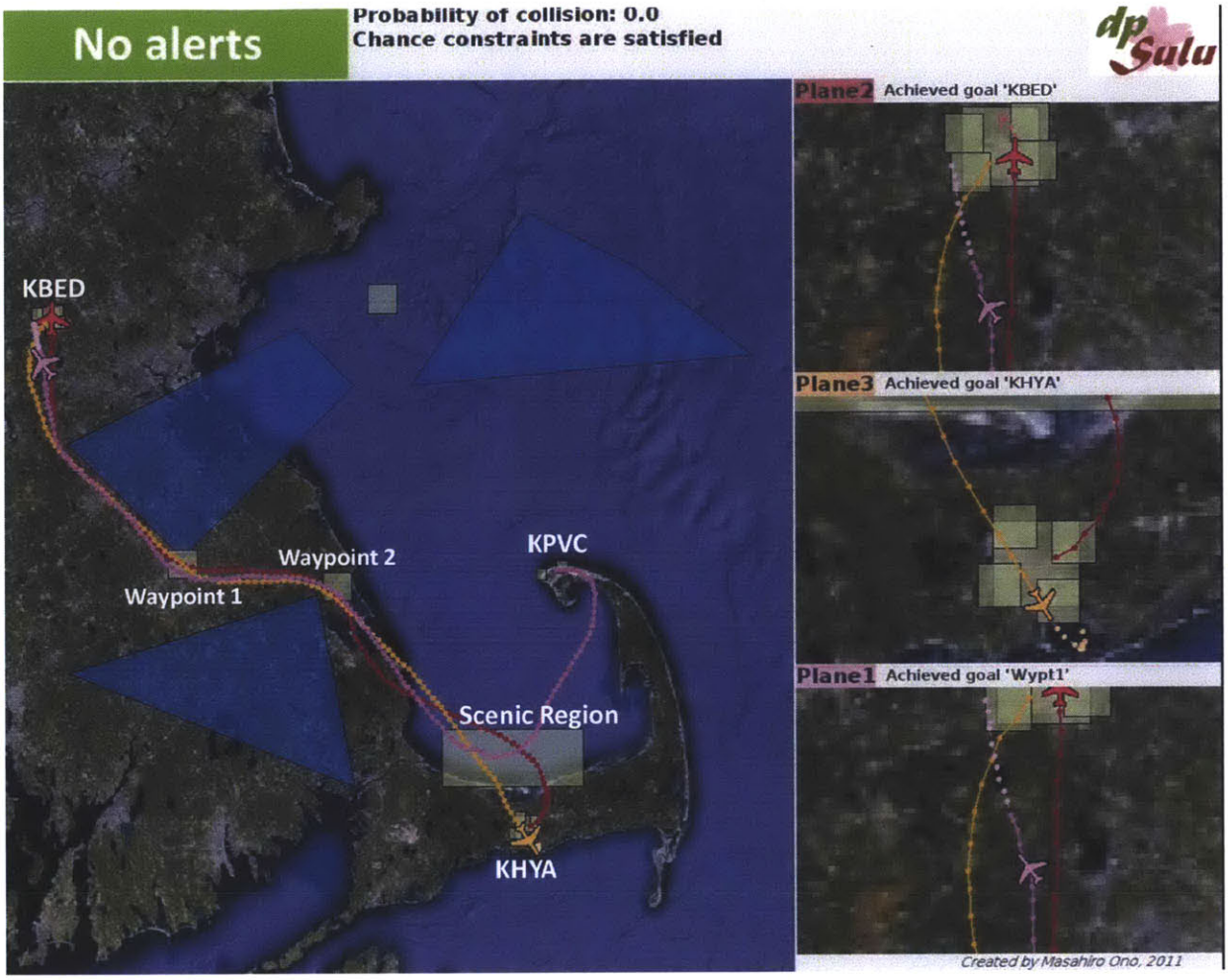


Figure 11-30: A PTS scenario with three PAVs, on which dp-Sulu is deployed.

11.2.4 Spiral 3 : Multi-agent, Integrated

Turning to Spiral 3, we demonstrate the capability of dp-Sulu to control multiple vehicles in a distributed manner.

Scenario

We revisit the PTS scenario presented at the very beginning of this thesis (Figure 1-2 in Section 1.1), with an additional vehicle. Figure 11-30 shows the environment of the scenario, while Figure 11-31 shows the plan. In the plan, PAV 1 takes off from Runway 7 of Provincetown Municipal Airport (KPVC) in Provincetown, Massachusetts, flies over a

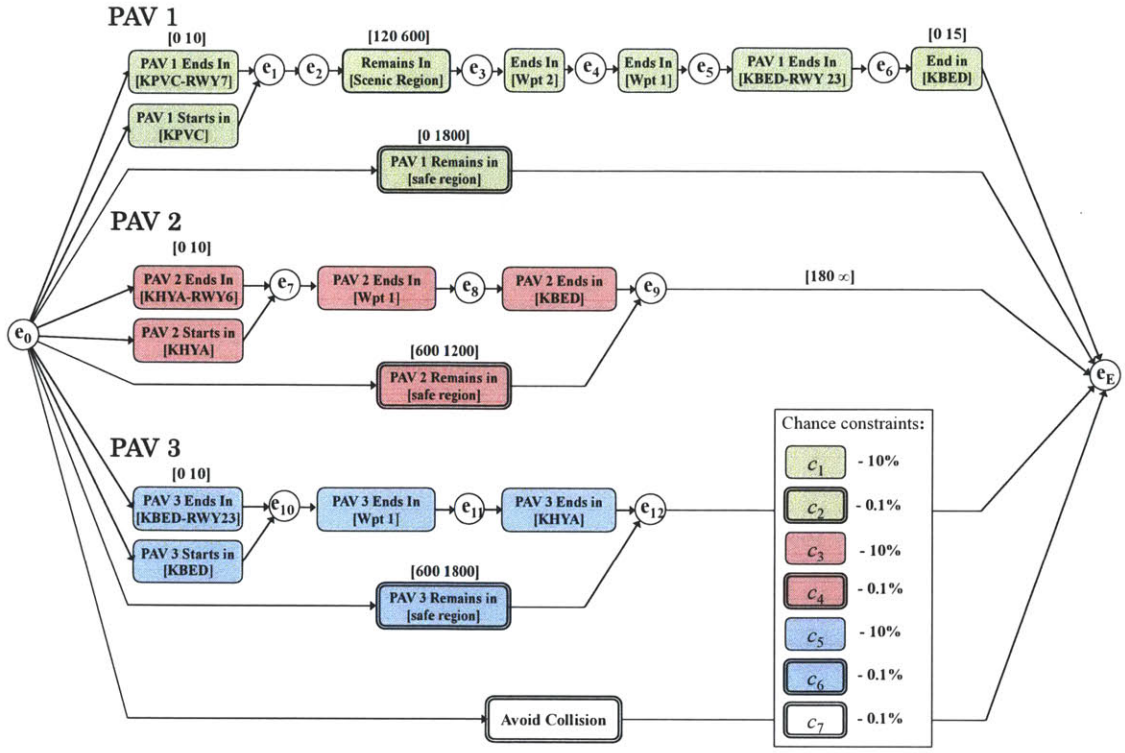


Figure 11-31: The CCQSP for the PTS scenario shown in Figure 11-30.

scenic region, goes through two waypoints, and lands on Runway 23 of Hanscom Field (KBED) in Bedford, Massachusetts. PAV 2, which takes off from Runway 6 of Barnstable Municipal Airport (KHYA) in Hyannis, Massachusetts, also heads to Hanscom Field. Meanwhile, PAV 3 starts from Hanscom Field and flies to Barnstable Municipal Airport, taking the opposite way of PAV 2. Each vehicle is required to satisfy its own chance constraints on avoiding obstacle and achieving goals, as shown in Figure 11-31. Additionally, the vehicles are required to limit the probability of collision below 0.1%. (Strictly speaking, we bound the probability that any vehicle approaches within 100 meters of another vehicle, instead of directly bounding the probability of collision.) All flights must be completed within 30 minutes. Since PAV 1 and PAV 2 land at the same airport, it is required to have at least a three minute interval between the two landings.

Simulation Results

Figure 11-30 is a screen capture of the central module of dp-Sulu, which plots the paths of the three PAVs. The main window on the left shows the overall paths, while the three small windows on the right show close-up views of the PAVs. Note that paths are disturbed by uncertainties. Despite the uncertainties, all three vehicles successfully achieve all goals.

On the way, PAVs 1 and 3 approach so close that the probability of collision exceeds the risk bound, 0.1%, as shown in Figure 11-32-(a). The paths of the two vehicles are immediately adjusted by Bi-stage Robust Collision Avoidance (Section 10.4), as shown in Figure 11-32-(b). Notice that, in the middle right window of 11-32-(a), the last waypoint of 'Plane 1' (shown in purple) is very close to a waypoint of 'Plane 3'. In 11-32-(b), the two waypoints are separated with a sufficient margin in order to make the probability of collision below the given risk bound.

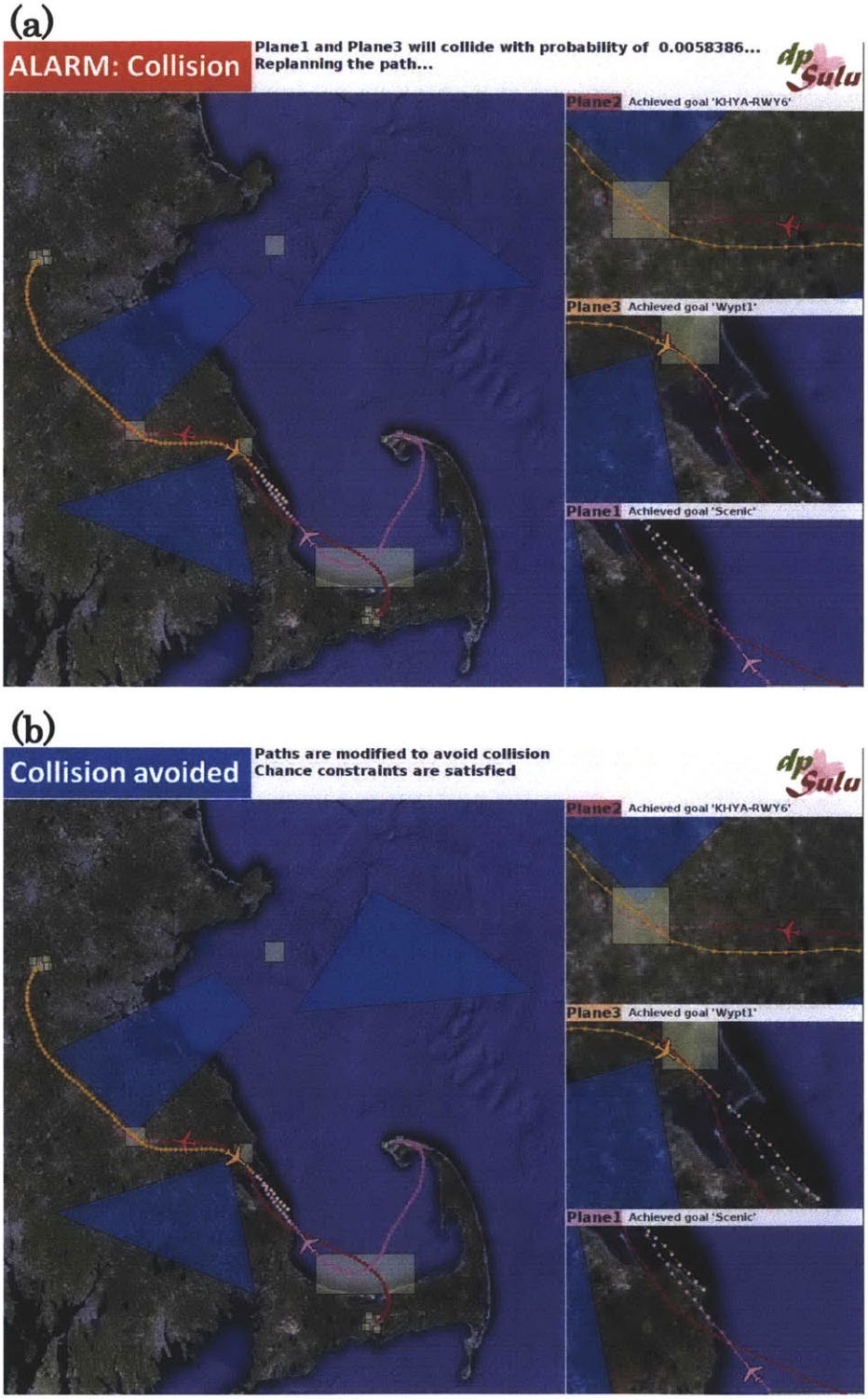


Figure 11-32: Demonstration of Bi-stage Robust Collision Avoidance (BRCA). (a) The central module finds that Planes 1 and 3 will collide with a probability above the risk bound. (b) BRCA adjusts the paths to avoid collision.

11.3 Application to Underwater and Space Vehicles

The application of our planners and executives are not limited by a specific plant model. The objective of this section is to demonstrate that our approach is applicable to a wide range of real-world problems. Section 11.3.1 apples the IRA algorithm to the depth planning of an autonomous underwater vehicle. Section 11.3.2 deploys p-Sulu FH on a space rendezvous scenario, and 11.3.3 applies dp-Sulu to space formation flight.

11.3.1 Depth Planning of an Autonomous Underwater Vehicle

In this subsection, we demonstrate the IRA algorithm on an autonomous underwater vehicle (AUV) depth navigation scenario[4].

Problem Setting

We consider an AUV that conducts scientific observations of the sea floor. Approaching to the sea floor improves the quality of observation, but increases the risk of crashing into the sea floor. Hence, we frame the problem as a minimization of the average altitude from the sea floor, while limiting the probability of crash. The plant model is taken from a Dorado-class AUV [52], which is developed and operated by Monterey Bay Aquarium Research Institute (Figure 11-33). We use the real bathymetry data (i.e., underwater landscape) of Monterey Bay. We discretize the plant model with the time interval $\Delta T = 5$. The deterministic planner that solves the subproblem (Problem 5 in Section 4.1.2) has been demonstrated in the actual AUV mission. The AUVs horizontal speed is assumed to be constant at 3.0 knots, so only the vertical position needs to be planned.

The AUV's plant model has six-dimensional state vector and takes one-dimensional control input, which corresponds to the elevator angle. A Gaussian-distributed disturbance w with the standard deviation $\sigma = 10$ [m] acts only on the third component of the state vector, which refers to the depth of the vehicle. The AUVs elevator angle and pitch rate are deterministically constrained.

[4] We acknowledge Monterey Bay Aquarium Research Institute for providing us with the plant model and the bathymetry data.

Figure 11-33: A Dorado-class autonomous underwater vehicle. Image courtesy of Monterey Bay Aquarium Research Institute.

The depth of the AUV is required to be less than the seafloor depth for the entire mission ($1 \leq t \leq 20$) with probability $\Delta = 0.05$. The objective is to minimize the average of AUVs nominal altitude above the floor.

The simulation is conducted on a machine with a Pentium 4 processor clocked at 2.80 GHz and 1.00 GB of RAM.

Compared Algorithms

We compare the performance of the following three algorithms:

(a) Ellipsoidal relaxation approach [97],

(b) Iterative Risk Allocation, and

(c) Particle Control (20 particles) [16]

Result

The three algorithms are run on 50 cases with different segments of the Monterey Bay sea floor. Figure 11-34 shows a typical result of the three algorithms with $\delta = 0.05$. Ellipsoidal relaxation yields a large safety margin, which touches the nominal path (i.e. constraint is active) only at a few points, just as Figure 4-1-(a). This is because ellipsoidal relaxation uniformly allocates risk to each step. On the other hand, the IRA algorithm generates a nominal path that touches the safety margin at most points, just as Figure 4-1-(b). This

implies that risk is allocated efficiently such that a large portion of risk is allocated to the critical points, such as the top of the seamount.

Figure 11-35 shows the optimal risk allocation computed by IRA algorithm on the same case as Figure 11-34 with $\delta = 0.05$. A large portion of the risk is allocated to the time steps when AUV go above the sea mountain, just as taking a large risk at the corner in the race car example.

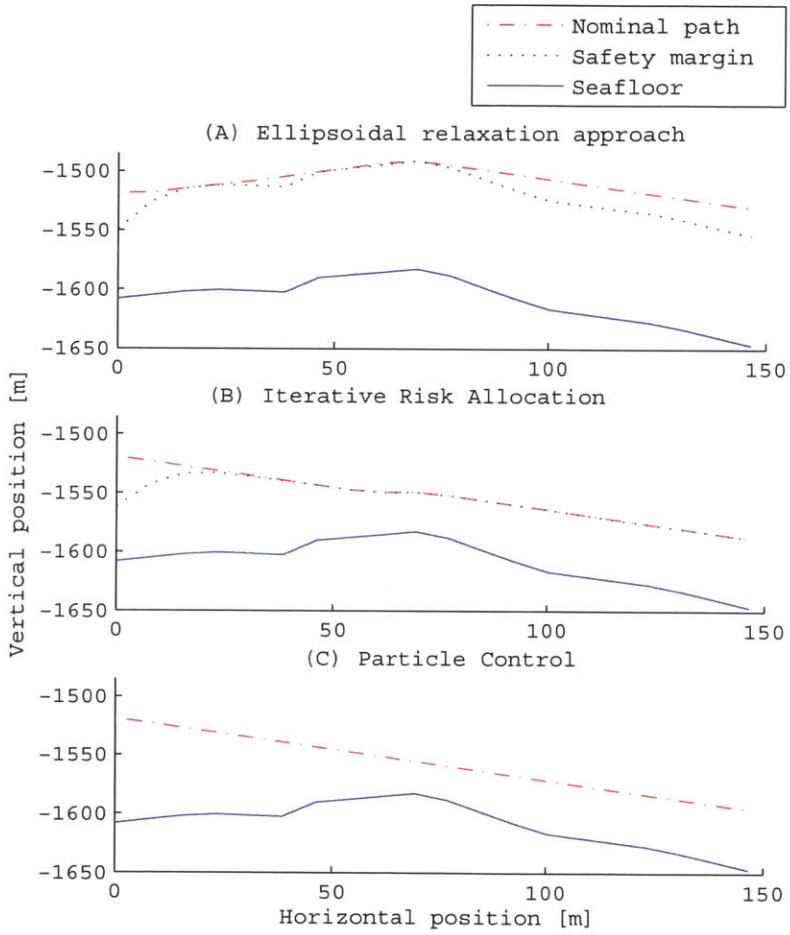


Figure 11-34: Nominal path of AUV and safety margin planned by three algorithms. Safety margin is not shown in (c) since Particle Control does not explicitly compute it.

The performance of the three algorithms is compared in Table 11.9. Values in the table are the average of 50 cases. The resulting probability of failure P_{Fail} is evaluated by Monte Carlo simulation with 100,000 samples. The plan generated by the ellipsoidal re-

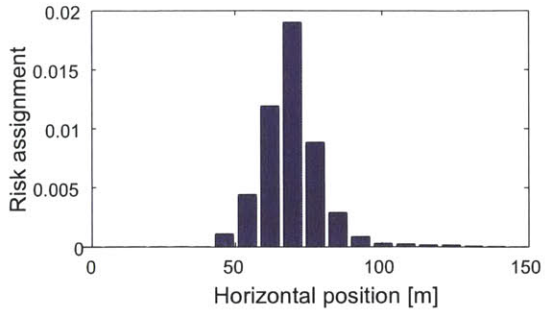


Figure 11-35: Risk allocation of Figure 11-34 (b)

Table 11.9: Performance comparison on the AUV depth planning problem with chance constraint $P_{Fail} \leq 0.05$. Values are the average of 50 runs on different segments of the Monterey Bay sea floor. Planning horizon is 20 steps (100 seconds). The names of the algorithms are abbreviated as follows - ER: Ellipsoidal relaxation approach, IRA: Iterative Risk Allocation, PC: Particle Control

Algorithm	ER	**IRA**	PC
Risk bound Δ	0.05		
Probability of failure P_{Fail}	$< 10^{-5}$	0.023	0.297
Cost J	88.6	64.1	56.2
Computation time (s)	0.093	1.36	915.6

laxation approach (ER) results in nearly zero probability of failure although the bound is $P_{Fail} \leq 0.05$, which shows its strong conservatism. The IRA algorithm (IRA) is also conservative, but much less so than ER. On the other hand, the probability of failure of Particle Control (PC) is higher than the bound, which means violation of the chance constraint. This is because Particle Control is a sample based stochastic algorithm that does not have guarantee that the chance constraint is strictly satisfied.

The ellipsoidal relaxation approach fail to find a solution in one case out of 50, while IRA find solutions in all 50 cases. This is because the strong conservatism of the ellipsoidal relaxation(i.e. large safety margin) makes the optimization problem infeasible.

The value of objective function J is a measure of optimality. Note that this problem is a minimization, hence smaller J means better performance. The true optimal value of J lies between (b) IRA and (c) Particle Control, since the former is suboptimal and the latter is "overoptimal" in the sense that it does not satisfy the chance constraint. The suboptimality

of the IRA is 14% at most, while the ellipsoidal relaxation has 57% suboptimality at most.

The computation time of Particle Control is longer than the planning horizon (100 sec). Although IRA is slower than the ellipsoidal relaxation approach, it is approximately one thousand times faster than Particle Control.

11.3.2 Space Rendezvous

In this section we consider an autonomous space rendezvous scenario of the H-II Transfer Vehicle (HTV) [105], shown in Figure 11-36, as our subject. HTV is an unmanned cargo spacecraft developed by the Japanese Aerospace Exploration Agency (JAXA)[5], which is used to resupply the International Space Station (ISS). Collision of the vehicle with the ISS may result in a fatal disaster, even if the collision speed is low. For example, in August 1994, the Russian unmanned resupply vehicle Progress M-34 collided with the Mir space station in a failed attempt to automated rendezvous and docking. As a result, one of the modules of Mir was permanently depressurized. In order to avoid such an accident, HTV is required to follow a specified safety sequence during the automated rendezvous, as described in the following subsection.

HTV Rendezvous Sequence

In HTV's autonomous rendezvous mission, the final approach phase starts from the Approach Initiation (AI) point, which is located 5 km behind the ISS, as shown in Figure 11-37. First, HTV moves to R-bar Initiation (RI) point, which is located 500 m below the ISS, guided by the relative GPS navigation. At the RI point, HTV switches the navigation mode to Rendezvous Sensor (RVS) Navigation. In RVS Navigation, HTV measures the distance to ISS precisely by beaming the laser to the reflector placed on the nadir (earth-facing) side of the ISS. Then, HTV proceeds to the Hold Point (HP), located 300 m below the ISS. HTV is required to hold at HP in order to perform a 180-degree yaw-around maneuver. The new orientation of HTV allows the vehicle to abort the rendezvous quickly in case of emergency. After the yaw-around maneuver, HTV resumes the approach, and holds

[5] We note that this work is not supported by, nor conducted in collaboration with, the Japanese Aerospace Exploration Agency

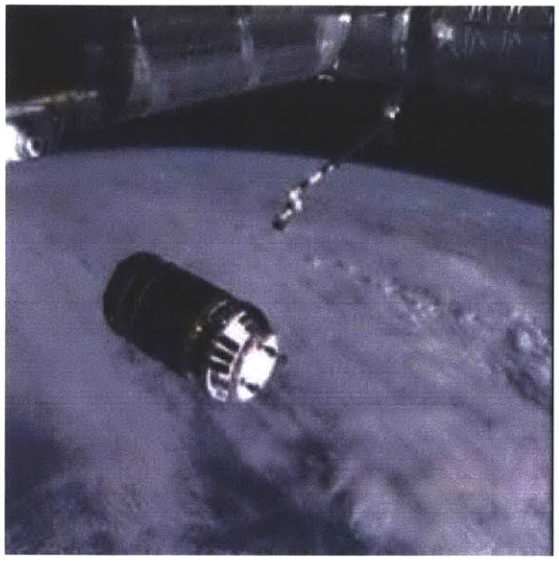


Figure 11-36: H-II Transfer Vehicle (HTV), a Japanese unmanned cargo vehicle, conducts autonomous rendezvous with the International Space Station. Image courtesy of NASA.

again at the Parking Point (PP), which is 30 m below the ISS. Finally, HTV approaches at a distance of 10 meters from the ISS, and stops within the Capture Box (CB) of the ISS's robotic arm. The robotic arm then grabs HTV and docks it to the ISS. This rendezvous sequence is described in detail in [46].

The rendezvous sequence described above is represented by the CCQSP shown in Figure 11-38. In addition to the temporally extended goals specified in the actual rendezvous sequence, we specify temporal constraints and chance constraints in the simulation, as shown in Figure 11-38. We require HTV to hold at each intermediate goals for at least 240 seconds. The transition between the goals must take at least 600 seconds, in order to make sure that the vehicle moves slowly enough. The entire mission must be completed within 4800 seconds (1 hour 20 minutes). We require HTV to stay within the Safe Zone, a conic area below the ISS, during the RVS navigation phase with 99.5% probability, since otherwise the laser may not be reflected back to HTV properly. We assume that the goals are square regions, with 10 m sides for RI and HP, 2 m sides for PP, and 1 m sides for CB. Finally, we require that HTV achieves all the temporally extended goals with 99.5% certainty.

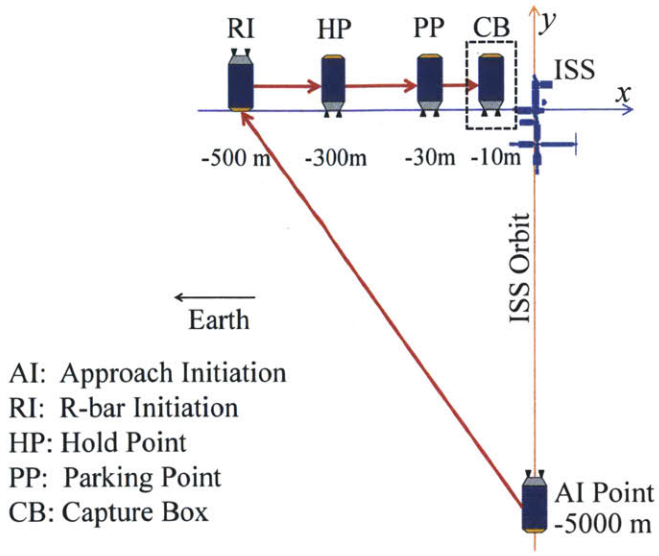


Figure 11-37: HTV's final approach sequence. The figure is created based on the information in [46].

Orbital Dynamics

The rendezvous can be considered as a two-body problem, where a chaser spacecraft (e.g., HTV) moves in relation to a target spacecraft (e.g., ISS), which is in a circular orbit. In such a problem, it is convenient to describe the motion of the chaser spacecraft using a rotating frame that is fixed to the target space craft, known as a Hill coordinate frame [84]. As shown in Figure 11-37, we set the x-axis pointing away from the center of the earth and the y-axis along the orbital velocity of the target spacecraft. Since HTV's path is within the x-y plane, we don't consider the z-axis.

It is known that the relative motion of the chase spacecraft in the Hill coordinate frame is described by the following the Clohessy-Wiltshire (CW) equation [95]:

$$\begin{aligned} \ddot{x} &= 2\omega\dot{y} + 3\omega^2 x + F_x \\ \ddot{y} &= 2\omega\dot{x} + F_y \end{aligned}$$

where ω is the angular speed of the target spacecraft's orbit, and F_x and F_y are the force per unit mass, or the acceleration in x and y directions. The first terms on the right-hand sides represent the Coriolis force.

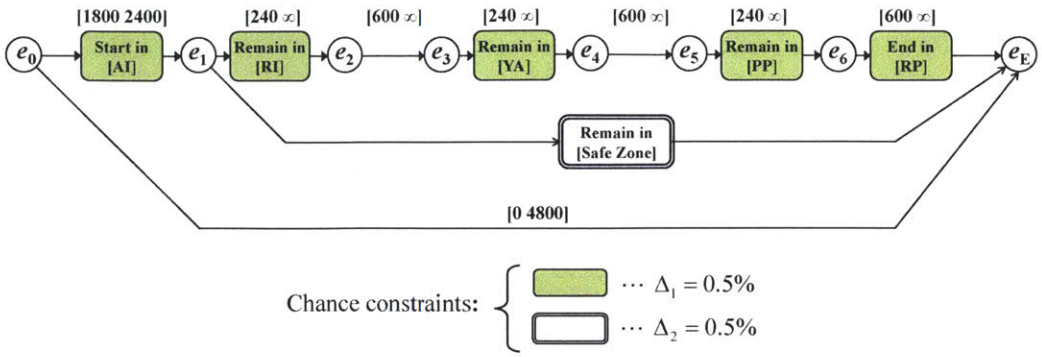


Figure 11-38: A CCQSP representation of the HTV's final approach sequence. We assume the same temporally extended goals as the ones used for actual flight missions. The temporal constraints and the chance constraints are added by the authors.

An object that follows the CW equation moves in an unintuitive manner. Its unforced motion is not in a straight line due to the Coriolis effect; in general, an object cannot stay at the same position without external force. For example, Figure 11-39 shows the fuel-optimal path to visit two waypoints, A and B, and come back to the start. As can be seen in the figure, the optimal path is not typically a straight line. The virtue of p-Sulu is that it can handle such irregular dynamic systems in the same way as regular systems, just by setting the $\boldsymbol{A}$ and $\boldsymbol{B}$ matrices of our plant model (2.3) appropriately.

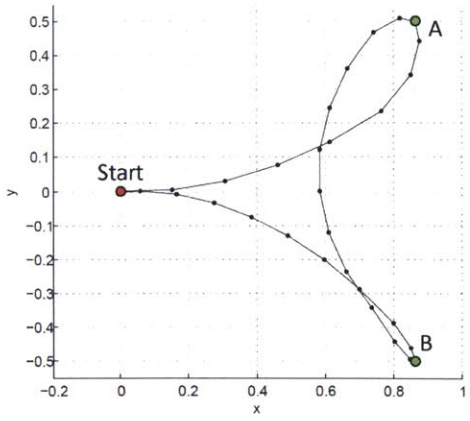


Figure 11-39: A typical motion of spacecraft in the Hill coordinate frame. The solid line is the fuel optimal path to visits A and B and returns to the Start in 30 minutes. Note that the optimal path is not a straight line in the Hill coordinate frame.

The state vector is consists of positions and velocity in the $x-y$ plane:

$$\boldsymbol{x} = [x\ y\ v_x\ v_y]^T$$

We obtain the discrete-time CW equation using the impulse-invariant discretization:

$$\boldsymbol{x}_{k+1} = A\boldsymbol{x}_k + B\boldsymbol{u}_k,$$

where

$$A = \begin{bmatrix} 4-3\cos(\omega\Delta T) & 0 & \frac{\sin(\omega\Delta T)}{\omega} & \frac{2\{1-\cos(\omega\Delta T)\}}{\omega} \\ -6\{\omega\Delta T - \sin(\omega\Delta T)\} & 1 & \frac{-2\{1-\cos(\omega\Delta T)\}}{\omega} & \frac{4\sin(\omega\Delta T)}{\omega} - 3\Delta T \\ 3\omega\sin(\omega\Delta T) & 0 & \cos(\omega\Delta T) & 2\sin(\omega\Delta T) \\ -6\omega\{1-\cos(\omega\Delta T)\} & 0 & -2\sin(\omega\Delta T) & 4\cos(\omega\Delta T) - 3 \end{bmatrix}$$

$$B = \begin{bmatrix} \frac{\sin(\omega\Delta T)}{\omega} & \frac{2\{1-\cos(\omega\Delta T)\}}{\omega} \\ \frac{-2\{1-\cos(\omega\Delta T)\}}{\omega} & \frac{4\sin(\omega\Delta T)}{\omega} - 3\Delta T \\ \cos(\omega\Delta T) & 2\sin(\omega\Delta T) \\ -2\sin(\omega\Delta T) & 4\cos(\omega\Delta T) - 3 \end{bmatrix}$$

We use the ISS's orbital angular speed, $\omega = 0.001164$ rad/sec, at which the station goes around the Earth in 90 minutes. We choose the interval $\Delta T = 120$ seconds. The number of time steps N is set to 40. Hence, the entire plan is 4800 seconds (1 hour and 20 minutes). In the discretization, we assumed impulse inputs as follows:

$$\begin{bmatrix} F_x \\ F_y \end{bmatrix} = \sum_{k=0}^{N-1} \delta(t - \Delta T \cdot k)\boldsymbol{u}_k,$$

where $\delta(\cdot)$ is the Dirac delta function. Such an assumption is justified because the thrusters of the Reaction Control System (RCS) of spacecraft, which are used for the final approach maneuver, operate for a very short duration ($0.01 - 5.0$ seconds) at each burn [104].

We consider stochastic uncertainty w, added to the discrete-time dynamic equation:

$$\boldsymbol{x}_{k+1} = A\boldsymbol{x}_k + B\boldsymbol{u}_k + w.$$

Such an assumption of additive uncertainty is commonly used in past research on autonomous rendezvous and formation flight in space, such as [87, 90, 26]. We assume that w has a zero-mean Gaussian distribution, with the following covariance matrix:

$$\Sigma_w = \begin{pmatrix} 10^{-6} & 0 & 0 & 0 \\ 0 & 10^{-6} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Objective Function

We employ an objective function J that requires for p-Sulu to minimize the fuel consumption. It follows from the Tsiolkovsky rocket equation that the fuel consumption of spacecraft is proportional to the total change in velocity, called Delta-V or ΔV [104]. The total fuel consumption is the summation of the fuel consumption of reaction jets in x and y directions for all time steps. Hence our objective function is described as follows:

$$\begin{aligned} J(\boldsymbol{u}_{0:N}) &= \Delta V_x + \Delta V_y \\ &= \int_0^{(N-1)\Delta T} |F_x| + |F_y| dt \\ &= \sum_{k=0}^{k=N-1} \left| \int_0^{(N-1)\Delta T} \delta(t - \Delta T \cdot k)\boldsymbol{u}_{x,k} dt \right| + \left| \int_0^{(N-1)\Delta T} \delta(t - \Delta T \cdot k)\boldsymbol{u}_{y,k} dt \right| \\ &= \sum_{k=0}^{k=N-1} |\boldsymbol{u}_{x,k}| + |\boldsymbol{u}_{y,k}|. \end{aligned}$$

Simulation Result

Figure 11-40 shows the planning result of p-Sulu FH. We compare the result with Sulu, as well as a nominal planning approach. In the nominal planning approach, we assume that HTV moves from AI to RI using a two-impulse transition (called "CW guidance law"),

which is commonly used in space rendezvous [70, 95]. From RI to CB, it follows a predetermined path that goes through the center of the Safe Zone, as shown in Figure 11-40-(b), with a constant speed.

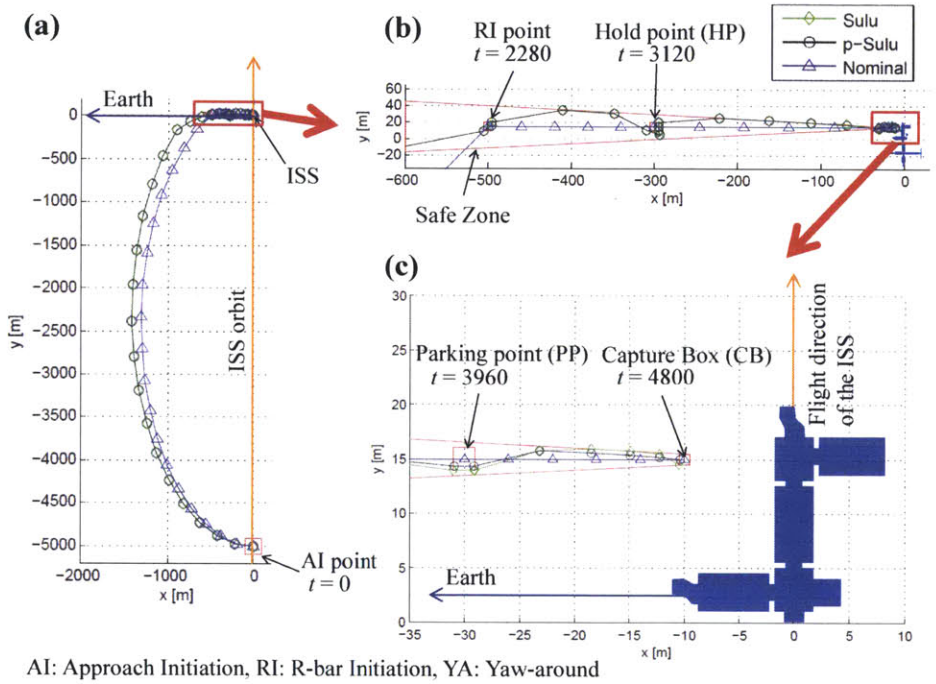


Figure 11-40: Planning results of Sulu, p-Sulu FH, and a nominal planning approach. The input CCQSP is shown in Figure 11-38.

As shown in Figure 11-40, the optimal paths generated by p-Sulu FH and Sulu are not straight. Such curved paths exploit the Coriolis effect to minimize fuel consumption.

Table 11.10 compares the performance of the three planning approaches. The two rows regarding the probabilities of failure correspond to the two chance constraints specified in the CCQSP, shown in Figure 11-38. The probabilities are evaluated by Monte Carlo simulations with one million samples.

As expected, the probabilities of failure of the path generated by p-Sulu FH are less than the risk bounds specified by the CCQSP, shown in Figure 11-38. On the other hand, Sulu's path results in almost 100% probability of failure. This is because Sulu minimizes the fuel consumption without considering uncertainty. The resulting path pushes against the boundaries of feasible regions, as is evident in Figure 11-40-(c). Also note that, although

p-Sulu FH significantly reduces the probability of constraint violation compared with Sulu, its cost (Delta V) is higher than Sulu only by 0.2%. p-Sulu FH results in a significantly smaller cost (Delta V) than the nominal planning approach. The 1.42 m/sec reduction in Delta V is equivalent to an 11.9 kg saving of fuel, assuming the $16,500$ kg mass of the vehicle and the 200 sec specific impulse (I_{SP}) of the thrusters. Although p-Sulu FH takes longer to compute the plan than the other two approaches, the 11.4 second computation time is negligible compared with the 1 hour and 20 minute plan duration.

Table 11.10: Performance comparison of Sulu, p-Sulu FH, and the nominal approach on the HTV rendezvous scenario.

Algorithm		Sulu	**p-Sulu FH**	Nominal
c_1 (Navigation)	Risk bound Δ_1	0.005		
	Probability of failure $P_{fail,1}$	0.92	0.0024	$< 10^{-6}$
c_2 (Goals)	Risk bound Δ_2	0.005		
	Probability of failure $P_{fail,2}$	1.0	0.0029	$< 10^{-6}$
Cost function value (Delta V) $J^\star$ (m/s)		7.30	7.32	8.73
Computation time (s)		3.9	11.4	0.09

11.3.3 Space Formation Flight

We next consider flying multiple space crafts in formation using dp-Sulu. Space formation flight has important applications ranging from astronomical interferometry to military surveillance. Guidance and control for space formation flight is an area that has been actively studied for decades. A comprehensive review of this field is provided by [83] and [82]. An import challenge is to precisely maintain specified formulations while avoiding collisions. Minimizing fuel consumption is another important challenge since it is very difficult to refuel space vehicles.

Scenario

Figure 11-41 shows the environment of the scenario and Figure 11-42 shows the plan. Initially the three satellites are put on the parking orbits, in which the satellites can maintain their relative positions without consuming fuel. The parking orbits are on the reference

orbit, which is depicted as the vertical white line in Figure 11-41. The satellites start from the parking orbits and form a triangular formation, specified by the three locations labeled as 'Formation 1', 'Formation 2', and 'Formation 3' in Figure 11-41. During the formation flight the three satellites exchange their positions in order to equalize fuel consumptions. Satellites are required to stay within each position for at least 10 minutes and at most 12 minutes. Transition between positions must take 20 to 22 minutes. After one rotation, the satellites go back to their parking orbits.

Simulation Settings

We use the CW equation-based plant model presented in Section 11.3.2. For collision avoidance, we require that any satellite does not approach within 100 meters of another satellite. The chance constraint c_4 in Figure 11-42 limits such a risk to be below 0.1%. The central module of dp-Sulu are run on a machine with a quad-core Intel Core i7 CPU clocked at 2.67 GHz and with 8 GB of RAM, while the distributed component (i.e., p-Sulu) of the three satellites are run on three machines: the first one with a quad-core Intel Xeon CPU clocked at 2.40 GHz and with 16 GB of RAM, the second one with a dual-core Intel Xeon CPU clocked at 1.69 GHz and with 512 MB of RAM, and the third one with a quad-core Intel Core i7 CPU clocked at 1.60 GHz and with 8 GB of RAM.

Simulation Results

Figure 11-41 shows the paths of the three satellites. Three satellites successfully completed a formation flight, within the specified temporal bounds. During the formation flight, there are several times when two satellites approach so close that the probability of collision exceeds the risk bound, 0.1%. For example, in Figure 11-43-(a), Satellites 1 and 2 would be within 100 meters from each other. The paths of the two satellites are immediately adjusted by Bi-stage Robust Collision Avoidance, as shown in Figure 11-43-(b). As a result, probability of collision is reduced below below the given risk bound. Note that the probability of collision shown in Figure 11-43-(a) exceeds 1. This is because dp-Sulu actually evaluates an *upper bound* of the probability using risk allocation; the upper bound is obtained by summing up the probabilities of collision at all times step within the current

horizon. It follows from Lemma 1 in Section 4.1.2 that if the summation is below the risk bound, then the original chance constraint is guaranteed to be satisfied.

11.4 Conclusion

This chapter presented simulation results of the planners and executives developed throughout this thesis. We demonstrated the proposed planners and executives in three ways. First, we evaluated their performances by extensive simulations with randomized conditions on benchmark problems. These simulations demonstrated the risk-sensitive planning capability to operate uncertain systems within user-specified risk bounds. Second, we deployed the proposed planners and executives on the personal transportation system (PTS). The PTS simulations demonstrated that our planners and executives can solve real-world scale problems. Third and finally, we applied our planners and executives to the control of underwater and space vehicles, both of which have significantly different plant models than the PTS. These simulation results showed that our approach is applicable to a wide range of real-world systems.

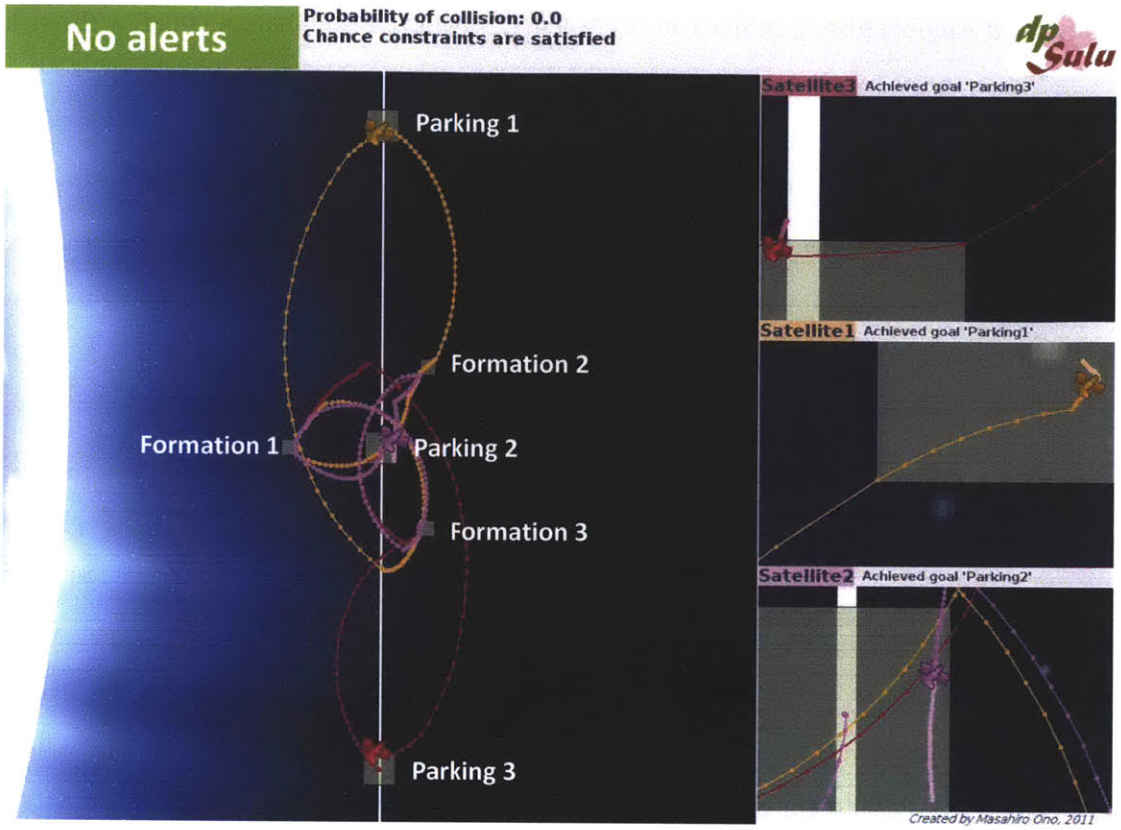


Figure 11-41: A space formation flight scenario with three satellites, on which dp-Sulu is deployed. The Earth illustrated on the right is not to scale.

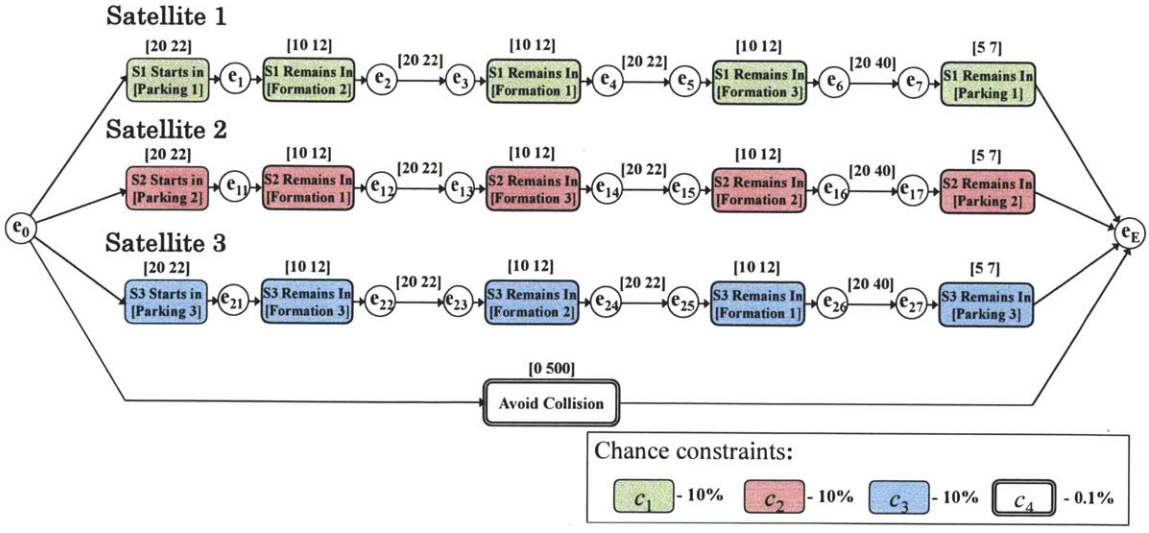


Figure 11-42: The CCQSP for the space formation flight scenario shown in Figure 11-41. The time is displayed in minutes.

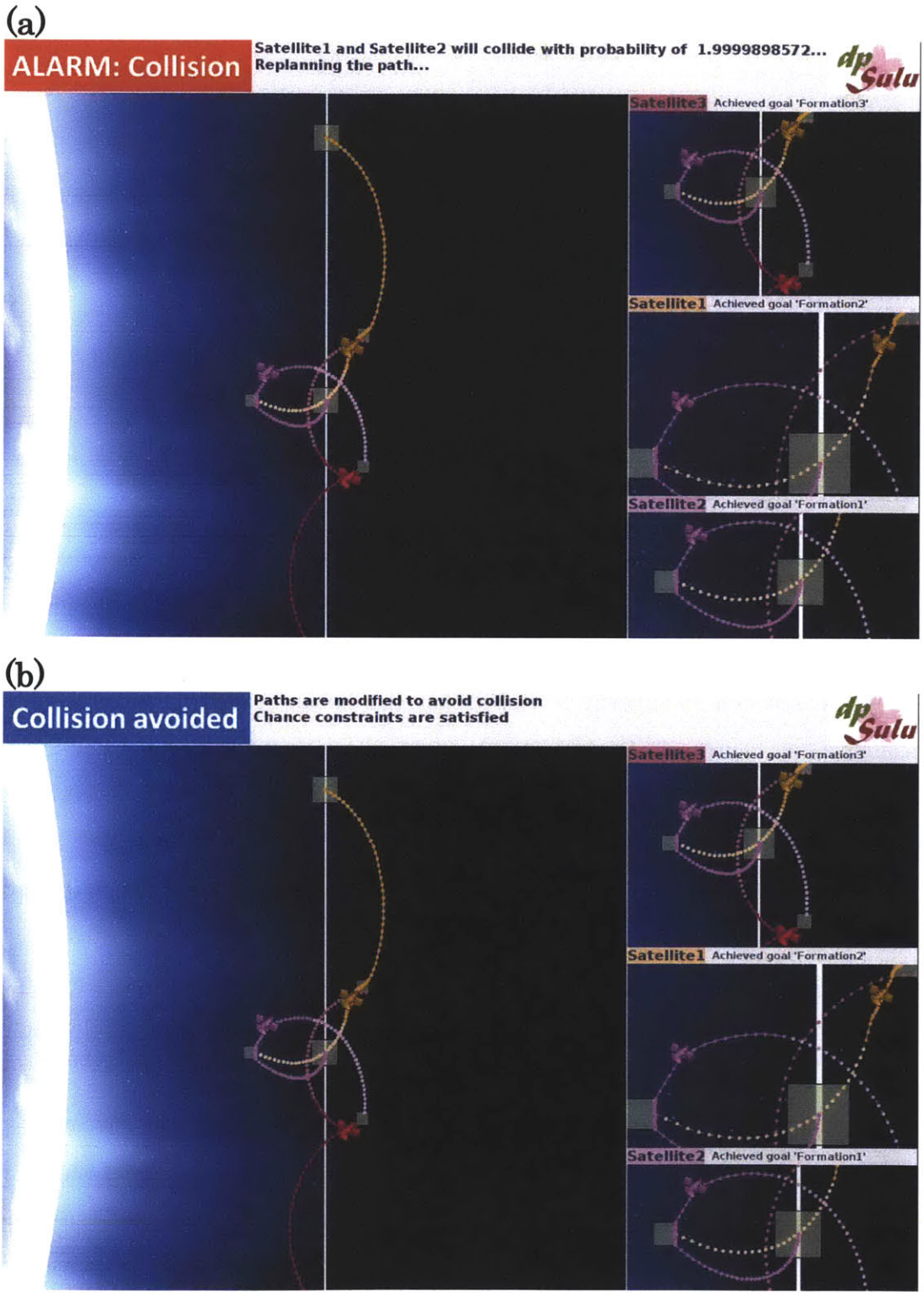


Figure 11-43: Collision avoidance performed by dp-Sulu during the formation flight. (a) The central module finds that Satellites 1 and 2 will collide with a probability above the risk bound. (b) dp-Sulu adjusts the paths to avoid collision.

Chapter 12

Thesis Conclusions

12.1 Thesis Contributions

We have developed risk-sensitive model-based plan executives, p-Sulu and dp-Sulu, which operate within user-specified risk bounds. p-Sulu and dp-Sulu optimize a continuous control sequence and a discrete schedule, given as inputs a continuous stochastic plant model, an objective function, and a newly developed plan representation called a chance-constrained qualitative state plan (CCQSP). A CCQSP involves temporally extended goals, simple temporal constraints, and chance constraints, which specify the user's acceptable levels of risk on subsets of the plan.

p-Sulu and dp-Sulu have been enabled by our new concept of risk allocation (Chapter 4). The risk allocation approach achieves tractability by allocating the specified risk to individual constraints and mapping the result into an equivalent deterministic constrained optimization problem. Risk allocation also enables a distributed plan execution for multi-agent systems by distributing the risk among agents.

The centralized CCQSP executive, p-Sulu, has been built upon the risk allocation approach in four spirals. In the first spiral (Chapter 5) we have solved a limited version of the CCQSP planning problem: a problem with only a convex state space, a fixed schedule, and a full planning horizon. Such a problem has been efficiently solved by the convex risk allocation (CRA) problem. In the second spiral (Chapter 6), we have developed the non-convex iterative risk allocation (NIRA) algorithm, in order to remove the above convexity

requirement. In the third spiral (Chapter 7), we have extended NIRA to handle *flexible* schedule, and developed a full horizon CCQSP planner, p-Sulu FH. Finally, in the fourth spiral (Chapter 8), we have developed a *receding horizon* CCQSP executive, p-Sulu, which has enabled an online execution of CCQSP in a centralized manner.

Next, we have developed the distributed CCQSP executive, dp-Sulu, in two spirals. In the first spiral of this part (Chapter 9), we have developed the Market-based Iterative Risk Allocation (MIRA) algorithm, which can solve a multi-agent CCQSP planning problem in a distributed manner with a convex state space, a fixed schedule, and a full planning horizon. Then, in the final spiral (Chapter 10), we have created the distributed receding-horizon plan executive, dp-Sulu, which can execute CCQSP in real time with a nonconvex state space and a flexible schedule.

The capabilities of the plan executives have been demonstrated by extensive simulations (Chapter 11). We have deployed the executives on aerial, underwater, and space vehicles, as well as various benchmark problems. Among the total of 1,120 simulations with various settings, none of them violated the chance constraints. The simulation results have demonstrated the executives' capability of robustly achieving the temporally extended goals while respecting the user-specified risk bounds and continuously adapting to environmental changes. Furthermore, our algorithms have been successfully deployed on three different plants: personal aerial vehicles (PAVs), autonomous underwater vehicles (AUVs), and autonomous cargo spacecraft. These results demonstrate the generality of our approach.

12.2 Future Work

Receding-horizon chance-constrained plan execution with resolvability

Recall that our proposed plan executives solve CCQSP planning problem repeatedly with a receding planning horizon. The solution at each planning horizon is guaranteed to satisfy chance constraints, if one exists. However, the existence of a solution is not guaranteed. A plan executive will be more robust if it can also guarantee the existence of a solution during the execution. In other words, the executive should be guaranteed to avoid a situation where

it cannot find any feasible control sequence in the middle of execution.

Such a guarantee can be obtained by making the planning problems *resolvable* or *recursively feasible*. The issue of resolvability has been extensively studied in the robust model predictive control (RMPC) community (e.g., [1, 2, 28, 30, 55, 59, 86]). However, most RMPC algorithms with a resolvability guarantee assume set-bounded uncertainties, meaning that the level of uncertainty at each time step is upper-bounded. To the best of our knowledge, there is no MPC algorithm that can guarantee resolvability with the more general assumption of unbounded, stochastic uncertainty. Therefore, it is our future work to develop a chance-constrained MPC algorithm with resolvability, and leverage that technology for CCQSP execution with guaranteed resolvability.

Extension of risk allocation approach to other planning domains

In this thesis we focused on a planning problem with continuous state variables, formulated as constrained optimization problems. However, the concept of risk allocation is more general than this specific form of planning problems. In particular, when Assumptions 4-1 to 4-3 in Section 4.2 are satisfied, the iterative risk allocation (IRA) algorithm proposed in Section 4.2 can be implemented on top of existing deterministic optimizers and give them a capability to solve corresponding chance-constrained optimization problems. Therefore, application of the risk allocation approach and the IRA algorithm to existing planning approaches is a promising direction of future research.

Deployment on hardware

In this thesis the capabilities of the proposed planners and executives are demonstrated by simulations. They need to be tested on real-world hardware in order to be more credible. Our algorithms are model-based, meaning that we assume that we have a perfect knowledge of plant models, including the probability distributions of uncertainties. Therefore, we need to combine our approach with model identification and/or learning techniques in order to obtain the plant model. A branch of robust model predictive control (RMPC) considers uncertainty in plant parameters. This approach would also be useful in order to deploy our algorithms on real-world hardware.

Bibliography

[1] B. Açikmeşe and J. M. Carson III. A nonlinear model predictive control algorithm with proven robustness and resolvability. In *Proceedings of American Control Conference*, 2006.

[2] B. Açikmeşe, J. M. Carson III, and D. S. Bayard. A robust model predictive control algorithm for incrementally conic uncertain/nonlinear systems. *International Journal of Robust and Nonlinear Control*, 21(5):563–590, 2011.

[3] Aircraft Owners and Pilots Association Air Safety Foundation. 2005 Joseph T. Nall Report - accident trands and factors for 2004, 2005.

[4] E. Altman. *Constrained Markov Decision Processes*. Chapman and Hall/CRC, 1999.

[5] R. Alur, T. Feder, and T. A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43, 1996.

[6] K. E. Atkinson. *An Introduction to Numerical Analysis, Second Edition*. John Wiley & Sons, 1989.

[7] F. Bacchus and F. Kabanza. Planning for temporally extended goals. *Annals of Mathematics and Artificial Intelligence*, (22):5–27, 1998.

[8] L. Baird. Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the Twelfth International Conference on Machine Learning*, 1995.

[9] E. Balas. Disjunctive programming. *Annals of Discrete Mathematics*, 1979.

[10] A. G. Banerjee and N. Roy. Learning Solutions of Similar Linear Programming Problems using Boosting Trees. CSAIL Technical Report MIT-CSAIL-TR-2010-045, Massachusetts Institute of Technology, 2010. Available at `http://dspace.mit.edu/handle/1721.1/58609`.

[11] N. M. Barr. *Wind Models for Flight Simulator Certification of Landing and Approach Guidance and Control Systems*. University of Michigan Library, 1974.

[12] K. Bell, A. I. Coles, M. Fox, D. Long, and A. J. Smith. The application of planning to power substation voltage control. In *Proceedings of ICAPS Workshop on Scheduling and Planning Applications*, 2008.

[13] D. Bertsekas and J. Tsitsiklis. Neuro-dynamic programming: an overview. In *Decision and Control, 1995., Proceedings of the 34th IEEE Conference on*, volume 1, pages 560 –564 vol.1, dec 1995.

[14] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 1999.

[15] D. P. Bertsekas. *Convex Optimization Theory*. Athena Scientific, 2009.

[16] L. Blackmore. A probabilistic particle control approach to optimal, robust predictive control. In *Proceedings of the AIAA Guidance, Navigation and Control Conference*, 2006.

[17] L. Blackmore. *Robust Execution for Stochastic Hybrid Systems*. PhD thesis, Massachusetts Institute of Technology, 2007.

[18] L. Blackmore, H. Li, and B. C. Williams. A probabilistic approach to optimal robust path planning with obstacles. In *Proceedings of American Control Conference*, 2006.

[19] L. Blackmore and M. Ono. Convex chance constrained predictive control without sampling. In *Proceedings of the AIAA Guidance, Navigation and Control Conference*, 2009.

[20] L. Blackmore, M. Ono, A. Bektassov, and B. C. Williams. A probabilistic particle control approximation of chance constrained stochastic predictive control. *IEEE Transactions on Robotics*, 26(3), 2010.

[21] J. Boyan and A. Moore. Robust value function approximation by working backwards. In *Proceedings of the Workshop on Value Function Approximation, Machine Learning Conference*, July 1995.

[22] J. A. Boyan and M. L. Littman. Exact solutions to time-dependent mdps. In *in Advances in Neural Information Processing Systems*, pages 1026–1032. MIT Press, 2000.

[23] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, mar 2004.

[24] L. Breger and J. P. How. Safe trajectories for autonomous rendezvous of spacecraft. *Journal of Guidance, Control, and Dynamics*, 2008.

[25] G. C. Calafiore and M. C. Campi. The scenario approach to robust control design. *IEEE Transactions on Automatic Control*, 51(5), 2006.

[26] M. E. Campbell and B. Udrea. Collision avoidance in satellite clusters. In *Proceedings of the American Control Conference*, 2002.

[27] A. Charnes and W. W. Cooper. Chance-constrained programming. *Management Science*, 6:73–79, 1959.

[28] H. Chen, C. Scherer, and F. Allgower. A game theoretic approach to nonlinear robust receding horizon control of constrained systems. In *American Control Conference, 1997. Proceedings of the 1997*, volume 5, pages 3073 –3077 vol.5, jun 1997.

[29] S. Chien, B. Cichy, A. Davies, D. Tran, G. Rabideau, R. Castano, R. Sherwood, D. Mandl, S. Frye, S. Shulman, J. Jones, and S. Grosvenor. An autonomous earth-observing sensorweb. *Intelligent Systems, IEEE*, 20(3):16 – 24, may-june 2005.

[30] L. Chisci, J. Rossiter, and G. Zappa. Systems with persistent disturbances: predictive control with restricted constraints. *Automatica*, 37(7):1019 – 1028, 2001.

[31] A. J. Coles, A. I. Coles, M. Fox, and D. Long. Temporal planning in domains with linear processes. In *Twenty-First International Joint Conference on Artificial Intelligence (IJCAI)*. AAAI Press, July 2009.

[32] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. In *Proceedings of the first international conference on Principles of knowledge representation and reasoning*, pages 83–93, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[33] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.

[34] D. Dolgov and E. Durfee. Stationary deterministic policies for constrained mdps with multiple rewards, costs, and discount factors. In *In Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05*, pages 1326–1331, 2005.

[35] W. B. Dunbar. Distributed receding horizon control of dynamically coupled nonlinear systems. *IEEE Transactions on Automatic Control*, 2007.

[36] W. B. Dunbar and R. M. Murray. Distributed receding horizon control for multi-vehicle formation stabilization. *Automatica*, 42(4):549 – 558, 2006.

[37] R. Effinger and B. C. Williams. Optimal temporal planning at reactive time scales via dynamic backtracking branch and bound. Master's thesis, Massachusetts Institute of Technology, 2006.

[38] Z. Feng, R. Dearden, N. Meuleau, and R. Washington. Dynamic programming for structured continuous markov decision problems. In *Proceedings of the Proceedings of the Twentieth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-04)*, pages 154–161, Arlington, Virginia, 2004. AUAI Press.

[39] M. Fox and D. Long. Pddl 2.1 : An extension to pddl for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 2003.

[40] M. Fox and D. Long. Modelling mixed discrete-continuous domains for planning. *Journal of Artificial Intelligence Research*, 27:235–297, 2006.

[41] C. E. Garcia, D. M. Prett, and M. Morari. Model predictive control: Theory and practice - a survey. *Automatica*, 25(3):335 – 348, 1989.

[42] P. Geibel and F. Wysotzki. Risk-sensitive reinforcement learning applied to control under constraints. *Journal of Artificial Intelligence Research*, 24:81–108, 2005.

[43] A. G. Hofmann and B. C. Williams. Robust execution of temporally flexible plans for bipedal walking devices. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS-06)*, 2006.

[44] I. hsiang. Shu. Enabling fast flexible planning through incremental temporal reasoning. Master's thesis, Massachusetts Institute of Technology, 2003.

[45] J. C. Jacobo, D. de Roure, and E. H. Gerding. An agent-based electrical power market. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS): Demo Papers*, 2008.

[46] Japan Aerospace Exploration Agency. HTV-1 mission press kit. Available on-line at `http://www.jaxa.jp/countdown/h2bf1/pdf/presskit_htv_e.pdf`, 2009.

[47] F. Kelly, A. Maulloo, and D. Tan. Rate control in communication networks: Shadow prices, proportional fairness and stability. *Journal of the Operational Research Society*, 49:237–252, 1998.

[48] E. C. Kerrigan. *Robust Constraint Satisfaction: Invariant Sets and Predictive Control*. PhD thesis, University of Cambridge, 2000.

[49] H. Kim, D. Shim, and S. Sastry. Nonlinear model predictive tracking control for rotorcraft-based unmanned aerial vehicles. In *American Control Conference, 2002. Proceedings of the 2002*, volume 5, pages 3576 – 3581 vol.5, 2002.

[50] P. Kim. Model-based planning for coordinated air vehicle missions. Master's thesis, Massachusetts Institute of Technology, 2000.

[51] P. Kim, B. C. Williams, and M. Abramson. Executing reactive, model-based programs through graph-based temporal planning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 487–493, 2001.

[52] W. J. Kirkwood. Development of the dorado mapping vehicle for multibeam, subbottom, and sidescan science missions. *Journal of Field Robotics*, 24(6):487–495, 2007.

[53] S. Knight, G. Rabideau, S. Chien, B. Engelhardt, and R. Sherwood. Casper: space exploration through continuous planning. *Intelligent Systems, IEEE*, 16(5):70 – 75, sep-oct 2001.

[54] M. V. Kothare, V. Balakrishnan, and M. Morari. Robust constrained model predictive control using linear matrix inequalities. *Automatica*, 32(10):1361–1379, October 1996.

[55] M. V. Kothare, V. Balakrishnan, and M. Morari. Robust constrained model predictive control using linear matrix inequalities. *Automatica*, 32(10):1361 – 1379, 1996.

[56] Y. Kuwata. Real-time trajectory design for unmanned aerial vehicles using receding horizon control. Master's thesis, Massachusetts Institute of Technology, 2003.

[57] Y. Kuwata. *Trajectory Planning for Unmanned Vehicles using Robust Receding Horizon Control.* PhD thesis, Massachusetts Institute of Technology, 2006.

[58] Y. Kuwata and J. P. How. Cooperative distributed robust trajectory optimization using receding horizon MILP. *IEEE Transactions on Control Systems Technology*, 19(2):423–431, March 2011.

[59] Y. Kuwata, A. Richards, and J. How. Robust receding horizon control using generalized constraint tightening. *Proceedings of American Control Conference*, 2007.

[60] J. Kvarnstrom and P. Doherty. Talplanner: A temporal logic based forward chaining planner. *Annals of Mathematics and Artificial Intelligence*, (30), 2000.

[61] T. Léauté. Coordinating agile systems through the model-based execution of temporal plans. Master's thesis, Massachusetts Institute of Technology, 2005.

[62] T. Léauté and B. C. Williams. Coordinating agile systems through the model-based execution of temporal plans. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI)*, 2005.

[63] O. Lemon, A. Bracy, A. Gruenstein, and S. Peters. An information state approach in a multi-modal dialogue system for human-robot conversation. *Perspectives on Dialogue in the new Millenium*, pages 229–242, 2003.

[64] O. Lemon, A. Gruenstein, and S. Peters. Collaborative activities and multi-tasking in dialogue systems. *Traitement Automatique des Langues*, 43(2), 2002.

[65] H. Li and B. C. Williams. Generalized conflict learning for hybrid discrete linear optimization. In *Proc. 11th International Conf. on Principles and Practice of Constraint Programming*, 2005.

[66] H. X. Li. *Kongming: A Generative Planner for Hybrid Systems with Temporally Extended Goals*. PhD thesis, Massachusetts Institute of Technology, 2010.

[67] J. Löfberg. *Minimax Approaches to Robust Model Predictive Control*. PhD thesis, Linköping Studies in Science and Technology, 2003.

[68] J. Lofberg. Yalmip : A toolbox for modeling and optimization in MATLAB. In *Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004.

[69] B. D. Luders, M. Kothari, and J. P. How. Chance constrained rrt for probabilistic robustness to environmental uncertainty. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, 2010.

[70] S. Matsumoto, S. Dubowsky, S. Jacobsen, and Y. Ohkami. Fly-by approach and guidance for uncontrolled rotating satellite capture. In *Proceedings of AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2003.

[71] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. M. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 2000.

[72] D. Mcdermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL - the planning domain definition language. In *The AIPS-98 Planning Competition Comitee*, 1998.

[73] R. Murray. Recent research in cooperative control of multivehicle systems. *Journal of Dynamic Systems Measurement and Control*, 129(5):571, 2007.

[74] N. Muscettola, P. Nayak, B. Pell, and B. C. Williams. Remote agent: to boldly go where no AI system has gone before. *Artificial Intelligence*, 103(1-2):5 – 47, 1998.

[75] A. Nemirovski and A. Shapiro. Convex approximations of chance constrained programs. *SIAM Journal on Optimization*, 17:969–996, 2006.

[76] T. Ohno and S. Imai. The 1987 tokyo blackout. In *Proceedings of Power Systems Conference and Exposition*, 2006.

[77] F. Oldewurtel, C. N. Jones, and M. Morari. A tractable approximation of chance constrained stochastic mpc based on affine disturbance feedback. In *Proceedings of Conference on Decision and Control*, 2008.

[78] M. Ono and B. C. Williams. An efficient motion planning algorithm for stochastic dynamic systems with constraints on probability of failure. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-08)*, 2008.

[79] A. Prékopa. The use of discrete moment bounds in probabilistic constrained stochastic programming models. *Annals of Operations Research*, 85:21–38, 1999.

[80] A. Richards and J. How. Decentralized model predictive control of cooperating uavs. In *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, volume 4, pages 4286 – 4291 Vol.4, dec. 2004.

[81] A. Richards, T. Schouwenaars, J. P. How, and E. Feron. Spacecraft trajectory planning with avoidance constraints using mixed-integer linear programming. *AIAA Journal of Guidance, Control, and Dynamics*, 25(4), 2002.

[82] D. Scharf, F. Hadaegh, and S. Ploen. A survey of spacecraft formation flying guidance and control (part 1): guidance. In *American Control Conference, 2003. Proceedings of the 2003*, volume 2, pages 1733 – 1739, jun 2003.

[83] D. Scharf, F. Hadaegh, and S. Ploen. A survey of spacecraft formation flying guidance and control. part ii: control. In *American Control Conference, 2004. Proceedings of the 2004*, volume 4, pages 2976 –2985 vol.4, 30 2004-july 2 2004.

[84] H. Schaub and J. L. Junkins. *Analytical mechanics of space systems*. American Institute of Aeronautics and Astronautics, Inc., 2003.

[85] T. Schouwenaars, B. DeMoor, E. Feron, and J. How. Mixed integer programming for multi-vehicle path planning. In *In European Control Conference 2001*, pages 2603–2608, 2001.

[86] P. O. M. Scokaert and D. Q. Mayne. Minmax feedback model predictive control for constrained linear systems. *IEEE Transactions on Automatic Control*, 43(8), 1998.

[87] J. Shields, S. Sirlin, and M. Wette. Metrology sensor characterization and pointing control for the formation interferometer testbed (fit). In *Proceedings of IEEE Aerospace Conference*, 2002.

[88] D. H. Shim, H. Chung, H. J. Kim, and S. Sastry. Autonomous exploration in unknown urban environments for unmanned aerial vehicles. In *in Proceedings of AIAA Guidance, Navigation, and Control Conference*, 2005.

[89] N. Z. Shor. *Minimization Methods for Non-Differentiable Functions*. Springer-Berlag, 1985.

[90] R. Smith and F. Hadaegh. Distributed estimation, communication and control for deep space formations. *IET Control Theory and Applications*, 2007.

[91] J. L. Stedl. Managing temporal uncertainty under limited communication: A formal model of tight and loose team coordination. Master's thesis, Massachusetts Institute of Technology, 2004.

[92] J. Tuinstra. *Price Dynamics in Equilibrium Models: The Search for Equilibrium and the Emergence of Endogenous Fluctuations*. Kluwer Academic Publishers, 2000.

[93] R. M. Turner and E. H. Turner. A two-level, protocol-based approach to controlling autonomous oceanographic sampling networks. *IEEE Journal of Oceanic Engineering*, 26, 2001.

[94] R. H. Tutuncu, K. C. Toh, and M. J. Todd. Solving semidefinite-quadratic-linear programs using SDPT3. *Mathematical Programming B*, 95:189–217, 2003.

[95] D. A. Vallado. *Fundamentals of Astrodynamics and Applications, Second Edition*. Microcosm Press, 2001.

[96] F. Vallée, J. Lobry, and O. Deblecker. Impact of the wind geographical correlation level for reliability studies. *IEEE Transactions on Power Systems*, 22(4), November 2007.

[97] D. H. van Hessem. *Stochastic inequality constrained closed-loop model predictive control with application to chemical process operation*. PhD thesis, Delft University of Technology, 2004.

[98] T. Vidal. Controllability characterization and checking in contingent temporal constraint networks. In *Proceedings Of Seventh International Conference on Principles of Knowledge*, 2000.

[99] T. Vidal and H. Fargier. Handling contingency in temporal constraint networks: from consistency to controllabilities. *Journal of Experimental and Theoretical Artificial Intelligence*, 11:23–45, 1999.

[100] T. Vidal and M. Ghallab. Dealing with uncertain durations in temporal constraint networks dedicated to planning. In *Proceedings Of 12th European Conference on Artificial Intelligence (ECAI-96)*, pages 48–54, 1996.

[101] H. Voos. Agent-based distributed resource allocation in technical dynamic systems. In *Proceedings of IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications*, 2006.

[102] X. Wang, V. Yadav, and S. N. Balakrishnan. Cooperative uav formation flying with obstacle/collision avoidance. *IEEE Transactions on Control Systems Technology*, 15(4), 2007.

[103] M. P. Wellman. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research*, 1:1–23, 1993.

[104] J. R. Wertz and e. Wiley J. Larson. *Space Mission Analysis and Design (Third Edition)*. Microcosm/Springer, 1999.

[105] K. Yamanaka. Rendezvous strategy of the japanese logistics support vehicle to the international space station. In *Proceedings of the 3rd ESA International Conference*, 1997.

[106] F. Ygge and H. Akkermans. Power load management as a computational market. In *Proceedings of Second International Conference on Multiagent Systems*, 1996.

[107] P. Yu and B. C. Williams. The diagnosis of temporal planning problems. Prepared for *22nd International Conference on Automated Planning and Scheduling (ICAPS-2012)*.