Hybrid Planning in Self-adaptive Systems

Ashutosh Pandey

CMU-ISR-20-100

February 2020

Institute for Software Research
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
David Garlan (Chair)
Jonathan Aldrich
John Dolan
Hausi Müller (University of Victoria, Canada)

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Software Engineering.

This work is supported in part by awards N000141310401 and N000141310171 from the Office of Naval Research (ONR), and FA87501620042 from the Air Force Research Laboratory (AFRL). Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the ONR, AFRL, DARPA or the U.S. Government.

Keywords: self-adaptive systems, formal model, automated planning, machine learning, probabilistic model-checking

Abstract

Planning is one of the fundamental design considerations when building a self-adaptive software system. Planning helps the adaptive system to determine an appropriate course of action at run time that seeks to change the system's behavior in response to faults, changing environments and security threats. Therefore, having an appropriate planner to find a plan is critical to a successful self-adaptation.

For many adaptive systems, an appropriate planner is the one that not only finds a plan quickly, particularly, in urgent circumstances but also the plan provides a near-optimal long-term performance. However, due to the fundamental trade-off between quality and timeliness of planning, today designers often have to compromise between an approach that finds a plan quickly and an approach that is slow but finds a higher-quality plan.

To deal with this trade-off, this thesis proposes a hybrid planning approach for self-adaptive systems that combines *off-the-shelf* deliberative and reactive planners to find a balance between quality and timeliness. The key idea is to use reactive planning to provide a quick (although potentially a sub-optimal) response, but simultaneously invoke deliberative planning to determine quality plans. Once the deliberative plan is ready, it takes over the execution from the reactive plan to provide a higher quality adaptation thereafter.

Such a combination of planners can, in principle, reap the benefits of both worlds: providing plans quickly when the timing is critical, while allowing (nearly) optimal plans to be generated when the system has sufficient time to do so. Moreover, instead of going through the non-trivial process of developing a new algorithm/heuristic, hybrid planning combines off-the-shelf planners; therefore, hybrid planning does not require software engineers to master the complexity of developing new planning algorithms/heuristics.

This thesis demonstrates that, compared to its constituent reactive and deliberative planners, hybrid planning can find a better balance between the timeliness and the quality of planning, thereby improve adaptation effectiveness as measured by a multi-dimensional utility function capturing different dimensions of a system's goal. In the process, the thesis makes contributions to both the theory and the practice of hybrid planning in self-adaptive systems. Specifically, the thesis provides: (a) a formal framework defining the problem of hybrid planning; (b) a practical approach (grounded in the formal model) to apply hybrid planning to self-adaptive systems; (c) informal guidelines and a quantitative approach to help engineers to select an appropriate set of planners to instantiate hybrid planning for a given domain, and (d) evaluation of hybrid planning using two realistic systems to bridge the gap between theory and practice.

To my mother and father for their selfless love and support.

Acknowledgements

I express the deepest gratitude to my advisor Prof. David Garlan for his guidance and support. He helped me to inculcate the skill of thinking about a problem at the right level of abstraction and switching between different abstraction levels as needed; this is perhaps the most important skill needed for complex problem solving. He gave me the freedom to pursue my ideas, fail, and learn from mistakes to eventually develop as an independent researcher. His patience as an advisor is phenomenal – he remained patient when I was not able to make progress, even for months.

I am also thankful to my thesis committee. By asking profound questions, Prof. Jonathan Aldrich helped me to refine the formal aspects of my thesis. The feedback provided by Dr. John Dolan and Prof. Hausi Müller helped to improve the thesis from the perspective of artificial intelligence and software engineering, respectively.

The credit of this thesis also goes to my mentors Dr. Bradley Schmerl and Prof. Javier Cámara. Bradley is a deep thinker who always surprises me with his ability to identify subtle gaps in an argument. His feedback helped to strengthen the thesis, but more importantly, my thought process improved by working with him. Prof. Javier Cámara is not only my mentor but also a great friend. Whenever I requested to meet him seeking his feedback on my research, despite being busy, he provided a meeting slot even on short notice. He used to patiently listen to my ideas and ask clarifying questions that helped me refine the ideas.

I am indebted to my bright colleagues and close friends – Dr. Gabriel Moreno and Dr. Ivan Ruchkin. I was fortunate to have them ahead of me in the Ph.D. program; just by observing them conducting their thesis research, I became a better researcher. For the thesis evaluation, rather than building the systems from scratch, I used the systems build by Gabriel; he used to patiently answer my questions related to the systems that enabled me to make progress rather quickly. Ivan touched my life in various ways. All these years, Ivan has been the biggest critic of my research; his honest (and sometimes harsh) feedback played a significant role in improving the overall quality of the thesis. Moreover, he helped me with technical aspects such as formalizing the hybrid planning problem, reviewing the design and analyzing the data of experiments, and co-authoring papers. In addition, my philosophical discussions with Ivan had been a great source of learning and intellectual flirtation. Ivan made me realize how all human beliefs are susceptible to change; therefore, it's worth being flexible with the beliefs. With this realization, now I am more open to new ideas/cultures/experiences and do not feel obligated to defend my beliefs. However, the biggest contribution of Ivan is taking me to a 10-day *Vipassana* meditation retreat, which changed the way I see and experience life. I also acknowledge my colleagues Vishal Dwivedi, Selva Samuel, and Cody Kinneer for their support during these years.

My special gratitude goes to my wife Varshika. This journey would not have been possible without her support and sacrifices. All these years, she has been a silent contributor to my graduation. While I was busy with my work, she took care of me and kids alone without complaining. Honestly, no words are sufficient to express my gratitude to her. I also want to mention my kids – Anushka and Saatvik for providing lovely stress-reducing moments during the challenging times as a graduate student.

Finally, there are several other friends and family members who made this thesis possible through their encouragement and support. My apologies for not being able to mention their names.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A typical control loop in many self-adaptive software systems has four fundamental computational components: Monitoring-Analysis-Planning-Execution (MAPE) [61]. Based on information collected by the monitoring component, if the analysis component decides that the system needs to adapt to meet its goals, then the planning component determines an adaptation plan, which is executed by the execution component.

The planning component determines an adaptation plan based on various factors such as the current state of a self-adaptive system and its operating environment, a set of possible adaptation actions, and the adaptation goal; such factors together constitute a planning problem. In other words, the planning component takes a planning problem as an input and returns an adaptation plan.

For the planning component, researchers in the self-adaptive community have proposed various planning approaches to determine plans.[1] Frameworks such as Rainbow [27] solve new problems based on solutions to similar problems from the past. When adaptation is needed, Rainbow chooses an adaptation strategy (i.e., a plan) from a predefined repertoire, which was created at design time by domain experts based on their past troubleshooting experience; such a repertoire can also be created in an automated manner [25]. In addition, researchers have demonstrated the potential of other techniques, such as reinforcement learning [62, 97], case-based reasoning [107], genetic algorithms [28], and fuzzy logic [53] that (similar to using expert knowledge) generate adaptation decisions offline but choose them at run time. In contrast to generating adaptation decisions offline, various automated planning techniques (e.g., model-checking [23, 85, 110], reinforcement learning [62, 97], and genetic algorithms [28, 63])) have been explored to generate adaptation plans at run time.

For the design of a MAPE-based system, an appropriate instantiation of the planning component is both critical and non-trivial. An appropriate instantiation is *critical* since it impacts the ability of a planning component to determine adaptation plans, and thus a system's potential to

[1]We use the term "planning" in a broad sense, referring to any decision-making approach that could be used to determine adaptation plans. Throughout the thesis, we use the term "planner" and "planning approach" interchangeably. As formalized in Chapter 3, both the terms refer to the black-box that takes a planning problem as an input and returns a plan. This black box encapsulates various planning aspects such as the planning tool that implements a planning algorithm/heuristic and its configuration options. Therefore, two instances of the same planning tool, but with different configuration options, will be considered as different planners.

meet adaptation goals. An appropriate instantiation of the planning component is *non-trivial* since there are numerous planning approaches, each having its own set of characteristics [44]; expertise is needed to identify and implement an approach that best meets the requirements.

For many self-adaptive systems, *quality* and *timeliness* are two particularly important requirements to be considered when choosing a planning approach to instantiate the planning component. Here "quality" of planning refers to the likelihood of a plan meeting the adaptation goals under the assumption that the plan is available instantaneously, when required. For many domains, such as safety-critical systems, quality of planning is important, especially since a bad plan could lead a system to an irreparable failure state that endangers lives [65]. In other domains such as an enterprise system, poor quality plans can hinder in meeting business goals.

In addition to quality, finding adaptation plans in a timely manner is another important requirement for planning [103]. For instance, after detecting a malware (e.g., a trojan) attack, if an enterprise system fails to determine a defense plan in a timely manner, the system risks being compromised, resulting in a failure to meet the goal of self-protection.[2]

Many systems need both: quick planning in urgent circumstances and near-optimal long-term performance. Ideally, such systems need a planning approach that can find optimal adaptation plans in a timely manner. For instance, to remain effective, commercial systems such as Amazon Web Services (AWS) have to maintain an up-time of at least 99.95% in any monthly billing cycle as per the service level agreement,[3] balancing it with other concerns such as cost minimization. When service-level constraints are violated, a rapid response is required to drive the system back to a desirable state (for AWS, maintaining availability). However, for long-term quality, adaptation plans should be as close to optimal as possible by considering other metrics (e.g., operating cost). Netflix is another example of such a system, where managing the overall latency of response to clients is critical to good user experience, in spite of the desire to minimize resource usage, and thus to lower operating cost.[4]

Unfortunately, for a planning approach, quality and timeliness are conflicting requirements. Planning, in essence, is a search/optimization process performed over the space of possible plans – more complete searches provide better quality guarantees, but require more time to complete. Hence, for urgent situations an approach can either provide a sub-optimal plan at the moment when it is needed, or provide a higher-quality plan, risking it being late. Moreover, this imbalance between quality and timeliness increases significantly with the increase in a search space that arises in the presence of large numbers of components, adaptation options, and multiple qualities of interest.

As a consequence, when choosing an off-the-shelf planning approach, self-adaptive systems today must compromise between one of the two requirements leading to systems that typically can either respond quickly, or provide a high-quality adaptation but not both (refer to Figure 1.1). Within the self-adaptive systems community, research has primarily focused on the quality of planning; timeliness of planning, in general, has not been treated as a first class concern [77].

One direction, explored by the artificial intelligence (AI) community, is to develop customized planning solutions applicable to a particular domain or a narrow class of planning problems,

[2] https://www.defenseone.com/technology/2019/02/russian-hackers-work-several-times-faster-chinese-counterparts-new-data-shows/154952/

[3] https://aws.amazon.com/compute/sla/

[4] http://techblog.netflix.com/2010/12/5-lessons-weve-learned-using-aws.html

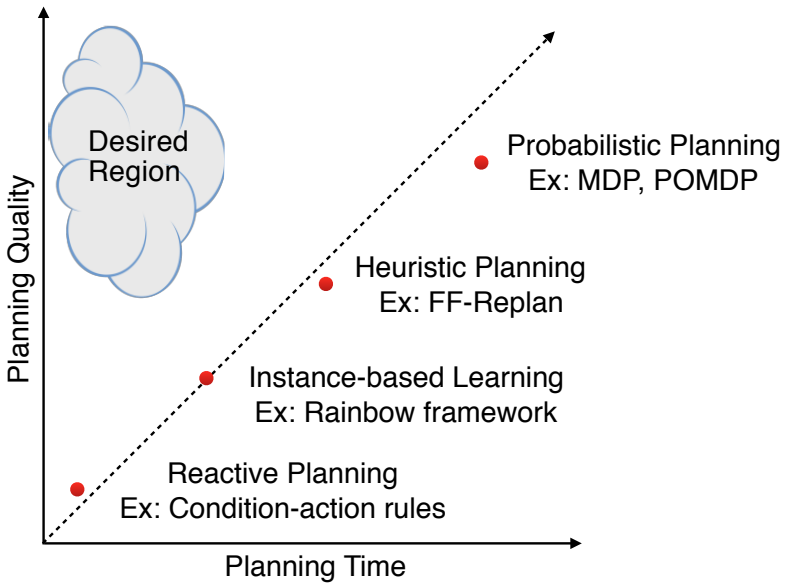


Figure 1.1: A notional representation of a space for planning approaches in domains with uncertainty. Ideally, one would like to move towards the desired region, i.e., high plan quality with low planning time

since such solutions can exploit specific knowledge about the search space. While sometimes successful [14, 49, 54], this approach is typically time-consuming and costly to develop since it requires deep understanding of the operating domain and experience in planning technology. Moreover, because these solutions are tailored to different domains, it is rare that successes are directly transferable to other domains; hence, these approaches are not general – an undesirable quality from a software engineering perspective.[5]

In contrast to inventing domain/problem-specific solutions, in this thesis we propose the idea of *hybrid planning* for self-adaptive systems that combines multiple off-the-shelf reactive planners with a deliberative one to address the trade-off between the timeliness and quality of planning.[6] The key idea is to use reactive planning to provide a quick (but potentially sub-optimal) response to a problem, but simultaneously invoke deliberative planning that is likely to provide a higher-quality plan (compared to reactive planning). Once a deliberative plan is ready, it takes over execution from the reactive plan to provide a higher-quality adaptation thereafter. Such a combination of planners can, in principle, reap the benefits of both worlds: providing plans quickly when the timing is critical, while allowing (nearly) optimal plans to be generated when the system has sufficient time to do so.

Hybrid planning has a number of potential advantages over custom planning solutions. Instead

[5]This fact is consistent with the *No Free Lunch Theorem*: for any search/optimization algorithm, performance gains over one class of problems are paid for by performance losses over another class [117].

[6]Having multiple reactive planners provides the flexibility to pick the best (reactive) planner for an emergency situation. For a complex system, it might be difficult to have a reactive planner that can deal with all (the possible) emergency situations.

of going through the non-trivial process of developing a new algorithm/heuristic, hybrid planning combines off-the-shelf planners. Using existing planners is likely to reduce development time and cost, since software engineers do not have to be AI experts or master the complexity of developing new algorithms/heuristics. In a sense, hybrid planning can be thought of as an instance of meta-planning that operates on a set of off-the-shelf planning approaches [21]; therefore, it raises the level of abstraction such that software engineers do not have to worry about developing new custom solutions.

Even though hybrid planning is a promising idea, its successful implementation faces substantial research challenges (as detailed later in Section 1.3):

- DEFINING HYBRID PLANNING (DEFHP) i.e., (formally) describing the hybrid planning problem in a way that helps to understand its general nature and describes the ideal behavior of a hybrid planner.
- INSTANTIATING HYBRID PLANNING (INSTHP) i.e., finding appropriate constituent approaches to instantiate hybrid planning such that quality and timeliness of planning can be balanced.
- PLANNING COORDINATION (PLNCRD) i.e., guaranteeing a seamless transition between plans determined by different constituent approaches.
- PLANNING SELECTION (PLNSEL) i.e., deciding which planning approach (among the constituents) should be invoked to solve a particular planning problem with hybrid planning, and when to stop using a plan produced by an approach and switch to a plan produced by another approach.

The rest of this chapter introduces a motivating example that will be used throughout the dissertation to present our approach, followed by the thesis investigated in this dissertation, an approach to address the research challenges listed above, and how the thesis claims were validated. Finally, the chapter lists the contributions from the thesis and provides organization for the rest of the dissertation.

1.1 Motivating Example for Hybrid Planning

To explain our approach, this section presents an exemplar system inspired by RUBiS [29] — an open-source benchmark application that implements the functionality of an auctions website that is widely used to evaluate research ideas for cloud-based systems [33, 36, 52, 82, 85, 98]. The exemplar system is a cloud-based self-adaptive system, as shown in Fig. 1.2, with a typical N-tiered architecture: a presentation tier, an application tier, and a database tier. Using the presentation tier, a client sends a request to the application tier, which interacts with the database tier to process the request. The system has different types of servers that cost more with increasing capacity (i.e., the ability to handle a number of requests per second). The system's workload depends on the request arrival rate, which is uncertain as it depends on external demand.

The system needs to optimize profit (i.e., maximizing revenue and minimizing operating costs) by means of various adaptation tactics. To maximize revenue, it is desirable to maintain the response time for user requests below some threshold (say T), since higher perceived user response time results in revenue loss [75]. Typically, an increase in request arrival rate causes a

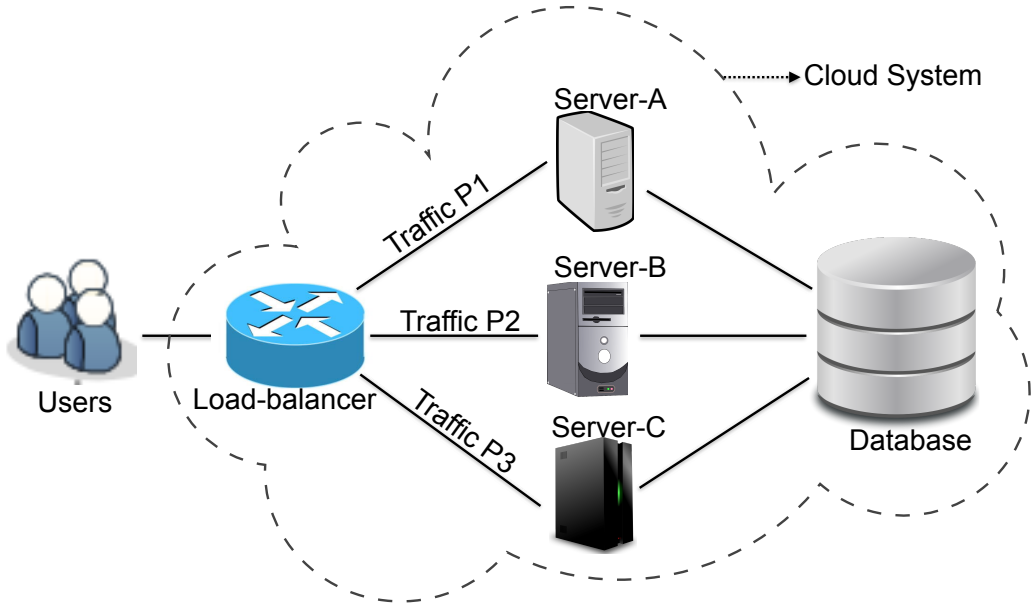


Figure 1.2: High-level view for the cloud-based system.

higher response time perceived by clients. In such situations, the system can add more servers (using tactic *addServer*<*type*>) to handle the increased workload, but also increasing the operating cost. To reduce costs, the system has an adaptation tactic (i.e., *removeServer*<*type*>) to deactivate a server.

In addition to manipulating servers, system response time can be controlled through "brownout" — reducing the amount of optional content (such as advertisements or product recommendations) [64]. Such optional content generates additional revenue, but requires more computational power and network bandwidth, which in turn increases the response time [32]. The system controls optional content with tactics *increaseDimmer* and *decreaseDimmer*, which respectively raise or lower the probability that a request will contain optional content. In other words, the number of requests with optional content decreases as the value of the dimmer setting decreases.[7] Different variants of brownout have been applied to cloud-based systems and shown to be effective in their respective contexts [118].

Since the system has servers of different capacity, a round-robin strategy for assigning client requests to active servers would not be efficient: the number of client requests delegated to a server should depend on its capacity. The load-balancer uses queueing theory [48] to decide on the optimal load-distribution among the active servers. To distribute the load efficiently, there is a tactic (i.e., *divert_traffic*<*traffic_server*$_1$ ... *traffic_server*$_n$>, where *traffic_server*$_i$ is the traffic, in terms of percentage of requests, for the i-th server, and n is the total number of servers), which helps the load-balancer manage the percentage of client requests assigned to each server.

[7]This formulation of the brownout mechanism provides more flexibility to the system compared to RUBiS, which is limited to a binary choice of having all or no responses include the optional content.

We assume there is a penalty, say P, for each request having a response time above the threshold. Therefore, in case of a high response time, the system needs to react quickly either by adding servers or decreasing the dimmer value. However, once response time is within acceptable limits, the system should execute adaptation tactics to bring down the operating cost or increase revenue in order to maximize long-term utility.

To summarize, the system needs to increase the revenue, keep response time below the threshold to avoid penalty P, and minimize the number of active servers to reduce operating cost. These objectives are captured in a multidimensional utility function shown in Formula 1.1; the adaptation goal of the system is to maximize the utility calculated using this formula. If the system runs for duration L, its utility function is defined as:

$$U = R_O x_O + R_M x_M - P x_T - \sum_{i=1}^{n} C_i \int_0^L s_i(t) dt \tag{1.1}$$

where R_O and R_M are revenue generated by a response with optional and just mandatory content respectively; P is the penalty for a request having a response time above the threshold; x_O, x_M, and x_T are number of requests with optional content, mandatory content, and having response time above the threshold, respectively; C_i is the cost of server type i, and s_i is the number of active servers of type i; n is the number of server types.

This multidimensional utility function captures conflicting requirements such as lowering response time, increasing revenue, and decreasing operating cost. Such a utility function captures both the quality and the timeliness of planning. The function captures quality since it has various adaptation goals as its constituents. Timeliness is captured since there is a penalty for response time above the threshold; in such a situation the system needs to react quickly to lower the penalty.

To determine adaptation plans for self-adaptive systems such as this cloud-based system, researchers have suggested a diverse set of planning approaches such as rule-based adaptation (RBA) [27], case-based reasoning (CBR) [107], fuzzy-logic [53], reinforcement learning [97], stochastic search (e.g., genetic algorithms) [28], that, generally speaking, fall into the category of reactive planning. These approaches determine an adaptation plan quickly because the plan is not generated at run time, but rather selected from an existing set of plans; however, the selected plan might not be optimal because it is difficult to have an optimal plan that was determined offline. For instance, a system with a rule-based adaptation might have a hard-coded rule saying that whenever the response time constraint is violated, add a server with the highest capacity. This plan could be sub-optimal if the spike in client requests is temporary; by the time the newly added server is active, response time would be below the threshold, but the system ends up paying an additional cost for this server.

In contrast, researchers have also proposed various deliberative approaches, such as planning based on Markov decision processes (MDP) to dynamically generate plans [23, 39]. Deliberative approaches such as MDP and partially observable Markov decision processes (POMDP) [55] planning at run time can be slower compared to the approaches discussed above, but they typically provide high-quality adaptation plans for uncertain situations since planning considers factors such as the current state of the system and its environment, predicted (but uncertain) values of future request arrival rate, and timing of tactic latency [82, 85]. [8]

[8]Compared to MDP, POMDP can additionally consider uncertainty in the underlying state.

For self-adaptive systems such as this cloud-based system, using a single (i.e., either a reactive or a deliberative) planning approach can be problematic. For instance, a reactive approach such as rule-based adaptation (RBA) might quickly provide a plan to a response time constraint violation, and thus improve the system's utility in the short-term.[9] However, the plan is likely to be sub-optimal due to uncertainty in the request arrival rate, which is difficult to predict/model at design time (i.e., when formulating the rule).[10] On the other hand, if a deliberative approach, such as one based on MDPs, is used for planning it is likely to provide high-quality plans but at the cost of having to wait, which would be an issue, particularly for situations such as a response time constraint violation. To balance quality and timeliness, hybrid planning (HP) seems a promising way to improve utility (e.g., Formula 1.1 presented above). For instance, hybrid planning can be instantiated using a reactive (e.g., RBA) and a deliberative (e.g., MDP) planning approach. RBA can provide a timely (i.e., quick) response to emergency situations (e.g., response time above the threshold), whereas MDP planning can be used to provide higher-quality plans, thus balancing quality with timeliness of planning.

1.2 Thesis

This research improves the current state of the art for planning in self-adaptive systems. Due to the trade-off between timeliness and quality of planning, when choosing a single planning approach, designers have two choices: (a) make an offline (i.e., at design time) compromise between finding adaptation plans quickly and finding quality plans demanding longer computation times, or (b) deal with the complexity of developing a customized planning approach that is likely to consume time and resources. Therefore, rather than choosing a single approach, we propose a hybrid planning approach that combines multiple off-the-shelf reactive planning approaches with a deliberative one to find a balance between quality and timeliness without incurring the overhead of developing a customized planning approach. In the context of self-adaptive systems, this thesis demonstrates that hybrid planning can be applied with the following three qualities:

- **Effectiveness:** Hybrid planning improves the effectiveness of a self-adaptive system compared to its constituent approaches used alone. Effectiveness is a measure of a system's ability to meet its adaptation goal. In this thesis, we assume that the system's adaptation goal is encoded in a multidimensional utility function (e.g., Formula 1.1) that captures both quality and timeliness of planning. To validate effectiveness, we demonstrate that hybrid planning provides higher utility.
- **Generality:** Hybrid planning is general enough to be applied (effectively) to self-adaptive systems operating in domains that differ in: (a) quality dimensions of concern, (b) the cost of poor/delayed actions, and (c) the ability to recover from poor/delayed actions.
- **Flexibility:** Hybrid planning can be instantiated (effectively) using different combinations

[9]Rule-based adaptation refers to a planning approach where adaptation plans are determined by predefined condition-action pairs (i.e, rules) indicating the action to be taken for a given condition. Typically, the conditions capture an abstract presentation of a system state.

[10]Theoretically, designers can formulate rules (at design time) for all the possible variations in the request arrival rate that the system can observe at run time. However, this approach might not scale for realistic cloud-based systems.

of reactive and deliberative approaches. Two instantiations are considered different if any of the constituent (reactive or deliberative) approaches are different between the instantiations.

Thesis Statement: *We can improve the* effectiveness *of self-adaptive systems by using a hybrid planning approach, which is* general *and* flexible*. This approach has the following elements:*

- *the use of off-the-shelf deliberative and reactive planning approaches to instantiate hybrid planning that can take advantage of both planning approaches to find a balance between quality and timeliness of planning;*
- *the ability to dynamically decide which constituent reactive planning approach should be invoked along with deliberative planning.*

1.3 Approach Overview

For a self-adaptive system such as the system presented in Section 1.1, hybrid planning seems to be a promising way to improve effectiveness by balancing quality and timeliness of planning. However, elaborating on what was said earlier, there are number of challenges in applying hybrid planning to realistic systems:

- DEFINING HYBRID PLANNING (DEFHP): Understanding the hybrid planning problem in its general nature is important in order to apply the idea realistically. When an arbitrary number of planning approaches with different time-quality trade-offs are combined together, currently, we lack answers to questions such as (a) what is a hybrid planning problem?, and (b) what does it mean to solve the problem?, and (c) what are the intermediate steps to solve the problem?

  Approach: To answer such questions, this dissertation formally defines the problem of hybrid planning in its general form (cf. Chapter 3). In addition, the formalization breaks the complex problem of hybrid planning into four sub- problems. In Chapter 4, this formalism is used to explain our approach to apply hybrid planning in a realistic context; solutions grounded in a formal model give us confidence that all relevant challenges are addressed. Moreover, as demonstrated in Chapter 6, this model can serve as a unifying evaluation framework to analyze and compare different instantiations of hybrid planning, and thereby understand their strengths and weaknesses.

- INSTANTIATING HYBRID PLANNING (INSTHP): Finding constituent approaches with the appropriate time-quality trade-off is critical to an effective instantiation of hybrid planning. However, for software engineers, finding such a set of approaches is a non-trivial process, since there are numerous candidate approaches to be used for reactive or deliberative planning [44]. Once we decide on constituent approaches, another obstacle to practical adoption of hybrid planning is lack of application guidance: When will hybrid planning outperform its constituent planning approaches for a given system?

  Approach: To address these questions, this thesis provides guidelines (in Chapter 7) to help software engineers to select planning approaches for instantiating reactive and deliberative planning. In addition, this dissertation uses findings from two case studies (discussed in Chapter 6) to provide data-driven guidance on appropriate conditions for using hybrid

planning. Specifically, we determine when hybrid planning is likely to be effective (i.e., outperform its constituent approaches).

- PLANNING COORDINATION (PLNCRD): To balance quality and timeliness, hybrid planning requires a smooth transition from a reactive plan to a possibly higher-quality deliberative plan. Suppose the system observes a response time constraint violation. As a result, assume reactive planning is invoked to provide a quick response to this problem. Now, for a seamless transition from the reactive plan to the deliberative plan, the latter needs to have provisioned for the system's state after executing the reactive plan. This is challenging for two reasons: (a) uncertainty about deliberative planning time makes it difficult to predict when the deliberative plan will be ready to take over, and (b) uncertainty in the system's environment makes it difficult to predict the expected system state after executing the reactive plan.[11]

  Approach: To solve PLNCRD, our approach has two distinguishing characteristics: (a) deliberative planning generates a universal plan (one containing state-action pairs for all the reachable states from the initial state), where a mapping from a state (say s) to an action (say a) indicates that a be executed in s [44], and (b) the operating domain is *Markovian*: the state after a transition depends only on the current state — not on the sequence of states that preceded it [78]. As explained in Chapter 4, these characteristics theoretically ensure a smooth transition if reactive and deliberative planning use the same initial state. That is, once the deliberative plan is ready, it can take over plan execution from the reactive plan because any state resulting from executing the reactive plan will be found in the deliberative plan.

- PLANNING SELECTION (PLNSEL): Assume that hybrid planning is instantiated using a deliberative approach and a finite set of reactive approaches.[12] For a planning problem, solving PLNSEL refers to choosing the reactive approach (from the given set) that in combination with deliberative planning will provide the highest utility.[13] The set has a special reactive approach that, for any planning problem, always suggests to wait until the deliberative plan is ready; therefore, using this approach in combination with deliberative planning is equivalent to using deliberative planning alone. This reactive approach is required to ensure that hybrid planning does not underperform deliberative planning in cases when none of the other reactive approaches (in the set) provides a better plan than just waiting for the deliberative plan to be ready.

  Approach: As the first approach to solve PLNSEL, we propose a condition-based (CB) invocation of reactive planning where a system's designer specifies up-front conditions (at design time) under which (a particular) reactive approach should be invoked. For example, whenever the response time constraint is violated for the cloud-based system, a reactive

[11] A state consists of system state and environment state.

[12] In this thesis, we restrict instantiations of hybrid planning to using a single deliberative approach to keep the problem of hybrid planning tractable as discussed in Chapter 4. However, in certain ways, our approach allows hybrid planning to be instantiated using multiple (reactive and) deliberative approaches as discussed in Chapter 8.

[13] As discussed in Chapter 4, the choice of using reactive planning *not* followed by deliberative planning is not considered since if a deliberative plan is ready to take over, it will provide a higher utility compared to a plan determined by any of the reactive approaches.

approach (from the given set of reactive approaches) can be invoked that might suggest to add a server with the highest capacity. Invoking a reactive approach based on predefined conditions is easy to apply when determining invocation conditions (e.g., emergencies) is straight-forward at design time; invoking reactive planning on such conditions reduces the risk of inappropriate quick decisions. However, the condition-based approach suffers from three major drawbacks: (a) it requires domain expertise to identify the conditions that should trigger reactive planning; (b) it relies on error-prone humans to identify the right and comprehensive conditions; and (c) it hinders reuse of hybrid planning since such conditions do not transfer to other systems/domains.

To overcome these drawbacks, this thesis also proposes a supervised *machine learning-based* (LB) approach to decide which reactive planning approach (from a given set, which includes waiting as a special case) in combination with deliberative planning would lead to improved performance for a given situation. In the training phase, using planning problems similar to those expected at run time, the approach trains a classifier to choose an appropriate reactive approach for a given problem. At run time, depending on how the current situation (i.e., the planning problem at hand) relates to problems in the training set, the classifier decides on the reactive approach to be invoked. This approach overcomes the disadvantages of condition-based (CB) invocation of reactive planning by removing the need for humans to determine the specific conditions at design time and being applicable to a broad range of systems/domains.

To train a classifier, one needs a set of labeled training problems such that the label of a problem indicates the reactive approach which, in combination with deliberative planning, will provide the best performance among the reactive approaches. To evaluate a combination, labeling requires evaluating the performance of the reactive plan (determined by the reactive approach for the problem) followed by the deliberative plan.

Obtaining sufficient and high-quality training data for a classifier is challenging on a real system: to label one planning problem, one would have to repeatedly put the system and its environment in the same exact state to test out different planning combinations. Further, in domains with uncertain dynamics, the environment evolution and the outcomes of the system's actions may change between attempts, so one would have to perform multiple trials of the same combination to determine the best average outcome.

We employ *probabilistic model-checking* that estimates the performance of a combination (of reactive and deliberative planning) for all possible execution paths in a planning problem with a single run of a model checker. For the estimation, we encode the combination and the problem in a probabilistic model checker specification and use it to calculate the performance assuming the specification represents the reality. Given a planning problem, a finite set of reactive approaches and a deliberative approach, this process is repeated for each reactive approach to evaluate its combination with the deliberative approach for the problem. By comparing the performances of the combinations, one can choose the best combination for the problem and label it accordingly.

Using a probabilistic model checker yields two benefits: (a) the probabilistic nature of the model-checking helps in accounting for uncertainty when evaluating a combination of plans

determined by reactive and deliberative planning; and (b) (multiple) existing probabilistic model checkers ease adoption, automation, and reuse of the learning-based approach by software engineers.

1.4 Validation of the Claims

The thesis of this research claims that hybrid planning is *effective*, *general*, and *flexible*. The dissertation uses two case studies (cf. Chapter 6 for details) to validate these claims. This section briefly summarizes the validation approach.

- **Effectiveness**: The two case studies demonstrate that hybrid planning significantly improves the effectiveness of a system in a variety of natural contexts. Specifically, in the case studies, hybrid planning achieved the adaptation goals better (i.e., provided higher utility) compared to its constituent planning approaches.[14]
- **Generality**: To validate generality, we apply hybrid planning on two different kinds of system — the RUBiS-inspired cloud-based system (as discussed earlier) and a safety-critical system (i.e., a team of unmanned aerial vehicles (UAV)). These systems differ in several significant ways:
  - *Quality dimensions of concern*: The cloud-based system aims at lowering response time, increasing revenue, and decreasing operating cost, whereas the UAV team intends to avoid threats and detect targets;
  - *The cost of poor/delayed actions*: Poor/delayed actions could lead to destruction of a UAV(s) in the team. Therefore, generally speaking, the (monetary) cost of such actions is higher for the team compared to the cloud-based system;
  - *The ability to recover from poor/delayed actions*: Even if the cloud-based system fails to maintain its critical constraint (i.e., response-time below the threshold) due to poor/delayed actions, it can still recover to a desired state later. However, in the case of the team of UAVs, a failure to avoid a crash (i.e., safety constraint) due to a poor/delayed action, could lead to a mission failure.
- **Flexibility**: To demonstrate flexibility, we use different combinations of off-the-shelf deliberative and reactive planning approaches for the two case studies. Specifically, the first case study uses MDP and deterministic planning as the deliberative and reactive approach, respectively. In contrast, the second case study uses MDP planning both as a deliberative and reactive approach; however, the reactive version of MDP planning uses a shorter planning horizon and only a subset of adaptation actions compared to the deliberative version of MDP planning.

[14]Both condition-based and learning-based approaches are able to solve PLNSEL such that hybrid planning is more effective compared to its constituent approaches. On comparing condition-based and learning-based approach in the two case-studies, we find that (on average) learning-based approach is more (or at least equally) effective compared to condition-based.

1.5 Thesis Contributions

The main contribution of this dissertation is to show that hybrid planning improves the current state-of-the-art for planning in self-adaptive systems by finding a balance between quality and timeliness of planning. More specifically, the dissertation contributes to both the theory and the practice of hybrid planning in self-adaptive systems.

The contribution to theory is:

- a formal model characterizing the general problem of hybrid planning;
- an illustration of how the formal model can be used as a unifying evaluation framework to compare/analyze instantiations of hybrid planning, and thereby understand their strengths and weaknesses.
- a formal analysis of the performance of the hybrid planning algorithm.

The contributions to practice are:

- a practical approach to applying hybrid planning under certain assumptions/restrictions that nonetheless apply to many self-adaptive systems;
- a demonstration of effectiveness, generality, and flexibility of hybrid planning for self-adaptive systems using the proposed solution approach;
- methods/tools to apply hybrid planning to self-adaptive systems, including
  - evaluation of hybrid planning using two systems (i.e., the cloud-based system and the UAV team) to illustrate how the proposed approach can be applied to realistic self-adaptive systems,
  - an implementation of the hybrid planning algorithm using a widely accepted MAPE-based self-adaptive framework (i.e., Rainbow [27]) to ease the adoption of hybrid planning among software engineers,
  - guidelines and a quantitative approach to help engineers to select an appropriate set of planners to instantiate hybrid planning for a given domain.

The research presented in this dissertation resulted in the following peer-reviewed publications:

1. Ashutosh Pandey, Gabriel A. Moreno, Javier Cámara and David Garlan. Hybrid Planning for Decision Making in Self-Adaptive Systems. In Proceedings of the 10th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2016), Augsburg, Germany, 12-16 September 2016.
   This paper introduced the idea for hybrid planning and condition-based approach to solve PLNSEL.
2. Ashutosh Pandey, Ivan Ruchkin, Bradley Schmerl, Javier Cámara and David Garlan. Towards a Formal Framework for Hybrid Planning in Self-Adaptation. In Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2017), Buenos Aires, Argentina, 22-23 May 2017.
   This paper provides the initial formalization for the problem of hybrid planning.
3. Ashutosh Pandey, Bradley Schmerl and David Garlan. Instance-based Learning for Hybrid Planning. In Proceedings of the 3rd International Workshop on Data-driven Self-regulating

Systems (DSS 2017) (in conjunction with SASO) , Tucson, AZ, USA, 18-22 September 2017.
This paper introduces the idea of the learning-based approach and provides initial evidence for the effectiveness of the approach.

4. Gabriel A. Moreno, Cody Kinneer, Ashutosh Pandey, and David Garlan. DARTSim: an exemplar for evaluation and comparison of self-adaptation approaches for smart cyber-physical systems. In Proceedings of the 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS@ICSE 2019, Montreal, QC, Canada, May 25-31, 2019.
This paper presents the DARTSim exemplar that simulates the team of UAVs. Dartsim is used as the second system for evaluating the thesis claims.

1.6 Dissertation Outline

The rest of the thesis is structured as follows. Chapter 2 discusses the related work relevant to hybrid planning. Chapter 3 formalizes the problem of hybrid planning, and therefore covers the theoretical aspect of hybrid planning. Moreover, the chapter lists (potential) applications of the formal model. Chapter 4 outlines our approach to instantiate hybrid planning; the approach is explained in the context of the formal model to ensure that all the relevant challenges are addressed. Chapter 5 describes a general hybrid planning algorithm executed in hybrid planners, and provides theoretical bounds on the performance of hybrid planning. Chapter 6 validates the thesis claims using the two case studies (i.e., self-adaptive cloud-based system, and a team of UAVs), and highlights the factors that influence the performance of hybrid planning. Also, the chapter illustrates how the formal model can be used as a unifying evaluation framework to compare/analyze instantiations of hybrid planning, and thereby understand their strengths and weaknesses. Chapter 7 provides informal guidelines to select an appropriate set of planners to instantiate hybrid planning; this chapter can, particularly, be useful for practitioners interested in applying hybrid planning to realistic systems. Finally, the thesis concludes with Chapter 8 that analyzes the contributions of the thesis, its limitations, and future work.

Chapter 2

Related Work

This chapter presents related work relevant to hybrid planning. To begin with, Section 2.1 compares and contrast hybrid planning with commonly used approaches that intend to deal with the trade-off between timeliness and quality of planning. The section highlights benefits of hybrid planning compared to the existing approaches. Once benefits of hybrid planning are discussed, Section 2.2 differentiates our notion of hybrid planning from other notions of hybrid planning; as discussed later, the term "hybrid planning" is being broadly used by researchers for their planning solutions that combine multiple planners. Section 2.3 further clarifies our notion of hybrid planning by discussing how our approach is inspired by the field of hyper-heuristics. Finally, Section 2.4 compares our approach to other similar instances of hybrid planning and discusses how the proposed approach is more general.

2.1 Approaches to Deal with the Trade-off between Timeliness and Quality of Planning

This section discusses commonly used approaches to deal with the timeliness-quality trade-off of planning and highlights the comparative benefits of hybrid planning. The section is structured as follows: Section 2.1.1 presents the planning approaches that find timely plans by not generating a plan at run time, but choosing it from a set of precomputed plans, thereby, reducing run-time overhead; Section 2.1.2 discuss the approach to develop customized algorithms/heuristics; Section 2.1.3 discusses reinforcement learning; and Section 2.1.4 presents hierarchical task network (HTN).

2.1.1 Using Precomputed Plans

To determine adaptation plans, researchers have suggested a diverse set of planning approaches such as rule-based adaptation [27], case-based reasoning [107, 115], that, generally speaking, determine an adaptation plan quickly because the plan is not generated at run time, but rather selected from an existing set of precomputed plans; however, quality (in a utility-theoretic sense) of plans might be bad, since the set of precomputed plans may not be sufficient to handle unforeseen problems or environments [1]. Similarly, fuzzy-logic determines plans in a quick time since it uses

a predefined set of rules to determine a plan [76]; however, the approach is not robust unless there is a comprehensive set of rules, and having such a set is non-trivial, particularly, for domains with uncertainty [8]. To summarize, approaches such as rule-based adaptation, case-based reasoning, and fuzzy-logic can find a plan in a quick time but the plan might be of low quality. However, since these approaches have potential to determine plans in a quick time, hybrid planning can be instantiated with these approaches (as reactive planners) to provide a quick (but potentially a sub-optimal) response to emergencies, and a deliberative planner that can handle uncertainty better than these reactive ones.

2.1.2 Search and Optimizing Algorithms/Heuristics

The AI community has been working towards finding better algorithms and heuristics to deal with the issue of planning delay. Generally speaking, these approaches reduce the planning time by selectively exploring the plan search space, thereby trading off quality against timeliness. For instance, many algorithms/heuristics have been developed to reduce the planning time for deterministic domains [13, 14, 40, 49].

For probabilistic domains, the state-of-the-art planning approaches based on Markov decision processes (MDP) [80], and partially observable Markov decision processes (POMDP) [55] can provide quality plans by considering uncertainty. If an MDP/POMDP policy can be determined offline (i.e., no run-time overhead), these approaches can provide a quick and a quality response to a situation. However, for many realistic systems such as the ones used for the thesis evaluation, offline planning is difficult because: (a) upfront consideration of all the possible states and transitions for planning might not scale for the systems[1], and (b) uncertainty in the operating domain could lead to difficulty in upfront probabilistic modeling of uncertainty in a planning problem specification used for the offline planning; imprecise modeling of uncertainty can negatively impact the quality of planning.[2] Therefore, for such systems, online MDP/POMDP planning could be more suitable than offline planning; however the online planning might not be quick enough to respond timely to emergency situations [73, 100]. Although various optimization algorithms have been suggested to improve the planning time for MDP [78] and POMDP [96, 100, 106] planning, planning delay in probabilistic domains is still an ongoing challenge.

To balance timeliness and quality of planning, in contrast to developing algorithms/heuristics that requires AI expertise, software engineers can use hybrid planning that combines off-the-shelf planners with a different time-quality profile. Moreover, assuming that in future AI researchers will develop better algorithms/heuristics to deal with the timeliness-quality trade-off, once they are available, the algorithms/heuristics can be used as a constituent approaches to instantiate hybrid planning. Therefore, hybrid planning helps software engineers benefit from such advances in dealing with the trade-off between timeliness and quality without being an AI expert.

[1] For instance, POMDP planning specification can have an infinite state space.

[2] In the systems used for the thesis evaluation, instead of doing offline planning by considering all the possible states and transitions over the entire execution period (for the systems), we do online planning with a shorter planning horizon as detailed in Chapter 6.

2.1.3 Reinforcement Learning

Reinforcement learning is the other commonly used approach that can determine a plan in quick time. The approach converges to an optimal MDP policy by trial and error over time, while a system is under operation [109]. Reinforcement learning can handle uncertainty, and once the optimal policy is determined, can provide a high-quality response to a situation in a quick time, thereby, balancing the timeliness and the quality of planning. However, there are two key drawbacks of reinforcement learning: (a) the trial and error approach might not be suitable for safety-critical system, and (b) the time to converge to an optimal policy increases exponentially with the increase in uncertainty in a domain; meanwhile, during the learning phase, the system is at risk of making a fatal decision.

2.1.4 Hierarchical Task Networks

To deal with the trade-off between the quality and timeliness of planning, in contrast to our hybrid planning approach where multiple planners plan at the same level of abstraction, researchers have proposed planning frameworks that create an HTN −− which combines multiple planners at different levels of abstraction and timing.

From the adaptive systems community, Kramer et al. [68] proposed a layered architecture inspired by Gat [38], which deals with the problem of planning delay through the hierarchical decomposition of the planning domains. Tajali et al. [112] extended the layered architecture by suggesting two types of planning: application planning and adaptation planning. Since hierarchical decomposition of a planning domain reduces the planning state-space at each layer, such an architecture helps to reduce the planning time. However, such a decomposition requires significant domain expertise to create a hierarchy and choose a planner for each layer. Our approach does not require a hierarchical decomposition; in fact, the approach complements hierarchical frameworks since hybrid planning could be deployed within a layer, for instance, to deal with planning delays in that layer.

The AI community has proposed various execution frameworks that, in general, also rely on the hierarchical decomposition of planning domains [67]. Quite different from these layered architectures, Musliner et al. propose a framework that ensures the execution of tasks meeting a specified deadline [90]. However, unlike hybrid planning, this framework requires hard deadlines to be specified in the planning specification.

2.2 Different Notions of Hybrid Planning

Generally, the term "hybrid planning" refers to solving a planning problem by combining multiple planning approaches/algorithms to benefit from their combined strengths. However, there is no universally accepted definition of hybrid planning; researchers have used the same term for different approaches. For example, some prior work interprets hybrid planning as combining domain-dependent and domain-independent planning approaches. To exemplify this definition, (as explained in Section 2.1.4) researchers combined hierarchical task networks (HTN), which is a domain-dependent approach since human expertise is needed to create a hierarchy, with domain-independent techniques from classical planning, such as partial-order planning (POP).

In that work, POP is modified to perform HTN planning [9, 57, 58, 104]. In contrast, Fox uses the term "hybrid planning" for a combination of planning approaches having specialized solvers, such as optimization algorithms and model-checking [35]. Quite differently, Li et al. [72] understand hybrid planning as planning for *hybrid* systems, which require handling of discrete and continuous action effects; therefore, the approach is not about combining multiple planning approaches/algorithms.

Although we use the same term, our approach is different from existing work. Our notion of hybrid planning combines multiple off-the-shelf planning approaches without a hierarchical relationship, i.e., the approaches plan at the same level of abstraction but the size/region of the planning state space may vary. For instance, the planning state space for one approach can be a subset of another planning approach or approaches can use different algorithms to find a plan in the same state space. These approaches are activated as necessary, ideally using the most appropriate approach in each situation. This notion focuses on reaping benefits of different constituent planning approaches at the right time and in the right context.

Specifically, this dissertation focuses on balancing timeliness and quality of planning by instantiating hybrid planning using reactive approaches in combination with a deliberative planning approach. As mentioned earlier, the reactive approaches provide plans quickly that could be useful in emergency situations. While a reactive plan is executed, the deliberative approach refines the reactive plan or provides a (different) higher-quality plan. This instance of hybrid planning is inspired by human decision-making: depending upon factors such as available planning time, humans apply different levels of deliberation to make real-life decisions [56]. Our notion assumes domain knowledge only in identifying an appropriate set of constituent planning approaches that can balance quality and timeliness. To help software engineers, Chapter 7 provides guidelines to build such a set.

2.3 Hyper-Heuristics

Our notion of hybrid planning is inspired by the research field of hyper-heuristics, which focuses on combining multiple lower-level heuristics (i.e., similar to constituent planning approaches in hybrid planning) and developing search methods or learning mechanisms for selecting or generating heuristics to solve computational search problems [20]. A hyper-heuristic is a high-level heuristic that, given a particular search problem instance and a number of low-level heuristics to solve the problem, selects and applies an appropriate low-level heuristic at each decision point. The field of hyper-heuristics is influenced primarily by two foundational frameworks. The first framework, formulated by Wolpert, suggests that it is impossible to devise a silver-bullet algorithm since all optimization algorithms yield equivalent performance on average [117]; therefore, the framework provides the reason to combine multiple heuristics. The second framework, suggested by Rice, suggests using approximation theory to select an appropriate algorithm/heuristic (from a set) for a given problem [99]; therefore, depending on the problem, the framework provides a way to select an appropriate algorithm/heuristic from the given set.

Since planning generally fits into the category of search/optimization problems, researchers have applied ideas from the field of hyper-heuristics to select a planning approach (from a set) to solve a planning problem. For instance, Gratch et al. [46, 47] proposed a system that uses

hill-climbing search in the space of possible control strategies (which can be understood as planning approaches) to solve scheduling problems (which constitutes a type of planning). But, unlike our approach, their work is based on the assumption that control strategies can be structured to facilitate a specific search method (e.g., hill-climbing), and therefore is limited to specific control strategies. As another example of combining multiple approaches, Lamghari et al. [70] (specifically) combine reinforcement learning [109] and Tabu search [45] to solve planning problems under uncertainty.

However, unlike the existing works, our notion of hybrid planning is more general since it is not limited to a specific search/optimization method. Moreover, for a planning problem, they focus only on the quality (not the timeliness) of planning by picking a method that is likely to provide the highest-quality plan; they evaluate their approach on a given set of offline problems rather than on a running system where timeliness of planning can be critical. In contrast, we explicitly deal with both timeliness and quality by applying a reactive and deliberative approach to solve a problem. To demonstrate the effectiveness of our approach we evaluate it on a running system (cf. Chapter 6).

2.4 Other Similar Instantiations of Hybrid Planning

Different research communities focused on self-adaptive software systems and AI have proposed instantiations of (our notion of) hybrid planning to deal with the quality/timeliness trade-off. However, these instantiations have been limited to a particular domain or a specific combination of reactive and deliberative planning.

From the adaptive systems community, researchers have proposed various instantiations of hybrid planning that use a condition-based approach to solve the planning selection problem (PLNSEL), discussed in Chapter 1. For instance, Iqbal et al. [51] and Ali-Eldin et al. [3] proposed hybrid controllers in the context of self-adaptive cloud systems; these instantiations of hybrid planning use threshold-based rules to invoke reactive planning. Bauer et al.[6] extended this idea with more sophisticated conditions. Broadly speaking, while these are instances of condition-based hybrid planning, they are (a) specific to a particular domain — self-adaptive cloud systems, and (b) limited to a particular combination of reactive and deliberative planning approach.

From the AI community, the instantiation proposed by Mausam et al. [79] works specifically with labeled real-time dynamic programming (RTDP) planning [15] and non-deterministic planning using a Model-Based Planner (MBP) [11] as deliberative and reactive planning, respectively. Beetz et al. [7] proposed an approach that projects the effects of contingencies on the plan (generated by reactive planning) under execution and, if required, revises the plan using a more deliberative planning approach. Their approach, again, assumes a specific combination of reactive and deliberative planning. In addition, the approach is restricted to a particular plan specification language to ensure a smooth transition from a reactive plan to a deliberative plan. To solve PLNSEL, both Mausam et al. [79] and Beetz et al. [7] propose to always use reactive planning and, if required, revise the plan using a more deliberative planning approach. In other words, they assume that reactive planning will always improve the current situation. Perhaps they made this assumption because their instantiation of hybrid planning is limited either to a specific combination of reactive and deliberative planning or a particular domain. However, this

assumption might not always hold, since it depends on the quality of a reactive approach and the nature of an operating domain. For instance, as we show later (cf. Chapter 6), invocation of a reactive approach is sometimes worse than waiting for a deliberative plan.

Tallavajhula et al. [113] proposed a notion of hybrid planning that combines only reactive planners (i.e., planning time is considered negligible), and thus only focuses on the quality of planning. To solve PLNSEL, a learning-based approach is used that helps to choose the best reactive approach for a planning problem. However, unlike our learning-based approach, since there is no deliberative approach, combinations of reactive and deliberative planning are not considered when choosing a reactive approach.

Researchers have also suggested a broad category of planning algorithms, based on the idea of incremental planning, known as "anytime" planning. Typically, anytime planning algorithms are optimizing (e.g., value iteration algorithm for MDP planning) in nature: the planning process can be interrupted at any time to get a sub-optimal plan, and longer planning times lead to better plans [122]. Anytime planning is a special case of hybrid planning, since anytime algorithms utilize the execution time of a low-quality plan to devise an improved plan. Once ready, the improved plan takes over the execution from the lower-quality plan. However, compared to anytime planning, the idea of hybrid planning is more general since it allows us to combine multiple search/optimization planning approaches.

2.5 Summary

The idea of combining multiple planning approaches has been suggested in various forms. However, existing works fall into (at least) one of these categories: they (a) require domain expertise for decomposition a planning problem, (b) do not explicitly deal with the trade-off between timeliness and quality of planning, and (c) are limited to a specific domain or a specific combination of reactive and deliberative planning. In contrast, our notion of hybrid planning does not require domain expertise to decompose a planning problem, explicitly deals with the timeliness-quality trade-off, and is not limited to a specific domain or a combination of planning approaches.

In addition to proposing the idea of hybrid planning, this dissertation goes a step further to formally define the problem of hybrid planning to describe its general nature (cf. Chapter 3) in the context of our notion. In Chapter 4, this formalism is used to explain our approach to applying hybrid planning in a realistic context. Moreover, Chapter 6 provides an example to demonstrate how the formal model can be used to analyze and compare existing hybrid planning instantiations, and thereby understand their strengths and weaknesses. In the rest of the thesis, the term "hybrid planning" will refer to our notion of hybrid planning.

Chapter 3

The Problem of Hybrid Planning

Although hybrid planning is a promising idea potentially applicable to a wide variety of domains, as discussed earlier, its successful implementation faces four substantial challenges: (a) defining the problem of hybrid planning, (b) instantiating hybrid planning using constituent approaches with an appropriate time-quality trade-off, (c) the planning coordination problem (PLNCRD) – i.e., guaranteeing a seamless transition between plans determined by different planning approaches, and (d) the planning selection problem (PLNSEL) – i.e., deciding which planning approach(es) should be invoked to solve a planning problem and when to stop using a plan produced by one approach and switch to a plan produced by another approach. This chapter addresses the first challenge: i.e., formally defining the problem of hybrid planning.

Suppose hybrid planning is instantiated using constituent planning approaches with an appropriate time-quality trade-off, and PLNCRD and PLNSEL are addressed. However, it is not known how to systematically analyze/evaluate such instantiations and compare them to other planning approaches. Hybrid planners are sometimes compared favorably to their constituent approaches [79], but that is a relatively conservative benchmark. Currently, any comparison between different hybrid planner implementations is difficult because we lack a fundamental description of the ideal behavior of a hybrid planner.

This dissertation takes a first step towards addressing the above challenge by providing a formal model to describe the hybrid planning problem. The model splits the problem of hybrid planning into *four subproblems:* (i) *Problem-Planner Generation* (PRBSEL), i.e., determining the set of well-formed subproblems of the initial planning problem, (ii) *Planner Assessment* (PLRAST), i.e., assessing constituent approaches (of a hybrid planner) on the generated problems, (iii) *Graph Construction* (GPHCON), i.e., deciding what plans to combine, and (iv) *Path Selection* (PTHSEL), i.e., selecting the optimal sequence of these plans. Moreover, the model connects these subproblems back to PLNCRD and PLNSEL. As explained later, this formalization helps inform, analyze, and compare hybrid planner implementations as approximations of the ideal solution to each subproblem.

Furthermore, to demonstrate practicality (i.e., the potential of this formal model to represent and analyze realistic instances of hybrid planning) of the formal model, we use the model to explain our approach in Chapter 4, and analyze an existing instantiation of hybrid planning [79] in Chapter 6. Our analysis not only provides insight into the strengths and weaknesses of these instantiations, but also highlights the (often implicit) assumptions behind the designs. Moreover,

we use the central concepts of the formal model (a posteriori utility, preemption, and timeliness) to formalize specific conditions for the hybrid planning instantiation to be valid. Violating these conditions will result in an invalid instantiation, which can lead to loss of utility and potential underperformance of the instantiation compared to its constituent approaches.

3.1 Summary of the Formal Model

This section summarizes the central concepts involved in our formalization of the problem hybrid planning, which is presented later in Sections 3.2 and 3.3. The section uses the cloud-based system discussed in Section 1.1 as an example to explain concepts from the formalism; nevertheless, the applicability of the formal model is not limited to any particular system.

Intuitively, hybrid planning refers to finding a concatenation of plans determined by different planners to solve a planning problem; the concatenated plan is known as a *hybrid plan*. To explain further, suppose from the current state of a system, we model all post-execution paths that result from executing (potentially) different hybrid plans; hybrid plans can be different if their constituent plans are different (e.g., determined by different planners) or the same set of constituent plans are concatenated in a different order. Hybrid planning is then equivalent to finding the hybrid plan that leads to the execution path leading to the solution of the planning problem.

The execution paths represented in such a model include all post-execution states resulting from (possibly non-deterministic) state transitions. Moreover, once the post-execution details are known, we can assume that planning time for a plan by a planner is also known. A model, such as this, which assumes that we know the post-execution paths and planning times is referred to as the *a posteriori* semantics for transitions. In contrast, an *a priori* semantics explicitly models uncertainty both in state transition and planning time representing the fact that before executing a sequence of transitions there is uncertainty in the action outcomes and planning times.

The *a posteriori* formalization is an appropriate theoretical model to formally define the planning problem and its solution (as detailed later in this chapter). The *a posteriori* formalization helps to characterize the hybrid planning problem and its solution, but solving a hybrid planning problem requires dealing in the *a priori* semantics because when determining a plan to be executed under uncertainty, it must be taken into account to estimate the effectiveness of the plan. Hence, as we will see in Chapter 5, this thesis also analyzes hybrid planning using *a priori* semantics.

Figure 3.1 further illustrates the difference between *a priori* and *a posteriori* models in context of the cloud-based system discussed in Section 1.1. Suppose at time t_0, response time for the system is above a predefined threshold (i.e., state S). In response to this emergency situation, suppose the system adds a server at time t_0 to bring the response time below the threshold. However, due to the uncertainty in the request arrival rate, workload on the system can change (i.e., increase or decrease). If the workload increases further then even after adding the server, at time t_1, suppose the response time still remains above the threshold (i.e., state S_1). In contrast, if the workload decreases, the response time can be below the threshold at time t_1; suppose the state is S_2. As shown in Figure 3.1, due to uncertainty in request arrival rate, there are two possible states at time t_1 in the *a priori* model. In contrast, in the *a posteriori* model, once the system's actual state has been observed at time t_1, we know whether the system is in state S_1 or

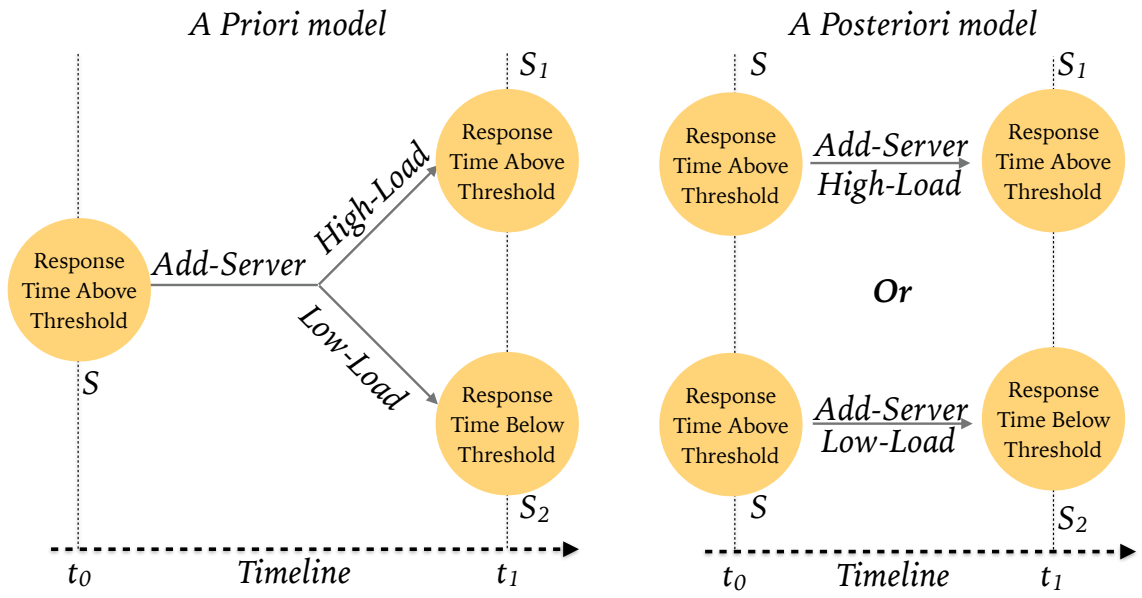


Figure 3.1: Due to uncertainty, there are two executions (i.e., state-transition) possible in the *a priori* model. In contrast, only one execution is possible in the *a posteriori* model since we know the resulting state post-execution. Therefore, in the *a posteriori* model, only one execution is represented, i.e., the one that has been realized post-execution.

The formal model decomposes the hybrid planning problem into four computational subproblems. Such a decomposition helps us tackle the complexity of hybrid planning by providing a framework to solve hybrid planning problems using composable solutions of subproblems. We start by introducing concepts that will be used later to explain the four subproblems.

A Planning Problem: A planning problem is a tuple consisting of: (a) the initial (i.e., current) state of the system and environment, (b) a set of possible adaptation actions (e.g., *addServer*), (c) a behavioral model of the environment informing which action (e.g., request arrival rate) will be taken by the environment for a given state, (d) a transition function informing the resultant state when a particular adaptation and environment action is applied to a given state, and (e) a utility function (say, U_e) that takes a plan as input and returns a real number indicating quality of a plan determined by a planner that takes a planning problem as an input.[1] Solving a planning problem (using a planner) means determining the plan that optimizes U_e.

Hybrid Planning and a Hybrid Plan: Given a planning problem and a set of planners, a hybrid plan is a sequence of plans, possibly generated by different planners, that optimized U_e from the planning problem; this sequence is known as a hybrid plan. As discussed later in Section 3.3.1, hybrid planning is equivalent to finding a path consisting of nodes and edges in a *reachability graph*. An example of such a path is illustrated in Figure 3.2.

[1] U_e is referred to as an *a posteriori* utility function, i.e., one that returns utility yielded after executing a plan. This is different from the traditional understanding of a planning problem, where a utility function returns the *a priori* (i.e., expected) utility and solving the problem means maximizing this utility.

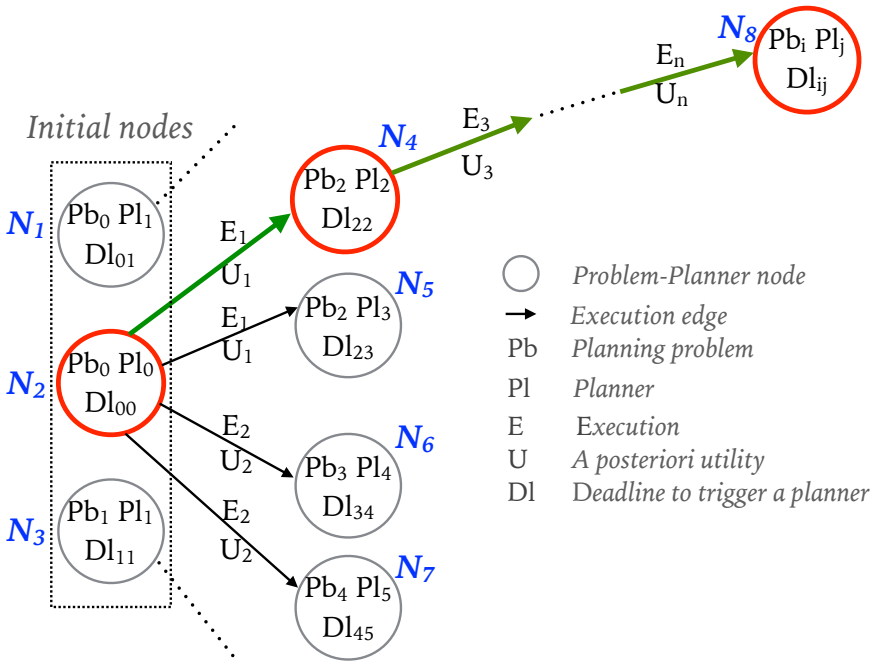


Figure 3.2: An example of a Reachability Graph. Red/green highlights indicate the selected path.

A Reachability Graph: A reachability graph is a directed graph, consisting of a set of nodes, a set of edges, and a set of initial nodes. Figure 3.2 shows a reachability graph with the nodes (e.g., N_4), the edges (e.g., E_1 joining nodes N_2 and N_4), and the initial nodes (i.e., N_1, N_2, and N_3).

A Node in a Reachability Graph: A node is a tuple (Pb, Pl, Dl) consisting of a planning problem (Pb), a compatible planner (Pl) that can solve Pb, and deadline (Dl), which is the planning time for Pl to solve Pb. A problem-planner node (say (Pb', Pl', Dl')) in a graph indicates that Pb' is compatible (i.e., could be solved) with Pl'.

An Edge in a Reachability Graph: An edge originating from a node (Pb', Pl', Dl') represents a complete or partial execution of the plan (say π') determined by planner Pl' for problem Pb'.[2] In the context of the cloud system, assuming π' has two actions (e.g., add a server then increase the dimmer), an example of a partial execution is to execute only the first action (i.e., add a server), whereas the complete execution would refer to executing both the actions in the plan. The utility of an edge is the same as the utility of the corresponding (full or partial) execution. Initial nodes V^i indicate the potential starts of executions in a reachability graph. Informally, an edge between a pair of nodes N_a and N_b indicates that the plan for the problem-planner pair in N_b can take over execution from the plan (after full/partial execution) for the problem-planner pair in N_a; therefore edges in a graph represent potential solutions to the planning coordination problem (PLNCRD).

[2]Before execution (i.e., *a priori*), the outcome of a plan might be uncertain due to uncertainty in an operating domain (e.g., cloud-based system). Conversely, the *a posteriori* notion of utility has no uncertainty, since the outcome of a plan/action execution is deterministic post-execution. Therefore, a reachability graph does not contain *a priori* uncertainty: edges represent actual executions rather than expected.

An edge can be constructed between N_a and N_b if and only if the two reachability conditions are met:

- $Timing$: the plan in N_b should be ready once the execution comes to it. Hence, the planner for N_b has to be invoked with enough time (i.e., deadline) for the planner to solve the problem before it is needed.
- $Preemption$: after executing the plan from N_a, the system should reach the initial state of the planning problem in N_b. Only then does the plan for N_b take over from the plan for N_a.

Satisfying the reachability conditions is necessary for the *correctness* of a hybrid planning instantiation; intuitively, by a correct instantiation we mean the one that performs a seamless transition between its constituent plans. For a smooth transition from plan π_a to π_b, the timing condition is necessary, since if it is violated π_b will not be ready to take over from π_a and the preemption condition is necessary, since it if it violated the transition state will not be found in the π_b; in either case a transition between the two plans will fail. In the rest of the thesis, the term "correctness" is used here to refer to the validity of a hybrid planning instantiation (i.e., one that satisfies the reachability conditions).

Given a planning problem (say Pb_0) and a set of planners (say $\{Pl_0, Pl_1, Pl_2...\}$), now we will walk through the corresponding reachability graph.

Suppose Pb_0 is compatible with planners Pl_0 and Pl_1 that solve Pb_0 in (the worst-case) time Dl_{00} and Dl_{01} respectively. Therefore, the reachability graph (in Figure 3.2) has initial nodes N_1 and N_2, indicating that Pb_0 could be solved by Pl_0 and Pl_1.

Figure 3.2 shows another initial node (i.e., N_3) consisting of a modified problem (say Pb_1) of Pb_0, a planner (say Pl_1) compatible with Pb_1, and deadline dl_{11}. A planning problem could be relaxed (i.e., modified) to reduce the state space to be searched for planning; in other words, only a smaller part of the state space is considered when planning for a relaxed problem. Therefore, such a modification helps reduce planning time. However, solving the relaxed (low-fidelity) problem is likely to result in a sub-optimal plan, since the optimal plan might exist in the part of the state space that has not been considered while planning. To exemplify, in the case of a constraint violation for the cloud system, suppose Pb_0 considers all the adaptation actions for planning. In contrast, assume Pb_1 considers only a subset of actions (e.g., *addServer*, *increaseDimmer* and *divert_traffic*). In this example, planning with fewer actions will reduce planning time due to the reduced search space; however, the plan is likely to yield a lower utility compared to the one determined using all the actions.

Using node N_2 as an example, we explain how a graph is expanded from an initial node. Suppose planner Pl_0 determines a plan π_{00} for problem Pb_0. An edge originating from node N_2 would indicate a full or a partial execution of plan π_{00}. Since different partial executions from node N_2 are possible for plan π_{00}, there are multiple outgoing edges from N_2 indicating various executions such as E_1 and E_2. Suppose execution E_1 takes the system to the initial state of problem Pb_2, which has two compatible planners Pl_2 and Pl_3. To capture a combination of Pb_2 with both the compatible planners, the graph has nodes N_4 and N_5. However, an execution (e.g., E_2) could also lead a system to different planning problems. Suppose execution E_2 of plan π_{00} takes the system to planning problem Pb_3. Node N_6 represents the combination of problem Pb_3 with a compatible planner Pb_4. Since problem Pb_3 could be modified, there is another reachable node N_7 that represents a modified problem (i.e., Pb_4) along with a compatible planner (i.e., Pb_5).

Using these possibilities the graph is expanded from all the nodes (including the initial nodes N_1 and N_3).

A path (consisting of nodes and edges) in a graph refers to a combination of plan executions for different problem-planner nodes in the path. Such a combination that optimizes *a posteriori* utility (i.e., U_e) is a hybrid plan. In other words, hybrid planning is about finding a sequence of problem-planner nodes that yields optimal U_e. Since an optimal path informs the planning (i.e., combination of problem-planner) approaches (and their ordering) that needs to be invoked to solve a planning problem, the path solves the planning selection problem (PLNSEL) (i.e., deciding which planning approach(es) should be invoked to solve a planning problem and when to stop using one and start using another)

To explain hybrid planning in the context of the cloud system introduced in Figure 1.2, suppose RBA and MDP planning is used to instantiate hybrid planning. Assume a situation in which the system needs to adapt in response to a constraint violation (say, planning problem Pb). To handle this situation, there could be different paths in the reachability graph representing various combinations of the two planning approaches. Examples of two such paths are: (a) use RBA planning alone, and (b) initially use RBA planning but later switch to MDP planning. Assuming the second path yields highest utility among all the possible paths then this path would be selected to formulate a hybrid plan.

To deal with the complexity of constructing a reachability graph and identifying an optimal path, the formal model breaks the problem of hybrid planning into four subproblems.

- *Path Selection* (PTHSEL): The Path Selection subproblem is, informally, to find a path in a reachability graph that yields the highest utility. As discussed earlier, this path implicitly solves the planning selection problem (PLNSEL).
- *Reachability Graph Construction* (GPHCON): The Graph Construction subproblem is to evaluate the reachability conditions between each pair of nodes in a reachability graph. If the reachability conditions are satisfied for a pair, the nodes are connected through an edge to construct the graph ensuring a smooth transition between plans, i.e., solves the planning coordination problem (PLNCRD).
- *Planner Assessment* (PLRAST): The Planner Assessment subproblem is, given a set of compatible problem-planner pairs, for each pair rate the performance of the planner on the respective problem. The metrics for the rating are execution utility (i.e., quality) and planning time (i.e., timeliness). These ratings are used by PLRAST and PTHSEL to construct the edges between nodes and to find an optimal path in a reachability graph, respectively.
- *Problem-Planner Generation* (PRBSEL): The problem-planner subproblem is, given a planning problem and a set of planners, generate compatible and relevant problem-planner pairs. Relevance means that problems are generated for a particular time t such that the problems' initial states are at time t. These pairs are used by PLRAST for assessment.

3.2 Foundational Concepts

This section defines the basic concepts needed to formalize hybrid planning.

Definition 3.2.1 (State)**.** A *state* (s) is a vector of values of the system's and environment's variables. Time is considered as a state variable. We denote the set of states by S.

Since time is a state variable, S is a potentially infinite set. Moreover, time imposes an implicit total order on states in S. By default we consider time continuous, and also allow its discretization.

Definition 3.2.2 (State time)**.** The function τ returns the time value of a state. Formally, $\tau : S \to \mathbb{R}_{\geq 0}$.

Definition 3.2.3 (Utility of state)**.** The *utility of a state* is a real number defined as a function $U_s : S \to \mathbb{R}$ that maps state s to its valuation.

Our formalization propagates the value of utility from the ground truth (utility of a particular state in a real system) to abstract notions that the MAPE loop manipulates (e.g., planners). We use this notion to create a formal underpinning for every planning decision of a self-adaptive system, rooted in the utility of the states that this action leads to. Although an obstacle for direct implementations, this model is beneficial for formalizing the problem and its idealized solution. In fact, by using information about the future (e.g., how much utility is accrued from an execution), we can establish a theoretical baseline for evaluation of downstream engineering solutions. These solutions will use relaxations (e.g., *a priori* utility or expected utility) of our utility notion to construct approximations of the idealized solution as discussed in the two instantiations of hybrid planning discussed in Chapter 4 and Chapter 6.

Definition 3.2.4 (Execution)**.** An *execution* e is a potentially infinite sequence of states: $e \stackrel{\text{def}}{=} \langle s_1, s_2, \dots \rangle$. We designate a set of executions by E.

We allow infinite executions to model reactive systems that can run indefinitely. To model goal-oriented systems, the above sets and sequences can be made finite.

Definition 3.2.5 (Partial execution)**.** For an execution $e \stackrel{\text{def}}{=} \langle s_1 \dots s_j \dots s_n \rangle$, a *partial execution* e_p^j is a prefix of e ending with s_j where $1 \leq j \leq n$. That is, $e_p^j \stackrel{\text{def}}{=} \langle s_1, \dots, s_j \rangle$.

Definition 3.2.6 (Duration of execution)**.** The *duration of an execution* is a function $D : E \to \mathbb{R}_{\geq 0} \cup \infty$ that maps execution $e \stackrel{\text{def}}{=} \langle s_1 \dots s_j \dots s_n \rangle$ to its duration $\tau(s_n) - \tau(s_1)$. For an infinite execution $e \stackrel{\text{def}}{=} \langle s_1, s_2, \dots \rangle$, function D will return infinity (i.e., ∞).

For infinite executions, the duration will be infinite; however, for partial executions the duration will be a finite value.

Definition 3.2.7 (Utility of execution)**.** The *utility of an execution* is a real number defined as a function $U_e : E \to \mathbb{R}$ that maps execution e to its valuation.

Even though an execution is a potentially infinite sequence of states, we assume its utility would be a finite value. This assumption is needed so that we can use utility values for comparison of planners. As an example, suppose U_e is defined as the utility of a state with the maximum utility (among all the states in the execution); here, the utility of all executions would be a finite value. In our model, we abstract away the particular function representing the utility of executions.

Definition 3.2.8 (Transition, action, and event)**.** *State transitions* are characterized by a transition function $T : S \times A \times Z \to S$, where A is a set of the *system's actions*, and Z is a set of *external events*. An element $\perp$ represents an empty action/event and is present in both sets: $A \cap Z = \{\perp\}$.

A self-adaptive system is characterized by controllable actions (e.g., adding/removing a server) and uncontrollable events (e.g., an arrival of a user request). Both actions and events cause state transitions. T captures both asynchronous (some action along with $\perp$ event, or vice

versa) and synchronous (neither the action nor the event are $\perp$) interactions of the system and its environment. To represent several actions/events happening at the same time, one can use composite actions/events. Since the model is *a posteriori*, the outcomes of actions and events are deterministic after a transition takes place. Thus, T is a function instead of relation.

In our model, we only consider *Markovian* domains, i.e., those where the conditional probability distribution of future states of a system depends only upon the present state, not on the sequence of states that preceded it [78]. We also assume that all transitions take time: the future state's time is always larger than that of the previous state's.

Definition 3.2.9 (Plan)**.** A *plan* π is a partial function $\pi : S \nrightarrow A$. A mapping from a state $s \in S$ to an action $a \in A$ suggests a to be executed in s. We denote a set of plans as Π.

Definition 3.2.9 of plan is general enough to capture different types of plans. For instance, some planners (e.g., deterministic planners) determine plans that do not have an action corresponding to every state in the state-space. Therefore, action for such a state ($s \in S$) will be undefined by function π (i.e., $s \notin \text{dom}(\pi)$). In contrast, universal plans (i.e., total functions), such as MDP policies suggesting an action for every $s \in S$, can also be represented by partial function π (since a total function is a special case of partial functions).

Definition 3.2.10 (Environment)**.** The *environment* is a total function $o : S \to Z$ encoding which event happens in each state. A set of possible environments is designated as O.

Definition 3.2.11 (Realization)**.** The *realization* function $\mathcal{R} : \Pi \times O \times S \to E$ maps a plan π, an environment o, and an initial state $s^i \in S$ to the execution e produced by the system executing in those conditions. That is, $\mathcal{R}(\pi, o, s^i) = e$.

Definition 3.2.12 (Partial realization)**.** The *partial realization* function $\mathcal{R}_p : \Pi \times O \times S \times S \to E$ maps a plan π, an environment o, an initial state $s^i \in S$, and an end state $s^n \in S$ to the partial execution $e_p^n \stackrel{\text{def}}{=} \langle s^i, \ldots, s^n \rangle$ produced by the system executing in those conditions. That is, $\mathcal{R}_p(\pi, o, s^i, s^n) = e_p^n$ such that $s^i, \ldots, s^n \in dom(\pi)$.

We use partial realizations to represent carrying out a plan for a given part of the state space from s^i to s^n, after which execution switches to another plan. Thus, a realization is a potentially infinite sequence of partial realizations.

Definition 3.2.13 (Utility of a plan)**.** The *utility of a plan* is a function $U_\pi : \Pi \times O \times S \times S \to \mathbb{R}$ that, given an environment o, the initial state s^i, and the end state s^n of execution e_p^n of plan π, returns the utility of that plan's realization. That is, $U_\pi(\pi, o, s^i, s^n) \stackrel{\text{def}}{=} U_e(\mathcal{R}_p(\pi, o, s^i, s^n))$. If plan π results in an infinite execution, the end state is specified as ∞.

This function can be used to calculate utility of the full or a partial execution of a plan. If the end state of an execution is also the goal state of the planning problem corresponding to the plan, the function U_π returns utility of full execution of a plan. However, if the end state is some intermediate state of an execution, then U_π returns the utility of partial execution, which ends at this state.

The function can also be used to calculate utility of plans corresponding to planning problems with no explicit goal states. For instance, MDP planning could be done for an infinite horizon with no explicit goal states. For such problems, execution of an MDP policy can happen indefinitely, thereby it is not always possible to specify the end state of a plan execution. In case of full execution ∞ needs to be specified as the end state; however, for a partial execution of such a policy, the end state of the execution is to be specified.

By linking the utilities of plans and executions, we have extended the ground truth to reasoning about planners. This bridge lets us establish utility-based comparison of concepts that normally exist before execution happens. Thus, we trade direct implementability of this model for a theoretical way of putting value on planning decisions.

Definition 3.2.14 (Planning problem)**.** A *planning problem* ξ is a tuple (S, s^i, A, T, o, U_e), where $s^i \in S$ is the initial state. Solving a planning problem means providing a plan that maximizes U_e for given S, s^i, A, T, and o. A set of planning problems is denoted by Ξ.

Self-adaptive systems have flexibility in the way an adaptation scenario is represented as a planning problem. For instance, the system can choose its lookahead horizon (the time bound on the future states to consider): should it consider a future of one minute or one hour ahead of the current moment [108]? Therefore, the same scenario can be represented as different problems. The space of such problems is encoded as Ξ.

Definition 3.2.15 (Planner)**.** A *planner* is a function $\rho : \Xi \to \Pi$ that solves a planning problem ξ and produces a plan π. We designate a set of potentially infinite planners by Ψ.

In this thesis, we use the term "planner" and "planning approach" interchangeably. Both the terms refer to the black box that takes a planning problem as an input and returns a plan. This black box encapsulates planning aspects such as the representation (e.g., relaxation) of a planning problem, the planning tool that implements a planning algorithm/heuristic and its configuration options. Two instances of the same planning tool will be considered as the different planners (i.e., planning approach) if their configuration options and/or representation of the input planning problems is different. To exemplify, as mentioned earlier in Section 6.3, the second case study uses the same MDP planning tool (i.e., PRISM [69]) for reactive and deliberative *planning*, but they have a different representation of a planning problem; reactive planning uses a shorter planning horizon and only a subset of adaptation actions compared to the deliberative planning.

Most planner implementations allow numerous customizations (e.g., value iteration and policy iteration used to solve an MDP). We formalize these customizations as individual planners without loss of generality: each planner is evaluated independently and with respect to compatible problem ξ (cf. Definition 3.2.16).

Definition 3.2.16 (Problem-Planner Compatibility Relation)**.** A problem ξ and a planner ρ are *compatible* if ρ can solve ξ, denoted $(\xi, \rho) \in \Upsilon$, where $\Upsilon : \Xi \leftrightarrow \Psi$ is a problem-planner *compatibility relation*. Given problem ξ, $\Psi^\xi \subseteq \Psi$ is a set of planners that are compatible with ξ (i.e., $\{\xi\} \lhd \Upsilon = \Psi^\xi$)

In practice, some planners (e.g., deterministic ones) are not applicable to problems that do not match their input format or algorithmic parameters (e.g., ones with non-deterministic transitions). Conversely, several planners can often solve the same problem. For instance, several decision-making approaches are applicable in self-adaptive cloud systems: rule-based adaptation (RBA) [27], case-based reasoning (CBR) [107], MDP planning [85]. Υ encodes such restrictions, naturally constraining the domain of planner functions in Definition 3.2.15.

Let us illustrate these definitions with the cloud-based system from Section 1.1. To adapt that system, suppose Ψ contains two planners: one based on Markov Decision Processes (MDP, ρ_{mdp}) and the other on a deterministic planner (Deterministic, ρ_{det}). The planning problem set (Ξ) consists of instances of ξ_{det} and ξ_{mdp}. To exemplify, ξ_{mdp} considers (probabilistic) uncertainty in request arrival rate whereas ξ_{det} ignores uncertainty in request arrival rate by assuming the request

arrival rate remains constant at the current value.

For planning problem ξ_{mdp}, ρ_{mdp} is slow to plan but provides high-quality plans, since it considers uncertainty in environment o. In contrast, ρ_{det} determines a plan quickly, but the plan is likely to be lower in quality (compared to the MDP plan), since it solves a relaxed problem instance (i.e., considering a smaller state space) ξ_{det} of ξ_{mdp} by ignoring uncertainty in the environment. To carry out hybrid planning, the self-adaptive system will find the best combinations of ρ_{det} and ρ_{mdp} selecting appropriate $\xi \in \Xi$ and assigning them to the highest-utility planner in advance (to account for their planning delays).

3.3 Decomposition of the Hybrid Planning Problem

We start with one of the central concepts of the thesis — a hybrid plan.

Definition 3.3.1. [Hybrid plan] A *hybrid plan* is a total function $\omega : S \to A$ based on partitioning of the full state space S into n partitions S_i, each governed by a planner ρ_i.

$$\begin{aligned} &\exists n : \mathbb{N} \cdot \forall i : 1..n \cdot && \text{There exists a number} \\ &\quad \exists \pi_i : \Pi, S_i \subseteq S \cdot S_i \neq \varnothing \ \wedge && \text{of plans and state partitions} \\ &\quad (\bigcup_{j:1..n} S_j = S) \wedge (\bigcap_{j:1..n} S_j = \varnothing) && \text{that partition the state space} \end{aligned}$$

such that the three following conditions hold:

Condition 1: actions in partitions are governed by their respective plans:

$$\forall s : S_i \cdot \omega(s) = \pi_i(s) \wedge dom(\pi_i) \subseteq S_i,$$

Condition 2: partitions are totally ordered in time:

$$\begin{aligned} &\forall k, l : 1..n, s_1 : S_k, s_2 : S_l \cdot \\ &\quad k < l \implies \tau(s_1) < \tau(s_2), \end{aligned}$$

Condition 3: the plans are solutions to some planning problems:

$$\exists \xi_i : \Xi,\ \rho_i : \Psi \cdot\ \pi_i = \rho_i(\xi_i).$$

For example, suppose there is an emergency situation such as a constraint violation. If hybrid planning is instantiated using RBA and MDP planners, let's assume the best combination is to execute the RBA plan until the MDP policy is ready. In such a case, there would be two partitions: S_{rba} containing the states whose time value is smaller than the time required to get the MDP policy ready, and S_{mdp} containing the states with time value after the policy is ready. A plan from ρ_{rba} would provide actions for states in S_{rba}, and a plan from ρ_{mdp} would provide actions in S_{mdp}.

The rest of this section describes the steps required to obtain hybrid plan ω through S_i, π_i for $i : 1..n$ in Definition 3.3.1.

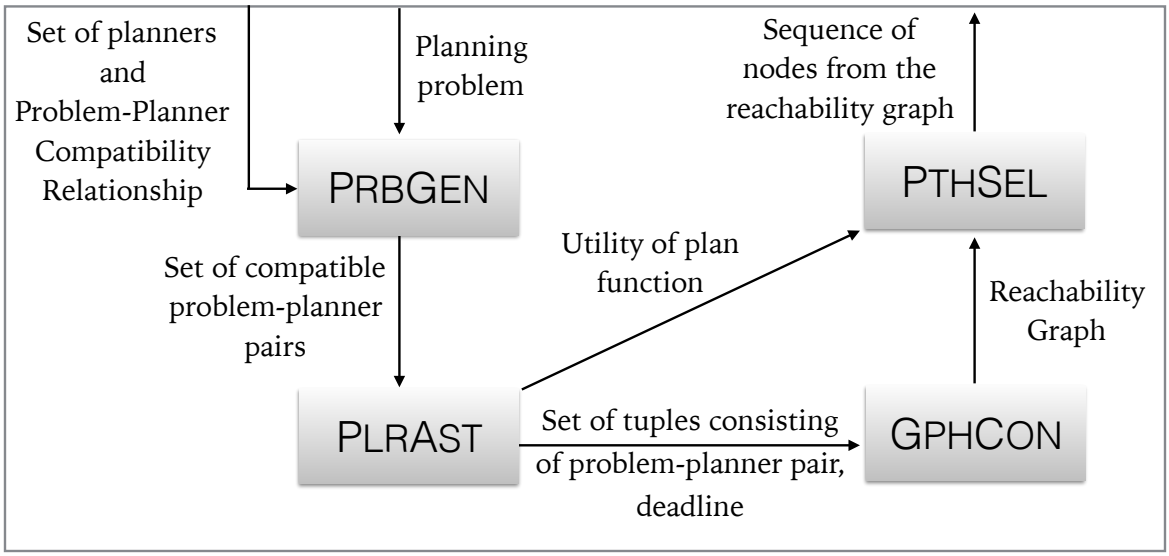


Figure 3.3: Decomposition of the hybrid planning problem.

Definition 3.3.2 (Hybrid Planning Problem)**.** The *Hybrid Planning Problem* (HPP) is, given an initial planning problem ξ^i, a set of planners Ψ, and a compatibility relation Υ, find a hybrid plan that maximizes the utility of execution U_e.

The formal framework decomposes HPP into four subproblems (starting from the end, see Figure 3.3):

1. *Path Selection* (PTHSEL): what is the sequence of planner invocations on planning problems that yields the maximum utility?
2. *Reachability Graph Construction* (GPHCON): what planning problems are reachable by solving other problems?
3. *Planner Assessment* (PLRAST): what are the quality and timeliness characteristics of each planner on a given planning problem?
4. *Problem-Planner Generation* (PRBSEL): what planning problems can be solved at any given time, given their compatibility with planners?

3.3.1 Path Selection

The *Path Selection* (PTHSEL) subproblem is, informally, to find the sequence of plans from different planners that yields the highest utility. This sequence of plans constitutes a hybrid plan ω according to Definition 3.3.1. The total number of plans in the sequence is *n* (possibly infinite). A plan π_i (i-th in the sequence), given an environment from the planning problem, can be realized

to an execution, which in turn can be mapped to a sequence of states. Therefore, each plan π_i can be mapped to the sequence of states.

The question posed in PTHSEL is where to end one execution in the sequence of states, and begin another one. To answer this question, one has to provide a sequence of partitions S_i that determine each plan's/execution's boundary, according to Conditions 1 and 2 of Definition 3.3.1. To satisfy Condition 3, we map these partitions to planning problems and planners. To construct this mapping, we formalize an input structure to PTHSEL that encodes potential choices of problems and planners.

Definition 3.3.3 (Reachability graph)**.** A *reachability graph* Γ is a directed graph defined as a tuple $(V, \mathcal{E}, V^i)$. V is a set of nodes, where each node v is a tuple (ξ, ρ, d) combining a problem, a planner, and a deadline $d \in \mathbb{R}_{\geq 0}$, which is the time instant when ρ needs to be invoked on ξ (detailed in Section 3.3.3). The set of edges ($\mathcal{E} \subseteq V \times V$) describes reachability between nodes in terms of executions: an edge $\epsilon = (v_1, v_2)$ means that full/partial execution of plan $v_1.\rho(v_1.\xi)$ with $v_1.\xi.o$ reaches s^i of $v_2.\xi$ (formally defined in Sec. 3.3.2). Initial nodes ($V^i \subseteq V$) indicate the potential starts of executions in Γ.

Paths (i.e., sequences of edges) in Γ mimic executions of the system, guided by a sequence of plans. A path indicates a sequence of switches between planning problems, which can be mapped to plans π_i and partitions S_i in Definition 3.3.1. PTHSEL selects a path based on the utility of its execution. We introduce several auxiliary concepts to express that selection.

Definition 3.3.4 (Edge execution)**.** *Edge execution* is a function $\eta : \mathcal{E} \to E$ that maps an edge ϵ to its partial execution e. Edge $\epsilon = (v_1, v_2)$ maps to a partial execution from the initial state of planning problem in the first node to the initial state of the planning problem in the second node: $\eta(\epsilon) = \mathcal{R}_p(v_1.\rho(v_1.\xi),\ v_1.\xi.o,\ v_1.\xi.s^i,\ v_2.\xi.s^i)$.

Definition 3.3.5 (Path execution)**.** *Path execution* is a function $\eta : \mathcal{E}^n \to E$ that maps a path to its execution. A path $\kappa = \langle \epsilon_1, \ldots, \epsilon_n \rangle$ maps to an execution composed of concatenation of edge executions: $\eta(\kappa) = \eta(\epsilon_1) \frown \ldots \frown \eta(\epsilon_n)$.

The utility of a path in Γ builds upon the utilities of its edges, which in turn build on the utilities of its executions.

Definition 3.3.6 (Utility of edges and paths)**.** The *utility of an edge* ϵ is a function $U_\epsilon : \mathcal{E} \to \mathbb{R}$ that maps ϵ to the utility of the edge's execution. Formally, $U_\epsilon(\epsilon) \stackrel{\text{def}}{=} U_e(\eta(\epsilon))$. Similarly, the *utility of a path* $\kappa = \langle \epsilon_1, \ldots, \epsilon_n \rangle$ is a function $U_\kappa : \mathcal{E}^n \to \mathbb{R}$ that is defined as $U_\kappa(\kappa) \stackrel{\text{def}}{=} U_e(\eta(\kappa))$.

Utility of an edge is same as the utility of the execution linked to that edge. To calculate utility of a execution, function U_π (Definition 3.2.13) is used. As discussed in Section3.3.3, this function is one of the outcomes of solving subproblem PLRAST.

Now we formalize the path selection (PTHSEL) subproblem.

Definition 3.3.7 (PTHSEL)**.** The *Path Selection* (PTHSEL) subproblem is, given a reachability graph Γ, to find a maximal-utility path starting from an initial node:

$$\text{PTHSEL}(\Gamma) \stackrel{\text{def}}{=} \arg\max_{\kappa \in \mathcal{E}^n} U_\kappa(\kappa).$$

Among the four subproblems of HPP, PTHSEL is the last one to be solved before a hybrid plan is ready. Once an optimal path is found, it translates into plans and partitions to define a hybrid plan. Due to the strict time ordering of partitions (Condition 3 in Definition 3.3.1), past

plans cannot be directly reused. As a result, whenever plans are switched, a new problem-planner node has to be created to represent a new partition of the hybrid plan.

3.3.2 Graph Construction

The *Graph Construction* (GPHCON) subproblem is to build reachability graph Γ to be used by PTHSEL. To this purpose, as Figure 3.3 shows, GPHCON uses output tuples from PLRAST. As Section 3.3.3 discusses, each tuple consists of a compatible problem-planner pair and an invocation deadline d.

The set of nodes for Γ is constructed as follows: for each tuple (ξ, ρ, d) from PLRAST, create a node consisting of (ξ, ρ, d). An edge is added between a pair of nodes v_1 and v_2, if and only if two conditions are met:

1. *Timing:* the plan in v_2 should be ready once the execution comes to it. Therefore, if $v_2.\rho$ takes planning time t to solve problem $v_2.\xi$ then $v_2.\rho$ needs to be invoked at least time t before the system reaches $v_2.\xi.s^i$. Mathematically, the only reason an early enough time could not be found is when $t < 0$. Therefore, for a node (ξ, ρ, d) that is the end of ϵ_n, the condition for ρ having enough time before its execution is: $d > \sum_{\epsilon_i \in \langle \epsilon_0 \dots \epsilon_n \rangle} D(\mathcal{R}_p(\epsilon_i))$, where function D returns duration of an execution, and edges $\epsilon_0 \dots \epsilon_{n-1}$ representing the system trajectory before ϵ_n.
2. *Preemption:* after executing the plan $\pi_1 = v_1.\rho(v_1.\xi)$ from v_1, the system should reach the initial state of the planning problem in v_2. Only then can the plan $\pi_2 = v_2.\rho(v_2.\xi)$ for v_2 take over from the previous plan. Formally, $v_2.\xi.s^i \in \mathcal{R}(\pi_1, v_1.\xi.o, v_1.\xi.s^i)$.

These two conditions are necessary for the correctness of a hybrid planning instantiation, which needs to ensure a smooth transition between plans. However, when a single planning approach (i.e., a special case of hybrid planning) is used to determine plans, these two conditions are automatically satisfied since no transition (between plans) happens during executions.

Now we are ready to define GPHCON formally.

Definition 3.3.8 (GPHCON)**.** The *Graph Construction* (GPHCON) problem is, given a set of tuples (ξ, ρ, d) where $\xi \in \Xi$, $\rho \in \Psi^\xi$, and d is the deadline corresponding to the problem-planner pair (ξ, ρ), find a reachability graph Γ with edges satisfying the preemption and timing conditions.

In practice, GPHCON is unlikely to be fully constructed for even moderately sized problems. Therefore, the goal of implementations is to build the most effective subgraph of Γ. For example, in the cloud-based self-adaptive system we can place the graph nodes at times of large expected changes in the incoming traffic. Edges can be made probabilistic (based on historic information and heuristics) to avoid an exhaustive traversal of the state space.

To build a reachability graph, the tuples are provided by PLRAST, which is discussed next.

3.3.3 Planner Assessment

The *Planner Assessment* (PLRAST) subproblem is: given a set of pairs consisting of compatible planning problem and planner, for each pair, evaluate the performance of the planner against the problem. The criteria for evaluation is timeliness and quality of planning.

As an outcome of the timeliness evaluation, PLRAST returns deadline $d \in \mathbb{R}_{\geq 0}$ for each problem-planner pair; these deadlines are used by GPHCON to evaluate the timing condition for reachability. As an outcome of the quality evaluation, PLRAST formulates the utility of plan function U_π (Definition 3.2.13 in Section 3.2) that returns utility of a full/partial execution of the plan corresponding to each pair; this function is used by PTHSEL to determine utility of each edge in a reachability graph.

Definition 3.3.9 (PLRAST)**.** The *Planner Assessment* (PLRAST) *problem* is, given a set of problem-planner pairs (ξ, ρ), where $\xi \in \Xi$ and $\rho \in \Psi^\xi$, find function U_π that returns utility of plans for all the pairs, and find the deadline d for each pair. The output of PLRAST is the function U_π and a set of tuples (ξ, ρ, d).

To solve PLRAST in practice, one needs to create algorithms to measure utilities and deadlines for planners. To the authors' knowledge, the majority of existing planner implementations do not provide up-front guarantees on either of these two characteristics. Therefore, two general approaches are possible: (i) design new planners with guarantees of quality and timeliness on given planning problems, and (ii) determine the characteristics of existing planners. While (i) is a challenging design problem, (ii) can be accomplished in a number of ways—from theoretical modeling to empirical profiling.

3.3.4 Problem Generation

The *Problem-Planner Generation* (PRBSEL) subproblem is to generate compatible and relevant problem-planner pairs. Compatibility means that in a pair, the problem and the planner belong to Υ. Relevance means that problems ξ are generated for a particular time t such that $\tau(\xi.s^i) = t$.These pairs, with deadline added, are eventually used by GPHCON as nodes of a reachability graph for a particular point in time.

The set of relevant problem-planner pairs is smaller than the set of all possible such pairs. At every moment, an infinite number of planning problems can be formulated: according to Definition 3.2.14, one can arbitrarily select the initial state, the subset of actions, the subset of the state space, the environment's choices of events, and the utility function. However, not all problems are relevant because as time passes, some initial states become unreachable, the state space evolves, and those problems become obsolete. Besides, not all problems and planners are compatible. PRBSEL helps in identifying the relevant problem set, which is fed into GPHCON.

So far we treated planning problems as timeless objects. While discussing PRBSEL we focus on time, which is encoded in states. This extension does not affect other subproblems of HPP, since time can be ignored by other subproblems.

Definition 3.3.10 (Time-bound Ξ)**.** Given a time t, a *time-bound planning problem set* Ξ_t is a set of planning problems whose initial states have time t: $\Xi_t = \{\xi : \Xi \mid \tau(\xi.s^i) = t\}$.

Each time-bound set Ξ_t needs to be filtered through the compatibility relation Υ: only problems that have at least one compatible planner need to be allowed. The result is a *filtered time-bound set of planning problems*: $\Xi_{t,\Upsilon} = \{\xi : \Xi_t \mid \Psi^\xi \neq \varnothing\}$.

For time instant t, PRBSEL generates a set of problem-planner pairs $\Xi_{t,\Upsilon}$. The inputs to PRBSEL are the initial problem ξ^i, the set of planners Ψ, and the compatibility relation Υ. The output of PRBSEL is a set of problem-planner pairs.

Definition 3.3.11 (PRBSEL)**.** The *Problem-Planner Generation* (PRBSEL) subproblem is, given the initial planning problem ξ^i, the set of planners Ψ, and the compatibility relation Υ, generate for each time instant t a set of pairs (ξ, ρ), where $\xi \in \Xi_{t,\Upsilon}$, and $\rho \in \Psi^{\xi}$.

The theoretical version of PRBSEL leaves most elements of ξ open, such as S and A. First of all, in practice, the time is often discretized, leading to fewer choices of time points for which to generate Ξ_t. Other elements are constrained by Υ: the existing planners consider certain types of states and actions, and it is often a matter of bounding them. Different bounds would then lead to different problem-planner pairs for a given time. Finally, further reduction of the number of considered pairs is possible using various heuristics. In the context of the cloud-based system, one could investigate more pairs at times when the environment is expected to rapidly change.

3.4 Applying the Formal Model

This section discusses potential applications of the formal model, the model's assumptions, and implementation barriers.

3.4.1 Formal Model Applications

The first potential application of the model is assisting in comprehending various instances of hybrid planners. The model provides a vocabulary to understand major design decisions in these instances: what planners are used, which problems they are run on, and how they are assessed and switched between each other. The model also helps determine assumptions made in a hybrid planner instance because creating an instance requires making the abstract formal concepts concrete, and assumptions are often required in this process.

The second possible application is analyzing whether a planner is a valid instantiation of hybrid planning. The analysis has three parts. First, whether the practical relaxations of the model's concepts rely on assumptions that are always satisfied in the instantiation. For instance, a hybrid planner may associate higher expected utility with using a particular planner, thus always preferring it to the other planners. However, if *a posteriori* utility is not always achievable with that planner, that violates the assumption and leads to suboptimal executions. Second, whether the timing and preemption conditions (Sec. 3.3.2) are satisfied. If there exists a possible scenario where these conditions are broken, then the interaction between planners may break down, thus making the instance invalid. Third, we can analyze the utility loss due to approximations of the utility functions described in Sec. 3.2. While some bounded utility loss may be acceptable, depending on the domain one can declare certain losses unacceptable, thus making the hybrid planner instance invalid.

A third application is comparison between planners. We can use the notion of utility as a uniform way to measure "goodness" of decisions made in implementations that address the subproblems. To estimate the difference between a benchmark planner and an implementation, our utility and reachability notions enable an *evaluation workflow*:

1. Implement a hybrid planner and a simulation of a system.
2. Obtain a hybrid plan ω and execute it with different o, logging complete execution traces.

3. Calculate utility of traces according to Definition 3.2.7.
4. Reconstruct a reachability graph for each scenario.
5. Perform what-if simulations to find (a) more optimal or timely paths, (b) other planning problems from $\Xi^*_{t,\Upsilon}$, (c) missing or inaccurate ϵ, and (d) other opportunities for improvement of ω.
6. The identified improvements characterize the delta between the empirical and benchmark utilities.

This is a repeatable evaluation procedure for hybrid planners, grounded in theoretical concepts defined by the formal model. It is applicable to a wide variety of planner combinations, including prior work on combining contingency plans [7]. Although such experiments can be computationally expensive, they yield valuable insight into the behavior and potential improvements of hybrid planners.

3.4.2 Assumptions

A distinctive feature of our formalization is its *parsimony*: we use only the essential concepts broadly applicable to planners, and we introduce the least restrictive assumptions that enable precise definition of subproblems. Below we summarize our assumptions to delineate the scope of the model.

Markovian domain: many of the domains explored by the self-adaptive community are assumed to be Markovian [110] [97] [4] (even though not always explicitly stated). Therefore, even with this assumption, the proposed formal model is applicable to various domains, particularly those investigated by the self-adaptive community.

No instantaneous transitions: in practice, no action or event takes exactly zero time to happen. Therefore, this assumption makes the formal model more applicable.

Instantaneous solutions to subproblems: we consider the delays of actions and planning itself, but not the delays of solving PTHSEL, GPHCON, PLRAST, and PRBSEL. This assumption holds if solving these problems takes negligible time compared to the time scale of planning and execution — or if the solutions are pre-computed offline.

Known planning time: currently most planners cannot provide a hard guarantee on their planning time. We hope, however, that extensive up-front profiling of planners can lead to empirical guarantees on average and worst-case planning times. In the future, relaxing this assumption will open a promising research direction—planners with predictable planning time.

Known and finite utility of states/executions: this assumption holds in most contexts of self-adaptation in software systems, except when experimental data are incomplete or inaccessible, or a utility function is not formulated in a convergent way. One example is complex cyber-physical systems where the physical state may be difficult to monitor and log entirely.

3.4.3 Implementation Barriers

Although hybrid planning has certain applications, implementing its general form (as defined in Section 3.1) to solve planning problems at run time is not practical because of three implementation barriers:

- **INFINITE-REACHABILITY-GRAPH**: In theory, since a planning problem could be modified in an unlimited number of ways and have an infinite planning horizon (i.e., no explicit goal/end state), a reachability graph (even with discretized time) could potentially have an infinite number of nodes and edges (connecting these nodes). Finding an optimal path in an infinitely large reachability graph is an intractable problem since it requires comparing utilities for an infinite number of infinitely long paths.
- **DELAY-IN-SOLVING-SUBPROBLEMS**: The model ignores the time to solve the four subproblems (i.e., PRBSEL, PLRAST, GPHCON, and PTHSEL). To solve a planning problem using hybrid planning, one needs to start with solving PRBSEL (i.e., generate problem-planner pairs), which is likely to take a non-negligible time since infinite combinations (due to problem modification) are possible for a planning problem and a given set of planners. Suppose PLNSEL could be solved in a negligible time by assuming the problem-planner pairs to be finite in number. Now, even with a finite number of problem-planner pairs, the time to construct a reachability graph is unlikely to be negligible, since the process requires solving PLRAST (i.e., rating the planner against the problem for each problem-planner node) and GPHCON (i.e., evaluating the reachability conditions for each pair of a problem-planner node). Moreover, finding an optimal path (i.e., PTHSEL) in the graph can take non-negligible time. Since hybrid planning aims at dealing with the run-time planning delay, an additional delay (due to solving the four subproblems) would further increase the complexity and decrease the effectiveness of applying hybrid planning to realistic self-adaptive systems.
- **REQUIRED-APRIORI-KNOWLEDGE-OF-EXECUTIONS**: Ideally, to solve a planning problem (at run time) using the hybrid planning approach, *a priori* (i.e., before determining and executing plans corresponding to problem-planner pairs) construction of the reachability graph is needed. This construction requires knowledge of utility of executions. However, this is a paradoxical requirement since the utility of an execution can't be determined without knowing a plan. Moreover, even if a plan is known, determining an execution path before executing the plan is impossible for systems operating under uncertainty. For instance, in our exemplar cloud-based system, due to uncertainty in the request arrival rate, it is hard to anticipate the state that would result from an execution of a tactic.

However, looking at the potential benefits of hybrid planning in balancing quality and timeliness, researchers have suggested various instantiations of hybrid planning [7, 79, 92, 113]. These instantiations make certain assumptions to make it tractable, and thereby, applicable to realistic systems. Often these assumptions are not explicitly stated, preventing one from understanding the strengths and the weaknesses of these instantiations. Moreover, even if assumptions are mentioned, it is difficult to compare different instantiations since there is no comprehensive comparison framework.

Our model provides a framework to systematically analyze existing instantiations of hybrid planning (or design new ones) in two ways. First, while analyzing/designing a hybrid planner, the model highlights how the implementation barriers are handled, describing the outcomes, assumptions, and limitations of the design choices. Second, the model breaks down the bigger design problem into four subproblems, allowing separate investigation of design decisions for each. Such an analysis, grounded in the formal model, not only highlights the implicit assumptions

made by designers of hybrid planners, but also gives confidence that all relevant challenges are addressed.

We use the formal model to analyze two instantiations of hybrid planning. In Chapter 4, we explain/analyze our approach to hybrid planning in the context of the formal model. Chapter 6 analyzes an instance of hybrid planning proposed by Mausam et al. [79]. In the two case studies, we will look at how (a) the implementation barriers have been handled, and (b) the four subproblems have been addressed. These analysis exemplify the first and the second application of the formal model as discussed in Section 3.4.1. In addition, the analyses demonstrate practicality of the formal model and provide examples of how to use the formalism for analyzing existing instantiations or designing a new one.

3.5 Summary

This chapter formalized the problem of hybrid planning and decomposes it into four computational subproblems. Moreover, the chapter links the four subproblems to the two fundamental challenges i.e., PLNCRD, and PLNSEL. As already discussed, there are several applications of this formal model. In particular, the formal definitions offered in this chapter can be used to analyze/evaluate existing solutions to hybrid planning or even instantiate a new one. However, the *a posteriori* nature of the formal model could initially be counterintuitive to users who want to apply the model to analyze/design hybrid planners. To exemplify this use of the model, the next chapter analyzes/evaluates our approach to solving the problem of hybrid planning in the context of the formal model. In addition, Chapter 6 analyzes/evaluates an instantiaton of hybrid planning not proposed by us but another researcher [79]. Users can use these two case studies as a handbook for applying the formal model.

Chapter 4

Solution to Hybrid Planning

Chapter 3 presented the formal model describing the problem of hybrid planning. The formal model breaks the hybrid problem into four subproblems as summarized in Table 4.1. Given a planning problem and a hybrid planner, solving a hybrid planning problem means finding a sequence of plans determined by different constituent planners such that the sequence yields the highest utility among all potential sequences. In the context of the formal model, this is equivalent to solving PTHSEL as illustrated in Figure 3.2.

This chapter presents our approach to solving the problem of hybrid planning. We explain the approach in the context of the formal model, giving us confidence that all the subproblems (i.e., PTHSEL, GPHCON, PLRAST, and PRBSEL) have been addressed; as discussed in Chapter 3, addressing GPHCON and PTHSEL implicitly solves PLNCRD and PLNSEL, respectively. In a way, this chapter presents and explains our approach in the theoretical context. For a practitioner interested in applying hybrid planning without going into theoretical details, later in Chapter 7, we provide informal guidelines to select an appropriate set of planners to instantiate hybrid planning.

To solve the problem of hybrid planning, essentially, we need to build a reachability graph with problem-planner nodes and edges connecting those nodes; once the graph is ready, the optimal path leads to the sequence of problem-planner nodes that gives a hybrid plan. As discussed in Chapter 3, to apply hybrid planning to a realistic system, the key challenge is to constrain a potentially infinite reachability graph (i.e., INFINITE-REACHABILITY-GRAPH). In our approach, the following assumptions are made to achieve this objective[1]:

- **TWO-LEVELS-OF-PLANNING**: Hybrid planning uses two levels of planning (i.e., reactive planning followed by deliberative planning) to solve a planning problem.[2] One level is provided by the reactive planner chosen from a set of reactive planners that determine a plan in a negligible time, and the other level is provided by the deliberative planner used to

[1]The thesis will revisit these assumptions and their impact on the scope of applicability of this thesis later in Chapter 8

[2]Levels of planning are differentiated by the amount of computation done to determine a plan. The computation is higher for deliberative planning compared to reactive planning, which is aimed at dealing with emergency situations (e.g., constraint violations) by providing plans in a negligible time; the quantification of negligible time is domain-dependent as discussed in Chapter 7.

Subproblem	Description
Path Selection (PTHSEL)	The Path Selection subproblem is, informally, to find a path in a reachability graph that yields the highest utility. As discussed earlier, this path implicitly solves the planning selection problem (PLNSEL).
Reachability Graph Construction (GPHCON)	The Graph Construction subproblem is to evaluate the reachability conditions between each pair of nodes in a reachability graph. If the reachability conditions are satisfied for a pair, the nodes are connected through an edge to construct the graph, ensuring a smooth transition between plans i.e., solves the planning coordination problem (PLNCRD).
Planner Assessment (PLRAST)	The Planner Assessment subproblem is, given a set of compatible problem-planner pairs, for each pair rate the performance of the planner on the respective problem. The metrics for the rating are execution utility (i.e., quality) and planning time (i.e., timeliness). These ratings are used by PLRAST and PTHSEL to construct the edges between nodes and to find an optimal path in a reachability graph, respectively.
Problem-Planner Generation (PRBSEL)	The problem-planner subproblem is, given a planning problem and a set of planners, generate compatible and relevant problem-planner pairs. Relevance means that problems are generated for a particular time t such that the problems' initial states are at time t. These pairs are used by PLRAST for assessment.

Table 4.1: Summary of the four subproblems of the hybrid planning problem.

instantiate hybrid planning.[3] As explained later, having only two levels of planning reduces the number of problem-planner nodes in a reachability graph, making the problem of hybrid planning tractable.

- **FINITE-HORIZON**: The planning problem has a finite planning horizon. An infinite horizon will lead to infinite nodes in the graph because time is a state variable according to the formal model.[4] This assumption restricts the number of problem-planner nodes in a reachability graph.
- **DISCRETE-STATE-VARIABLES**: The value of state variables (e.g., time) is discrete. Otherwise, a reachability graph would have infinite nodes.
- **DELIBERATIVE-PREFERRED**: For any planning problem, a deliberative plan always provides higher expected utility compared to a reactive. This implies that whenever a deliberative plan is ready for a planning problem, it is preferred over the plans determined by reactive planners. This assumption ensures that there can never be a path in a reachability graph that has deliberative planning followed by reactive planning, and thereby restricts the number of paths in a reachability graph.

Once the size of a reachability graph is constrained, the next challenge is to deal with the issue DELAY-IN-SOLVING-SUBPROBLEMS. For a practical application of hybrid planning, one needs to minimize the delay of solving the four subproblems (PRBSEL, PLRAST, GPHCON and PTHSEL). As explained later, the first three subproblems are simplified by the assumptions (listed above) in such a way that negligible time is consumed to solve them.

As already discussed, solving PTHSEL amounts to finding an appropriate reactive approach (from the given set) for a planning problem until the deliberative plan is ready; PTHSEL implicitly solves PLNSEL. To address PTHSEL in a negligible time, as discussed later in Section 4.2, we propose two approaches: condition-based (CB) and learning-based (LB). Each of these addresses PTHSEL in a negligible time (**NEGLIGIBLE-PLNSEL-DECISION-TIME**). The condition-based approach is useful for the domains where manually determining invocation conditions (e.g., emergencies) to invoke reactive planning is straight-forward at design time; invoking reactive planning on such conditions reduces the risk of inappropriate quick decisions. To pick a reactive approach using a condition-based approach, designers specify up-front conditions under which the reactive approach should be invoked [3, 51]; checking whether the conditions are satisfied (in the current state) can be done in a negligible time.

While the condition-based approach can be useful for effective hybrid planning, it suffers from three drawbacks: (a) it requires domain expertise to identify the conditions that should trigger reactive planning; (b) it relies on error-prone humans to identify the right and comprehensive conditions; and (c) such conditions do not transfer to other systems or domains, hindering reuse of hybrid planning.

To overcome these shortcomings, as detailed later, we propose a learning-based approach to solve PTHSEL. It is called learning-based since it maps a problem space to the solution space

[3] Although our approach can theoretically support any number of deliberative planners (to instantiate hybrid planning) as discussed later in Section 4.2.2, we restrict the thesis claims to a single deliberative planner since the validation is done using one deliberative planner.

[4] Although condition-based hybrid planning can support planning problems (e.g., represented as MDP) with infinite horizon, the learning-based approach requires the planning problem to have a finite horizon, as discussed later.

using training instances; this mapping is used at run time to solve a new problem instance [81]. Compared to explicitly finding an optimal path in a reachability graph, the approach approximates a solution to PTHSEL quickly since most of the computation is done offline. In the context of hybrid planning, the assumption (**INDUCTIVE-BIAS**) behind the application of the approach is that the reachability graphs for two "closely related" planning problems are also similar. In other words, for two similar planning problems, an effective combination of reactive and deliberative planning for one problem will also work for the other problem. Moreover, this approach does not require *a priori* knowledge of execution utility to approximate the solution to a hybrid planning problem, and therefore the implementation barrier REQUIRED-APRIORI-KNOWLEDGE-OF-EXECUTIONS (cf. Section 3.4) is not an issue.

Now, using the formal model, we provide a detailed analysis of our approach and how it approximates a solution to a hybrid planning problem.

4.1 Constructing a Reachability Graph

As explained earlier, hybrid planning requires constructing a reachability graph and finding an optimal path in the graph. The first step towards constructing a reachability graph is to choose the problem-planner nodes, and then to connect a pair of nodes if the reachability conditions (i.e., preemption and timing) are satisfied between the nodes.

4.1.1 Restricting the Number of Nodes

Assumptions FINITE-HORIZON, DISCRETE-STATE-VARIABLES and TWO-LEVELS-OF-PLANNING help to restrict the number of nodes in a reachability graph corresponding to a planning problem. Assumption FINITE-HORIZON restricts the number of nodes by assuming the planning horizon of all the planning problems to be finite (e.g., having an explicit goal/end state with a finite value for the state variable *time*). However, even with a finite horizon, theoretically a planning problem can have a reachability graph with an infinite number of problem-planner nodes if time (i.e., a state variable) is treated as a continuous variable. Therefore, assumption DISCRETE-STATE-VARIABLES is made to ensure that planning problems use a discrete notion of time.

Even after FINITE-HORIZON and DISCRETE-STATE-VARIABLES, given an infinite set of planners, there could be an infinite number of problem-planner nodes since (a) a planning problem can be modified in an unlimited number of ways (as discussed in Section 3.1), and (b) an infinite number of planning approaches can be used to solve each of these modified problems therefore, infinite combinations (i.e., nodes) of problem-planner are possible.

To address this problem, our approach assumes TWO-LEVELS-OF-PLANNING, which limits the number of (deliberative and reactive) planning approaches, thereby constraining the number of problem-planner nodes to a finite value. To explain, suppose hybrid planning is instantiated using a deliberative approach, and a set (say, $\mathcal{F}$) of N reactive approaches, which include a special reactive approach (i.e., ρ_{wait}) that, for any planning problem, always suggests to wait until the

deliberative plan is ready.[5] Because of assumption TWO-LEVELS-OF-PLANNING, (only) $1 + N$ kinds of problem-planner nodes are possible for a planning problem (say Pb) in a reachability graph. The first kind of node consists of the deliberative approach (suppose, MDP planning) and a modified version (say Pb_d) of Pb that is compatible with the planner. Realistically, even MDP (i.e., deliberative) planning might be solving a modified (i.e., relaxed) version of an original planning problem. For example, suppose Pb has uncertainty in both – action outcomes and observations of an underlying system state.[6] Now, if an MDP planner is used for deliberative planning, the input problem specification to the planner needs to ignore uncertainty in observations since MDP planners can handle uncertainty only in action outcomes.

In addition to the first node, there are N nodes corresponding to each reactive approach. From these N nodes, one node corresponds to wait planning (i.e., ρ_{wait}). Planning problems do not matter for ρ_{wait} since it always returns the same plan (i.e., to wait until the deliberative plan is ready). Since ρ_{wait} does not take anything from Pb into account, it does not matter what the planning problems are (except $\tau(Pb.s^i)$), so all nodes of this kind are equivalent at a given time point.

However, even after TWO-LEVELS-OF-PLANNING, solving PRBSEL (i.e., generating compatible and relevant problem-planner pairs) is likely to consume non-negligible time. Even with a finite (i.e., $1 + N$) number of possible problem-planner nodes for a planning problem, there could be a large number of nodes due to intermediate planning problems (i.e., problems resulting from partial executions of a plan) and corresponding modified problems. In the proposed approach, solving PRBSEL is not required, since the output of PRBSEL is used by PLRAST, which is not explicitly required to be handled, as explained later in Section 4.1.2.

4.1.2 Connecting the Nodes

Once the set of problem-planner nodes is finite, the next step is to connect the nodes in the reachability graph. To this end, the approach to solve the problem of hybrid planning needs to solve PLRAST and GPHCON. Given pairs of problem-planner nodes, a solution to PLRAST would return the utility of plan function (i.e., U_π defined by Definition 3.2.13), and a set of tuples containing the problem-planner pair and the deadline, as discussed in Section 3.3.3. The deadline is used by GPHCON to evaluate reachability between nodes and the partial utility function (returning the utility of a full/partial execution) is used by PTHSEL to find an optimal path.

As discussed earlier, a practical application of hybrid planning needs to deal with the issue of DELAY-IN-SOLVING-SUBPROBLEMS; however, solving (both) PLRAST and GPHCON in negligible time is infeasible. To solve PLRAST, for each problem-planner node, one needs to rate the planner with respect to the planning problem; the time consumed for this process is unlikely to be negligible. GPHCON is also likely to consume a non-negligible time. To solve GPHCON, nodes in a reachability graph need to be connected. This requires evaluating the reachability conditions between each pair of nodes in a reachability graph; if the reachability conditions (i.e., timing

[5]Using ρ_{wait} in combination with deliberative planning is equivalent to using deliberative planning alone. ρ_{wait} is required to ensure that hybrid planning does not underperform deliberative planning in cases when none of the other reactive approaches (in $\mathcal{F}$) provide a better plan than just waiting for the deliberative plan to be ready.

[6]POMDP-based planners can handle both kinds of uncertainty; however, these planners can be slow in determining plans compared to other planners such as MDP.

and preemption) are satisfied for two nodes, they are connected through an edge. However, this computation is likely to take a non-negligible time for realistic systems.

In our approach, no time is consumed to solve PLRAST because it is not handled explicitly. The approach does not solve PLRAST since the deadline and the utility function for full/partial executions are not required to approximate a solution to GPHCON and PTHSEL, respectively. The deadline is not required since, as explained next, reachability from reactive to deliberative planning nodes is approximated without knowing deadlines; in other words, deadline is not used to approximate a solution to subproblem GPHCON. The utility function is not required since the proposed condition-based or learning-based approach approximates a solution to PTHSEL without having knowledge of the function.

The approach solves GPHCON in a negligible time because it does not explicitly evaluate the reachability conditions between each pair of nodes. If reactive and deliberative planning use the same initial state, our approach relies on two assumptions that increase the chances of a seamless transition from a reactive plan (including an empty plan generated by ρ_{wait}) to a deliberative plan; however, as explained later, in practice there is still a possibility that the transition might fail. As already mentioned in Chapter 1, the two assumptions are:

- **UNIVERSAL-DELIBERATIVE-PLAN**: Deliberative planning generates a universal plan (i.e., one containing state-action pairs for all the reachable states from the initial state), where a mapping from a state (say s) to an action (say a) suggests a be executed in s [44].
- **MARKOVIAN-DOMAIN**: The operating domain is *Markovian* (an assumption made by the formal model): the state after a transition depends only on the current state — not on the sequence of states that preceded it [78].

Fig. 4.1 explains how a universal plan (e.g., MDP policy) increases the chances of (but does not guarantee) the transition of execution from a reactive plan to a deliberative plan. Explaining in the context of the exemplar system, suppose at time t_0 in state s, there is a response time constraint violation. To deal with this situation, both reactive planning and deliberative planning are invoked simultaneously. Suppose reactive planning is designed such that it ignores uncertainty in the external environment by assuming the future request arrival rate will remain the same as in the current state. Since reactive planning time is negligible, suppose it suggests an action a_1 to be executed at time t_0. Meanwhile, using a time-series predictor, suppose deliberative planning takes predicted, but uncertain, values of future request arrival rate into consideration and comes up with a deliberative plan, suppose at time t_1. On executing the action a_1, due to uncertainty in the client request arrival rate, suppose the system could reach one of three possible outcome states: s_1, s_1' or s_1''. If the predicted values for the request arrival rate (used for the deliberative planning) are correct, these states will be found in the deliberative plan, because the plan contains the state-action pair for all the reachable states from the initial state s. Therefore, once the deliberative plan is ready (suppose at time t_1), it can take over the plan execution from the reactive plan because any state in the reactive plan will be in the deliberative plan.

Moreover, due to the $Markovian$ nature of the operating domain, optimality of the action prescribed by the deliberative plan for states such as s_1, s_1' and s_1'', depends only on that state, and not on any of the previous states. This implies that once deliberative planning solves the planning problem corresponding to the state space shown in Figure 4.1, the resulting plan would suggest an optimal action for each state reachable from the initial state s.

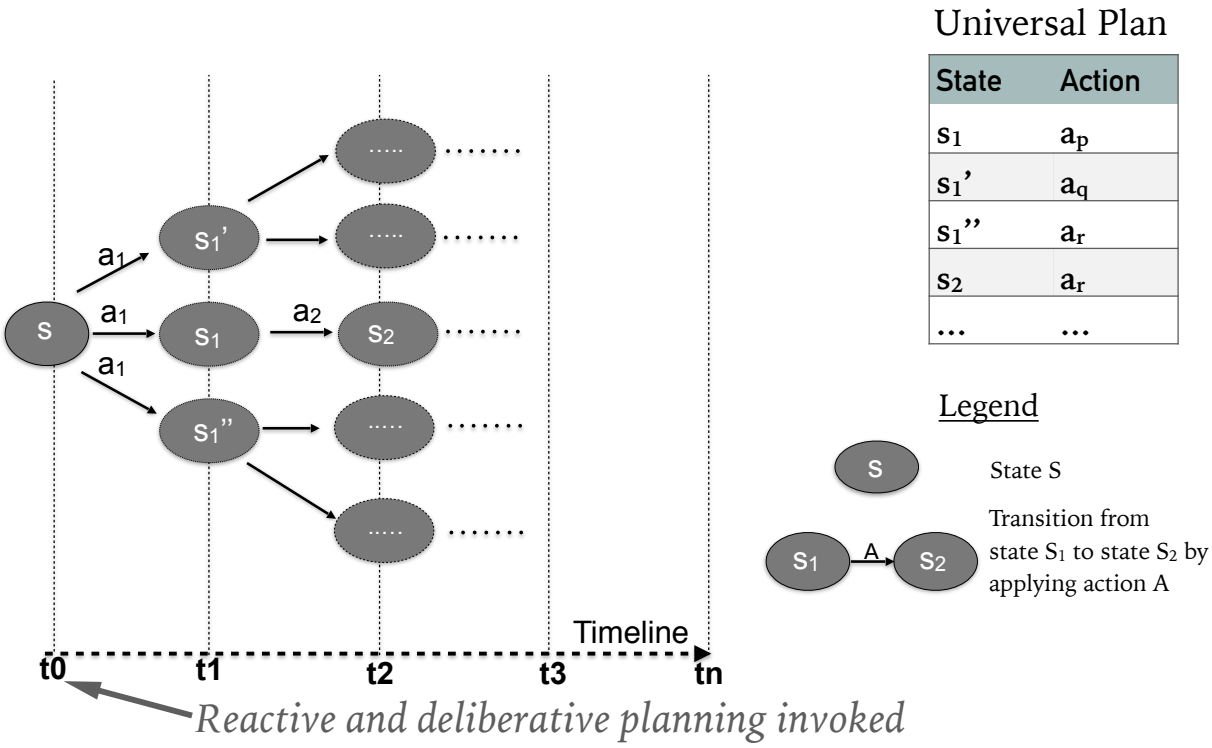


Figure 4.1: Transition from a reactive plan to a deliberative plan

The structure of a deliberative plan and the Markov property increase chances of a transition from a reactive plan to the deliberative plan; however, in practice, there is still a possibility that this transition might fail due to violation of either the timing or the preemption condition (cf. Chapter 3) between two nodes. Continuing with the example, the timing condition is violated if the deliberative plan is not ready by the time the system observes one of the states s_1, s_1' or s_1''. In such cases, there is no need to restart deliberative planning since, due to the "universal" nature of the plan (contingent on the accuracy of predictions made by the time-series predictor), once the plan is ready it can take over the execution if the preemption condition is satisfied; meanwhile, the system continues with reactive planning i.e., TWO-LEVELS-OF-PLANNING.

As an example of an unsatisfied preemption condition, suppose at time t_1 the system ends up in states such as s_1' or s_1'' not anticipated by the deliberative planning; this can happen, for instance, if a prediction of the future request arrival rate is incorrect. In such a case, even if the deliberative plan is ready at time t_1 the transition from a reactive plan to the deliberative plan would be infeasible because states s_1' and s_1'' will not be present in the deliberative plan. Therefore, if the system still needs to adapt at time t_1, deliberative planning needs to be restarted; however, the choice of reactive planning depends on the solution to PTHSEL as discussed later in Section 4.2.

4.2 Finding a Path in a Reachability Graph

Up to this point, we have discussed the approach to deal with PRBSEL, PLRAST, and GPHCON. The last subproblem is PTHSEL, which amounts to finding a path consisting of problem-planner nodes that would maximize the system's utility. As already discussed, solving PTHSEL implicitly solves PLNSEL. This section analyzes the approach to solve PTHSEL in the light of the formal model.

Given assumptions FINITE-HORIZON and TWO-LEVELS-OF-PLANNING and a finite set of (N) reactive approaches, as explained earlier, for a planning problem only $1 + N$ kinds of node are possible in a reachability graph, i.e., nodes corresponding to deliberative planning, and N reactive approaches. Therefore, initially when a system observes a planning problem, one of these $1 + N$ nodes has to be selected as the first node of the path corresponding to the hybrid plan for the planning problem. Given assumption DELIBERATIVE-PREFERRED, if a deliberative plan is ready for a planning problem, the deliberative node gets precedence over the other nodes corresponding to reactive planning. If the timing and the preemption condition is satisfied as defined in Chapter 3, once a deliberative plan is ready no more planning is required; in other words, no further nodes need to be selected to construct a path. However, due to the time-consuming nature of deliberative planning, initially the plan is unlikely to be ready (i.e., the timing condition for reachability is violated) therefore, the deliberative planning node cannot be selected.

While deliberative planning is in process, a decision is required to decide which reactive approach to apply among the given (N) approaches. Given TWO-LEVELS-OF-PLANNING, once a system picks an appropriate reactive approach, it sticks with it until the deliberative plan is ready. To explain in terms of a reachability graph, first the node corresponding to the reactive approach is selected among the N nodes corresponding to the reactive approaches. Then using the execution edge for the related reactive plan, this node is connected to the deliberative planning

node, which is the first node in the timeline (i.e., as soon as the deliberative plan is ready) that has the two reachability conditions satisfied. The execution edge from this node extends until the planning horizon is reached for the planning problem.[7] For such a reachability graph, formally, given set Ξ of all planning problems for the system and set $\mathcal{F}$ of reactive planning approaches, solving GPHCON (or PLNSEL) problem means approximating function $\mathcal{G} : \Xi \rightarrow \mathcal{F}$ suggesting which reactive approach should be invoked for a planning problem $\xi \in \Xi$.

Since reactive planning time is negligible, nodes corresponding to reactive approaches are always available (i.e., reachable) when constructing a path. Condition-based and learning-based approaches help in choosing between among reactive nodes in negligible time (i.e., assumption NEGLIGIBLE-PLNSEL-DECISION-TIME).

4.2.1 Condition-based Approach

To choose among reactive approaches, this thesis investigates a condition-based (CB) invocation of reactive planning where a system's designer specifies up-front conditions (at design time) under which (a particular) reactive approach should be invoked. Chapter 6 demonstrates the effectiveness of hybrid planning using the condition-based approach to solve PLNSEL. To explain in the context of the cloud system, on a response time violation, the system invokes reactive planning to provide a quick response (say *addServer*) to the violation. However, while a new server becomes active, deliberative planning would determine a (possibly higher) quality plan that will take over the execution once it is ready. However, as discussed earlier, the condition-based approach requires domain expertise and relies on error-prone humans to identify the conditions. Moreover, the identified conditions cannot be transferred to other systems or domains, hindering reuse of hybrid planning.

4.2.2 Learning-based Approach

To overcome these drawbacks, this thesis proposes a *machine learning-based* (LB) approach to decide which reactive planning (from a given set, which includes waiting as a special case) in combination with deliberative planning would lead to improved performance for a given situation. Using planning problems similar to the ones expected at run time, the approach trains a classifier to choose an appropriate reactive approach for a given problem. At run time, depending on how the current situation (i.e., the planning problem at hand) relates to problems in the training set, the classifier chooses which reactive approach (including ρ_{wait}) to invoke. This approach overcomes the disadvantages of condition-based (CB) invocation of reactive planning by removing the need for humans to determine the specific conditions at design time and being applicable to a broad range of systems/domains.

The learning-based approach has two phases: *offline* and *online*. During the offline phase, the first step is to collect/identify a training set of planning problems similar to the ones expected at run time. In the second offline step, using a probabilistic model-checker, these problems are labelled with the preferred reactive approach to be used for a problem in combination with

[7] If the plan execution is interrupted for some reason, then a new planning problem will be formulated, which will result in a new reachability graph; the current reachability graph is abandoned.

deliberative planning, as discussed later. The third and last offline step is to decide appropriate features in the training set and use them to train a machine learning classifier, which will determine the best reactive planner for each situation. In the online phase, on facing a planning problem ξ (representing the current situation) at run time, the system invokes the classifier on the features of ξ. The classifier picks a reactive planner, which is used by the system until a deliberative plan is ready.

The offline phase

In the offline phase, a classifier is trained using planning problems that the system expects to observe at run time. The offline phase has three steps: (a) identify sample planning problems to profile the hybrid planner; (b) profile the hybrid planner on these problems to determine the label (i.e., which reactive planning outperforms others); and (c) select features and hyper-parameter values to train a classifier.

(1a) Identifying Sample Problems: To select reactive planners effectively, it is crucial to cover the planning problem space comprehensively. However, identifying a set of representative problems is challenging due to a potentially infinite problem space and its unknown structure. No single selection strategy fits all systems and domains, and we suggest tailoring the sample set to the system's context and requirements. Fortunately, modern-day systems produce large amounts of data available to train a classifier. For instance, in our evaluation systems, we mine sample planning problems from the available traces containing the typical load patterns [37] (for the cloud-based system) and randomly sample the space of missions (for the UAV).

(1b) Labeling the Sample Problems: This step determines the reactive approach $\rho_r^i \in \mathcal{F}$ that performs best in combination with deliberative planning for a sample planning problem ξ, and label it accordingly (i.e., ρ_r^i). At the end of this step, we obtain a set of labelled training data, which is critical to (supervised) learning in our learning-based approach [101]. However, in the presence of uncertainty in the environment (which is often the case for realistic systems), it is difficult to evaluate a combination given that its performance may vary across plan executions (for the same problem) because of different possible outcomes leading to different plan execution paths. For example, suppose a self-adaptive cloud-based system proactively adds a server anticipating an increase in the future workload (i.e., the number of requests received by clients). However, if the workload increases or decreases further, adding the server might not have the desired effect. Therefore, an approach is needed that can take uncertainty into account when evaluating the combination.

To overcome this problem, this thesis proposes to use a *probabilistic model checker*, which considers probabilistic uncertainty when evaluating a combination of reactive and deliberative plan. Moreover, existing model checkers ease adoption, automation, and reuse of the learning-based approach by software engineers. One can encode each combination of planners and planning problem in a model checker specification, and the model checker gives expected utility that the combination will provide for the problem. This use of model-checking is fundamental to our learning-based approach: a model checker labels training problems by evaluating plan combinations under probabilistic uncertainty, by considering all possible execution paths weighted with their probabilities. Such a use of probabilistic model-checking to assess the quality of a (combined) plan is based on the assumptions that the model checker specification of a planning

problem captures reality. There might be other potential techniques/tools that can be explored to evaluate combined plans. However, any such technique/tool should: (a) have an input specification that captures reality, and (b) be able to evaluate a combined plan in the context of planning goal; for instance, if the goal is to maximize expected utility, the tool should be able to calculate expected utility for the combined plan in the context of the input specification.

Figure 4.2 illustrates how a model checker can be used to evaluate the combination of reactive (ρ_r^i, producing plans π_r^i) and deliberative planning (ρ_d, producing plan π_d in time t_d). The outcomes of executing actions from each plan are uncertain, and a model checker handles this uncertainty by aggregating the quality of possible outcomes as expected utility, denoted U_r^i.

To compute U_r^i for ξ, the model checker calculates the expected utility for the combination of plans π_r^i (until time step t_d) and then π_d. If set $\mathcal{F}$ has N reactive planners, then each sample problem ξ requires N evaluations corresponding to each $\rho_r \in \mathcal{F}$. Specifically, to calculate the expected utility of a combination, we used PRISM [69] as a model checker.[8]

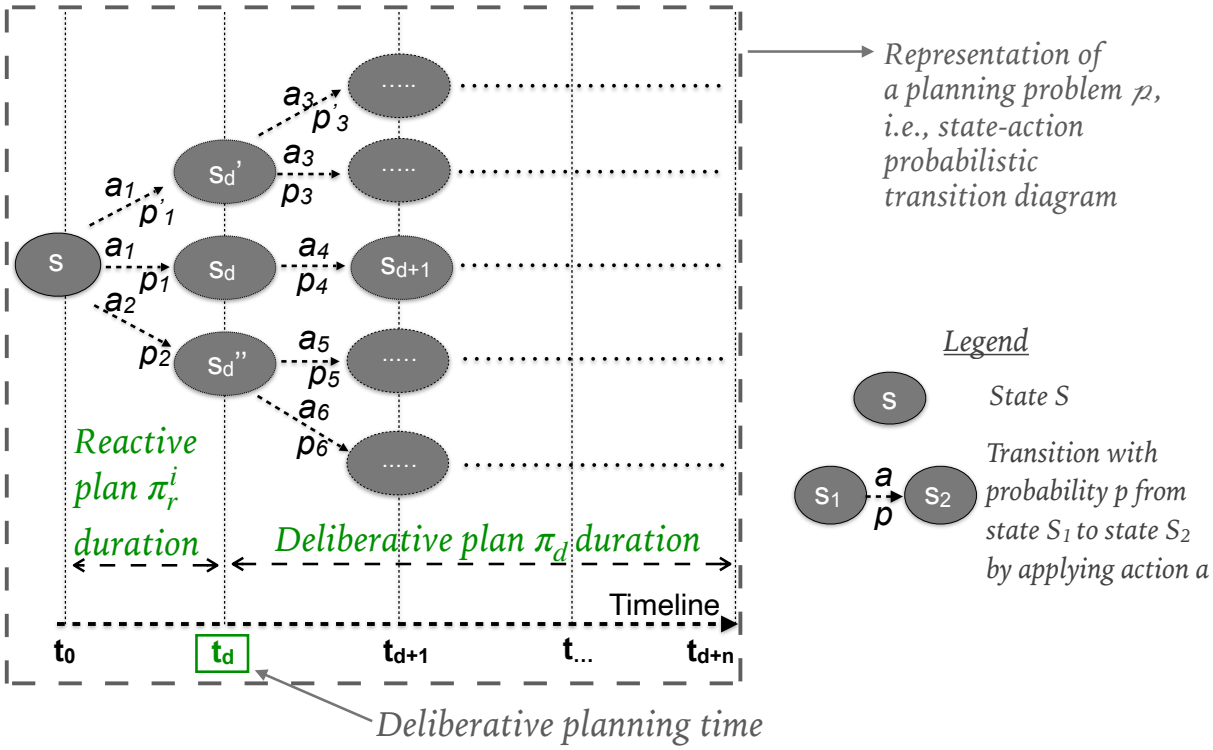


Figure 4.2: Evaluating a reactive approach ρ_r^i, i.e., calculating the utility for the combination of reactive and deliberative plan.

Finally, we need to compare expected utilities for each combination. For problem ξ, suppose

[8]Our evaluation uses PRISM since it supports model-checking for the (MDP) domains with probabilistic uncertainty in action outcomes, which is the case for the two systems used for evaluation. However, our approach is not limited to any specific model checker. For Markovian domains (i.e., partially observable MDP) that also have uncertainty in the underlying state, one can use model checkers that support such domains.

the plan determined by $\rho'_r \in \mathcal{F}$ (in combination with the deliberative plan) provides the highest utility, and ξ is assigned the label corresponding to ρ'_r. If more than one reactive approach provides the highest utility, any of those approaches can be chosen. Thus, each sample problem is labeled with one of the *N* labels, given *N* reactive approaches. This approach can be naturally extended to also support any number of deliberative approaches (rather than one); basically, the labeling process can help in deciding the best combination of a reactive and a deliberative approach.

Although Section 6.3 demonstrates that labeling the sample problems using model checking works in practice, this approach is limited to the problems with a finite horizon. To explain further, as shown in Figure 4.2, a model checking specification needs to keep track of states before time t_d (i.e., before the deliberative plan is ready), and t_d onwards (i.e., when the deliberative plan is available). This requires having a state variable that captures the time for a state. Since time is a state variable, a problem cannot have an infinite planning horizon. Otherwise, the state space will be infinite, therefore, intractable for probabilistic model checking. However, this problem of infinite state space does not arise due to assumption FINITE-HORIZON.

(1c) Training a Classifier: The first step to train a classifier on the labeled planning problems is to identify relevant features of planning problems that help separating the *N* classes. To this end, we use two complementary sets of features: ones representing the current state of the system, and ones describing how the system will evolve in the future. For the two evaluation systems, future evolution of the systems depend on the external environment, which is uncertain but predictable. To exemplify, for the cloud-based system, we use state variables such as the number of active servers to capture the current state. To capture the future evolution of the system, we use real-time predicted request arrival rates for the future within the planning horizon [85]. Similarly, for the team of UAVs, we use state variables such as flying altitude to capture the current state. To capture future evolution, we use predicted value of threats and targets as discussed in Section 6.1.2.

We believe these features reasonably capture a planning problem, which has current (i.e., initial) state and transitions as the fundamental elements. One could also use techniques such as principal component analysis (PCA) to further reduce the set of features [2]. Once features are identified, using the sample problems, we use cross-validation to train and test different classifiers [66]; we pick the classifier which provides the best performance during cross-validation.

The online phase

When a self-adaptive framework requires planning (e.g., periodically or in response to a constraint violation [10]), it formulates a planning problem ξ. The offline-trained classifier is used on ξ to assign the label corresponding to an appropriate $\rho_r \in \mathcal{F}$. Typically, such a supervised learning classifier can classify instances (e.g., planning problems) near-instantaneously: therefore, assumption NEGLIGIBLE-PLNSEL-DECISION-TIME is not violated.

4.3 Summary

This chapter presented our approach to instantiate a hybrid planning using a deliberative approach and a finite number of reactive approaches. The chapter explained the approach in the context of the formal model, giving us confidence that all relevant subproblems (i.e., i.e., PTHSEL, GPHCON,

PLRAST, and PRBSEL) are handled. Consequently, the chapter also discussed how the two fundamental challenges (i.e., PLNCRD and PLNSEL) are handled because solving subproblems GPHCON and PTHSEL implicitly solves PLNCRD and PLNSEL, respectively.

To solve PLNSEL, the chapter discussed two approaches: condition-based and learning-based. The condition-based approach is easy to apply for systems where appropriate conditions to invoke reactive planning can be identified; however, for complex systems, it can be difficult to determine a fixed set of predefined conditions at design time that captures all possible constraint violations.

To overcome the shortcomings of the condition-based approach, the chapter proposed a learning-based approach that overcomes the limits of relying on predefined conditions to choose among reactive approaches. Now, domain expertise is not necessary to decide which reactive approach needs to be invoked. Instead, engineers can rely on planning problems encountered in the past to answer the same question, without committing to specific up-front conditions. Moreover, full/partial automation is possible for the learning-based approach, which can relieve designers from the painstaking and error-prone process of identifying the conditions. In Chapter 6, we investigate the performance-related advantages of the learning-based approach over the condition-based approach.

Chapter 5

Design and Analysis of Hybrid Planning

The previous two chapters presented the formal model describing the hybrid planning problem, and a practical approach to apply hybrid planning under certain assumptions/restrictions that nonetheless apply to many self-adaptive systems. This chapter takes a step closer to implementing hybrid planning in realistic systems. First, the chapter describes a general algorithm, executed in hybrid planners, to determine an adaptation plan for a given situation. This algorithm can be used with either the condition-based approach or the proposed learning-based approach. Second, the chapter analyzes the performance of hybrid planning: specifically, it proves a theorem, which states that under ideal conditions (listed in Section 5.2), hybrid planning cannot underperform its constituent approaches used alone.

5.1 The Hybrid Planning Algorithm

The goal of the hybrid planning algorithm (see Algorithm 1) is to determine a plan for system adaptation using a combination of a reactive approach ($\rho_r \in \mathcal{F}$) with a deliberative one. The input to the algorithm is a planning problem (ξ) that contains the current state of the system ($\xi.s_{curr}$), and the output is a plan stored in variable π, protected from race conditions by a mutex μ. Since, as discussed later, reactive and deliberative planners can be invoked simultaneously using different threads, we need mutex μ to handle race conditions.

To find a plan corresponding to the problem ξ, the HYBRIDPLANNING algorithm first refers to the existing plan (line 6). If a plan is present and matches the current state (i.e., contains $\xi.s_{curr}$), then the algorithm does not change π. However, if the plan does not exist or $\xi.s_{curr}$ is not in the plan, then the planner computes a suitable new plan (lines 10–20).

First, the algorithm needs to decide its reactive response (lines 10–13), which requires choosing an appropriate reactive planning approach $\rho_r \in \mathcal{F}$. This decision is made by the function PICKREACTIVEPLANNING, the role of which is to solve PLNSEL. This function can be implemented by checking predefined conditions on ξ, or by learning which reactive approach is (the most) suitable for a given planning problem. As an input, function PICKREACTIVEPLANNING takes planning problem ξ, set $\mathcal{F}$, and deliberative planner ρ_d, and returns the reactive approach that will provide the plan that provides the highest utility when combined with the deliberative plan, when it is ready.

Algorithm 1 A general hybrid planning algorithm.

```
1:  global π ← null                                          ▷ System's plan for execution
2:  global μ ← new Mutex                                     ▷ Mutex for π
3:  global T ← new Thread                                    ▷ Deliberative thread
4:  function HybridPlanning(Problem ξ, Planners F, Planner ρ_d)
5:      μ.lock()
6:      if π ≠ null and π.has(ξ.s_curr) then
7:          μ.unlock()
8:          return                                           ▷ Replan only if needed
9:
10:                                  ▷ Pick an appropriate reactive planning approach
11:     Planner ρ_r = PickReactivePlanning(ξ, F, ρ_d)
12:
13:     π ← ρ_r.Plan(ξ)                                      ▷ Determine the reactive plan
14:     μ.unlock()
15:
16:     if not T.isRunning() then
17:         T.run [                                          ▷ Deliberate in the background
18:             π' ← ρ_d.Plan(ξ)
19:             μ.lock()
20:             π ← π'
21:             μ.unlock() ]
```

Regardless of the above decision, deliberative planning (function DELIBERATIVEPLANNING) is started afterwards in a separate thread ($\mathcal{T}$, lines 16–18), in order to eventually arrive at a plan that is expected to yield higher utility than any of the reactive planning approaches. For each planning problem, deliberative planning is invoked once, allowing only one thread at a time. Once the computation of the deliberative plan is complete, the system's plan is thread-safely updated to it (17–19). As discussed earlier, the structure of the plan enables a smooth transition (i.e., the global plan π is updated) from a reactive to the deliberative plan, thus resolving PLNCRD.

Algorithm 1 is implemented by a self-adaptive framework, which can use the global variables π, μ, and $\mathcal{T}$ to configure HYBRIDPLANNING. For example, to find a reactive plan without stopping deliberative planning, the framework can lock μ, set π to *null* and, if $\mathcal{T}$ is still running, execute HYBRIDPLANNING. If needed, deliberative execution can be reset by stopping $\mathcal{T}$.

5.2 Analysis of the Performance of the Hybrid Planning

In Chapter 3, we formalized the problem of hybrid planning using an *a posteriori* (i.e., after executing a hybrid plan) semantics, which was useful in the theoretical formulation of the problem and its solution. Moreover, as we demonstrate in Chapter 6, the formalism can be used to analyze and compare different hybrid planners. However, when applying hybrid planning, one also needs to analyze hybrid planning in an *a priori* (i.e., before executing a hybrid plan) semantics, for instance, to investigate bounds (e.g., the worst-case) on the performance of hybrid planning even before applying it. Knowing the performance bounds of an approach helps to understand associated risks. This section provides the worst-case bound on the performance of hybrid planning, and in the process, formulates an *a priori* definition for the concepts defined in Chapter 3 in an *a posteriori* semantics.

A critical part of the hybrid planning algorithm is realized by the function PICKREACTIVEPLANNING, which solves PLNSEL in the algorithm (or subproblem PTHSEL in the context of the formal model). Under idealized conditions (i.e., (a) DELIBERATIVE-PREFERRED– deliberative planning provides higher expected utility than any of the reactive planners in $\mathcal{F}$, and (b) PICKREACTIVEPLANNING picks the best reactive approach for a given planning problem), hybrid planning will never provide a lower expected utility compared to utilities provided by reactive and deliberative planning used alone. This section provides a formal proof of this statement.

Intuitively, comparing hybrid planning to a reactive approach used alone, while deliberative planning is in-process, no reactive approach can be better than the choice of PICKREACTIVEPLANNING given the (ideal) implementation as explained earlier. And, once the deliberative plan is ready, it will outperform any reactive plan due to assumption DELIBERATIVE-PREFERRED. On comparing hybrid planning to deliberative planning used alone, the only way for the latter to be better than hybrid planning is when PICKREACTIVEPLANNING inappropriately returns a reactive approach other than ρ_{wait}, which might be the best choice.[1] However, this is not possible given the ideal implementation of PICKREACTIVEPLANNING.

We start with some basic definitions defining the concepts, which will be used in stating the theorem and its proof. In contrast to Section 3.2 (in Chapter 3), which lists formal definitions in

[1] As explained earlier, using deliberative planning alone (i.e., wait until the deliberative plan is ready) is equivalent to using ρ_{wait} in combination with deliberative planning; ρ_{wait} always suggests to wait.

the context of an *a posteriori* semantics, this section provides definitions using *a priori* semantics. Consequently, some concepts (e.g., a planning problem) have been redefined in this section. The key difference between an *a posteriori* and an *a priori* semantics is that the latter considers uncertainty in the environment. As explained in Section 3.1, there is no uncertainty in an *a posteriori* semantics. In contrast, as explained later, in an *a priori* semantics (i.e., before a plan execution) uncertainty in the environment is considered, if it exists. Since we evaluate expected utility to compare hybrid planning with its constituent planners, only probabilistic uncertainty in the *a priori* definitions and the proof (i.e., non-deterministic uncertainty is not covered). For completeness of this formal system, this section also reiterates the definitions that are the same as those specified in Section 3.2.

Definition 5.2.1 (State)**.** A *state* (s) is a vector of values of the system's and environment's variables. Time is considered as a state variable. We denote the set of states by S.

Since time is a state variable, S is a potentially infinite set. Moreover, time imposes an implicit total order on states in S. By default we consider time continuous, and also allow its discretization.

Definition 5.2.2 (State time)**.** The function τ returns the time value of a state. Formally, $\tau : S \rightarrow \mathbb{R}_{\geq 0}$.

Definition 5.2.3 (*A priori* transition)**.** *A priori state transitions* are characterized by a transition function $T_a : S \times A \times Z \rightarrow P(S)$, giving a probability distribution (P) over states S telling the probability of $s' \in S$ given that system action $a \in A$ and environment action $z \in Z$ are applied to system state $s \in S$. Here, A is a set of the *system's actions*, and Z is a set of *external events*. An element $\perp$ represents an empty action/event and is present in both sets: $A \cap Z = \{\perp\}$. We write $T_a(s, a, z, s')$ for the probability of ending in state s', given that a system starts in state s and, actions a and z are taken.

Definition 3.2.8 of an *a posteriori* transition in Section 3.2 does not consider uncertainty since, as explained in Chapter 3, there is no uncertainty in an *a posteriori* semantics. In contrast, an *a priori* semantics has uncertainty, therefore, Definition 5.2.3 considers (probabilistic) uncertainty in transitions. Referring to Figure 3.1 that explains the difference between the *a priori* and the *a posteriori* semantics in the context of the cloud-based system, suppose at time t_0, response time for the system is above a predefined threshold. In response to this emergency situation, suppose the system adds a server to bring the response time below the threshold. However, due to the (probabilistic) uncertainty in the request arrival rate, workload on the system can change (i.e., increase or decrease). If the workload increases further then even after the adding the server, the response time can still remain above the threshold. In contrast, if the workload decreases, the response time can be below the threshold. Therefore, due to uncertainty in the request arrival rate, the system can end up in one of the two states, i.e., response time above (S_1) or below (S_2) the threshold.

Definition 5.2.4 (*A priori* environment)**.** *An priori environment* is a function $o_a : S \rightarrow P(Z)$ giving probability distribution (P) of external events happening in each state. A set of possible *a priori* environments is designated as O_a such that $o_a \in O_a$. We write $o_a(s, z)$ for the probability of external event z happening in state s.

Again, in contrast to Definition 3.2.10 of environment, Definition 5.2.4 defines environment in an *a priori* semantics, therefore, (probabilistic) uncertainty is considered.

Definition 5.2.3 and Definition 5.2.4 restrict the proof to domains with probabilistic uncertainty

(i.e., non-deterministic uncertainty is not covered). However, the proof is still useful, since uncertainty can be represented probabilistically in a large number of domains, where planning based of MDP and POMDP is applied [44].

Definition 5.2.5 (Utility of a transition)**.** *Utility of a transition* is a function $U_T : S \times A \times Z \to \mathbb{R}$ giving the expected immediate utility gained by a system for taking $a \in A$ and environment action $z \in Z$ in $s \in S$.

Definition 5.2.6 (*A priori* plan)**.** A *plan* π^a is a total function $\pi^a : S \to A$. A mapping from a state $s \in S$ to an action $a \in A$ suggests a to be executed in s. We denote a set of plans as Π.

This definition of *plan* as a total function is different from Definition 3.2.9, which defines *plan* as a partial function in an *a posteriori* semantics. As explained later, in context of the definition (i.e., Definition 5.2.11) of a planner, Definition 5.2.6 captures different types of plans (e.g., policies) in the context of this formal system.

Definition 5.2.7 (Expected utility of a plan for a state over a horizon)**.** The *expected utility of a plan for a state over a horizon* is a function $U_\pi : \Pi \times S \times O_a \times \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ that returns the expected discounted sum of future utility that a system gets over horizon (i.e., time steps) $h \in \mathbb{R}$ for a given plan $\pi \in \Pi$, the initial state $s \in S$ for plan (π) execution in environment $o_a \in O_a$, and discount factor $\gamma \in \mathbb{R}$ such that $0 < \gamma < 1$. The expected utility of a plan can be inductively calculated as

$$U_\pi(\pi, s, o_a, h, \gamma) = \Sigma_{z \in Z} o_a(s, z) U_T(s, a, z) + \gamma \Sigma_{s' \in S} T_a(s, a, o_a(s), s') U_\pi(\pi, s', o_a, h - 1) \quad (5.1)$$

Approaches such as MDP planning can find a plan (i.e., policy) for an infinite horizon. To calculate U_π for such plans, *horizon* can be set to infinity (i.e., ∞).

Definition 5.2.8 (*A priori* planning problem)**.** The *a priori planning problem* ξ_a is a tuple $(S, s^i, A, T_a, o_a, h, \gamma, U_T)$, where $s^i \in S$ is the initial state and h is the planning horizon. Solving an *a priori* planning problem refers to finding a plan (π^a) with maximum utility U_π for horizon h given s^i, o_a, U_T and discount factor γ. Here $h = \tau_e - \tau_i$ such that $\tau_i = \tau(s^i)$ and $\tau_e = \tau(s^e)$ for the end state s^e of the plan π execution. If h is specified as ∞ then provided plan π will maximize U_π over an infinite planning horizon. A set of *a priori* planning problems is denoted by Ξ_a.

This definition of a planning problem is general enough to represent commonly used probabilistic planning approaches such as MDP and POMDP planning. An MDP planning problem can be directly represented by Definition 5.2.8. However, to represent a POMDP planning problem, a state is treated as a belief space, and the problem can be modelled as a belief MDP [55]. Definition 5.2.9 defines a belief space, which is a probability distribution over states. Belief space helps in representing uncertainty in the underlying state since (typically) states cannot be directly observed in a POMDP domain.

Definition 5.2.8 does not assume sets, such a S, A, and Z, to be finite. However, some algorithms to solve planning problems require these sets to be finite. For instance, the value-iteration method to solve MDPs assumes the sets are finite. In contrast, for POMDPs, the set of belief spaces (Definition 5.2.9) is infinite. By not restricting these sets to be finite, Definition 5.2.8 is flexible enough to capture both cases (i.e., when the sets are finite or infinite).

Definition 5.2.9 (Belief state)**.** *Belief* state $b \in B$ is a probability distribution over S, where B is the set of belief states such that $0 \leq b(s) \leq 1$ for state $s \in S$, and

$$\sum_{s \in S} b(s) = 1 \quad (5.2)$$

Due to uncertainty in the underlying state space, utility of a belief space is the expected utility of all the states in the belief space as formulated by Equation 5.2.10.

Definition 5.2.10 (Utility of a belief state)**.** *Utility of a belief state* is a function $U_b : B \times A \times Z \rightarrow \mathbb{R}$ giving the expected immediate utility gained by a system for taking $a \in A$ and environment action $z \in Z$ in belief state $b_{S'} \in B$. This can be calculated as

$$U_b(b_{S'}, a, z) = \sum_{s \in S'} b_{S'}(s) U_T(s, a, z) \tag{5.3}$$

Definition 5.2.9 and Definition 5.2.10 provide an insight into how Definition 5.2.8 can represent a POMDP planning problem.

Definition 5.2.11 (Planner)**.** A *planner* is a function $\rho : \Xi_a \rightarrow \Pi$ that solves a planning problem ξ and produces a plan π. We designate a set of potentially infinite planners by Ψ.

Definition 5.2.11 abstracts the process of problem modification and replanning if a plan fails. When a planning problem (ξ_a) is assigned to a planner, it modifies/relaxes ξ a problem (say, ξ'_a) to make it compatible and determines a plan, which is applied to ξ. For some planners (e.g., deterministic), there is a possibility that the plan might fail since the planner uses a modified problem (i.e., ξ'_a) having a subset of states $S' \in S$; therefore, the plan cannot provide for all the states. In case of a plan failure, suppose at state s_f, the planner will replan for a new problem (ξ^1_a) having s_f as the initial state. Definition 5.2.11 encapsulates the process of problem modification and replanning, and returns a plan, which has provided for all the states. Therefore, the plan is defined as a total function in Definition 5.2.6. A plan as a total function is agnostic to the planning approach (i.e., whether it is generated by an MDP planner which generates a policy, or a deterministic planner, which generates a sequential plan).

Definition 5.2.12 (Plan merge)**.** *Plan merge* is a function $\varphi : \Pi \times \Pi \times \mathbb{R} \rightarrow \Pi$ that returns hybrid plan $\omega \in \Pi$ by merging plans π_r and π_d w.r.t. $t \in \mathbb{R}$ such that $s \in S$ before t (i.e., $\tau(s) < t$) will be directed by π_r and remaining states (i.e., $\tau(s) \geq t$) will be directed by π_d. Here, $\omega = \varphi\,(\pi_r, \pi_d, t)$.

We use the same Definition 3.3.1 of a hybrid plan as mentioned in Chapter 3. In simple words, for an *a priori* planning problem ξ_a, a hybrid plan merges plans from different planners such that different sets of states $S_i \subset \xi_a.S$ are directed by different plans. Moreover, these subsets (i.e., partitions of state space) are totally ordered in time. To exemplify, for problem ξ_a, suppose reactive (say, ρ_r) and deliberative (say, ρ_d) planners determine plans as π_r and π_d, respectively, and deliberative planning time is t. In this case, the hybrid plan will merge π_r and π_d such that states $s \in \xi_a.S$ before t (i.e., $\tau(s) < t$) will be directed by π_r and remaining states (i.e., $\tau(s) \geq t$) will be directed by π_d. Due to assumption TWO-LEVELS-OF-PLANNING, a hybrid plan ($\omega^t_{\pi_r \pi_d}$) merges only two plans (i.e., a reactive and a deliberative), therefore $\omega^t_{\pi_r \pi_d}$ divides $\Xi_a.S$ into two partitions.

Definition 5.2.13 (PickReactivePlanning)**.** Function $PickReactivePlanning : \Xi_a \times \psi_r \times \Psi \rightarrow \mathcal{F}$ returns reactive planner $\rho_r \in \mathcal{F}$ for a given planning problem $\xi_a \in \Xi_a$, a finite set of reactive planners $\mathcal{F} \in \psi_r$, deliberative planner $\rho_d \in \Psi_d$, where $\psi_r \in \mathbb{P}\,\Psi_r$ such that Ψ_d and Ψ_r is are infinite set of all deliberative and reactive planners, respectively.

Given planning problem ξ_a, set of reactive planners $\mathcal{F}$, deliberative planner ρ_d and deliberative planning t to solve problem ξ_a, an ideal implementation of function $PickReactivePlanning$ will output the reactive planner that provides the plan, which when merged with the deliberative plan (π_d) w.r.t. t, provides the hybrid plan with highest expected utility. Formally, for an ideal

implementation of the function, given $\omega_\rho = \varphi(\rho(\xi_a), \pi_d, t)$ and $\omega_{\rho_r} = \varphi(\rho_r(\xi_a), \pi_d, t)$ such that $\rho, \rho_r \in \mathcal{F}$

$$\begin{aligned} \rho = PickReactivePlanning(\xi_a, \mathcal{F}, \rho_d) \implies \\ \forall \rho_r : \mathcal{F} \cdot U_\pi(\omega_\rho,\ \xi_a.s^i,\ \xi_a.o_a,\ \xi_a.h,\ \xi_a.\gamma) \geq U_\pi(\omega_{\rho_r},\ \xi_a.s^i,\ \xi_a.o_a,\ \xi_a.h,\ \xi_a.\gamma) \end{aligned} \tag{5.4}$$

As proved later, given DELIBERATIVE-PREFERRED, and an ideal implementation of PICK-REACTIVEPLANNING, hybrid planning will never provide a lower expected utility compared to utilities provided by reactive and deliberative planning used alone. Here is the theorem that formally states this fact.

Theorem 1. Given

1. *a priori* planning problem $\xi_a \in \Xi_a$
2. Set of reactive planners $\mathcal{F} = \{\rho_{wait}, \rho_{r1}, \ldots, \rho_{rn}\}$
3. Deliberative planner ρ_d, which determines plan ρ_d for ξ_a with deliberative planning time as $t \leq \xi_a.h$
4. And an ideal implementation of function PickReactivePlanning such that ρ = PickReactivePlanning(ξ_a, $\mathcal{F}$, ρ_d) and $\omega_\rho = \varphi(\rho(\xi_a), \pi_d, t)$

then $U_\pi^{hp} \geq U_\pi^d$ and $U_\pi^{hp} \geq U_\pi^r$, where
$U_\pi^{hp} = U_\pi(\omega_\rho,\ \xi_a.S.s^i,\ \xi_a.o_a,\ \xi_a.h,\ \xi_a.\gamma)$
$U_\pi^d = U_\pi(\omega_{\rho_{wait}},\ \xi_a.S.s^i,\ \xi_a.o_a,\ \xi_a.h,\ \xi_a.\gamma)$ such that $\omega_{\rho_{wait}} = \varphi(\rho_{wait}(\xi_a), \pi_d, t)$
$U_\pi^r = U_\pi(\rho_r(\xi_a),\ \xi_a.S.s^i,\ \xi_a.o_a,\ \xi_a.h,\ \xi_a.\gamma)$ for $\rho_r \in \mathcal{F}$

Proof. Given *PickReactivePlanning*(ξ_a, $\mathcal{F}$, ρ_d) = ρ and deliberative planning time t, hybrid planning determines plan ω_ρ by merging reactive plan π_r (i.e., ρ (ξ_a)) and deliberative plan π_d such that $\omega_\rho = \varphi$ (π_r, π_d, t). In contrast, when using deliberative planning alone, ρ_{wait} is used for reactive planning until t_d since no system's action is taken; however, from t onwards, deliberative planning is used. Therefore, the resulting plan can be represented as plan $\omega_{\rho_{wait}} = \varphi(\rho_{wait}(\xi_a), \pi_d, t)$ (i.e., the states before t are directed by plan π_r and the remaining states (i.e., with time value $\geq t$) are directed by plan π_d). The expected utility of plan π_r determined by using reactive planning alone is U_π^r.

Given the four conditions (listed above), Theorem 1 has two claims. The first claim is that the expected utility (i.e., U_π^{hp}) of hybrid plan ω_ρ cannot be less than the expected utility (i.e., U_π^d) of $\omega_{\rho_{wait}}$ i.e., $U_\pi^{hp} \geq U_\pi^d$. The second claim is that the expected utility (i.e., U_π^{hp}) of hybrid plan ω_ρ cannot be less than the expected utility (i.e., U_π^r) of plan $\omega_{\rho_r} = \varphi(\pi_r, \pi_d, t)$ for $\rho_r \in \mathcal{F}$ such that $\pi_r = \rho_r$ (ξ_a); formally, $U_\pi^{hp} \geq U_\pi^r$. To summarize the two claims, hybrid planning cannot underperform either deliberative or reactive planning used alone. To prove Theorem 1, we will separately investigate the two claims.

The fact that deliberative planning will outperform hybrid planning implies $U_\pi^d > U_\pi^{hp}$. However, given an ideal implementation of function $PickReactivePlanning$, the function will output a reactive planner that, when merged with deliberative plan π_d, provides a merged plan with the highest expected utility among $\rho_r \in \mathcal{F}$. Therefore, plan $\omega_{\rho_{wait}}$ can never outperform ω_ρ because if that had been the case, $PickReactivePlanning$ would have output ρ_{wait} instead of any other reactive planner in $\mathcal{F}$. Hence, $U_\pi^{hp} \geq U_\pi^d$, i.e., performance of hybrid planning will always be

greater or equal to that of deliberative planning. The performance will be equal when both ρ_{wait} and some other planner $\rho_r \in \mathcal{F}$ provide equal expected utility when merged with π_d.

Now, let us analyze the second claim (i.e., $U_\pi^{hp} \geq U_\pi^r$) of the theorem. According to assumption DELIBERATIVE-PREFERRED, for any planning problem, a deliberative plan provides higher or equal expected utility compared to a reactive plan. Each state $s \in S_t$, where $S_t \subseteq \xi_a.S$ and $\forall s \cdot S_t : t \leq \tau(s) < (\xi_a.h - t)$, can be treated as the initial state of an *a priori* planning problem $\xi_a^{'}$ such that $\xi_a^{'} = \{S_t, s, \xi_a.A,\ \xi_a.T_a,\ \xi_a.O_a,\ \xi_a.h - \tau(s),\ \xi_a.\gamma,\ \xi_a.U_T\}$ (i.e., each element is the same between ξ_a and $\xi_a^{'}$ except state space S_t (i.e., starting from t onwards), the initial state and the planning horizon). According to DELIBERATIVE-PREFERRED, plan $\pi_d^{'} = \rho_d(\xi_a^{'})$ will provide higher or equal expected utility compared to a reactive plan π_r determined by planner $\rho_r \in \mathcal{F}$. Formally,

$$\forall s \cdot S_t : \forall \rho_r \cdot \mathcal{F} : U_\pi(\rho_r(\xi_a^{'}),\ s,\ \xi_a.o_a,\ \xi_a.h - \tau(s),\ \xi_a.\gamma) \leq U_\pi(\pi_d^{'},\ s,\ \xi_a.o_a,\ \xi_a.h - \tau(s),\ \xi_a.\gamma) \tag{5.5}$$

Since the domain is assumed to be Markovian, plan $\pi_d^{'}$ will be subsumed by plan π_d, i.e., $\pi_d^{'}(s) = \pi_d(s)$, where $s \in dom(\pi_d^{'})$. Therefore, we can deduce that deliberative plan π_d will provide higher or equal utility compared to a reactive plan π_r determined by a reactive planner in $\mathcal{F}$. Formally, $\pi_d^{'}$ in Equation5.5 can be replaced with π_d to formulate Equation 5.6 below.

$$\forall s \cdot \xi_a.S : \forall \rho_r \cdot \mathcal{F} : U_\pi(\rho_r(\xi_a),\ s,\ \xi_a.o_a,\ \xi_a.h - \tau(s),\ \xi_a.\gamma) \leq U_\pi(\pi_d,\ s,\ \xi_a.o_a,\ \xi_a.h - \tau(s),\ \xi_a.\gamma) \tag{5.6}$$

Reactive plan π_r can be treated as a merged plan such that $\pi_r = \varphi(\pi_1, \pi_2, t)$ where π_1 and π_2 direct the states before t and the states from t onwards, respectively. Due to Equation 5.6, the expected utility of π_2 for any state $s_d \in S_d$, such that $\forall s \cdot S_d : \tau(s) = t$ and $S_d \subset \xi.S$, cannot be greater than the utility provided by π_d. In other words, the utility of plan π_r cannot be greater than the merged plan $\omega_1 = \varphi(\pi_1, \pi_d, t)$. On comparing plans ω_1 and hybrid plan ω_ρ, due to the ideal implementation of $PickReactivePlanning$, the expected utility of ω_1 cannot be greater than ω_ρ, otherwise the function would have output planner ρ_r, i.e., the one that provides the maximum expected utility when merged with π_d. Therefore, since utility of ω_1 cannot be less than π_r, we can conclude that the utility of plan ω_ρ cannot be less than plan ρ_r. Hence, hybrid planning cannot underperform reactive planning, i.e., $U_\pi^{hp} \geq U_\pi^r$.

□

5.3 Summary

This chapter presented Hybrid planning algorithm that can be used both with the condition-based and the learning-based approaches discussed in Chapter 4. In the context of MAPE-K loop [61], this algorithm can be implemented in the planning component. however, this algorithm is not limited to any specific self-adaptive framework, since it can be implemented in the component of the framework that is responsible for determining an adaptation plan. Moreover, while explaining the algorithm, the chapter highlights how condition-based and learning-based approaches fit into the algorithm (i.e., the approaches are implemented inside $PickReactivePlanning$).

In addition, the chapter proves Theorem 1, which states that (theoretically) hybrid planning cannot underperform reactive or deliberative planning used alone. However, in practice, implementations of PICKREACTIVEPLANNING solve PLNSEL imperfectly, affecting the performance of hybrid planning. In the next chapter, we experimentally evaluate the hybrid planning approach, and demonstrate how the formal model can be used to evaluate/analyze/compare instantiations of hybrid planning.

Chapter 6

Validation

This chapter serves three purposes: It (a) validates the thesis claims using two realistic systems as a testbed, (b) presents an empirical analysis of the experimental data from the two systems that reveal the factors that influence the performance of hybrid planning, and (c) illustrates how the formal model can be used as a unifying evaluation framework to compare/analyze instantiations of hybrid planning, and thereby understand their strengths and weaknesses.

Chapter 1 stated the thesis claims, which are restated below:

We can improve the effectiveness *of self-adaptive systems by using a hybrid planning approach, which is* general *and* flexible*. This approach has the following elements:*

- *the use of off-the-shelf deliberative and reactive planning approaches to instantiate hybrid planning that can take advantage of both planning approaches to find a balance between quality and timeliness of planning;*
- *the ability to dynamically decide which constituent reactive planning should be invoked along with deliberative planning.*

Chapter 4 presented the two elements of our hybrid planning approach. The chapter explained how to instantiate hybrid planning using off-the-shelf approaches and introduced the condition-based and the learning-based approach to address PLANNING SELECTION (i.e., the ability to dynamically decide which constituent reactive planning should be invoked along with deliberative planning). This chapter validates the thesis claims that hybrid planning is:

- **Claim 1**: *effective*, i.e., provides a higher utility compared to its constituent planning approaches used alone.
- **Claim 2**: *general*, i.e., can be applied to different kinds of system.
- **Claim 3**: *flexible*, i.e., can be instantiated using different combinations of off-the-shelf deliberative and reactive approaches.

To validate these claims, we use two realistic systems – a self-adaptive cloud-based web system and a team of UAVs. These systems are from different domains and differ in a variety of ways, as detailed later in Section 6.3.2. The systems are realistic since: (a) they belong to real-world domains, and (b) the quality attributes and adaptation actions considered for the two systems are similar to the ones considered in the real-world for their respective domains. A validation using these systems demonstrate that hybrid planning is effective. Moreover, an *effective* application of hybrid planning on such different kinds of systems demonstrates *generality*

of the approach. Furthermore, hybrid planning is shown to be *flexible* since it is instantiated using a different set of off-the-shelf planning approaches in the two systems. Additionally, this chapter presents an empirical analysis of the data from the case studies that reveals that the performance of hybrid planning is correlated to the performance of (i) deliberative planning, and (ii) the relatively better-performing approaches among the reactive ones. These findings can inform software engineers who need to prioritize their investment of resources in planners.

Finally, using an example, this chapter demonstrates how the formal model describing the hybrid planning problem (cf. Chapter 3) can be used to analyze and compare instantiations of hybrid planning. To this end, the chapter uses the model to analyze an existing hybrid planning instantiation and compare it with the instantiation used in one of the case-studies (i.e., the cloud-based system). Notably, the instantiation used as an example is not proposed by us but another researcher [79].[1].

The chapter is organized as follows. Section 6.1 introduces the two systems used for evaluation; Section 6.2 explains the implementation of the learning-based approach for the two systems; Section 6.3 presents the evaluation results and discusses how they validate the thesis claims; Section 6.4 highlights findings from the empirical analysis; Section 6.5.1 presents an example demonstrating how to analyze and compare an instantiation of hybrid planning; Finally, threats to validity are discussed in Section 6.6.

6.1 Validation Systems

This section presents the two systems that are used to evaluate the thesis claims. These systems are a cloud-based load balancing system and a team of UAVs on a reconnaissance mission. As discussed later, these two systems are used because balancing timeliness and quality of planning is critical to their success. In addition, developing a single planning approach from scratch can be challenging for software engineers. The two systems let us investigate different compositions of constituent planners, which vary in their action sets, planning horizons, and treatment of uncertainty. The differences are further discussed in Section 6.3.3.

To compare various planning approaches such as condition-based and learning-based hybrid planning and its constituent planners, we conducted controlled experiments by keeping all the experimental parameters constant except the planning approach, and the traces/missions used as inputs for the two systems respectively (cf. Section 6.1.1 and Section 6.1.2). We controlled the parameter values to isolate the effects of the planning approach on the utility.

6.1.1 The Cloud-based Load Balancing System

As the first system, we adopted a cloud-based load balancing system already introduced in Section 1.1. Here we provide more details about its implementation, instantiation of hybrid planning, and experimental setup for evaluating the thesis.

[1]Compared to our approach to hybrid planning, this instantiation is limited to a specific combination of a reactive and deliberative approach.

Implementation

As an implementation of the cloud-based system, we used SWIM, which is a well-accepted artifact in the self-adaptive research community [87]. We made two key extensions to SWIM to suit our goals. First, we added support for non-negligible planning times to make it comparable to a realistic system. This change is needed to implement the hybrid planning algorithm (i.e., Algorithm 1) discussed in Section 5.1. The extended SWIM runs planners during experiments, therefore planning delays are real (i.e., not artificially induced). Our second extension, as detailed later, added support for different types (i.e., 3) of servers, with their associated tactics as described earlier in Section 1.1. Specifically, as discussed later, we implemented the M/G/1/PS queueing model to distribute load among active servers of different capacity. The servers allow incrementing/decrementing dimmer values, and the load-balancer allows addition/removal of servers and distributing load among active servers. The extended version of SWIM is open-source, and available online.[2]

Instantiation of Hybrid Planning

As parts of hybrid planning, we use two reactive approaches, i.e., $\mathcal{F} = \{\rho_{det},\ \rho_{wait}\}$, and ρ_{mdp} as deliberative planning. Here, ρ_{det} and ρ_{wait} refers to deterministic planning and wait planning (as discussed in Chapter 4), and ρ_{mdp} refer to MDP planning. ρ_{det} ignores uncertainty in the request arrival rate by assuming it to be constant at the current value. This reduction in the search space greatly reduces the planning time for ρ_{det}, making it practically instantaneous in the context of the this system. When using the condition-based approach to solve PLNSEL, ρ_{det} is invoked when response time is above the threshold; therefore, the intent behind using ρ_{det} is to avoid penalty P for having the response time above the threshold, as discussed in Section 1.1.

In contrast to ρ_{det}, ρ_{mdp} considers predicted (but uncertain) values of the request arrival rates. We use a time-series predictor to anticipate the future workload on the system, similar to others [85]. When deliberative planning (i.e., ρ_{mdp}) is triggered, a time-series predictor feeds predicted values as an environment model formulating an MDP, mapping each possible request arrival rate to an outcome of a probabilistic action taken by the environment. Moreover, ρ_{mdp} generates a universal plan (i.e., MDP policy), which is the requirement for deliberative planning as discussed in Section 4.1.2. However, due to explicit modeling of uncertainty, ρ_{mdp} has a larger (on average, 22 times) state space compared to ρ_{det}: thereby, ρ_{mdp} is relatively time-consuming. The goal of both reactive and deliberative planning is to maximize utility (Formula 1.1) for their lookahead horizon. In addition to labeling sample planning problems to implement the learning-based approach (as discussed in Section 6.2), we use PRISM both as a deterministic and a MDP planner – similar to what has been done by other researchers [39, 108].

The supplementary material provides a PRISM planning specification for non-wait reactive (i.e., deterministic) and deliberative (i.e., MDP) planning [93]. These specifications have some constants (e.g., *MAX_ARRIVALA_CAPACITY*, *penalty*) that remain constant for all the planning problems. However, there are variables (e.g., *ini_servers_A*, and *ini_traffic_A*) that depend on the current state (i.e., initial state of the planning problem) of the system.

[2] `https://bitbucket.org/ashutosh_pandey/hybridplanning-omnet5`

As mentioned earlier, a deliberative planning specification includes a model of the environment as an MDP. To build a model for the environment, as proposed by Moreno et al. [85], we build a probability tree that represents both the predicted interarrival rates and the probabilistic uncertainty in the request arrival rate. To discretize the probabilistic distribution of transitions from a state, we use Extended Pearson-Tukey (EP-T) three-point approximation, which consists of three points that correspond to the 5th, 50th, and 95th percentiles of the estimation distribution, with probabilities 0.185, 0.630, and 0.185, respectively [59].

Experimental Setup

To construct a realistic environment of users accessing the cloud-based system, we adapted a research dataset with online traffic common in web analytics — the daily traces of user requests from the FIFA WorldCup website [5]. These traces are independent day-by-day recordings of user website activity during the championship, with rapid changes in system load, as well as periods of low variation. We picked these traces for several reasons. First, this trace set is considered as a benchmark for traffic in web analytics [6, 50, 121]. Second, modern-day web applications (such as content delivery and video streaming) are typically bursty, with periods of low load contrasting with occasional flash crowds caused by an important event; these traces represent such workloads [22]. Finally, they contain the patterns for high-demand cloud systems as classified by Ghandhi et al. [37] and illustrated in Figure 6.2. These patterns are: slowly varying, quickly varying, big spike, dual phase, large variations, and steep tri phase. As shown in Figure 6.1, a trace can have multiple patterns. We performed experiments on 87 traces (out of 92), ignoring 5 empty/partial ones. The plots illustrating the traces' pattern are made available in Appendix B. Each day's trace contains timestamps representing inter-arrival time between two client requests, abstracting away the details of user requests to focus on their frequency.

As discussed in Section 1.1, the system uses a queueing theory model to predict the system's response time depending on the number of active servers, their capacity to handle a number of requests per minute, and the request arrival rate. For the model to work, we scaled each trace (keeping the trace pattern intact) to the length of 105 minutes such that it does not exceed the maximum capacity of the testbed in terms of the ability to serve requests per minute. The scaling also ensured that the starting request arrival rate is not greater than the capacity of server of type A (the only active server at the start of a simulation). Out of 105 minutes in each trace, the first 15 minutes are used to train the time-series predictor, and the performance of different planners on a trace is evaluated during the remaining 90 minutes. We fix the length of the traces to normalize aggregate utility values.

As mentioned earlier, we extended SWIM to have three types of servers: A, B and C. Servers of type A are the most expensive, but have the highest capacity to handle requests. Servers of type C are the cheapest, but have the least request-handling capacity. We assume the time for a server to serve a request is normally distributed with the mean as the server's capacity and the variance as the maximum possible delay calculated using a variation of the M/G/1/PS queueing model that supports different-capacity servers operating in parallel. The costs and capacities are assigned according to Table 6.1. The ratio between cost and capacity is constant, which is inspired by the

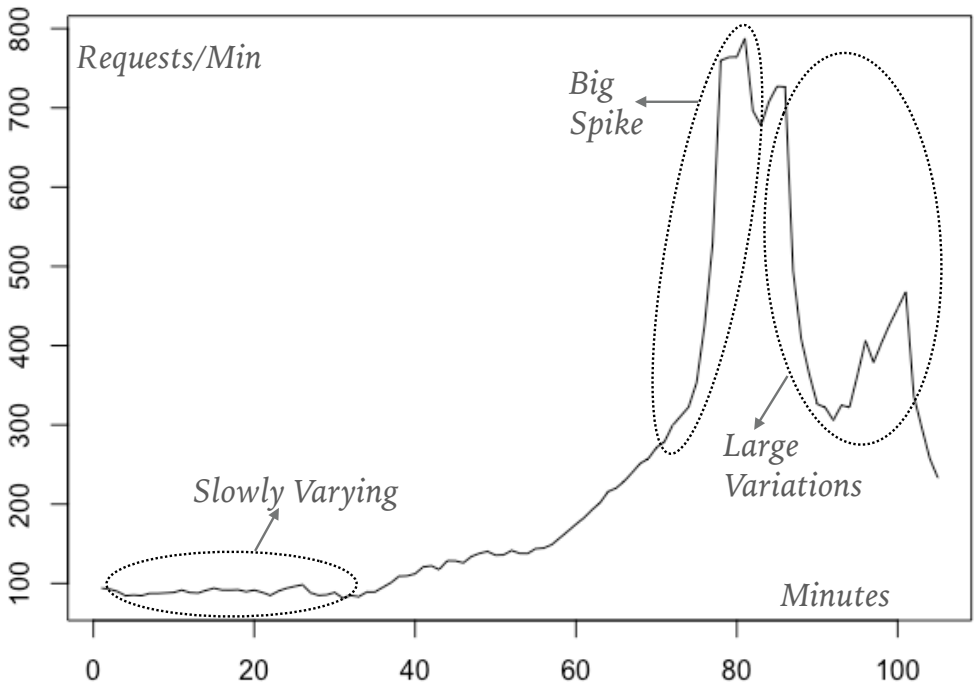


Figure 6.1: Illustration of multiple workload patterns in day-46 trace of FIFA worldCup. The trace has slowly-varying, big-spike and large-variation patterns.

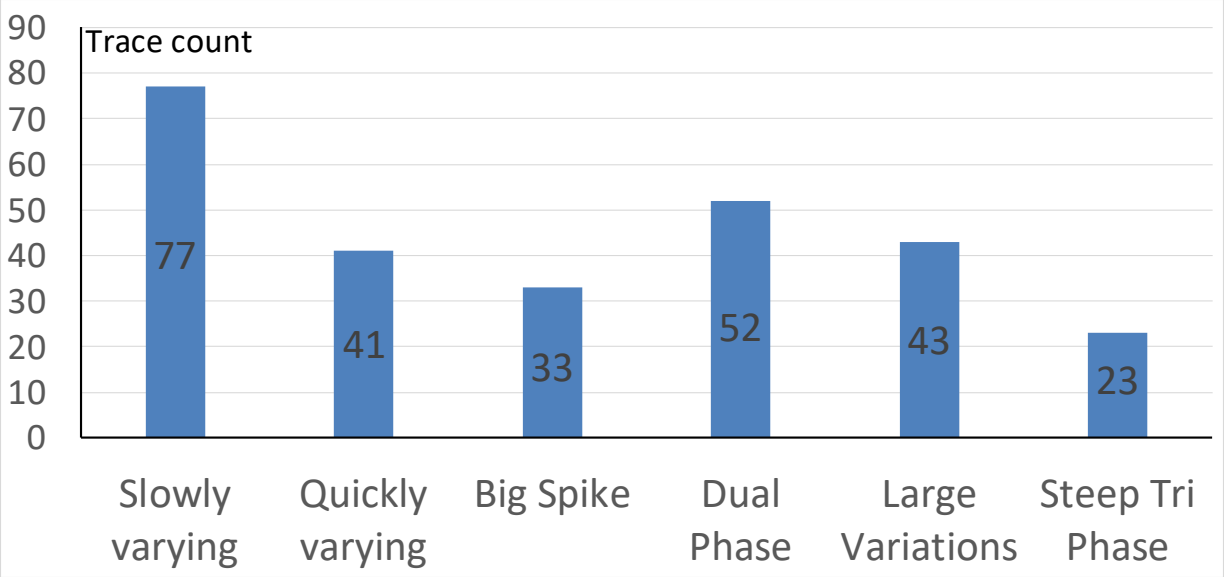


Figure 6.2: The number of traces (out of total 87 traces) having a specific pattern. A trace can have more than one pattern.

cost model of Amazon Web Services[3] where the system capacity improves in the same ratio as the increase in cost. In the experiments we have three dimmer levels and one server of each type.

		Capacity (ability to server requests per minute)	
Server type	Cost (units per minute)	With Optional Content	Without Optional Content
A	1.0	200	400
B	0.7	140	280
C	0.5	100	200

Table 6.1: Cost/capacity parameters for each server type.

In our experiments, the cost of a server can be covered by the revenue of handling 1/10 of its maximum capacity with optional content and the revenue of handling 2/3 of its maximum capacity without optional content. If the server cost per minute is C, capacity with optional content is c_O and without optional content is c_M, then the revenue for a server with optional content is $R_O = \frac{10}{c_O}C$ and without optional content would be $R_M = \frac{3/2}{c_M}C$. For each request having response time above the threshold of 1 second, there is a penalty of -0.25 units.

The system evaluates the need for adaptation at each minute, i.e., the length of an evaluation cycle. To this end, a planning problem is formulated that represents the current state of the system and future transitions. This problem is passed as an input to the hybrid planning algorithm, which returns an appropriate adaptation action(s); if adaptation is not needed then the algorithm returns an empty action, suggesting the system to wait until the next invocation of the algorithm.

We assume a fixed server boot-up time of 2 minutes (i.e., 2 evaluation cycles) and only one server can be booted up (at a time) in response to an increase in requests arrival rate; however, multiple servers can be active (i.e., serve requests) simultaneously. Since at a given point, we would have a maximum of 2 inactive servers and each server has 2 minutes of boot-up time, our planning horizon for the deliberative (i.e., ρ_{mdp}) planning is 5 minutes. This heuristic gives a planning horizon long enough to go from 1 active server to 3 active servers plus 1 additional evaluation cycle to observe the resultant utility.

When looking up the current state in a plan (Line 6 in the hybrid planning algorithm, i.e., Algorithm 5), the cloud-based system needs to deal with the possibility of not finding any matches; in such cases the plan fails and needs to be recomputed. As mentioned earlier, planning is done based on predicted request arrival rate; not the actual values. Since the prediction discretizes the values, it is possible that the actual value is not one of the discrete values. To account for this situation, we used a matching heuristic. Specifically, we use two criteria for states in a plan being matched to a given (current) state: (1) all state variables (except the request arrival rate) have the same values; (2) the rate is within $min(0.5 * current\_rate, 100)$ of the current arrival rate.[4] If no state meeting both the criteria is found in a plan, the matching fails. If several states meet both criteria, one that minimizes the difference between request arrival rates is picked. Appendix A

[3] `https://aws.amazon.com/ec2/pricing/on-demand`

[4] By experimentation, we found that this criterion provides a reasonable balance between matching states and failing plans in our experiments.

formalizes the timing and the preemption condition (cf. Chapter 3) for this system and discusses how the state matching heuristic influences the two conditions.

We conducted the experiments on a Ubuntu 14.04 virtual machine having 8.5 GB RAM and 4 processors at 2.9 GHz. The state space for deterministic planning (i.e., ρ_{det}) varies between 25K and 100K, for deliberative planning (i.e., ρ_{mdp}) between 1.6 million and 2.8 million. The planning time for reactive planning ρ_{det} is considered negligible, i.e., less than a second. The planning time for deliberative planning ρ_{mdp} varies between 35-45 seconds.

6.1.2 A Team of Unmanned Aerial Vehicles

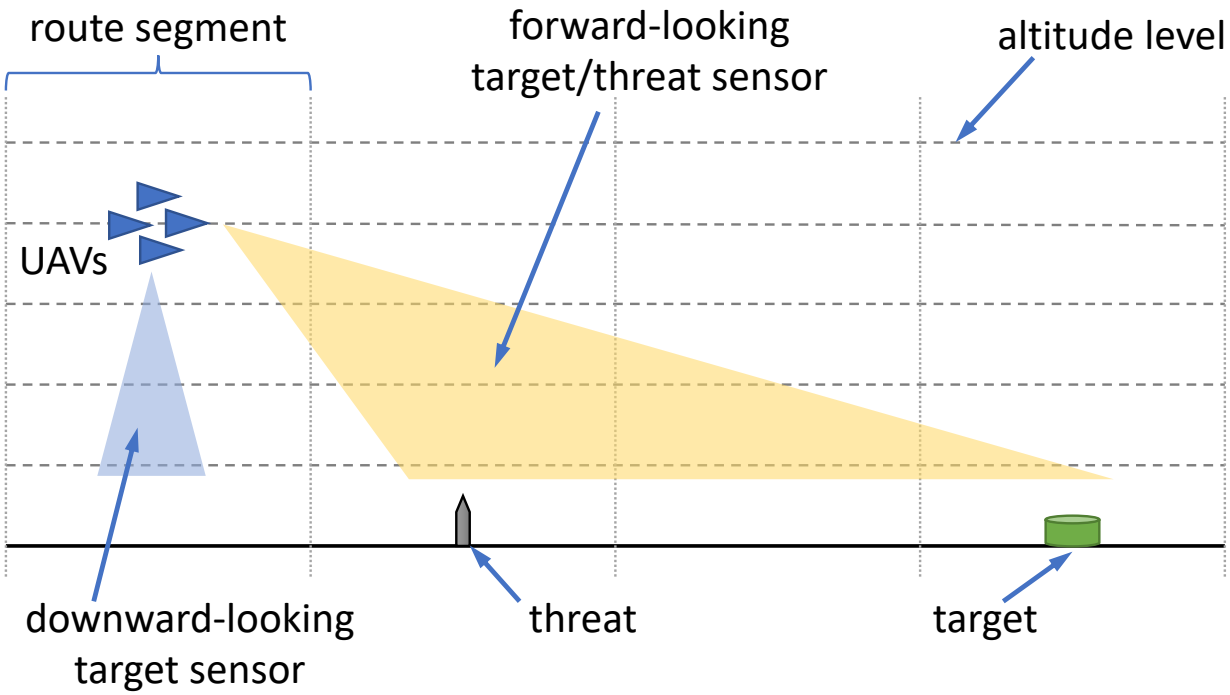


Figure 6.3: Simulation overview [88].

As the second evaluation system, we used a simulated team of unmanned aerial vehicles (UAVs) performing a reconnaissance mission in a hostile environment. The predefined route of the team is a straight line, divided into equal segments of fixed length, as shown in Figure 6.3. Each segment can have threats and detection targets depending on how they are randomly placed in the route. The mission of the team is to detect the targets and avoid being shot down by the threats, which would lead to the mission failure (no more targets can be detected further). However, it is difficult to meet the two requirements simultaneously since there is no action available that increases the chances of both target detection and survival for the team (see Table 6.2). The team uses these actions to maximize the number of targets detected, taking into account that if the team is lost to a threat, the mission fails. If the team chooses to execute an action, then all of its UAVs in the team execute the same action.

Pre-planning the execution of the mission is not feasible because the environment (i.e., the location of targets and threats) can only be discovered as the team flies during the mission, and

Action	Description	Survival/Detection Chance
`IncAlt`	Climb one altitude level	increases/decreases
`DecAlt`	Descend one altitude level	decreases/increases
`IncAlt2`	Climb two altitude levels	increases/decreases
`DecAlt2`	Descend two altitude levels	decreases/increases
`GoTight`	Change to tight formation	increases/decreases
`GoLoose`	Change to loose formation	decreases/increases
`EcmOn`	Turn ECM on	increases/decreases
`EcmOff`	Turn ECM off	decreases/increases

Table 6.2: Adaptation actions for the team of UAVs.

even then, only with uncertainty.[5] Moreover, even though both targets and threats are static, their number is not known *a priori*. The team has different sensors to detect targets and threats as it flies a route at constant speed. For each route segment within the range, the sensor reports whether it detects a target or threat, depending on the sensor type. However, due to sensing errors, these reports may include false positives and false negatives. An adaptation manager can get multiple observations to construct a probability distribution of threat or target presence in a cell.

The team configuration has an effect on the probability of being destroyed by a threat and the probability of detecting a target, which is important when deciding how to adapt. A threat can destroy the team only if both are in the same segment. However, a threat has range r_T, and its effectiveness is inversely proportional to the altitude of the team, denoted by $\mathcal{A}$. In addition, the formation of the team affects the probability of it being destroyed. The team can be in two different formations: loose ($\phi = 0$), and tight ($\phi = 1$). The latter reduces the probability of being destroyed by a factor of ψ [116]. When the team uses ($E = 1$) electronic countermeasures (ECM), the probability of being destroyed is reduced by a factor of α. Taking altitude, formation, and the use of ECM into account, the probability of the team being destroyed, d, is given by (6.1).

$$d = \frac{\max\big(0, r_T - \mathcal{A}\big)}{r_T} \left((1-\phi) + \frac{\phi}{\psi}\right)\left((1-E) + \frac{E}{\alpha}\right). \tag{6.1}$$

The probability of detecting a target with the downward-looking sensor, given that the target is in the segment being traversed by the UAVs, is inversely proportional to the altitude of the team [111]. Furthermore, flying in tight formation reduces the detection probability due to sensor occlusion or overlap, and the use of ECM also affects target detection, reducing the probability of detection by a factor of β. The probability g of detecting a target is given by (6.2).

$$g = \frac{\max\big(0, r_S - \mathcal{A}\big)}{r_S} \left((1-\phi) + \frac{\phi}{\sigma}\right)\left((1-E) + \frac{E}{\beta}\right), \tag{6.2}$$

where r_S is the range of the sensor (i.e., at an altitude of r_S or higher, it is not possible to detect

[5]Theoretically, pre-planning can be done for all the possible states and transitions, but that is difficult to scale; a large number of variables (e.g., the number of segments, threats and targets) in a typical system leads to a well-known combinatorial explosion of states, making the task of off-line calculation intractable in practice.

targets), and σ is the factor by which the detection probability is reduced due to flying in tight formation. Given constants μ and λ, and the number of segments survived and targets detected for a mission is S and T respectively, the utility of the mission is calculated as

$$U = \mu S + \lambda T. \tag{6.3}$$

Both timeliness and quality of planning are needed to maximize utility for this system. A timely (i.e, quick) response is needed in response to threats, which could lead to mission failure. Simultaneously, a quality plan is needed for the long-term utility gains that requires not only surviving, but also detecting targets; this requires considering factors such as uncertainty in the threat and target locations.

Implementation

As an implementation of a team of UAVs, we used DARTSim, which is a published benchmark in the research community [88]. We extended DARTSim to support non-negligible planning times to make it comparable to a realistic system. Although DARTSim is a simulator, planning was invoked at run time, therefore, planning delays are real during evaluation. The locations of targets and threats depend on a seed, which is an input parameter to DARTSim. Thus, by varying seeds we created different missions to generate training problems and evaluate the learning-based approach, as detailed later. The extended DARTSim is open-source and available online.[6]

Instantiation of Hybrid Planning

To instantiate hybrid planning, we use two reactive approaches ($\mathcal{F} = \{\rho_{mdps},\ \rho_{wait}\}$) and deliberative planning ρ_{mdpl}. Both ρ_{mdps} and ρ_{mdpl} use MDP planning, however, ρ_{mdps} plans with a shorter horizon compared to deliberative planning ρ_{mdpl}. Moreover, while planning, ρ_{mdps} does not consider adaptation actions `IncAlt`, `DecAlt`, and `EcmOn`, and `EcmOff`. Using a shorter horizon in combination with a subset of actions results in a smaller state space in ρ_{mdps} compared to ρ_{mdpl}. Since ρ_{mdps} uses actions `IncAlt2` and `DecAlt2`, it can increase/decrease two altitude levels in response to a threat or an opportunity to detect a target. The goal for both reactive and deliberative planning is to detect targets on the ground and avoid being shot down by threats. When using the condition-based approach, this instantiation invokes ρ_{mdps} if $\mathcal{A} < r_T$, i.e., the team is in the range of threats, else ρ_{wait} is used; therefore, ρ_{mdps} is used to provide a quick response when the team is in danger.

Appendix A provides PRISM planning specifications for non-wait reactive (i.e., the short-horizon MDP with a subset of actions) and deliberative (i.e., the long-horizon MDP) planning. These specifications have some constants (e.g., *threatRange*, *sensorRange*) that remain the same for all the planning problems (i.e., specifications). However, there are variables (e.g., current altitude *ini_a*, and formation *ini_f*) that depend on the current state (i.e., initial state of the planning problem) of the system.

These specifications also include modeling of the environment that is proposed by Moreno et al.[84]. To calculate these probabilities, by sampling the information captured by the target and threat

[6] `https://github.com/Ashutoshp/pladapt`

sensors, we describe the probability densities of a target and a threat in a segment using the beta distribution [12]. This continuous distribution is then discretized using the EP-T three-point approximation [59], allowing the planners to consider three possible realizations of the environment for each segment.

Experimental Setup

We fixed the mission length for DARTSim at 40 segments. The total number of targets and threats is 20 and 10, respectively, and they are placed randomly depending on the random seed. The total number of altitude levels is 4, threat (r_T) and target (r_S) range is 3 and 4 respectively, the tight configuration reducing the probability of being destroyed (i.e., ψ) by a factor of 1.5. When using ECM the probability of being destroyed and target detection is reduced by a factor of 0.15, and 0.3, respectively. The threshold for the Manhattan distance is 1.0, which was decided after trying values 0.25, 0.5, 1.0, and 1.5. 1.0 provided the best performance for deliberative planning. The reward for surviving a segment is (μ =) 0.2 and for detecting a target is (λ =) 1.

We set the time-related parameters as follows. Similar to the cloud-based system, this system evaluates the need for adaptation at each minute, i.e., the length of an evaluation cycle. Time to observe effects of action `IncAlt2`/`DecAlt2` and `IncAlt`/`DecAlt` is equal to 1 minute, which is also the duration that the team takes to cross a segment. For the remaining actions, effect can be observed instantaneously. For the team, the planning horizon for ρ_{mdps} and ρ_{mdpl} is 2 and 5, respectively.

When looking up the current state in a plan (i.e., Line 6 in hybrid planning algorithm in Chapter 5), DARTSim needs to deal with the possibility of not finding any matches; in such cases, the plan fails and needs to be recomputed. As mentioned earlier, planning is done based on probability for segments having a target and a threat; this probability is calculated by sampling the observations by target and threat sensors. However, when the team reaches a segment, the probability value can change due to additional data collected during the mission. To find the closest matching state corresponding to the current state, we use two criteria: (1) all state variables (except the target and threat probabilities in the current segment) have the same values, and (2) the *Manhattan* distance between the pairs of target and threat probabilities is less than a predefined threshold. If no state meeting both criteria is found in a plan, the matching fails. If several states meet both criteria, the one with the smallest distance is picked.

In hybrid planning modes, to identify an appropriate *Manhattan* distance to find the current state in an MDP policy, we evaluated the performance of deliberative mode with different distances, as shown in Table 6.3. We focused on the deliberative mode because, when using hybrid planning, Manhattan distance is used by deliberative planning. We finalized on Manhattan distance as 1.0 since deliberative mode detects maximum targets (i.e., 493 from 70 missions generated by 70 random seeds) without being destroyed more compared to other values for *Manhattan* distance.

Manhattan Distance	Targets	Destroyed
0.25	315	29
0.5	229	29
1.0	493	29
1.5	229	49

Table 6.3: Influence of Manhattan distances on the performance of deliberative mode on 70 missions.

We conducted the experiments on a Ubuntu 14.04 virtual machine having 8.5 GB RAM and 4 processors at 2.9 GHz. The state space for the short-horizon MDP planning (i.e., ρ_{mdps}) varies between 75K to 250K, and for the long-horizon MDP planning ρ_{mdpl} between 3 million to 5 million. The planning time for reactive planning ρ_{mdps} is considered negligible, i.e., less than a second. The planning time for deliberative planning ρ_{mdpl} varies between 40-60 seconds.

6.2 Learning-based Approach Implementation

This section explains the implementation of the offline and the online phase for the learning-based approach for the two case studies.

6.2.1 The Offline Phase

As already explained in Section 4.2.2, the offline phase involves three steps: identifying sample problems, labeling the sample problems, and training a classifier.

Identifying Sample Problems

To generate sample problems for the two systems, our goal was to create a set of problems similar to the ones expected at run time. For the cloud-based system, we executed each trace in a mode where ρ_{det} was always invoked in the combination with ρ_{mdp}. This mode is different from using a learned classifier, which does not always invoke ρ_{det}. Therefore, the training data are less likely to include the exact problems that the system would observe at run time, thus providing us with data similar to what can often be mined from system execution logs. In total, we generated 1651 planning problems from 87 traces. For the UAV team, we simulated 630 missions (using 630 different seeds) in the mode similar to the cloud-based system, i.e., always invoke ρ_{mdps} in combination with ρ_{mdpl}. In total, 16822 planning problems were generated.

Labeling the Sample Problems

In both the systems, for the labeling process, we configured the worst-case planning time (t_d cf. Chapter 4) for ρ_{mdp}/ρ_{mdps} (i.e., deliberative planning depending on the case study) as 1 minute, chosen as an over-approximation after a large number of trial runs. Since in both the systems set $\mathcal{F}$ has two elements, the offline phase of the learning-based approach labels each sample problem (say, ξ) with one of three classes (i.e., *UseReactive*, *UseWait*, or *UseEither*). Suppose

the expected utility (after model-checking) for the combination ρ_{det}/ρ_{mdps} (i.e., non-wait reactive planning depending on the case study) and deliberative planning is U_R and for the combination of ρ_{wait} and deliberative planning is U_w. if $U_r > U_w$, then the problem is labeled to invoke the reactive planning (i.e., $Y(\xi)$ = *UseReactive*); if $U_r < U_w$, then the problem is labeled to wait for the deliberative plan to be ready (i.e., $Y(\xi)$ = *UseWait*). Finally, if $U_r = U_w$, then the choice between reacting and waiting does not matter (i.e., $Y(\xi)$ = *UseEither*). One can also include a small margin (δ such that $U_r > U_w + \delta$, or vice versa) when comparing U_r and U_w. For the cloud-based system, 111, 253, and 1287 problems were labeled as *UseWait*, *UseReactive*, and *UseEither*, respectively. The UAV team had 358, 8391, and 8073 problems labeled as *UseWait*, *UseReactive*, and *UseEither*, respectively.

Training a Classifier

Next we choose a classifier such that the test data used in the online phase are not considered while training a classifier. For the cloud system, this was accomplished through leave-one-out cross-validation. First, we left out a test trace (iterating through all 87 traces) on which the classifier would later be used to evaluate the learning-based approach. Using the problems generated from the remaining 86, we did 10-fold cross-validation to train a classifier. Classifier performances are then averaged over all validation folds, and the best one is picked for the test trace. For the UAV team, we used 630 missions (using 630 seeds) to train a classifier using 10-fold cross-validation. Once the best classifier is identified and trained, we simulated 70 missions (using seeds other than the 630 seeds) in the online phase to evaluate the learning-based approach. During cross-validation for the systems, each fold had the same proportion of classes as the overall dataset to preserve the real-world imbalance between classes.

To define the "best" classifier in cross-validation (CV), we did not use the typical measure of *accuracy*. To explain in the context of the cloud-based system, due to the data being skewed towards *UseEither*, even a trivial classifier that always predicts *UseEither* would have a relatively high accuracy (1287/1651 = 0.78). Instead, we analyzed *recall*, *precision*, and *F1 score* for each of the three classes to judge classifier performance. As it turned out, the limitations of the training data made it challenging to discover situations when ρ_{wait} is the best choice. Therefore, in CV we maximized the recall value for *UseWait*. Using this criterion we determined that an ensemble classifier known as *extremely randomized trees* [42] achieved the best performance in both systems. For the cloud, this classifier had recall/precision for *UseWait* above 0.8, and the same was above 0.9 for *UseReactive* and *UseEither*. However, even after trying several classifiers for UAVs, *UseWait* recall cannot go beyond 0.70 and precision above 0.72. Both recall and precision for *UseReactive* and *UseEither* were between 0.8 and 0.85.

6.2.2 The Online Phase

As discussed earlier, both the systems periodically (i.e., once per minute) evaluate if an adaptation is needed. When a system observes a problem (ξ) at run time, the hybrid planning algorithm is invoked once per planning problem, thus committing to either invoking ρ_{det}/ρ_{mdps} or ρ_{wait} until a deliberative plan is ready. In the learning-based approach, the offline-trained classifier is used on ξ to assign it to one of the three classes discussed above in Section 6.2.1. If the returned

class is *UseWait* or *UseReactive*, the system invokes ρ_{wait} or ρ_{det}/ρ_{mdps}, respectively. However, if the class is *UseEither*, then the choice is not fully defined by the profiling information. To deal with this ambiguity, we consider two variants of the learning-based approach: LB-W chooses to wait in the case of *UseEither*, and LB-R chooses *UseReactive*. Both variants are studied in the evaluation. As already mentioned, for the cloud-based system, using the classifier trained on 86 traces we executed one left-out trace, and repeated this process for each trace for evaluation. For the team of UAVs, using the classifier trained on 630 missions, we simulated 70 different missions for evaluation. Except aggregate utility (based on Formulas 1.1, and 6.3), all the evaluation parameters (e.g, choice of reactive and deliberative planning, instantiation of the condition-based, LB-W, and LB-R) are independent.

6.3 Claims Validation

This section presents experimental results from the two case studies, and discusses how these results support the thesis claim that hybrid planning is effective, general, and flexible.

6.3.1 Effectiveness

The thesis claims that hybrid planning is more effective than its constituent planning approaches; for each trace/mission, we define higher effectiveness of a planning as greater utility accrued over the trace/mission. To validate the effectiveness claim, we investigate if hybrid planning provides higher utility compared to its constituent approaches used alone. We compare two variations of hybrid planning to its constituent approaches; The first uses the condition-based approach and the second uses the learning-based approach to solve PLNSEL.

For validation of the effectiveness claim, each trace/mission was evaluated in seven modes:

1. Non-wait reactive — only ρ_{det}/ρ_{mdps} is used (i.e., used ρ_{det} for the cloud and ρ_{mdps} for the UAVs);
2. Wait — only ρ_{wait} is used, which essentially means the system does not adapt;
3. Deliberative — the system invokes ρ_{mdp}/ρ_{mdpl} (i.e., used only deliberative planning ρ_{mdp} for the cloud and ρ_{mdpl} for the team), and waits until a deliberative plan is available;
4. Non-wait hybrid planning (NW-HP) — when ρ_{det}/ρ_{mdps} is *always* invoked until a deliberative plan is ready;
5. Condition-based hybrid planning — when a deliberative plan is not available, ρ_{det} and ρ_{mdps} are invoked only when the predefined conditions are met as described in Section 6.1.1 and Section 6.1.2 for the cloud-based system and the UAV team, respectively;
6. LB-W hybrid planning — the learning-based approach solves PLNSEL and invokes ρ_{wait} if classification is uncertain; and
7. LB-R hybrid planning — the same learning-based approach solves PLNSEL, but invokes ρ_{det}/ρ_{mdps} if classification is uncertain.

Given ρ_{mdp}/ρ_{mdps} and ρ_{wait}, non-wait reactive and wait modes represent the two possible modes when only reactive planning is used. The condition-based mode that calls ρ_{wait} until a

deliberative plan is ready is not considered separately since it is equivalent to the deliberative mode. In both the case studies, although the classifier performed well during the cross-validation, comparison of the learning-based modes (i.e., LB-W and LB-R) with NW-HP and deliberative mode will further indicate whether the learned classifier was able to switch effectively between the reactive approaches (i.e., ρ_{wait} and ρ_{det}/ρ_{mdps}); NW-HP and deliberative mode use only one of the reactive approaches.

The results of our experiments showed that on average (both the condition-based and the learning-based) hybrid planning outperforms its constituent planners. The experiments further demonstrate that the learning-based approach is more effective than the condition-based. This indicates that we can instantiate hybrid planning more effectively by solving PLNSEL using the learning-based approach instead of the error-prone condition-based approach since it relies on human judgment to identify the right and comprehensive conditions.

Hybrid Planning Outperforms its Constituent Planners

Our experiments in both systems indicate that hybrid planning provides **more utility** than individual planning, as depicted in Figure 6.4 and Figure 6.5. The box-plots in Figure 6.4 show the differences in accrued utility (per trace/mission) when comparing pairs of planning approaches. Bars in Figure 6.5 show performance comparison of different planning approaches in terms of traces/missions count.

In Figure 6.4, the boxes represent the median 50% of traces (in terms of the difference between a pair of planners), with the horizontal lines inside showing the median difference across the traces. The whiskers show the minimum and maximum difference in utilities. For example, the leftmost box compares the condition-based approach to only using reactive (i.e., ρ_{det}/ρ_{mdps}). The fact that the lower edges (i.e., the first quartile) of the six leftmost boxes are above zero indicates that for most of the traces/missions hybrid planning provides equal-or-higher utility compared to non-hybrid approaches. Specifically for the cloud-based system, the condition-based approach, LB-W, and LB-R show equal-or-higher utility than *both* reactive and deliberative planners on 57 (66%), 60 (69%), and 57 (66%) traces respectively (out of 87 total); moreover, for the respective boxes, the positive whisker is longer than the negative one, indicating higher maximum gain than loss when choosing hybrid planning.

Based on the estimator of true probability with a confidence level of 95%, the true probability ranges for the three hybrid planning approaches to match or improve over both non-hybrid planners are (0.55; 0.76), (0.58; 0.80), and (0.55; 0.76). For the UAVs, the condition-based approaches, LB-W and LB-R, show equal-or-higher utility than *both* reactive and deliberative planners on 51 (71%), 55 (78%), and 56 (80%) traces respectively (out of 70 total). The longer negative whiskers for the 1st, 3rd, and 5th box-plot are explained by the team being averse to destruction in reactive mode, which avoids the threats at all costs; therefore, in certain missions the team survives whereas it gets destroyed (i.e., mission failure) in hybrid planning modes, which lost significantly in the overall utility for those missions. Based on the estimator of true probability with a confidence level of 95%, the true probability ranges for the condition-based approach, LB-W, and LB-R to match or improve over *both* non-hybrid planners is (0.58; 0.82), (0.65; 0.89), and (0.67; 0.9), respectively.

We also found that it is unlikely that hybrid planning performs worse than *both* reactive and

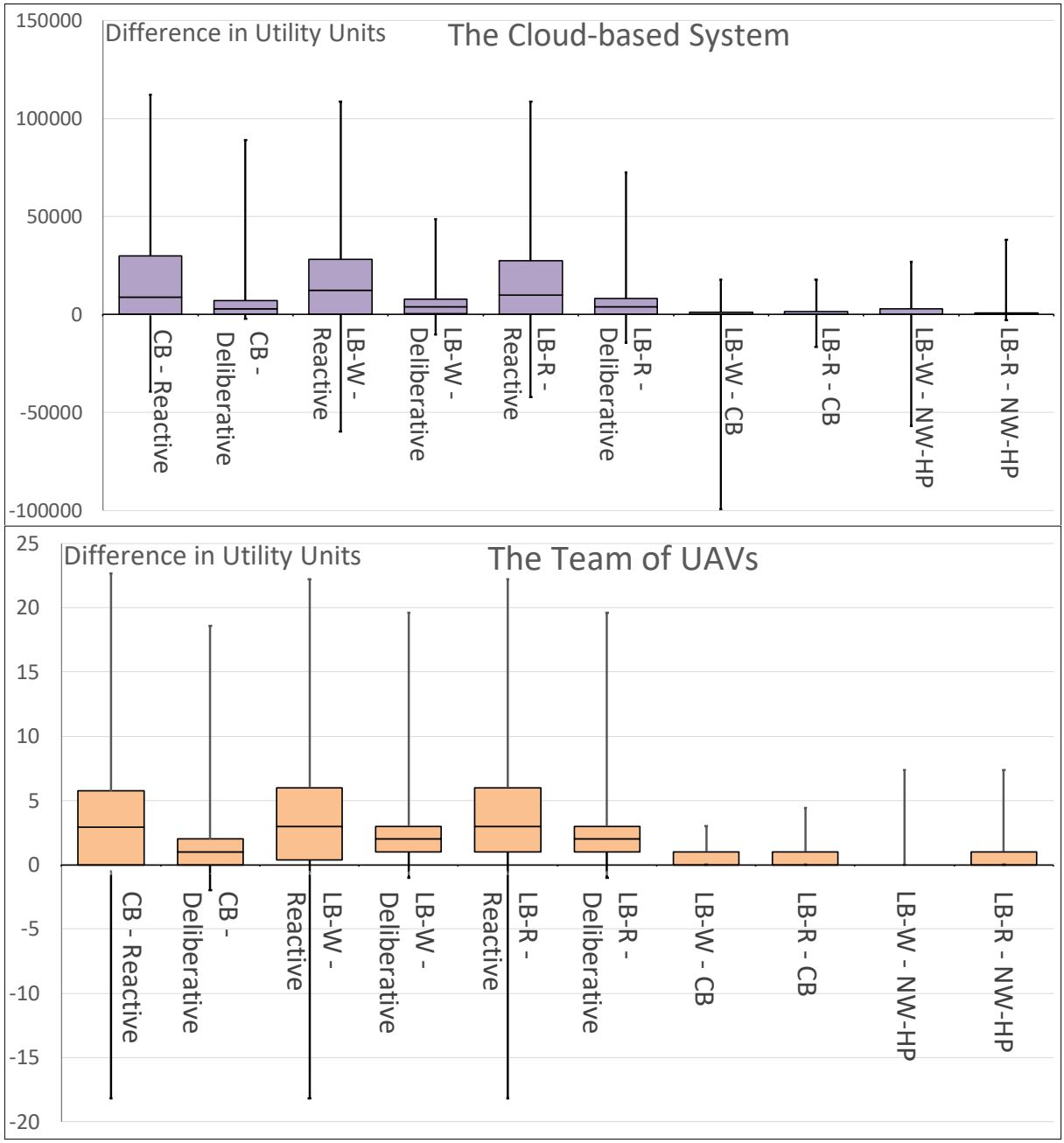


Figure 6.4: Utility differences per trace/mission added up for all traces/missions. Each bar represents a sum of differences for a pair of planning approaches.

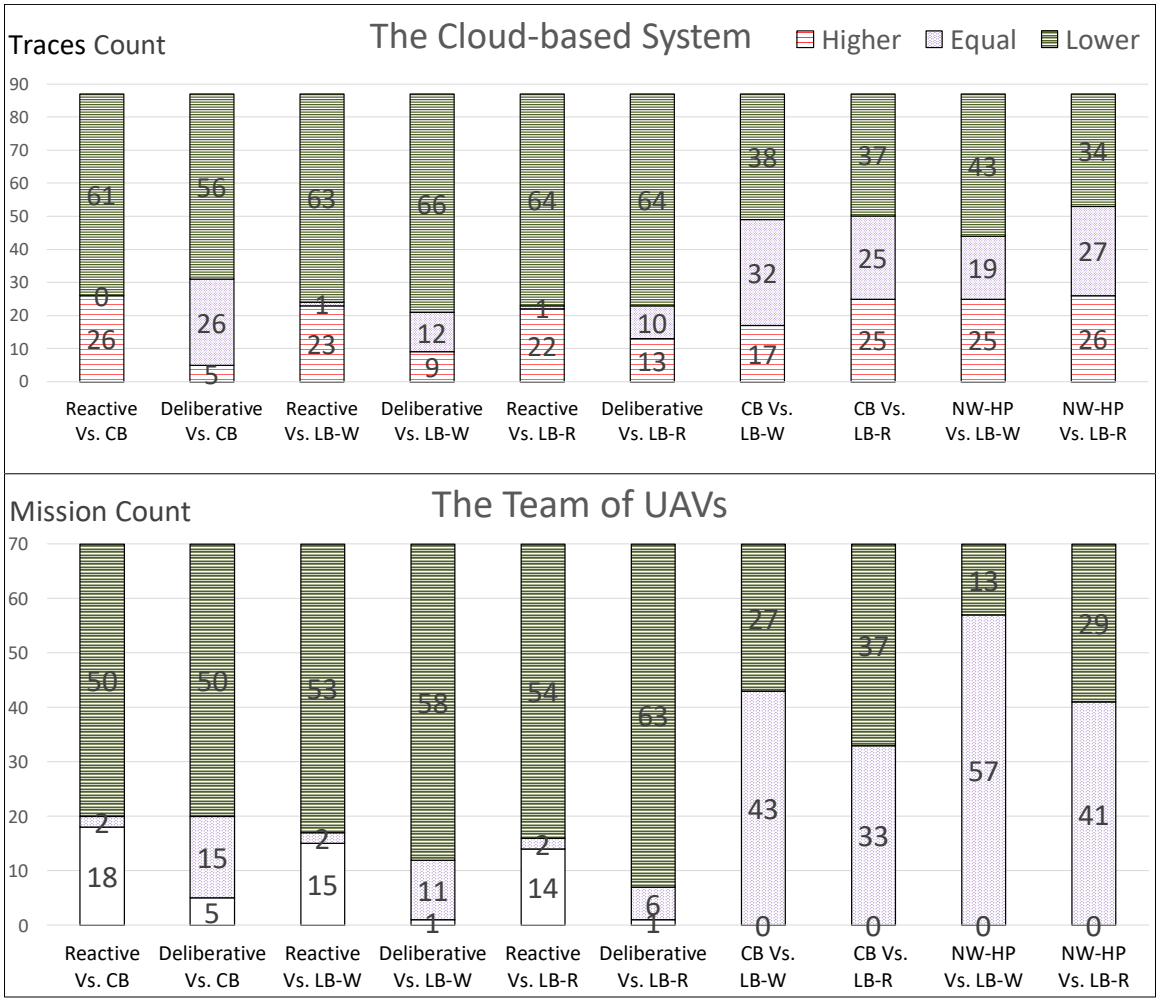


Figure 6.5: Pairwise performance comparison of planning approaches. Each bar is for a pair of approaches, labeled with the counts (out of the total traces/missions) of traces/missions where the first approach provides higher/equal/lower utility compared to the second approach in the pair.

deliberative planning; therefore, using hybrid planning is **less risky** compared to reactive and deliberative planning. Out of 87 traces, hybrid planning does worse than *both* of its constituent planners only in 1 (1%), 5 (6%), and 5 (6%) traces for the condition-based approach, LB-W, and LB-R planning, respectively. This leads us to, respectively, $(0, 0.12)$, $(0, 0.16)$, and $(0, 0.16)$ probability ranges of both reactive and deliberative planning outperforming hybrid planning according to the estimator of true probability, with 95% confidence. For the UAVs, hybrid planning does worse than *both* non-hybrid planners only in 4 (6%), 2 (3%), and 2 (3%) missions respectively (out of 70 total). The true probability ranges for the three hybrid planning approaches to match or improve over both non-hybrid planners are (0; 0.17), (0; 0.15), and (0; 0.15). Therefore, when choosing between deliberative, reactive, and hybrid planning, the latter is the least risky choice.

The Learning-based Approach Outperforms the Condition-based Approach

Our experiments show that the learning-based approach provides **more utility** than the condition-based on average. In Figure 6.4, the 7th and 8th box is above zero, indicating that for the majority of traces/missions the learning-based hybrid planning does equal-or-better than the condition-based. Specifically, out of 87 traces, LB-W and LB-R provided higher or equal utility for 70 (80%) and 62 (71%) traces, respectively. The estimator of true probability suggests with a confidence of 95% that the true probability range for the condition-based hybrid planning yielding higher utility than LB-W and LB-R is $(0.09, 0.3)$ and $(0.18, 0.39)$, respectively. For the UAV team, out of 70 missions, both LB-W and LB-R provided higher or equal utility for all the 70 missions. With a confidence of 95%, the true probability range for the condition-based hybrid planning yielding higher utility is $(0, 0.12)$. Thus, it is **less risky**, and in many cases advantageous, to use the learning-based over the condition-based hybrid planning.

However, the magnitude of the utility difference between the condition-based and the learning-based is smaller than that between hybrid planning and its constituent planners. The reason is that in response to the condition-based constraint violations, reactive planners typically propose conservative measures such as `addServer`, `decreaseDimmer`, and `IncAlt2`. These actions decrease the worst-case utility loss, which is particularly high for the second system due to the possibility of destruction. In contrast to the condition-based, despite not falling behind in performance, the learning-based enables the system to (automatically) learn when utility could be gained by using reactive planning, even without violations. Thus, we conclude that the condition-based is more **risk-averse**, whereas the learning-based is **more opportunistic** since it does not limit the use of non-wait reactive planning to constraint violations.

The outperformance of the learning-based approach is less significant compared to the NW-HP mode, as shown in the right-most two boxes, because in both systems invoking the non-wait reactive planner (i.e., ρ_{det} or ρ_{mdps}), in general, was preferred over using ρ_{wait}.[7] However, compared to NW-HP modes, the learning-based approach was able to automatically learn a classifier that switches effectively between the reactive approaches.

[7]This fact is supported by the class imbalance of the labelled data, which is skewed against using ρ_{wait} as presented in Section 6.2; this indicates the model-checking was able to label the problems reasonably well.

6.3.2 Generality

The thesis claims that hybrid planning is *general* enough to be applied (effectively) to self-adaptive systems operating in domains. The two case studies validate this claim since the systems used in the two case studies differ in various significant ways. The first three differences mentioned below were stated in Chapter1 and restated here. However, the fourth difference was realized while conducting the experimental study.

- *Quality dimensions of concern*: The cloud-based system aims at lowering response time, increasing revenue, and decreasing operating cost, whereas the UAV team intends to avoid threats and detect targets;
- *The cost of poor/delayed actions*: Poor/delayed actions could lead to destruction of a UAV(s) in the team; Therefore, generally speaking, the (monetary) cost of such actions is higher for the team compared to the cloud-based system;
- *The ability to recover from poor/delayed actions*: Even if the cloud-based system fails to maintain the critical response time constraint due to poor/delayed actions, it can still recover back to a desired state later. However, in case of the UAV, a failure to avoid a crash (i.e., safety constraint) will lead to a mission failure, as illustrated by the negative long whiskers in Figure 6.4;
- *Significance of wait/non-wait planning*: The cloud-based system has the majority of (both) training and testing problems labelled/classified as either *UseWait* (i.e., use wait planning) or *UseEither* (i.e., use either wait or non-wait reactive approach as they are equally preferred), indicating that for a large number problems wait planning was the preferred or the equally preferred choice compared to non-wait planning. In contrast, for the UAV team, the majority of problems were labelled as *UseReactive* (i.e., use non-wait reactive), indicating the significance of wait planning was relatively lower compared to non-wait planning. Despite this difference between the two systems, in both cases hybrid planning outperformed its constituent approaches. Moreover, the learning-based approach outperformed the condition-based approach indicating, the potential of the former to address PLNSEL.

6.3.3 Flexibility

This thesis claims that Hybrid planning is *flexible* enough to be instantiated (effectively) using different combinations of reactive and deliberative planners. Two instantiations are considered different if any of the constituent (reactive or deliberative) planners are different between the instantiations.

To demonstrate flexibility, we use different combinations of off-the-shelf deliberative and reactive planning approaches for the two case studies. Specifically, the first case study uses MDP and deterministic planning as the deliberative and reactive approach, respectively. In contrast, the second case study uses MDP planning both as a deliberative and reactive approach; however, the reactive version of MDP planning uses a shorter planning horizon and only a subset of adaptation actions compared to the deliberative version of MDP planning.

6.4 Other Findings: Influence of Constituent Planners on Hybrid Planning

This section presents an empirical study (using the data from the two case studies) that aims to characterize the impact of constituent planners on the performance of a hybrid planner. Knowing such dependencies (up front) can help engineers to instantiate hybrid planning using a right set of constituent planners. Our evaluation shows that the performance of hybrid planning depends on the performance of deliberative planning, and the (relatively) effective reactive planners. Below is the evidence and implications for software engineers.

Deliberative planning performance has a **consistent positive impact** on the performance of hybrid planning. We observe a medium-to-strong correlation ($p < 0.01$) between the deliberative mode and each of the three hybrid planning modes. For the cloud system, the **Pearson correlation** is 0.95 for the condition-based approach, 0.97 for LB-W, and 0.95 for LB-R. For the UAVs team, the correlation is 0.6 for the condition-based approach, 0.61 for LB-W, and 0.59 for LB-R. The interpretation of this finding is that, once a deliberative plan is ready, it inevitably takes over from any reactive plan, hence the performances of hybrid planning and deliberative planning are tightly coupled. This finding is further illustrated by Figure 6.6 for the cloud-based system and the UAV team; the x-axis represents traces/missions sorted in ascending order in terms of aggregate utility (y-axis) accrued by deliberative mode. However, the performance of hybrid planning is not linked to a reactive approach. Intuitively, if a particular reactive approach (e.g., ρ_{det} or ρ_{mdps}) is not effective, the hybrid planner can choose another reactive approach (e.g., ρ_{wait}) in $\mathcal{F}$.

To further investigate this correlation, we conducted a **Chi-square** independence test, which also showed that the ability of hybrid planning to perform better than or equal to its constituent planners significantly depends ($p < 0.01$) on deliberative planning performing better than or equal to reactive ρ_{det}/ρ_{mdps}. For the cloud-based system, the χ^2 values for the condition-based approach, LB-W, and LB-R are 43.79, 32.38, and 19.02, indicating strong-to-moderate dependency. For the UAVs, the χ^2 values for the condition-based approach, LB-W, and LB-R are 18.97, 22.16, and 20.37, also indicating strong-to-moderate dependency. This finding supports our assumption that an effective deliberative planning approach is a foundation for hybrid planning. As the chi-square test suggests, one should prefer hybrid planning to reactive planning if deliberative planning consistently provides higher or equal utility compared to reactive approaches.

In addition, we found that the performance of each reactive planner has a **positive impact** on the performance of hybrid planning, **moderated by the relative performance** of the reactive planner. Our analysis found that among the reactive approaches, the more effective ones had a stronger influence on the hybrid planning performance. The above holds under the assumption that the classifier performance is reasonably good (in our evaluation this meant having precision/recall above 0.7 for all classes). Therefore, we suggest identifying the more effective approaches (via comparing their utilities or respective class counts in training data) and focusing the resources on improving them further.

We discovered this dependency by fitting a regression model to the utility of a hybrid planner U_{hp}, using the deliberative utility U_d and reactive utility U_r as independent variables. The fitting is done over all the traces/seeds in both the case studies. Furthermore, to represent the moderating effect, these utilities were weighed with the following ratios:

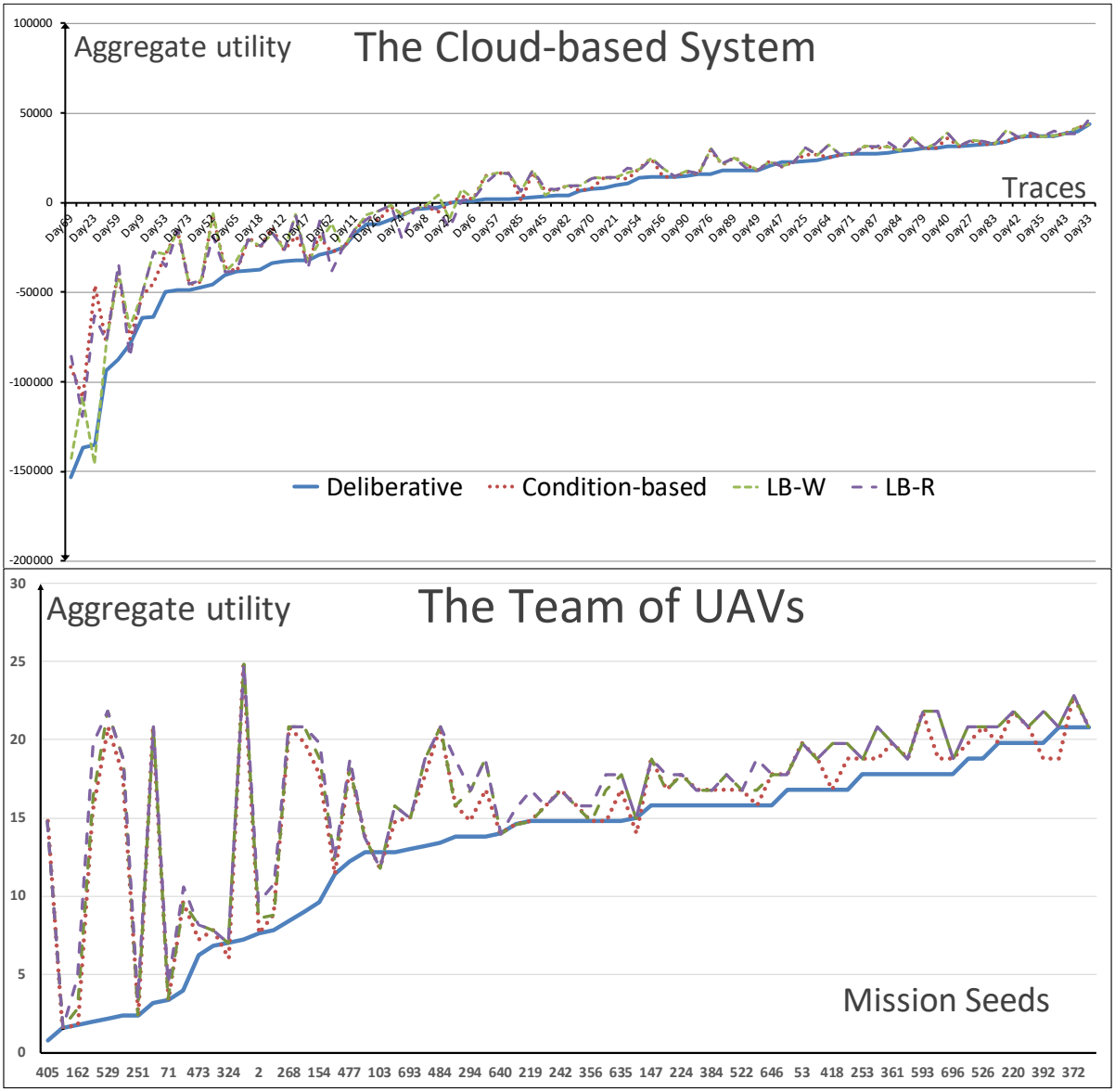


Figure 6.6: Performance of the hybrid planning modes improves with the performance of deliberative planning mode.

- U_d/U_{nw} for the deliberative utility, where U_{nw} is the utility of the NW-HP planner on that day. In this case, U_d is a proxy for the utility of the wait planner, which is invoked instead of a reactive planner for U_{nw}. Thus, this ratio represents how much the reactive planning is better than waiting planning.
- U_{nw}/U_d for the reactive utility, thus amplifying it on days when it is preferable to the wait planning, and reducing it on days when wait planning is preferable.

Thus, we arrive at the following regression model:

$$U_{hp} = a \cdot \frac{U_d}{U_{nw}} \cdot U_d + b \cdot \frac{U_{nw}}{U_d} \cdot U_r + c,$$

where a, b, and c are regression coefficients determined by fitting the above function to the utility data. In the second case study we fit this exact function, and in the first case study we had to adjust the utility numbers such that none is negative or zero (otherwise the meaning of ratios is lost). Thus, we performed the following operation for the utility U of each type of planning:

$$U := U + min(U) + 1.$$

In the fit models corresponding to the two systems, the coefficients a and b were found positive. We tested the hypothesis that a and b are not zero with a t-test, yielding a highly significant result ($p < 0.01$) that indeed the dependency exists.

6.5 Applications of the formal model

Chapter 3 listed some potential applications of the formal model describing the hybrid planning problem. One of the key applications is using the model as a unifying evaluation framework to compare/analyze instantiations of hybrid planning. Grounding the analysis of an instantiation on the formal model has several benefits. First, while analyzing/designing a hybrid planner, the model highlights how the implementation barriers are handled, describing the outcomes, assumptions, and limitations of the design choices. Second, the model breaks down the bigger design problem into four subproblems, allowing separate investigation of design decisions for each. Therefore, such an analysis not only highlights the implicit assumptions made by designers of hybrid planners, but also gives confidence that all relevant challenges are addressed. In Chapter 4, we explained our approach to hybrid planning in the context of the formal model, thereby demonstrating the utility of the model in analyzing hybrid planning intantiations.

This section provides another example demonstrating how the formal model can be used to analyze instantiations of hybrid planning. The section uses the formal model to analyze an existing instantiation of hybrid planning that has been proposed by another researcher and has shown to be effective in its context. Our analysis not only provides insight into the strengths and weaknesses of the instantiation, but also highlights the (often implicit) assumptions behind the designs. Moreover, to indicate how the formal model can be used to compare hybrid planning instantiations, we compare this instantiation with the learning-based hybrid planner applied to the cloud-based system as described in Section 6.1.1.

6.5.1 Analysis of Hybridized Planner

To demonstrate practicality of the formal model, we analyze the hybrid planning instantiation proposed by Mausam et al. [79]; they refer to this instantiation as a *hybridized planner*. The instantiation uses three kinds of planning:

- *MDP planning:* deliberative planning uses an exact MDP solver *General Planning Tool* (ρ_{gpt})[16]. ρ_{gpt} uses labeled real-time dynamic programming (RTDP), which finds an optimal solution to an MDP once planning is complete. Labeled RTDP is *anytime* in nature: the planning process could be stopped anytime to get a sub-optimal plan; however, more planning time leads to a better plan. Since RTDP converges to an optimal plan slowly, ρ_{gpt} can be preempted after planning for a (predefined) fixed amount of time to get a sub-optimal plan.
- *Non-deterministic planning:* reactive planning uses a Model Based Planner (ρ_{mbp}) [11]. For a planning problem with probabilistic uncertainty and utility function to be optimized, ρ_{mbp} relaxes the problem in two ways: (a) it treats probabilistic transitions as non-deterministic transitions, and (b) it ignores the utility function while planning. The goal of planning is to reach one of the absorbing goal states — ones that end the process once reached. Model Based Planner (MBP) finds strong cyclic plans (i.e., one that admits loops but for every non-goal state there always exists a path to reach a goal unless the state is a dead end); however, such plans might be (highly) sub-optimal. ρ_{mbp} solves a relaxed problem, so it is more scalable than ρ_{gpt}.
- *Wait planning:* similar to our approach presented in Chapter 4, ρ_{wait} is one of the constituent reactive approaches for this instantiation. While formalizing this instantiation, the formal model helped us discover the implicit assumption that wait planning is one of the reactive approaches.

Here is a high-level overview of how a combination of ρ_{gpt} and ρ_{mbp} is used to balance quality and timeliness of planning. For a planning problem, ρ_{gpt} is used to plan for a fixed duration (say, t_{hyb}). Once t_{hyb} elapses, ρ_{gpt} returns a (possibly) sub-optimal plan (say, π_{gpt}). ρ_{mbp} is also expected to be ready with a plan (say, π_{mbp}) by t_{hyb}. Both ρ_{gpt} and ρ_{mbp} determine a universal plan (i.e., a policy) on the same state space. Therefore, in theory coordinating between their plans is not an issue (i.e., PLNCRD is solved). Policies determined by the ρ_{gpt} and ρ_{mbp} are merged together into a consolidated policy, which is a hybrid plan ω. Once a hybrid plan is determined, as discussed in Section 3.3.1, PLNSEL (or subproblem PTHSEL) is solved. Since the system waits (i.e., executes no action) until t_{hyb}, we can say that the system is governed by ρ_{wait} during that period.

The instantiation uses an algorithm to merge π_{gpt} and π_{mbp}. For a state s, π_{mbp} is used if s is neither marked as *solved* by RTDP, nor has been visited a number of times above a certain user-defined threshold (V) in RTDP. Intuitively, a number of visits of state s lower than V indicates low confidence in the quality of $\pi_{gpt}(s)$, therefore $\pi_{mbp}(s)$ is preferred. Sometimes, ρ_{mbp} may suggest that no solution exists from a state s; this might happen due to a choice of action in any of the proceeding states. In such cases, the algorithm recursively re-visits proceeding states to ensure if action for any of those states could be modified in order to find a solution from s. Further details can be found in the paper [79].

Similar to our approach to instantiate hybrid planning as outlined in Chapter 4, this instantiation makes certain assumptions to constrain a potentially infinite reachability graph (i.e., issue INFINITE-REACHABILITY-GRAPH). Here are the assumptions:

- **TWO-LEVELS-OF-PLANNING**: Hybrid planning is instantiated using one deliberative planning (i.e., using ρ_{gpt}) and two reactive planning approaches (i.e., using ρ_{wait} and ρ_{mbp}). ρ_{wait} and ρ_{mbp} determine a plan in negligible time and within t_{hyb} respectively. This assumption reduces the number of problem-planner nodes in a graph, making the problem of hybrid planning tractable in practice.
- **FINITE-HORIZON**: Each planning problem has a finite planning horizon. In other words, planning problems have explicit goal/end states. This assumption restricts the number of problem-planner nodes in the reachability graph.
- **DISCRETE-STATE-VARIABLES**: The value of state variables (e.g., time) is discrete. Otherwise, a reachability graph would have infinite nodes.

Once the size of the reachability graph is constrained, the next challenge is to deal with issue DELAY-IN-SOLVING-SUBPROBLEMS, i.e., solve the four subproblems (i.e., PRBSEL, PLRAST, GPHCON and PTHSEL) in a negligible time. The first three subproblems are simplified by the assumptions in such a way that no time is consumed to solve them, therefore the remaining concern is to solve PTHSEL in negligible time. To this end, it is assumed that the time required to merge π_{gpt} and π_{mbp} is negligible (**NEGLIGIBLE-MERGE-TIME**), and as already discussed, an approximate solution to PTHSEL is found once these two policies are merged. The merge algorithm relies on a user-defined variable (V) to decide whether to use an action from π_{gpt} or π_{mbp} for a particular state. This criterion is a kind of heuristic rather than a principled approach, therefore we treat the criterion as an assumption (say, **DELIBERATIVE-PREFERRED-CONDITIONALLY**).

Finally, issue REQUIRED-APRIORI-KNOWLEDGE-OF-EXECUTIONS also needs to be handled for a practical application of hybrid planning; i.e., solve PTHSEL without having *a priori* knowledge of utility of execution. However, this is not a problem, since merging π_{gpt} and π_{mbp} policies does not require this knowledge. Merging is either done based on expected utility (when a state is marked as solved by RTDP) or assumption DELIBERATIVE-PREFERRED-CONDITIONALLY.

Next, we analyze the proposed instantiation as we did for our approach in Chapter 4.

Constructing a Reachability Graph

Construction of a reachability graph has two steps: (i) restricting number of nodes in the graph to make it tractable, and (ii) connecting nodes if the timing and the preemption condition is satisfied; a (direct) connection guarantees a seamless transition between two nodes.

Restricting number of nodes: Similar to our approach, assumptions TWO-LEVELS-OF-PLANNING, FINITE-HORIZON, and DISCRETE-STATE-VARIABLES help restrict the number of problem-planner nodes in the reachability graph. Due to assumption TWO-LEVELS-OF-PLANNING, for a planning problem (say Pb), three nodes are possible. They correspond to (a) MDP (i.e., deliberative) planning that consists of planner ρ_{gpt} and problem description that represents probabilistic uncertainties and utility function, (b) non-deterministic (i.e., reactive) planning that consists of planner ρ_{mbp} and problem description that *ignores* probabilistic uncertainties and utility function,

and (c) wait planning that consists of planner ρ_{wait} and problem Pb. As in our approach, for the proposed instantiation, solving PRBSEL is not required, since output of PRBSEL is used by PLRAST, which is not explicitly handled, as explained in Section 6.5.1.

Connecting the nodes: Once the problem-planner nodes are finite, the next step is to connect the nodes in the reachability graph. For this purpose, the instantiation needs to solve PLRAST and GPHCON. A practical application of hybrid planning needs to deal with the issue of DELAY-IN-SOLVING-SUBPROBLEMS; however solving (both) PLRAST and GPHCON in negligible time is infeasible.

For this instantiation, no time is consumed to solve PLRAST because it is not handled explicitly. Solving PLRAST is not required since the outputs (the deadline and partial utility function) are not required to approximate a solution to a hybrid planning problem. Deadline is not required since, as discussed later, GPHCON does not require evaluating the timing condition. Partial utility function is not required since the merge algorithm (as discussed earlier) approximates a solution to PTHSEL without having knowledge of utility of executions.

The proposed instantiation does not explicitly handle GPHCON, since the timing and the preemption condition is satisfied by design. The timing condition is satisfied because non-deterministic plan execution waits until t_{hyb}. Since both ρ_{gpt} and ρ_{mbp} generate a policy for the same state space, once policies π_{gpt} and π_{mbp} are ready, they are ready to take over plan execution from each other for any state in the policies; thus, the timing condition between the two policies is satisfied. The timing condition between wait planning, and the other two planning approaches is satisfied because transition from the former planning to either of the later approaches only happens after t_{hyb}; that is, when ρ_{gpt} and ρ_{mbp} are ready.

The preemption condition between different planning approaches is satisfied because (a) both ρ_{gpt} and ρ_{mbp} generate a policy, and (b) domain is assumed to be *Markovian*. These properties provide a theoretical guarantee of a smooth transition from the empty action $\perp$ (suggested by ρ_{wait}) to π_{gpt} and π_{mbp}. In addition, these two properties also facilitate interleaving between π_{gpt} and π_{mbp}, once they are merged together to formulate a hybrid plan.

Finding a Path in a Reachability Graph

By now, we have analyzed how the instantiation restricts the number of problem-planner nodes and deals with the issue of connecting them. There are some similarities between this instantiation and our approach discussed in Chapter 4. First, FINITE-HORIZON, DISCRETE-STATE-VARIABLES, and TWO-LEVELS-OF-PLANNING are similar for both the cases. Second, they do not explicitly handle PRBSEL, PLRAST and GPHCON.

However, the approach to find a path (i.e., PTHSEL) linked to a hybrid plan is quite different between this instantiation and our approach. In our approach, for a planning problem (say, Pb), once a deliberative plan is ready it is preferred. However, while deliberative planning is in process, both reactive and ρ_{wait} nodes are available for selection, since both are ready with a plan in a negligible time. In contrast, for the instantiation analyzed in this section, the problem-planner node corresponding to MBP planning is not used until t_{hyb} (i.e., fixed time to preempt GPT). As a result, initially the wait planning node is a default choice in the path. However, nodes corresponding to both, MDP and non-deterministic planning, are available for selection after t_{hyb}.

Once π_{GPT} and π_{MBP} are merged, a hybrid plan is ready, thereby PTHSEL is solved. In other words, empty action $\perp$ is used until t_{hyb}, but merged policy π_{hyb} is executed thereafter.

To summarize this instantiation, certain assumptions help in restricting the size of a reachability graph. Similar to our approach, PRBSEL, PLRAST, and GPHCON are not explicitly handled. The solution to PTHSEL is found by merging the policies determined by MDP (i.e., deliberative) and non-deterministic (i.e., non-wait) planning based on the algorithm/heuristic discussed earlier. We have formalized the implicit assumptions about the utility that would make this instantiation valid.

6.5.2 Comparison Between the Learning-based and the Hybridized Planner

This section shows how hybrid planning instantiations can be compared using the formal model as a framework. Table 6.4 illuminates the similarities and differences between learning-based hybrid planning applied to the cloud-based system (as discussed in Section 6.2) and *hybridized planner*.[8] The basic ingredients of the graph are similar (three planners, similar node types, and a restricted space of possible nodes), but the planners approach reachability, timing, preemption, and path selection in different ways. For example, to solve PTHSEL, learning-based planner uses machine learning to decide between deterministic and wait planning node until an MDP policy is ready. In contrast, hybridized planner selects wait planning node until t_{hyb}, and afterwards, states are handled by π_{gpt} or π_{mbp} as per the merge algorithm. This suggests that once approaches diverge in one subproblem, they are likely to differ on the downstream subproblems. These observations raise a question: can subproblem solutions be reused across planners that solve the preceding subproblems differently? We anticipate that future work will provide more evidence for this question.

6.6 Threats to Validity

A central *construct* of the experiments, using the two systems, is the effectiveness of hybrid planning. The *internal validity* of our validation for the *effectiveness* claim is threatened by four potentially confounding factors. First, to measure effectiveness, we use a cumulative utility function (presented in Section 6.1). This function expresses the conflicting goals of such systems, and similar functions are used to measure performance of cloud-based systems and UAV team throughout related work [27, 28, 53, 63, 83, 85, 86, 94, 97, 107]. Such utility functions are applicable to systems that need to accumulate correct behavior while avoiding undesirable behavior (which is penalized), by performing actions with uncertain outcomes in uncertain environments (modeled as MDPs).

Second, our objective function for cross-validation (recall on *UseWait*) could lead to increased performance of the learning-based approach. This threat is mitigated by precision and recall for other classes also being high for our chosen classifier, and that the patterns are observed

[8]Learning-based hybrid planner is a specific instance of the hybrid planning approach presented in Chapter 4. Therefore, the planner has the same set of assumptions and addresses the four subproblems as discussed in Chapter 4.

Formal Aspect	**Learning-based Hybrid Planner**	**Hybridized Planner**
Planners	Prism model-checker (ρ_{mdp}) is used both for deterministic and MDP planning.	ρ_{mbp} and ρ_{gpt} is used for non-deterministic and RTDP planning, respectively.
Graph node types	Three kinds of nodes correspond to deterministic, MDP, and wait planning.	Three kinds of nodes correspond to non-deterministic, RTDP, and wait planning.
Handling PRBSEL	For learning-based and hybridized planner, potential combinations of problem-planner nodes are decided/restricted by various assumptions listed in Chapter 4 and Section 6.5.1, respectively.	
Handling PLRAST	Learning-based planner approximates deadline and partial utility function based on similar problems seen in the past.	Since both π_{gpt} and π_{mbp} are assumed to be ready by t_{hyb}, there is no need to calculate deadline. The partial utility function is not required because the merge algorithm approximates a solution to PTHSEL without knowing the function.
Handling GPHCON	The timing and preemption conditions are guaranteed given the assumptions formalized in Appendix A. When the respective assumptions are not satisfied, the conditions are not met.	The timing condition is satisfied once π_{gpt} and π_{mbp} are ready. Between wait and non-deterministic/RTDP planning the preemption condition is satisfied since both ρ_{gpt} and ρ_{mbp} generate a policy. Between non-deterministic and RTDP planning the preemption condition is satisfied since ρ_{gpt} and ρ_{mbp} generate policy on the same state space.
Handling PTHSEL	Learning is used to decide between deterministic and wait planning node until an MDP policy is ready.	Wait planning node is selected until t_{hyb}. Afterwards, states are handled by π_{gpt} or π_{mbp} as per the merge algorithm.

Table 6.4: Comparison between learning-based planner for the cloud-based system and hybridized planner in the context of the formal model.

in experiments with a broad range of classifier performances. However, it is possible that the classifier could have led to higher utility than that of LB-W and LB-R.

Third, the relative performances of the condition-based and the learning-based approaches are due to the specific conditions for triggering reactive planning. Although this condition is tied to the system's utility function, it is possible to fine-tune it further, to approach the theoretical limit of perfectly matching a situation to a reactive approach. However, this fine-tuning is difficult in practice due to the multi-dimensional utility function and uncertainty in the external environment that leads to uncertainty in (reactive) action outcomes. Therefore, we expect this tuning to have a minor effect on the evaluation results.

Fourth, the performance of the learning-based and the condition-based approach may depend on system parameters (e.g., server costs, ECM factors). Different parameter values might change the penalties for reacting incorrectly. This threat is mitigated by two different test-beds and hybrid planners, and a sizable set of traces/missions with substantial variation, which leads to a robust assessment of planner performance through cross-validation. To our knowledge, this is the largest set of traces ever used for an evaluation of a cloud-based system.

The *external validity* of our conclusions is threatened by the use of only two systems and three reactive planners (ρ_{det}, ρ_{mdps}, and ρ_{wait}). In theory, the learning-based approach should apply to any number of reactive approaches in set $\mathcal{F}$; however, we evaluate using only two planners at a time. As a sanity check, we compare the learning-based approach with deliberative only and NW-HP mode; these modes are constrained to use only one of the reactive approaches. The fact that the learning-based approach outperforms them indicates that the classifier was able to switch effectively between the reactive approaches used to instantiate a HP. This conclusion is also supported by the precision/recall values from the cross-validation of the classifier. Furthermore, labeled training data can be used as a basis for narrowing down the set of constituent planners.

The dependencies between constituent and hybrid planners are dependent on various factors, including the utility function and assumptions behind the approach. We expect these dependencies to hold for any utility function that is accrued over states of traces/missions and reflects that fast reactions are vital to the system's goals, yet the choice of when to react is not obvious. We further mitigate the threat to validity by evaluating on two published testbeds (i.e., SWIM [87] and DartSim [88]) for self-adaptive research. The domains for these testbeds differ in significant ways, as already discussed in Section 6.3.2.

6.7 Summary

This chapter presented results that support the claims of the thesis. Using the two case studies that differ in significant ways, we have demonstrated that hybrid planning is effective, general, and flexible. The case studies also showed that the proposed learning-based approach to solve PLNSEL is more effective than the condition-based. In addition, we analysed a hybrid planning instantiation suggested by other researchers and compared it to the learning-based hybrid planner used for the cloud-based system. Users can use these examples as a handbook to apply the formal model for analyzing and comparing hybrid planning instantiations.

Chapter 7

Guidelines to Apply Hybrid Planning

Prior chapters outlined the formal model describing the hybrid planning problem (cf. Chapter 3), an approach to solve the problem (cf. Chapter 4), and the hybrid planning algorithm (cf. Chapter 5). Suppose a practitioner is interested in applying hybrid planning. This chapter provides guidelines for the practitioner to apply the principles outlined earlier in order to use hybrid planning for a realistic self-adaptive system.

7.1 Introduction

Alice is designing a self-adaptive system for a domain where both the timeliness and the quality of planning is critical. She has passed a graduate level course on artificial intelligence (AI) that included topics such as automated planning and machine learning. Consequently, although not an expert in planning and machine learning, she has a general understanding of various planning approaches (e.g., classical planning, search heuristics, MDP/POMDP planning), and different machine learning algorithms (e.g., supervised and unsupervised learning), models (e.g., decision trees, support vector machines) and techniques (e.g., cross-validation).

While designing the system, she is struggling to find a planner that can balance the timeliness and the quality of planning for her particular adaptive system and application domain. She has tried different off-the-shelf planning approaches but the approaches that, in general, provide quality plans tend to take longer to plan, leading to loss in utility, particularly in emergency situations. In contrast, the approaches that can provide a timely response tend to provide lower-quality plans. Now, she is left with two options: (a) compare the existing approaches and pick the one that is "best" (e.g., performs better than others on average), or (b) develop a customized planner that can outperform the off-the-shelf planners. She has already explored the first option, and is inclined to explore the second option in search of a better planner. However, not being an AI researcher, she anticipates that it will be difficult for her to develop such a customized planner.

While exploring potential planning solutions, she came to know about the idea of hybrid planning. She is interested in applying hybrid planning, but wonders about questions such as (a) how to identify an appropriate set of reactive and deliberative planners that can handle the trade-off between the timeliness and the quality of planning, (b) whether to use condition-based or learning-based hybrid planning, and (c) how to implement learning-based hybrid planning.

This chapter aims at helping Alice to answer these questions. It is structured as follows: Section 7.2 provides informal guidelines and a quantitative approach to select an appropriate set of planners to instantiate hybrid planning; Section 7.3 provides insights on how to decide between condition-based and learning-based hybrid planning; and Section 7.4 highlights challenges to implementing learning-based hybrid planning and potential solutions to those challenges.

7.2 Instantiating Hybrid Planning

For applying hybrid planning, a key step is to instantiate hybrid planning using a set of (deliberative and reactive) planning approaches that can balance quality and timeliness of planning. However, for a domain, choosing such a set is a non-trivial decision due to a large number of choices for a planning approach.[1] For instance, assume the domain has uncertainty in action outcome, therefore MDP planning can be a suitable choice to determine plans. But, to instantiate MDP planning, one can configure options such as the algorithm (e.g., value-iteration and policy iteration) to solve an MDP, optimization threshold[2], planning horizon, and the subset of actions to be considered for planning as done for the DART system (cf. Chapter 6) [78]. Even with a small number (e.g., 10) of binary configuration options, a large number of MDP planners can be instantiated.

This section aims at providing guidelines and a quantitative approach to identify an appropriate set of planners to instantiate hybrid planning. To identify the set, a practitioner can use the guidelines followed by the quantitative approach; they complement each other. However, the guidelines and the approach are independent of each other, therefore can be used in isolation.

7.2.1 Informal Guidelines to Instantiate Hybrid Planning

The choice of a (deliberative or reactive) planner for a domain depends on the properties of planning problems used to represent adaptation situations. The properties of a planning problem have several dimensions such as whether (a) the objective of planning is to reach a predefined goal state, or to maximize a reward (i.e., utility) function, (b) action outcome is deterministic, or non-deterministic, (c) there is full or partial observability of the current state, (d) uncertainty in the domain is captured using probabilistic models, (e) state variables are discrete or continuous, and (f) there is a single agent or multiple agents to execute a plan. Based on certain assumptions in this thesis, we scope the kinds of planning problems under consideration in the following ways:

- The objective of planning is to maximize the expected utility calculated through a multi-dimensional function that captures both quality and timeliness of planning (cf. Chapter 1).

[1] As a reminder, we use the term "planning" in a broad sense, referring to any decision-making approach that could be used to determine adaptation plans. Throughout the thesis, we use the term “planner” and “planning approach” interchangeably. As formalized in Chapter 3, both the terms refer to the black-box that takes a planning problem as an input and returns a plan. This black box encapsulates various planning aspects such as the planning tool that implements a planning algorithm/heuristic and its configuration options. Therefore, two instances of the same planning tool, but with different configuration options will be considered as different planners.

[2] The value to decide when the improvement in a policy between two successive iterations is not significant enough to continue the optimization process.

- Planning problems have either no uncertainty (i.e., they are deterministic) or use probabilistic models (e.g., MDPs and POMDPs) to capture uncertainty (i.e., non-determinism) in a domain.[3]
- The planning problem has a finite planning horizon (cf. assumption FINITE-HORIZON stated in Chapter 4).
- The value of state variables (e.g., time) is discrete (cf. assumption DISCRETE-STATE-VARIABLES stated in Chapter 4).
- We assume a single agent executes a plan, therefore multi-agent planning approaches are out of scope.

Instantiating a Deliberative Planner

To balance the quality with the timeliness of a plan, hybrid planning requires a smooth transition from a reactive plan to a possibly higher-quality deliberative plan. For such a transition, according to the formal model, both the *timing* and the *preemption* condition need to be satisfied (cf. Chapter 3). In short, the timing condition is that the deliberative plan should be ready at the moment of transition, and the preemption condition is that the deliberative plan should have a provision for the state of a system at the point of transition.

To solve this transition problem (which we referred to earlier as the planning coordination problem (PLNCRD)), as detailed in Chapter 4, our approach has two assumptions: (a) deliberative planning generates a universal plan/policy (one containing state-action pairs for all the reachable states from the initial state), where a mapping from a state (say s) to an action (say a) suggests a be executed in s; and (b) the operating domain is assumed to be *Markovian*: the state after a transition depends only on the current state — not on the sequence of states that preceded it [78].[4] The combination of these two characteristics increase the chances of successful preemption if reactive and deliberative planning use the same initial state (cf. Chapter 4).

Given the constraints that deliberative planning needs to handle probabilistic uncertainty, generate a policy and plan for *Markovian* domains, MDP and POMDP planning are two potential choices to determine a plan. Typically, MDP planning is used for domains with the probabilistic uncertainty in outcomes of actions [78].[5] POMDP planning is a generalization of MDP planning since it captures the probabilistic uncertainty both in outcomes of actions and in observability of the underlying state [55]. MDP/POMDP planning generate a policy-structured plan that helps to deal with uncertainty. To explain further, when executing the policy, due to the uncertainty, a system might end up in one of the several anticipated states; however, irrespective of the current state of the system, the action corresponding to that state can be found since the policy has state-action pairs for all states reachable from the initial state. Finally, both MDP and POMDP are suitable to plan for *Markovian* domains [44].

[3]Probabilistic modeling of uncertainty is needed to calculate expected utility calculated through the utility function [44].

[4]A *non-Markovian* domain can be represented as a *Markovian* domain using additional state variables to capture history; however, this may increase the state space and could lead to an increase in planning time, thereby negatively impacting the timeliness of planning.

[5]By specifying transition probability as 1, deterministic transitions can also be captured by MDP planning as is done for the cloud-based system used for the thesis evaluation.

Due to their ability to handle uncertainty, which is often required by realistic systems, MDP/POMDP planning can potentially determine quality plans; therefore, MDP/POMDP planning can be a good choice for deliberative planning. Both MDP and POMDP planning have been used in a variety of domains such as robotics [71], and cyber-security [41, 120]. MDP planning can also be extended to game-theoretic planning that can incorporate competitive or collaborative behavior, modeled as (turn-based) stochastic multi-player games (SMGs) [26].[6] SMGs can be particularly useful to provide quality plans to deal with cyber-attacks by modeling a defender (i.e., a self-adaptive system) and attackers as distinct agents such that the goal of planning is to determine a plan that helps the defender to protect the systems against the attackers [31].

Although MDP/POMDP planning can provide quality plans they might fail to provide a timely plan when invoked at run time (i.e., online) [73]; specifically, solving a POMDP is often intractable except for small problems due to their complexity [100]. Although various optimization algorithms have been suggested to improve the planning time for MDP [78] and POMDP [96, 100, 106] planning, planning delay in probabilistic domains is still an ongoing challenge.

If an MDP/POMDP policy can be determined offline (i.e., no run-time overhead), these approaches can provide a quick and a quality response to a situation as suggested by Mostafa et al. [89]. However, for many realistic systems such as the two systems used for the thesis evaluation, offline planning is often difficult since: (a) upfront consideration of all the possible states and transitions for planning might not scale for the systems, and (b) uncertainty in the operating domain could lead to difficulty in upfront probabilistic modeling of uncertainty in a planning problem specification used for the offline planning; imprecise modeling of uncertainty can negatively impact the quality of planning.[7] Therefore, for such systems, online MDP/POMDP planning could be more suitable than offline planning.

To summarize, given the constraints as mentioned earlier, MDP and POMDP planning could be good choices for deliberative planning. MDP and POMDP planning can provide quality plans, but the planning might be time-consuming, and therefore, not suitable to provide quick response to emergencies. But, by combining POMDP/MDP planning with reactive planning, one can instantiate a hybrid planner that can deal with the timeliness-quality trade-off. The next section discusses various ways to instantiate reactive planning.

Instantiating Reactive Planners

To provide a quick response, our approach to hybrid planning combines a deliberative planner with a set of reactive planners, which provide plans, particularly in emergencies (e.g., constraint violations), in negligible time. Intuitively, planning time is considered negligible when utility loss during the planning process is insignificant compared to the (potential) utility gain from the plan determined from the process. The threshold for planning time to be considered as negligible depends upon the operating domain. Moreover, even for the same domain, the threshold for reactive planning might vary depending upon context. For instance, for a self-driving car, the reactive planning time to avoid a crash with another vehicle next to it would likely be in the range

[6] A tool for SMG planning can be available online: https://www.prismmodelchecker.org/bibitem.php?key=CFK+13

[7] In the systems used for the thesis evaluation, instead of doing offline planning by considering all the possible states and transitions over the entire execution period (for the systems), we do online planning with a shorter planning horizon, as detailed in Chapter 6.

of milliseconds; however, in case the other vehicle is few meters away, the car could tolerate a longer reactive planning time in return for a better a (reactive) plan. Therefore, it is often difficult to define this threshold manually at design time.

To deal with this issue, Section 7.2.2 provides a quantitative approach that helps to identify an appropriate set of reactive planners (to instantiate hybrid planning for a domain) by taking reactive planning time and the plan quality into consideration so that designers do not have to define the reactive planning time threshold manually. However, before finalizing the set of reactive planners, one needs a candidate set of reactive planners to evaluate using the quantitative approach. To this purpose, this section discusses three techniques to build the candidate set of reactive planners. These techniques are: (a) anytime planning, (b) using precomputed plans, and (c) planning with a relaxed deliberative planning problem.[8]

(a) Anytime Planning: To solve an MDP/POMDP, the state-of-the-art algorithms (e.g., value iteration and policy iteration) are based on the idea of incremental planning, known as "anytime" planning. Typically, anytime planning algorithms are optimizing in nature: the planning process can be interrupted at any time to get a sub-optimal plan, and longer planning times lead to better plans [122]. For example, in Figure 7.1, the planning (i.e., optimization) process can be interrupted at time t_0, t_1, or t_2 to obtain a valid but potentially a sub-optimal plan. However, the quality of plan will be lowest at t_0 and highest at t_2.

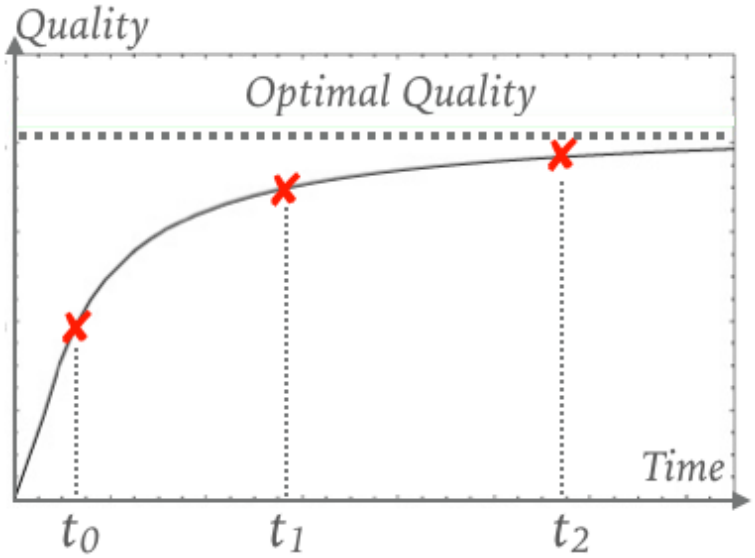


Figure 7.1: Anytime algorithms are optimizing in nature and can return a valid plan to a planning problem even if planning is interrupted before the optimizing process ends.

This anytime nature of these algorithms can be utilized to provide a reactive response from the deliberative (i.e., MDP/POMDP) planner itself. To explain further, when a system observes a (planning) problem, deliberative planning can be invoked that will provide a quality plan once the planning is over. Meanwhile, intermediate plans (e.g., at time t_0, t_1, or t_2) can be used to

[8]These techniques can be used in combination with the quantitative approach (as recommended in this thesis) as well as in isolation (i.e., without the approach) to identify the set of reactive planners.

provide a reactive response. However, the key challenge to use this approach (to provide a reactive response) is to decide how long to wait before one can get a "reasonable" (e.g., non-fatal) plan. In the context of this thesis, intermediate plans are assumed to be ready in a negligible time (cf. Chapter 4), and if the plans are not good enough, the learning-based hybrid planning will not use the anytime approach (from the set of reactive plans) to provide reactive plans. For more general solutions to this challenge, one can refer to different variations/frameworks proposed by researchers in the context of MDP [17, 60, 114] and POMDP [96, 119] planning.

(b) Using Precomputed Plans: To determine adaptation plans, researchers have suggested a diverse set of planning approaches such as rule-based adaptation [27] and case-based reasoning [107, 115] that, generally speaking, determine an adaptation plan quickly because the plan is not generated at run time, but rather selected from an existing set of precomputed plans; however, quality (in a utility-theoretic sense) of plans might be bad, since the set of precomputed plans may not be sufficient to handle unforeseen problems or environments [1]. Similarly, fuzzy-logic determines plans quickly since it uses a predefined set of rules to determine a plan [76]; however, the approach is not robust unless there is a comprehensive set of rules, and having such a set is non-trivial, particularly for domains with uncertainty [8]. To summarize, approaches such as rule-based adaptation, case-based reasoning, and fuzzy-logic can find a plan quickly but the plan might be of low quality. However, since these approaches have potential to determine plans quickly, hybrid planning can be instantiated with these approaches (as reactive planners) to provide a quick response to emergencies, and a deliberative planner that can handle uncertainty better than these reactive ones.

(c) Planning with the Relaxed Deliberative Planning Problem: Another technique to instantiate reactive planning is to plan for relaxed planning problems compared to the one used for deliberative planning. The deliberative planning problem can be relaxed by reducing the planning search space and/or by relaxing the planning goal. Planning with a reduced search space and/or a relaxed goal is likely to result in reduced planning time. Once the planning problem has been relaxed one can use either the deliberative planner itself or search heuristics that find quick, but potentially sub-optimal plans. This section discusses techniques to relax a planning problem. These techniques are summarized in Figure 7.2.

One approach to relax the deliberative planning problem is to reduce the search space; planning with the smaller space can potentially reduce the planning time. The search space can be reduced by decreasing the states and/or the transitions, for instance, by planning with a subset of actions, ignoring low-probability transitions [19], and/or reducing the planning horizon (when the planning goal is not an explicit state). Figure 7.3 illustrates how such techniques can reduce a search space for planning. Researchers have also suggested heuristics such as ignoring transitions that lead to negative outcomes with respect to the planning goal [13].

Planning time can also be reduced by relaxing the planning goals. For example, instead of finding an optimal plan, which typically requires finding all the possible plans and comparing them to identify the optimal plan, a planner can also settle for a sub-optimal plan (e.g., the first plan determined by the planner). Another way to relax the planning goal is to plan for a subset of goals. For instance, for the cloud-based system presented in Chapter 1, a planner can focus on

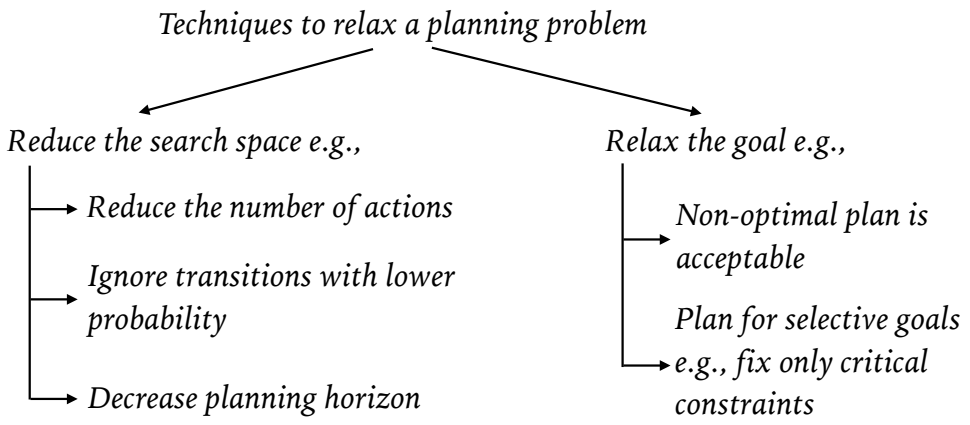


Figure 7.2: Summary of potential techniques to relax a planning problem.

just fixing a response time constraint violation instead of optimizing the utility as calculated by Formula 5.1; the formula considers other quality attributes such as decreasing the cost of servers and increasing revenue. When planning for a subset of goals, a planner is likely to find a plan quickly compared to finding an optimal plan that maximizes the utility.

This section presented guidelines and the techniques to relax a planning problem. Although the guidelines and techniques are not comprehensive, these can be a good starting point to identify a potential deliberative planner and a set of reactive planners to instantiate hybrid planning. Due to the informal nature of the guidelines and the techniques, even after applying them, there is a possibility that one might end up with some poor planners in the set of reactive planners. When using learning-based hybrid planning, the classifier will automatically learn not to invoke them. However, when using condition-based hybrid planning, one might need to do some experimentation and manual analysis to identify the most effective set of reactive planners in the context of the predefined conditions to be used to invoke reactive planning. Section 7.3 discusses a quantitative approach to map a predefined condition with an appropriate reactive planner.

There is also a possibility that even after applying the guidelines, one is left with more than one choice for deliberative planners; however, only a single deliberative planner is allowed by the hybrid planning approach proposed in this thesis (cf. Chapter 4). Section 7.2.2 proposes a quantitative approach to identify the most effective deliberative planner from a given set of deliberative planners; this planner can be used to instantiate hybrid planning in combination with the set of reactive planners. Although one can rely only on the guidelines to instantiate hybrid planning, we recommend using both the guidelines and the quantitative approach. Using the guidelines in combination with the quantitative approach would act as two levels of filtering, and therefore is likely to identify a better set of planners to instantiate hybrid planning.

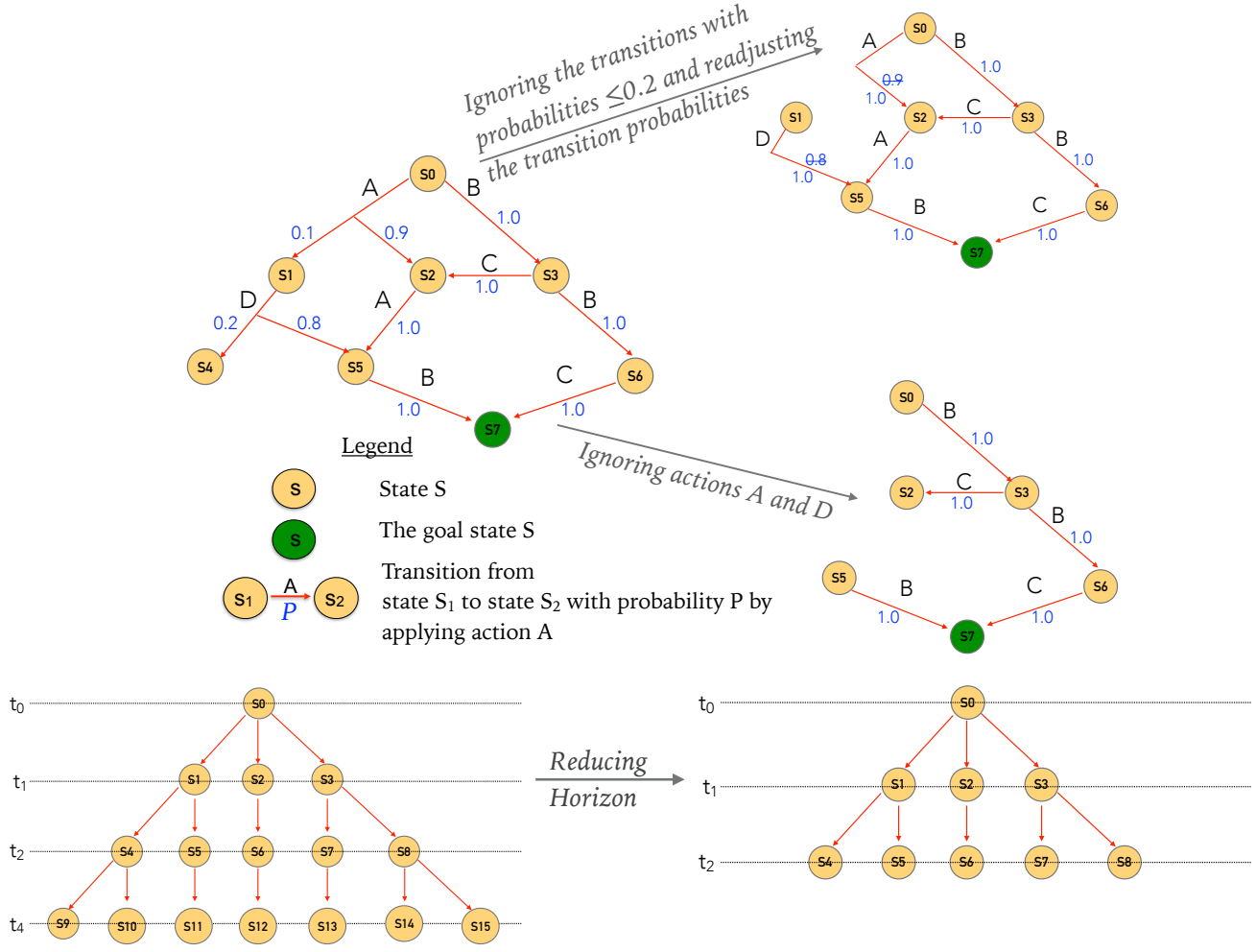


Figure 7.3: Illustration of reduction in a planning search space by applying the potential techniques such as ignoring actions, ignoring low probability transitions, and/or reducing the planning horizon.

7.2.2 Quantitative Approach to Instantiate Hybrid Planning

Given set $\mathcal{F}$ of reactive planners and set $\mathcal{D}$ of deliberative planners, the quantitative approach helps to identify the most effective deliberative planner in $\mathcal{D}$ such that hybrid planning is instantiated using the deliberative planner and the reactive planners in set $\mathcal{F}$. The quantitative approach is inspired by the labeling process for learning-based hybrid planning detailed in Chapter 4. In fact, when using learning-based hybrid planning, as discussed later, the steps of the quantitative approach are naturally captured by the offline phase of learning-based hybrid planning; therefore, applying the quantitative approach will not incur extra efforts or time.

The quantitative approach has two steps: (a) collect/identify a training set of planning problems similar to the ones expected at run time, and (b) use these problems and a probabilistic model checker to evaluate the deliberative planners to identify the deliberative planner, which is most effective when used in combination with the reactive planners in set $\mathcal{F}$.

Identifying Sample Problems

The first step to apply the quantitative approach is to collect/identify a set of sample problems similar to the ones expected at run time.[9] To evaluate the set of deliberative planners, it is crucial to cover the planning problem space comprehensively. However, this is challenging due to a potentially infinite problem space and its unknown structure. There is no single selection strategy that fits all systems and domains. Therefore, one needs to tailor the sample set to the system's context and requirements. Fortunately, modern-day systems produce large amounts of data that can be utilized to build the sample problem set. For example, in our evaluation systems, we mine sample planning problems from the available traces containing the typical load patterns [37] (for the cloud-based system) and randomly sample the space of missions (for the UAVs).

Choosing the Deliberative Planner

This step determines the deliberative approach $\rho_r^d \in \mathcal{D}$ that performs best in combination with reactive planners in $\mathcal{F}$ for a sample planning problem ξ. In the process, at the end of this step, we obtain the most effective deliberative planner in the context of the set of sample planning problems.

To evaluate a combination of a reactive and a deliberative planner for problem ξ, we need to estimate how well the planning goals are met when the reactive plan (determined by the reactive planner) is executed followed by the execution of the deliberative plan (determined by the deliberative planner) when it is ready; in other words, we estimate the performance of hybrid planning (for ξ) when the reactive planner is invoked in combination with the deliberative planner. However, as explained in Chapter 4, in the presence of uncertainty in the environment, it is difficult to evaluate a combination of plans given that its performance may vary across plan executions (for the same problem) because of different possible outcomes leading to different plan execution paths. To overcome this problem, similar to the labeling process for the learning-based approach, we propose using a *probabilistic model checker*, which considers probabilistic uncertainty when evaluating a combination of reactive and deliberative plan. For each sample problem, a model

[9] The same set can be used to train a classifier if learning-based hybrid planning is used.

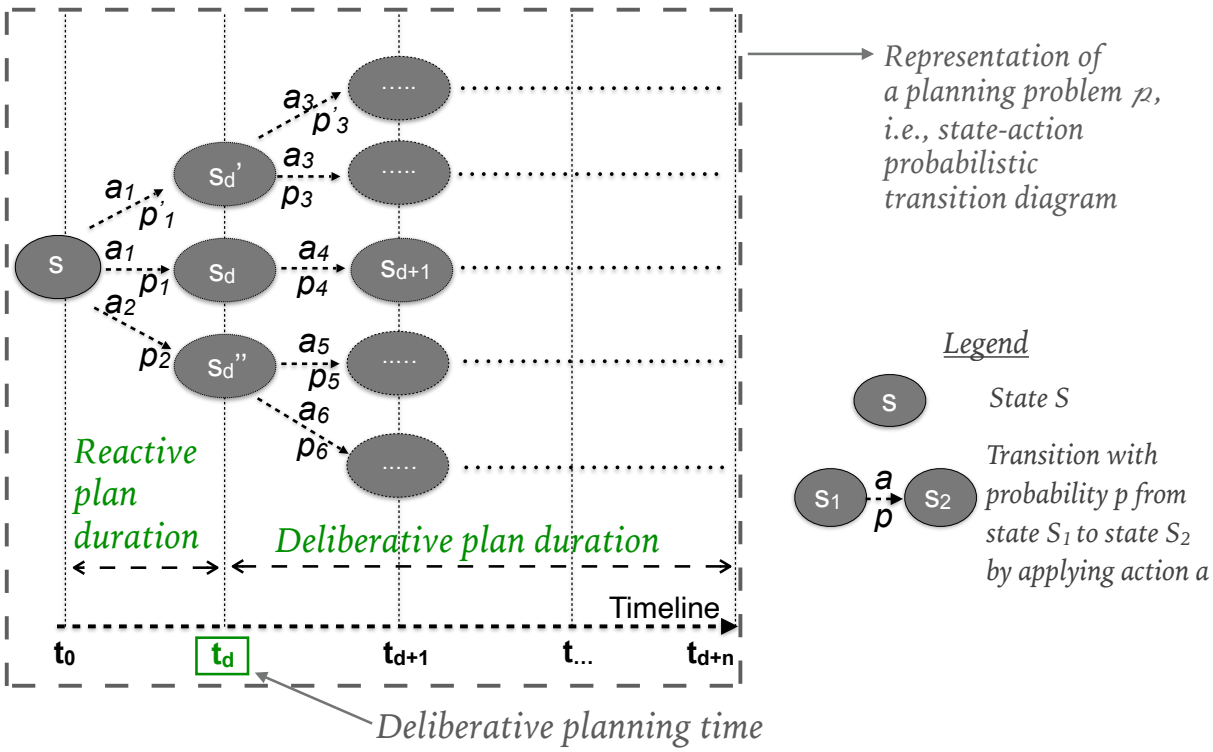


Figure 7.4: Evaluating the combination of reactive planner ρ_r^i and deliberative planner ρ_d^j, i.e., calculating the utility for the combination of reactive and deliberative plan.

checker evaluates a pair of a reactive and a deliberative planner under probabilistic uncertainty, by considering all possible execution paths weighted with their probabilities. Finally, the most effective (e.g., the one that provides the highest expected utility on average) deliberative planner is selected to instantiate hybrid planning.

Figure 7.4 illustrates how a model checker can be used to evaluate the combination of reactive planner (ρ_r^i in $\mathcal{F}$, producing plan π_r^i) and deliberative planner (ρ_d^j, producing plan π_d^j in time t_d). The outcomes of executing actions from each plan are uncertain, and a model checker handles this uncertainty by aggregating the quality of possible outcomes as expected utility, denoted U^{ij}. To compute U^{ij} for ξ, the model checker calculates the expected utility for the combination of plans π_r^i (until time step t_d) and then π_d^j. If set $\mathcal{F}$ and $\mathcal{D}$ have *M* and *N* planners respectively, then each sample problem ξ requires *MxN* evaluations corresponding to each pair $\langle \rho_r, \rho_d \rangle$ such that $\rho_r \in \mathcal{F}$ and $\rho_d \in \mathcal{D}$. For the (MDP) domains with probabilistic uncertainty in action outcomes, one can use PRISM [69] as a probabilistic model checker to calculate the expected utility of a combination. However, the quantitative approach is not limited to any specific model checker. For instance, in the case of POMDP domains, which also have uncertainty in the underlying state, one can use model checkers that support such domains.

Finally, we need to compare expected utilities for each pair to determine the effective delibera-

tive planner. There can be various heuristics to define the "most effective" planner. For instance, for all the sample problems, the deliberative planner can be the one that provides: (a) the highest mean utility, or (b) the highest median utility, or (c) the best worst-case performance (e.g., never provides expected utility below a predefined threshold). Depending upon a system's requirements, practitioners can decide on an appropriate heuristic. For example, in the cyber-security domain, going with the deliberative planner that provides the best worst-case performance might be useful to prevent an attack; such a planner is likely to protect the system more reliably compared to a planner that provides highest mean utility. In contrast, for the domains (i.e., cloud-based systems) where systems can recover from failures without significant damage, one might choose the deliberative planner that provides the highest mean utility. Chapter 6.3 illustrates how the evaluation of combinations of a reactive and a deliberative plan using model checking works in practice.[10]

7.3 Choosing Between Condition-based and Learning-based Hybrid Planning

Suppose Alice instantiates a hybrid planner, which has the potential to balance the timeliness and the quality of planning in the context of her self-adaptive system. The next challenge is to decide whether to use the condition-based or the learning-based approach to solve the planning selection problem (PLNSEL), i.e., choose an appropriate reactive planner to solve a planning problem. We recommend using the learning-based approach since: (a) it does not require domain expertise to decide which reactive planner needs to be invoked, and (b) full/partial automation is possible for the approach (including the specification generation for model-checking), which can relieve her from the painstaking and error-prone process of identifying the conditions to invoke (reactive) planners. Moreover, the experimental results presented in Chapter 4 demonstrate that the learning-based approach is likely to provide *more utility* and is *less risky* compared to the condition-based approach.

However, if Alice is still interested in applying condition-based hybrid planning, she needs to answer two questions: (a) given the system requirements and utility function, can the conditions that require invoking a reactive planner be manually identified at design time, and (b) can those conditions be manually mapped to an appropriate reactive planner (in set $\mathcal{F}$) to solve PLNSEL. If both the steps are feasible, condition-based hybrid planning can be applied.

The evaluation results from the probabilistic model checking (used by the quantitative approach) can be referred to mapping a condition to a reactive planner. To elaborate further, suppose after the quantitative approach, it was decided to instantiate hybrid planning using the reactive planners in set $\mathcal{F}$, and deliberative planner ρ_d. Using the model checking data for the pairs $\langle \rho_r, \rho_d \rangle$ where $\rho_r \in \mathcal{F}$, one can calculate the correlation between the planning problems capturing a specific condition, and the reactive planner that provided the highest utility (in combination with ρ_d) for that problem. If a (medium to strong) correlation is found between the planning problems capturing a specific condition, and a particular reactive planner, then the condition can be mapped to the planner; in other words, when that condition is observed, that reactive planner is invoked to

[10]In that Chapter, the evaluation was used for labeling the problems when implementing learning-based hybrid planning).

provide a quick response.

7.4 Implementing Learning-based Hybrid Planning

Suppose Alice decides to use learning-based hybrid planning. To implement the approach, the two key challenges are: (a) identifying a set of sample problems to train a classifier, and (b) training a classifier to solve the planning selection problem (PLNSEL). As already mentioned, to select reactive planners effectively, it is crucial to cover the planning problem space comprehensively. However, to build such a set of sample problems, no single selection strategy fits all systems and domains, and we suggest tailoring the sample set to the system's context and requirements. Fortunately, modern-day systems (e.g., Amazon Web Services (AWS), Netflix, autonomous vehicles) produce large amounts of data (e.g., planning problems) that can be used to train a classifier.

Assuming the labeling process goes well using probabilistic model checking, the second challenge is to train a classifier, which requires: (a) feature selection, and (b) identifying the machine learning algorithm that can classify the planning problems.

To train a classifier, we need to identify relevant features of planning problems that help separating the classes corresponding to each reactive planner as explained in Chapter 4. For the two systems used for the thesis evaluation, we use two complementary sets of features: ones representing the current state of the system, and ones describing how the system will evolve in the future. These features reasonably represent a planning problem by capturing the initial state and future transitions of the problem. However, one can also investigate techniques such as principal component analysis (PCA) to identify the optimal set of features [2].

The final step is to identify a machine learning algorithm that can classify the sample problems; this step includes the training of a classifier because before evaluating the classifier one needs to train it. A commonly used technique to determine a set of potential classifiers is to plot the sample problems in a plane, and (visually) observe the shape of the boundary that can separate different classes (corresponding to each reactive planner). Depending on the shape of the classifying boundary, an appropriate set of algorithms can be selected for further evaluation using cross validation. For instance, if the shape is linear, one can try algorithms such as logistic regression and stochastic gradient descent [81].

For cross validation, it is critical to have an appropriate metric to evaluate the performance of a classifier; typical metrics are *accuracy*, *recall*, *precision*, and *F1 score* (combines precision and recall). There is no formal approach to decide which metric is to be used; the decision usually involves analysis of the data. For example, in our experiments, we did not use the typical measure of *accuracy* to define the "best" classifier in cross-validation. To explain in the context of the cloud-based system, due to the data being skewed in favor of using a particular reactive planner (say, $\rho_r^{'}$), even a trivial classifier that always predicts to use that planner would have a relatively high accuracy. Instead, we analyzed recall, precision, and F1 score for the class corresponding to each reactive planner to judge a classifier's performance. More specifically, as it turned out in our experiments, the limitations of the training data made it challenging to discover situations when a reactive planner (other than $\rho_r^{'}$) is the best choice; therefore, in cross validation we maximized the recall value for the class corresponding to that reactive planner (cf. Chapter 6).

7.5 Summary

The previous chapters presented the theoretical aspects of hybrid planning. This chapter intends to ease the adoption of hybrid planning by addressing the questions a practitioner needs to answer when applying hybrid planning. These questions are: (a) how to identify a set of constituent planners to instantiate a hybrid planner that can balance the timeliness and the quality of planning, (b) how to choose between condition-based and learning-based hybrid planning, and (c) how to implement learning-based hybrid planning that includes finding a set of sample problems, selecting the feature set of a planning problem, and identifying the machine learning algorithm that can classify the planning problems.

To address the first question, the chapter provides both guidelines, and also a quantitative approach to identify constituent planners to instantiate hybrid planning. To address the second and the third questions, the chapter provides guidelines both for choosing between condition-based and learning-based hybrid planning, and implementing learning-based hybrid planning; these guidelines are built upon the empirical findings (cf. Section 6.4) as discussed in Chapter 6.

Chapter 8

Discussion and Conclusion

The previous chapters presented the thesis contributions that help to understand the problem of hybrid planning and apply it in realistic contexts such as a self-adaptive cloud-based system and team of UAVs. Specifically, Chapter 3 formulated the problem of hybrid planning; Chapter 4 outlined our approach to solve the problem; Chapter 5 presented the hybrid planning algorithm; Chapter 6 validated the thesis claims (i.e., the effectiveness, the generality, and the flexibility of hybrid planning), and demonstrated the applicability of the formal model describing the problem of hybrid planning; Chapter 7 provided informal guidelines and a quantitative approach to instantiate hybrid planning, decide between condition-based and learning-based hybrid planning, and implement learning-based hybrid planning. This chapter analyzes the thesis contributions in detail, and discusses the assumptions behind the proposed hybrid planning approach, and how to relax the assumptions that are not fundamental to our approach. It also provides a potential list of short-term and long-term research projects in the future.

8.1 Thesis Contributions

This thesis contributes to both the theory and the practice of hybrid planning. This section analyzes these contributions and discusses their broader impact.

8.1.1 Theoretical Contributions

The theoretical contributions of this thesis are as follows:

A Formal Model Describing the Problem of Hybrid Planning

Understanding the hybrid planning problem is a critical step towards solving it. This thesis formally defines the problem to describe its general nature, and decomposes it into four computational subproblems (cf., Table 4.1). Moreover, the model links the four subproblems to the two fundamental challenges (i.e., PLNCRD, and PLNSEL) of hybrid planning.

The formal model uses the *a posterioi* semantics of utility and planning time. This means that the model assumes that we know the post-execution states and how the non-determinism

in state transitions was resolved (cf. Chapter 3). In contrast, an *a priori* (i.e., pre-execution) notion assumes uncertainty both in state transition and planning time, which makes it difficult to understand and define the problem of hybrid planning. Compared to the *a priori* view, using the *a posteriori* notion is different in two notable ways: (a) it is not required to handle uncertainty since the state transitions are deterministic after a transition has taken place, and (b) the planning time is known. Despite using the *a posteriori* notion of utility and planning time, the formal model defines the problem of hybrid planning without the loss of generality. However, as a step closer to implementing hybrid planning in realistic systems, the thesis also analyzes the hybrid planning problem in the *a priori* semantics (cf. Chapter 5.

There are various applications of the formal model: it (a) can be used to represent and analyze existing instances of hybrid planners to understand their strengths and weaknesses, (b) is a unifying framework to compare existing hybrid planners, and (c) sets the stage for going beyond the solution proposed in this thesis to find even better solutions to hybrid planning.

An Illustration of the Applicability of the Formal Model

The *a posteriori* semantic could initially be counterintuitive to users who want to apply the model to analyze/design hybrid planners. To help the users, using two different hybrid planners as examples, the thesis demonstrates how the model can be used to analyze and compare hybrid planners. For instance, Chapter 4 uses the model to analyze our approach to hybrid planning; the analysis grounded in the formal gives us confidence that all relevant challenges are addressed. To demonstrate how the model can be used to compare different hybrid planners, using the model, Chapter 6 compares a hybrid planning instantiation proposed by another researcher [79] with the hybrid planning instantiation proposed in this thesis (cf. Table 6.4). Users can use these examples as a handbook to apply the formal model for analyzing and comparing hybrid planning instantiations.

A Formal Analysis of the Performance of Hybrid Planning

The formal model uses the *a posteriori* (i.e., after executing a hybrid plan) semantics, which was useful in the theoretical formulation of the problem and its solution. However, when applying hybrid planning, one also needs to analyze hybrid planning in an *a priori* (i.e., before executing a hybrid plan) semantics. To this end, the thesis also analyzes the hybrid planning problem in a priori semantics. Specifically, Chapter 5 provides the worst-case bound on the performance of hybrid planning, and in the process, formulates an *a priori* definition for the concepts defined in Chapter 3 in an *a posteriori* semantics. By formal analysis of the hybrid planning problem both in the *a priori* and the *a posteriori* semantics, the thesis aims at providing a broader understanding of the problem, and its potential solutions.

8.1.2 Practical Contributions

In addition to the theoretical contributions, the thesis makes practical contributions, which will help a practitioner to apply hybrid planning in realistic contexts. The practical contributions are as follows:

An Approach to Solve the Hybrid Planning Problem

The thesis proposes an approach to solve a hybrid planning problem to apply it to realistic self-adaptive systems. As listed in Chapter 4, the approach is applicable under certain assumptions/restrictions, but (still) it can be applied to many self-adaptive systems as discussed later. To ensure the soundness of the approach, we represent and analyze it in the context of the formal model.

To solve the planning selection problem (PLNSEL), the thesis proposes learning-based hybrid planning. This approach has both qualitative and quantitative benefits over condition-based hybrid planning. In terms of the qualitative benefits, compared to the condition-based approach, the learning-based approach: (a) does not rely on predefined conditions to choose among reactive planners, (b) automatically maps problems to an appropriate reactive planner using a machine-learning classifier, and (c) can be fully/partially automated. In terms of quantitative benefits, the learning-based approach is shown to be more effective and less risky compared to the condition-based approach (cf. Chapter 6).

Using probabilistic model checking to label the planning problem is fundamental to the proposed learning-based approach (cf. Chapter 4). Model checking helps label training problems by evaluating plan combinations under probabilistic uncertainty, by considering all possible execution paths weighted by their probabilities. Moreover, existing probabilistic model checkers ease adoption, automation, and reuse of the learning-based approach by software engineers.

Evaluation of the Thesis Claims Using Realistic Systems

The thesis uses two realistic systems to evaluate its claims about the effectiveness, generality, and flexibility of hybrid planning. These systems are: (a) a self-adaptive cloud-based load balancing system that has become a de facto benchmark for researchers in the self-adaptive community [28, 53, 85, 94, 97, 107], and (b) a team of UAVs as used by other researchers [86]. As an implementation of these systems, we used well-accepted exemplars – SWIM [87] for the cloud-based systems, and DART [88] for a simulated team of UAVs.

There are various benefits of using these systems. First, these systems/domains are widely used in the self-adaptive community, therefore, the performance of hybrid planning can be compared with other planning approaches proposed by the community. Second, the application of hybrid planning in these systems is an illustration of how the proposed approach can be applied to realistic self-adaptive systems. Third, using these systems gives us confidence about the validation of thesis claims (cf. Chapter 6).

Methods/tools to apply hybrid planning to self-adaptive systems

To facilitate the adoption of hybrid planning, the thesis provides methods and tools as discussed below:

Guidelines to Apply Hybrid Planning: To ease the adoption of hybrid planning, the thesis provides informal guidelines and a quantitative approach to apply hybrid planning. For a practitioner interested in applying hybrid planning, the guidelines and the approach help to answer

questions such as (a) how to identify an appropriate set of reactive and deliberative planners that can handle the trade-off between the timeliness and the quality of planning, (b) whether to use condition-based or learning-based hybrid planning, and (c) how to implement learning-based hybrid planning.

An implementation of the Hybrid Planning Algorithm: We implemented the hybrid planning algorithm (cf. Chapter 5) using an established self-adaptive framework (i.e., Rainbow [27]). Implementing the algorithm in Rainbow has two key benefits: (a) it indicates the generality of the algorithm, and (b) the same implementation can be used (with minor modifications e.g., the set of constituent planners used to instantiate hybrid planning) by researchers/practitioners to apply/test hybrid planning in their context.

8.2 Scoping Assumptions

As discussed in the previous chapters, the thesis makes certain assumptions in order to apply hybrid planning in realistic contexts such as the two systems used for evaluation (cf. Chapter 6). Some of these assumptions are fundamental to the proposed hybrid planning approach, and therefore difficult to relax. This section discusses various assumptions (as summarized in Table 8.1) made by the thesis.[1]

Category	Assumption	Description
Assumptions to Make a Hybrid Planning Problem Tractable	TWO-LEVELS-OF-PLANNING	Hybrid planning uses two levels of planning (i.e., reactive planning followed by deliberative planning)
	FINITE-HORIZON	Planning problems have a finite planning horizon.
	DISCRETE-STATE-VARIABLES	The value of state variables (e.g., time) is discrete.
	DELIBERATIVE-PREFERRED	For any planning problem, a deliberative plan always provides higher expected utility compared to a reactive one.

[1]The assumptions behind the formal model and the validation have already been discussed in Chapter 3 and Chapter 6, respectively.

Assumptions to Address the Planning Coordination Problem	UNIVERSAL-DELIBERATIVE-PLAN	Deliberative planning determines a universal plan (i.e., a policy).
	MARKOVIAN-DOMAIN	The operating domain is assumed to be Markovian.
Assumptions Related to Learning-based Hybrid Planning	USE-OF-UTILITY-FUNCTION	Different conflicting quality attributes for a self-adaptive system can be represented as a multi-dimensional utility function, and the planning goal is to maximize expected utility.
	AVAILABILITY-OF-MODEL-CHECKERS	There are probabilistic model checkers available to deal with different kinds of probabilistic uncertainty.
	NEGLIGIBLE-PLNSEL-DECISION-TIME	The time to solve the planning selection problem (i.e., deciding the reactive planner) is negligible
	AVAILABILITY-OF-TRAINING-PROBLEMS	A comprehensive set of sample planning problems is available to train a classifier.
	IDENTIFIABLE-FEATURES	One can identify the set of features that can help to map a problem to a reactive planner.

	INDUCTIVE-BIAS	For two planning problems having a similar set of features, an effective combination of reactive and deliberative planning for one problem will also work for the other problem.
Other Assumptions	ONE-DELIBERATIVE-APPROACH-ONLY	The thesis claims are subject to using a single deliberative planner to instantiate hybrid planning.
	PLANNING-PROBLEM-REPRESENTATION	An adaptation situation can be represented as a planning problem representation that realistically captures the current state of a system and its future evolution.
	IGNORED-PLANNING-RESOURCE-CONSUMPTION	The likely resources to be consumed by planning are not considered when choosing a planner to solve a planning problem.
	DESIGNERS-HAVE-FAMILIARITY-WITH-AI	The person applying hybrid planning has a broad understanding of automated planning and machine learning.

	USING-EXISTING-PLANNERS-IS-COST-EFFECTIVE	The cost to instantiate hybrid planning (using off-the-shelf planners) is lower compared to developing a hand-crafted planning solution that can balance timeliness and quality of planning.

Table 8.1: Summary of Assumptions.

8.2.1 Assumptions to Make a Hybrid Planning Problem Tractable

The hybrid planning problem in its general form is intractable to solve, as suggested by the formal model describing the problem (cf. Chapter 3). To make the problem tractable, the thesis makes the following assumptions to scope the problem.

- **TWO-LEVELS-OF-PLANNING**: Hybrid planning uses two levels of planning (i.e., reactive planning followed by deliberative planning) to solve a planning problem such that one level is provided by a reactive planner chosen from a set of reactive planners that determine plans in a negligible time, and the other level is provided by the deliberative planner used to instantiate hybrid planning. Reactive planners with a non-negligible planning time are not considered as it would increase the level of planning (from two) to three. To explain further, the first level of planning will be done by ρ_{wait} that determines a plan in negligible time (i.e., always suggests to wait), the second level (of planning) will be done by the reactive planner (say, ρ_r) that determine a plan in a non-negligible time (but quickly compared to a deliberative planner), and the third level is done by the deliberative planner (say, ρ_d).

  There is a potential to relax assumption TWO-LEVELS-OF-PLANNING in the context of learning-based hybrid planning. Theoretically, when using learning-based hybrid planning, probabilistic model checking (to label training problems) can evaluate a combination of plans determined by multiple planners. Explaining model checking for the three levels of planning (caused by a non-negligible reactive planning time) as discussed earlier, suppose for a planning problem (say, ξ), ρ_r and ρ_d determine plans in time t_r and t_d, respectively such that $t_r < t_d$. To evaluate the performance of the combination of ρ_{wait}, ρ_r and ρ_d for ξ, probabilistic model checking can calculate the expected utility for the combination of their plans such that until time t_r no action is considered, between t_r and t_d the plan determined by ρ_r is considered, and from t_d onwards, the plan determined by ρ_d is considered. When all the potential combinations of multiple planners are evaluated for problem ξ, the best-performing combination can be identified for ξ, and label it accordingly; this process can be repeated for all the training problems. As a result, we have a set of labeled training problems, which can be used to train a classifier that solves PLNSEL. As detailed later, it is worth investigating (in the future) how learning-based hybrid planning performs with

multiple levels of planning.

Even with assumption TWO-LEVELS-OF-PLANNING, using specific instantiations of reactive and deliberative planning in different domains, researchers from the self-adaptive community have demonstrated the potential of hybrid planning [43, 94, 110, 112]. However, the existing work is limited to condition-based hybrid planning, i.e., invoking reactive planning only on faults (i.e., for self-healing [10]). In contrast, in this thesis, we extend the idea of hybrid planning to learning-based hybrid planning that not only overcomes the shortcomings of the condition-based approach but also supports other self-* properties such as self-optimization. Moreover, we consider different kinds of instantiation (different reactive-deliberative) combinations, thereby broadening the effectiveness/generality of our approach.

- **FINITE-HORIZON**: The thesis assumes that a planning problem has a finite planning horizon. An infinite horizon will lead to infinite nodes in a reachability graph because time is a state variable according to the formal model, and infinite nodes will lead to an infinite reachability graph (i.e., intractable problem) (cf. Chapter 4). Although condition-based hybrid planning can theoretically support planning problems (e.g., represented as MDP) with infinite horizon, but the learning-based approach requires the planning problem to have a finite horizon due to the use of probabilistic model checking, as detailed in Chapter 4.

  Assumption FINITE-HORIZON restricts the planning horizon to a finite value. For realistic self-adaptive systems such as cloud-based systems, planning for an infinite horizon is not (typically) recommended, since as the planning horizon increases, the planning time increases exponentially, while the quality of planning decreases due to decrease in accuracy of predictions (e.g., request arrival rate). Researchers from the self-adaptive community have demonstrated that planning even with a finite horizon is effective [39, 85, 108].

- **DISCRETE-STATE-VARIABLES**: The value of state variables (e.g., time) is discrete. Otherwise, a reachability graph would have infinite nodes (cf, Chapter 4). Even with this assumption, our approach can be applied to a variety of realistic systems since many of the commonly used planning algorithms/heuristics (e.g., classical planning, MDP/POMDP planning, Reinforcement learning) assume the state space to be discrete [44, 71].

- **DELIBERATIVE-PREFERRED**: For any planning problem, a deliberative plan always provides higher expected utility compared to a reactive one. This implies that whenever a deliberative plan is ready for a planning problem, it is preferred over the plans determined by reactive planners. This assumption ensures that there can never be a path in a reachability graph that has deliberative planning followed by reactive planning, and thereby restricts the number of paths in a reachability graph. This is a realistic assumption since, as discussed in Chapter 7, reactive planning either ignores parts of the operating domain state-space or does not optimize a plan (e.g., anytime planning); this is likely to result in lower-quality plans compared to ones determined by deliberative planning.

8.2.2 Assumptions to Address the Planning Coordination Problem

When using hybrid planning, a key to balancing the timeliness and quality of planning is to have a smooth transition from a reactive plan to a deliberative plan. For a seamless transition from a reactive plan to a deliberative plan, as suggested by the formal model, both the *timing* and the *preemption* condition need to be satisfied. However, this is challenging for two reasons: (a) uncertainty about deliberative planning time makes it difficult to predict when the deliberative plan will be ready to take over, and (b) uncertainty in the system's environment makes it difficult to predict the expected system state after executing the reactive plan. The dissertation makes the following fundamental assumptions to address these challenges:

- **UNIVERSAL-DELIBERATIVE-PLAN:** The thesis assumes that the deliberative planner determines a universal plan (i.e., a policy). Once the deliberative plan is ready, it can take over plan execution from the reactive plan because any state resulting from executing the reactive plan will be found in the deliberative plan.[2]
- **MARKOVIAN-DOMAIN:** The operating domain is assumed to be *Markovian*: the state after a transition depends only on the current state — not on the sequence of states that preceded it [78]. This implies that once deliberative planning solves the planning problem, the resulting plan would suggest an optimal action for each state reachable from the initial state. Therefore, after a transition happens from a reactive plan to a deliberative plan, it ensures an optimal execution thereafter.

Even with assumptions UNIVERSAL-DELIBERATIVE-PLAN and MARKOVIAN-DOMAIN, hybrid planning is applicable to different domains, since potential choices (such as MDP and POMDP planning) for deliberative planning determine policy-structured plans (to deal with uncertainty) and are applicable to Markovian domains. As discussed in Chapter 7, these approaches can be used both for reactive and deliberative planning. Moreover, by specifying transition probability as 1, deterministic transitions can also be captured by MDP/POMDP planning, as was done for the cloud-based system used for the thesis evaluation. Furthermore, as mentioned in Chapter 7, even though MDP/POMDP planning is used only for a *Markovian* domain, they can also be used for a *non-Markovian* domain by representing it as a *Markovian* domain using additional state variables to capture history leading to a much larger state space, and therefore negatively impacting the timeliness of planning.

8.2.3 Assumptions Related to Learning-based Hybrid Planning

One of the key contributions of this thesis is the learning-based approach to solve the planning selection problem (PLNSEL). Although the approach is broadly applicable as discussed earlier, there are certain assumptions that are fundamental to the approach.

- The following assumptions are due to using a probabilistic model checker to evaluate a combination of planners.
  - **USE-OF-UTILITY-FUNCTION:** This thesis assumes that different conflicting quality attributes for a self-adaptive system can be represented as a multi-dimensional utility

[2]However, in reality, there is still a possibility that transition between the plans might fail due to violating the timing or preemption condition, thus affecting the quality of adaptation (cf. Chapter 6).

function such as Equations 1.1 and 6.3, and that the planning goal is to maximize expected utility. In addition, the thesis assumes that planning problems have either no uncertainty (i.e., deterministic) or probabilistic uncertainty. The combination of these two assumptions enables the use of a probabilistic model checker to evaluate a combination of reactive and deliberative planner by calculating the expected utility (cf. Chapter 7); expected utility cannot be calculated for non-deterministic uncertainty. The representation of a planning goal using a multi-dimensional utility function is general enough to also capture goals having an explicit state. The assumption about probabilistic uncertainty is broad enough to represent various domains [44, 71, 103].

  - **AVAILABILITY-OF-MODEL-CHECKERS:** The other key assumption is that there are probabilistic model checkers available to deal with different kinds of probabilistic uncertainty. The thesis already discusses and demonstrates the use of probabilistic model checker PRISM that can handle MDP-based models (i.e., uncertainty in action outcomes) [69]. For domains that also have uncertainty in the underlying state one needs to use a model checker that supports partially observable MDPs. To this end, an extended version of PRISM that deals with a POMDP model can be explored [91]; however, the tool is still in early stages.

- **NEGLIGIBLE-PLNSEL-DECISION-TIME:** This thesis assumes that the time to solve the planning selection problem (i.e., deciding the reactive planner) is negligible, as elaborated in Chapter 4. A delay in deciding the reactive planner (e.g., by classifying a planning problem) will delay the response from the planner, thereby decreasing the effectiveness of hybrid planning. Assumption NEGLIGIBLE-PLNSEL-DECISION-TIME limits learning-based hybrid planning to the kinds of machine learning algorithms that can classify in negligible time. For instance, supervised learning approaches such as logistic regression and support vector machines (SVMs) can classify a planning problem quickly because they construct a mathematical formula (using training problems) to map the problem to its label [101]; predicting using such formulas is nearly instantaneous. In contrast, there are lazy learning algorithms such as k-nearest neighbors that can be slow in classifying a problem, therefore, might not be suitable for our approach [30].

- As explained in Chapter 4, learning-based hybrid planning uses supervised learning to train a classifier that solves the planning selection problem (i.e., map a problem to an appropriate reactive planner). For an effective training of a classifier, the thesis makes the following assumptions [81]:

  - **AVAILABILITY-OF-TRAINING-PROBLEMS:** A comprehensive set of sample planning problems is available to train a classifier. This set should be a good representation of the space of planning problems. Fortunately, modern-day systems (e.g., two systems used for validation) produce large amounts of data that are available to train a classifier.

  - **IDENTIFIABLE-FEATURES:** One can identify a set of features that can help to map a problem to a reactive planner. To identify such a feature set, the thesis proposes using two complementary sets of features: ones representing the current state of the system, and ones describing how the system will evolve in the future (cf., Chapter 4). These

features reasonably represent a planning problem by capturing the initial state and future transitions of the problem. However, one can also investigate techniques such as principal component analysis (PCA) to identify the optimal set of features [2].

- **INDUCTIVE-BIAS:** This thesis assumes that for two planning problems having a similar set of features, an effective combination of reactive and deliberative planning for one problem will also work for the other problems; this is a fundamental assumption to apply the learning-based approach.

.

8.2.4 Other Assumptions

Here are some other assumptions made by the thesis.

- **ONE-DELIBERATIVE-APPROACH-ONLY:** The thesis claims are subject to using a single deliberative planner to instantiate hybrid planning. We restrict the thesis claims to this assumption since our validation is done using single deliberative planning. However, as discussed in Chapter 4, the proposed learning-based hybrid planning can naturally be extended to support multiple deliberative planners. In short, given a finite set of reactive planners and a finite set of deliberative planners, the proposed use of probabilistic model checking can help in deciding the best pair (consisting of a reactive and a deliberative planner) for a problem, and label it accordingly.
- **PLANNING-PROBLEM-REPRESENTATION:** The thesis assumes that an adaptation situation can be represented as a planning problem (cf. Definition 5.2.8), particularly, for deliberative planning.[3] This is a fundamental assumption of our approach; however, even with this assumption, the scope of hybrid planning is broad enough to be applied to many realistic contexts. For example, there are a variety of domains (e.g., cloud-based systems, robotics, disaster management systems) explored by self-adaptive researchers that represent an adaptation situation as a planning problem [39, 86, 102, 103].

  For effective hybrid planning, it is critical to have a planning problem representation that realistically captures the current state of a system and its future evolution. For instance, while formulating a planning problem for the cloud-based system used for the thesis evaluation, the expected values of request arrival rates used in problem formulation, and their transition probabilities should be close to what the system would experience in *a posteriori* semantics. Specifically, for deliberative planning, if the planning problem does not capture reality: (a) the plan is likely to provide low utility on execution, and (b) during execution, the plan (i.e., policy) will often fail (i.e., the current state of a system will not be found in the policy) leading to more use of sub-optimal reactive planning, which may result in a possible decrease in the overall quality of planning.
- **IGNORED-PLANNING-RESOURCE-CONSUMPTION:** The likely resources to be consumed by planning are not considered when choosing a planner to solve a planning problem. Specifically, the thesis ignores the fact that deliberative planning is likely to consume more

[3]Reactive planning (e.g., rule-based) does not necessarily require representing a situation as a planning problem.

resources (e.g., CPU cycles, energy) compared to reactive planning since the former has to do more computation to determine a plan. If resources are limited, it might not always be a good idea to invoke deliberative planning. As Kahneman and others have pointed out, humans tend to avoid tasks that require deliberative thinking [56] to conserve energy. Consideration of planning resource consumption to decide when invoking deliberative planning is "good enough" is an open problem both for humans and systems.

- **DESIGNERS-HAVE-FAMILIARITY-WITH-AI:** As mentioned in Chapter 6.3.1, the thesis assumes that the person applying hybrid planning has a broad understanding of various planning approaches (e.g., classical planning, search heuristics, MDP/POMDP planning), and different machine learning algorithms (e.g., supervised and unsupervised learning), models (e.g., decision trees, support vector machines) and techniques (e.g., cross-validation). This might not be an unrealistic assumption considering the increased awareness of these technologies among practitioners, particularly in the self-adaptive community, in recent years.
- **USING-EXISTING-PLANNERS-IS-COST-EFFECTIVE:** The thesis assumes that the cost to instantiate hybrid planning (using off-the-shelf planners) is lower compared to developing a hand-crafted planning solution that can balance timeliness and quality of planning. Burke et al. suggested that, in general, the cost of developing new search/optimizing algorithms is higher than using a combination of off-the-shelf algorithms [20]. In the context of this thesis, the cost of developing a customized solution is assumed to be higher because, due to the complexity of the task, developing the solution can be more time-consuming, and would require the skills of AI experts that many software engineers may not have. Moreover, because these solutions are tailored to different domains, successes are rarely directly transferable to other domains; hence, these approaches are not general, and therefore, the investment in developing a planning solution cannot be utilized again. In contrast, using existing planners is likely to reduce development time and cost since software engineers do not have to be AI experts or master the complexity of developing new algorithms/heuristics. However, a user study needs to be conducted to validate if this assumption holds in reality.

8.3 Future Work

The previous section provided a consolidated list of assumptions made in this thesis. Such a list will help a practitioner to know all the assumptions up front, and thereby evaluate the feasibility of applying hybrid planning in his context. In addition, there is a detailed discussion on how to relax the assumptions that are not fundamental to our approach.This section presents the short-term and long-term research projects (in the context of hybrid planning) that can be explored in the future.

8.3.1 Short Term Projects

This section lists the research projects that can be investigated in the short-term.

Supporting Multiple Deliberative Planners

Currently, the thesis assumes that hybrid planning is instantiated using a single deliberative planner. However, as discussed earlier, when using learning-based hybrid planning, the use of probabilistic model checking can be extended to support multiple deliberative planners. Using multiple deliberative planners is likely to improve the effectiveness of hybrid planning because it provides more choices for deliberative planners (in combination with reactive planners) to solve a planning problem. In addition, using multiple reactive and deliberative planners to instantiate hybrid planning would strengthen the case for *flexibility* for hybrid planning.

Application of Active Learning

Currently, as discussed in Chapter 4, the offline phase of learning-based hybrid planning is used to train a classifier that learns the mapping function between the set of reactive planners (used to instantiate hybrid planning), and the set of planning problems that a system expects to observe at run time. However, over time, a system might face new planning problems that are not represented in the set of sample problems. This might happen due to various reasons, such as a change in workload patterns in the context of a cloud-based system. Therefore, the system needs to identify such new planning problems to actively improve the mapping function. To this end, one might apply a machine learning technique known as *active learning* in which a learning algorithm can interactively query the user (or some other information source) to obtain the desired output for a new data point (e.g., a planning problem) [105]. For the new data point, if the classifier's output is different from the desired output, then that data point can be used to further tune the classifier.

In the context of hybrid planning, probabilistic model checking can be used as a source of information to decide which reactive planner should be invoked for a problem observed at run time, and if the prediction of the classifier is different from the decision using model checking then the problem can be treated as a new data point, and therefore used to train the classifier.

8.3.2 Long Term Projects

This section lists some potential long-term research projects in the context of hybrid planning.

Support for Multiple levels of Planning

The thesis assumes that hybrid planning has only two levels of planning. However, as explained earlier, theoretically learning-based hybrid planning can support multiple levels of planning because model checking can evaluate a combination of plans determined by multiple planners for a planning problem. It would be worthwhile to investigate such multi-level hybrid planning since: it (a) has the potential to outperform two-level hybrid planning due to availability of more choices of planner combinations to solve a planning problem, and (b) naturally relaxes the assumption that reactive planning time should be negligible. If multi-level hybrid planning works in practice, it would further broaden the flexibility claim for hybrid planning, and demonstrate the generality of the learning-based approach including the use of model checking to label planning problems.

However, using multi-level hybrid planning has various challenges. First, instantiating hybrid planning for multi-level planning can be overwhelming for a practitioner. Currently, with the two

levels of planning, any planner with negligible planning time is in the set of reactive planners, else it can be a potential candidate for deliberative planning. However, with multi-level planning, one needs to evaluate a larger set of planners with different timeliness-quality profiles to instantiate hybrid planning effectively.

Another challenge with multi-level hybrid planning is that the potential combinations of planners to solve a problem will increase exponentially with the increase in the levels of planning; therefore the number of labels (i.e., classes) will also increase exponentially. This will increase the overall complexity in applying the learning-based approach. For instance, one needs to ensure that no class is underrepresented or overrepresented in the set of training problems; finding such a set can be challenging with a large number of classes.

Hybrid Planning in the Context of Human-in-the-loop

Further ahead, one might explore hybrid planning in the context of human-in-the-loop adaptation by treating the human (e.g., a system administrator) as a reactive planner that is used in combination with some deliberative planner such as an MDP planner. Self-adaptive systems such as a smart-grid often require intervention by system administrators, particularly for emergency situations; an administrator could provide a quick decision based on past troubleshooting experience. However, for domains such as cyber-security, humans might be treated as a deliberative planner that provides a high-quality decision after investigating logs of a system.

To apply hybrid planning in such a context, an interesting challenge will be to build a model of human decision-making that can be used to decide when it is safe for a human to make a decision. To this purpose, one might explore the framework proposed by Eskins et al. [34] to model human behavior as done by Cámara et al. [24] and Lloyd et al. [74].

8.4 Conclusion

This dissertation presents a hybrid planning approach that improves the current state of the art of planning for self-adaptive systems. The approach deals with the fundamental timeliness-quality trade-off by combining multiple off-the-shelf planners. The key idea is to compose planners with different time-quality tradeoffs. When a time-critical adaptation becomes necessary, "fast" (*reactive*) planning determines a quick (but potentially a sub-optimal) plan, while "slow" (*deliberative*) planning computes a better plan that can take over once it is ready. This idea of hybrid planning is akin to human decision making: depending upon factors such as available planning time, humans apply different levels of deliberation while making real-life decisions [56].

Using two realistic systems the dissertation demonstrates that hybrid planning can improve the overall utility of a self-adaptive system by finding a right balance between timeliness and quality. From the software engineering perspective, instead of going through the non-trivial process of developing a new planning algorithm/heuristics, engineers can potentially reduce development time and cost by combining off-the-shelf planning approaches using hybrid planning [20]. As mentioned earlier, the dissertation contributes to both the theory and the practice of hybrid planning in self-adaptive systems:

The contribution to theory is:

- a formal model characterizing the general problem of hybrid planning;
- an illustration of how the formal model can be used as a unifying evaluation framework to compare/analyze instantiations of hybrid planning, and thereby understand their strengths and weaknesses.
- a formal analysis of the performance of the hybrid planning algorithm.

The contributions to practice are:

- a practical approach to applying hybrid planning under certain assumptions/restrictions that nonetheless apply to many self-adaptive systems;
- a demonstration of effectiveness, generality, and flexibility of hybrid planning for self-adaptive systems using the proposed solution approach;
- methods/tools to apply hybrid planning to self-adaptive systems, including
  - evaluation of hybrid planning using two systems (i.e., the cloud-based system and the UAV team) to illustrate how the proposed approach can be applied to realistic self-adaptive systems,
  - an implementation of the hybrid planning algorithm (cf. Chapter 5) using a widely accepted MAPE-based self-adaptive framework (i.e., Rainbow [27]) to ease an adoption of hybrid planning among software engineers,
  - informal guidelines and a quantitative approach to help engineers to select an appropriate set of planners to instantiate hybrid planning for a given domain.

This thesis formalizes the sophisticated problem of hybrid planning and decomposes it into four computational subproblems. There are several applications of this formal model (cf. Chapter 3). First, it helps to understand the problem of hybrid planning in its general form. Second, it helps to analyze whether a hybrid planning instantiation is valid. Third, this model serves as a unifying evaluation framework for different such solutions. In addition, to demonstrate the applicability of the formal model, the thesis analyzes and compares two hybrid planing instantiations.

In the past, the promising idea of hybrid planning has been studied from algorithmic [6, 7, 79, 113] perspectives. This thesis improves engineering aspects of hybrid planning by providing: (i) a learning-based approach to the planning selection problem (PLNSEL), which aims to replace domain-specific hard-coded conditions for invoking reactive planning (cf. Chapter 4); (ii) the hybrid planning algorithm, its formal analysis, and implementation in the Rainbow framework (cf. Chapter 5); (iii) informal guidelines and a quantitative approach to help engineers to select an appropriate set of planners to instantiate hybrid planning (cf. Chapter 7).

One of the barriers to adopting learning-based hybrid planning is the difficulty of obtaining a labeled set of training planning problems. We overcome this by using probabilistic model checking to label the training problems. Moreover, this enables the steps (including model checking) of the learning-based approach to be automated. Our evaluation indicates the generality of learning-based hybrid planning since the evaluation uses: (a) two realistic systems from different domains, and (b) different combinations of constituent planners to instantiate learning-based hybrid planning.

This thesis uses a cloud-based self-adaptive system and a team of unmanned aerial vehicles to evaluate *effectiveness*, *generality*, and *flexibility* of hybrid planning (cf. Chapter 6). Using these relistic systems gives us confidence about the validation of thesis claims. Although our hybrid

planning approach appears to be effective, the approach has a number of assumptions as discussed in Chapter 8; the chapter also highlights potential directions of future research in hybrid planning.

To conclude, this thesis sets the stage for the application of hybrid planning in realistic self-adaptive systems. The thesis demonstrates that hybrid planning is a promising way to improve self-adaptation, thus increasing the potential for industrial adoption. However, the complexity of hybrid planning creates a possibility for many diverse solutions to solve the problem. Therefore, further research is needed to provide efficient, usable, and general approaches to combine multiple planners for self-adaptation.

Appendix A

Formalization of Timing and Preemption Conditions for the Cloud-based System

For the cloud-based system, as discussed in Chapter 6, in practice the timing and preemption conditions might not be satisfied; then, a transition from a deterministic plan to an MDP policy would fail. Therefore, this instantiation makes assumptions about the timing and the preemption condition that must hold to guarantee a smooth transition from a deterministic plan to an MDP policy. In other words, these assumptions make this instantiation valid. To highlight these assumptions and make them checkable in practice, we formalize the timing and the preemption condition for this instantiation in the context of the cloud-based system.

Definition A.0.1 (Time)**.** The *time* T is an infinite set containing all possible discretized time-stamps.

Definition A.0.2 (Universal state space)**.** The *universal state space* S_u is an infinite set containing all the possible states.

The universal state space captures states corresponding to all possible request arrival rates. Therefore, deterministic and MDP planning state spaces (i.e., S_{det} and S_{mdp} respectively) are subset of the universal state space: $S_{det} \subseteq S_u$ and $S_{mdp} \subseteq S_u$.

Definition A.0.3 (Environment realization)**.** The *environment realization* is a function $Env : T \rightarrow \mathbb{R}_{\geq 0}$ that returns the actual (i.e., ground truth) request arrival rate for a given time-stamp $t \in T$.

Definition A.0.4 (Predictor)**.** The *predictor* is a function $P : T \rightarrow \mathbb{P}\, \mathbb{R}_{\geq 0}$ that returns expected request arrival rates for a given time stamp $t \in T$.

Definition A.0.5 (State similarity classifier)**.** The *state similarity classifier* is a function $\mathcal{S} : S_u \times S_u \rightarrow \{true, false\}$, which returns $true$ if the two input states have same value for all the state variables that represent the system's state.

For system state similarity, state variables such as number of servers, dimmer value, and traffic distribution among servers are compared because these variables represent the system state. Request arrival rate is not compared for two states since it represents environment state.

Definition A.0.6 (Timing condition)**.** For an MDP planning problem ξ_{mdp} with planning horizon h, time discretization step t_d, and planning time t_{mdp}, the timing condition is satisfied if $h \geq t_d + t_{mdp}$.

In the instantiation, a planning problem is discretized such that the planning horizon is divided into equal intervals of time t_d. The timing conditions is satisfied if MDP policy is ready *at least* t_d

time before the horizon. If an MDP policy not ready until time h - t_d then the policy can't take over from a deterministic plan since no execution is needed once horizon is reached.

Definition A.0.7 (Preemption condition)**.** For deliberative plan π_d, request arrival rate threshold $E \in \mathbb{R}_{\geq 0}$, and state $s_{curr} \in S_u$ representing the current state of the system and the environment, successful preemption requires the following condition to be satisfied:[1]

$$\exists s_{min} : dom(\pi_d) \cdot \mathcal{S}(s_{curr}, s_{min}) \wedge (|s_{curr}.rae - s_{min}.rae| \leq E) \wedge$$
$$(\forall s : dom(\pi_d) \cdot S(s_{curr}, s) \implies |s_{curr}.rae - s_{min}.rae| \leq |s_{curr}.rae - s.rae|).$$

To ensure that this preemption condition is always satisfied, the combination of the following two conditions must be satisfied:

1. The state space S_{det} considered by deterministic planning is a subset of state space S_{mdp} considered by MDP planning, i.e., $S_{det} \subset S_{mdp}$. For a planning problem, this condition ensures that the MDP policy consists of a state $s \in dom(\pi_d)$ such that all the state variables representing the system's state are same as the current state s_{curr} that resulted from executing the deterministic plan. Formally, $\exists s : dom(\pi_d) \cdot \mathcal{S}(s_{curr}, s)$.
2. Given planning horizon h, state $s^i \in S$ representing the initial state of system, and environment $\forall t \in T \cdot \tau(s^i) \leq t \leq h \implies \min_{s_e \in P(t)} |s_e - Env(t)| \leq E$. This condition ensures that, at any time t within planning horizon h, request arrival rate s_e observed for the current environment (i.e., $Env(t)$) will be with in error bound E. In other words, this condition guarantees that there will be $s \in dom(\pi_d)$ that represents the current request arrival rate.

[1] We denote the immediate request arrival rate of state s as $s.rae$

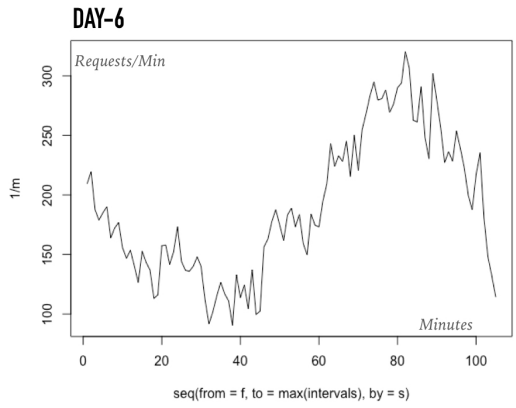
Appendix B

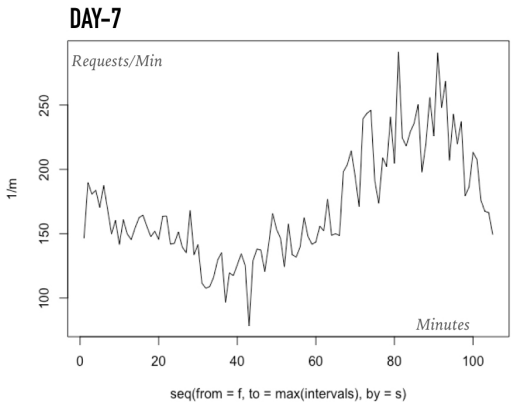
Plots of the FIFA Traces Used for Validation

Total traces were 92 but 5 were incomplete/corrupted. Therefore, we used 87 traces for validation. Each trace is scaled such that

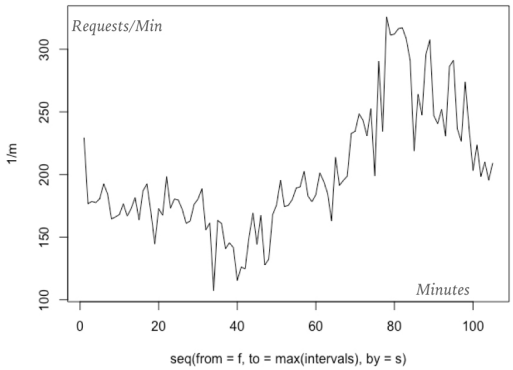
- duration is 105 minutes;
- starting request arrival rate is about 200 requests/min since active servers beginning of a simulation can server 200 requests/min. If workload goes beyond the capacity, the queueing model does not work;
- similarly, the highest workload is about 800, which is 90of the the total capacity (including all available servers with no optional content) of the system.

Below are the visualizations of load patterns for each day.

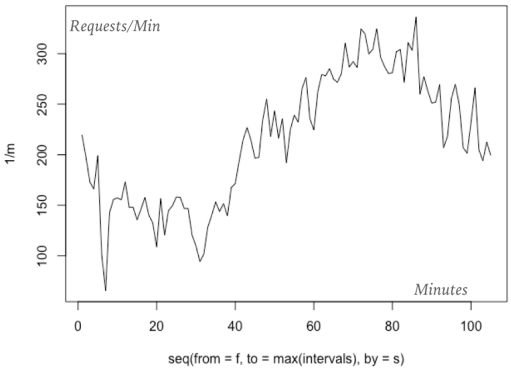




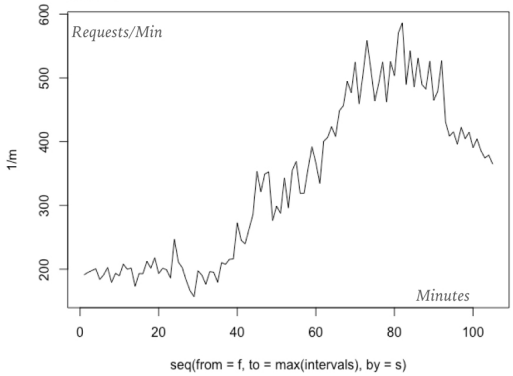
DAY-8



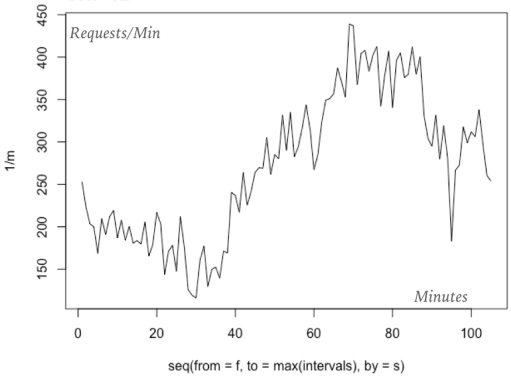
DAY-11



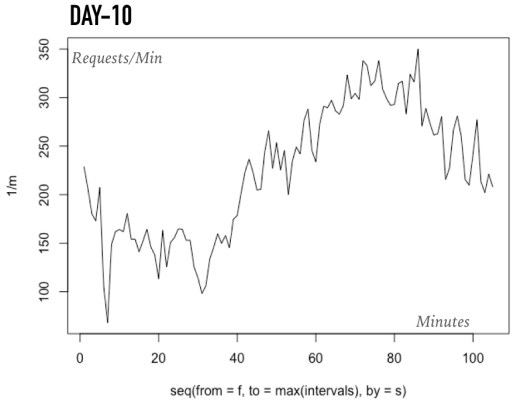
DAY-9



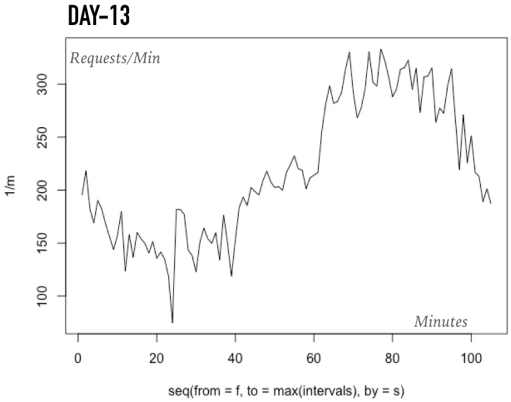
DAY-12



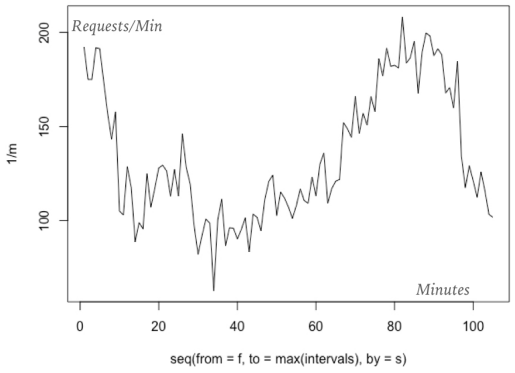
DAY-10



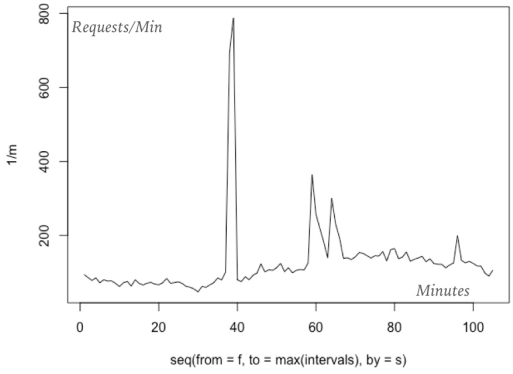
DAY-13



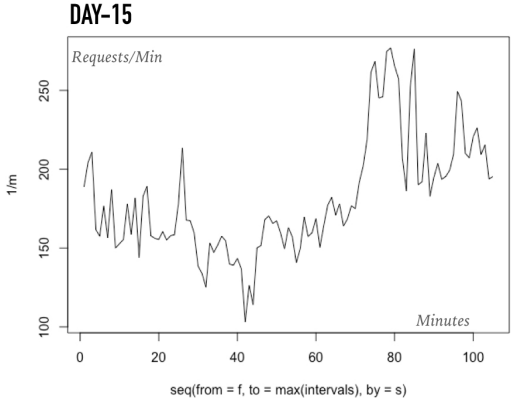
DAY-14



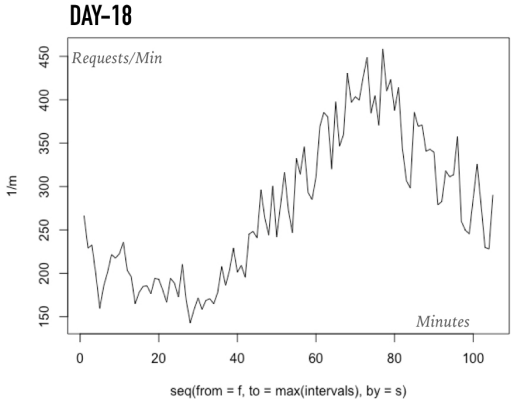
DAY-17



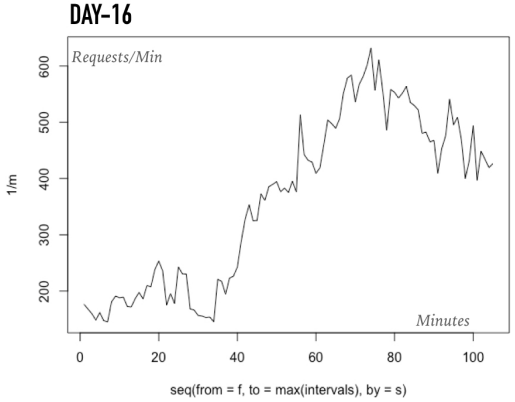
DAY-15



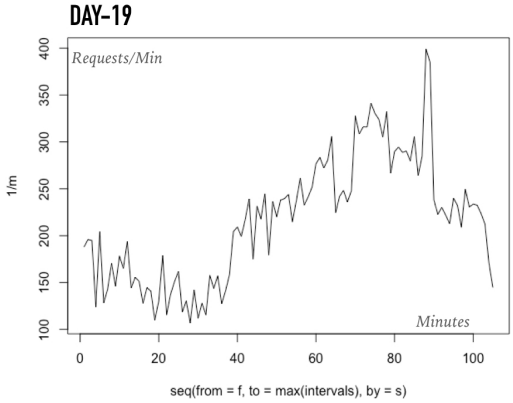
DAY-18



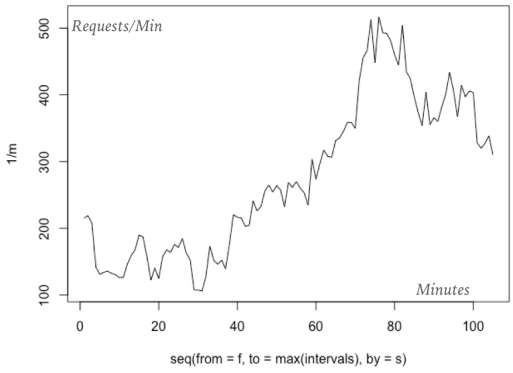
DAY-16



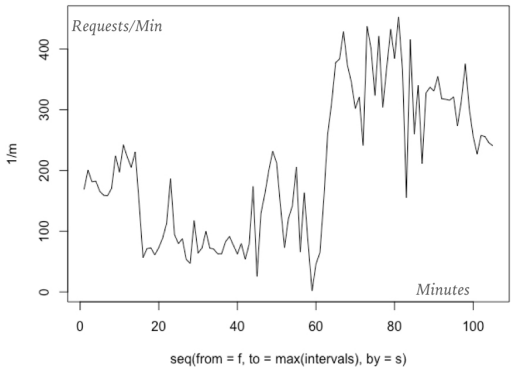
DAY-19



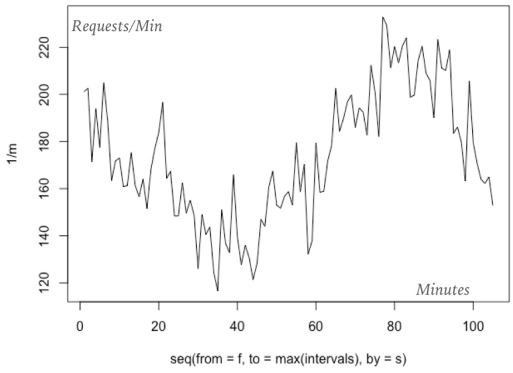
DAY-20



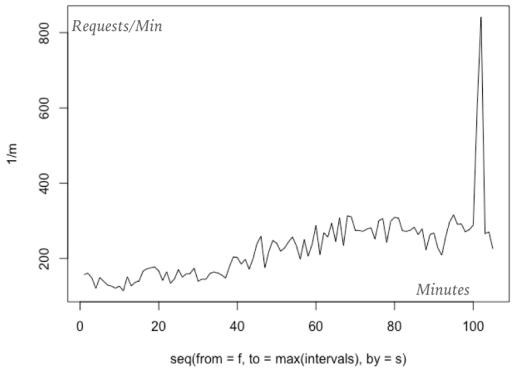
DAY-23



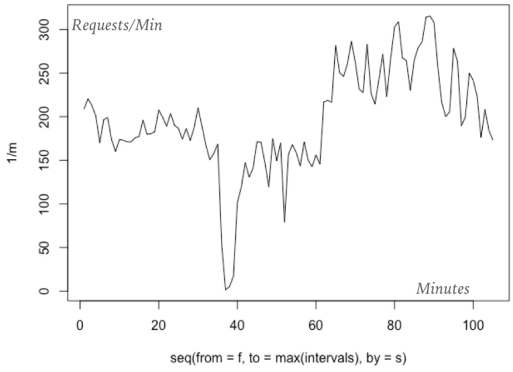
DAY-21



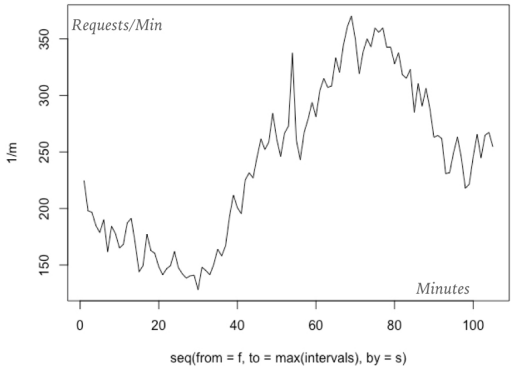
DAY-24



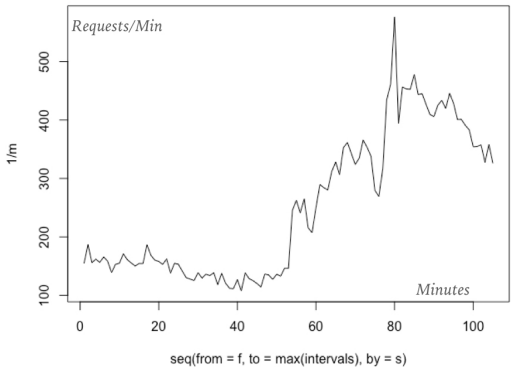
DAY-22



DAY-25



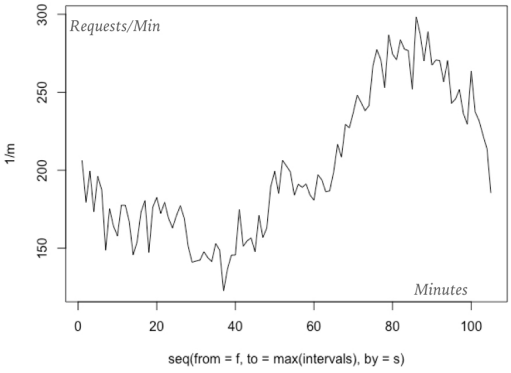
DAY-26



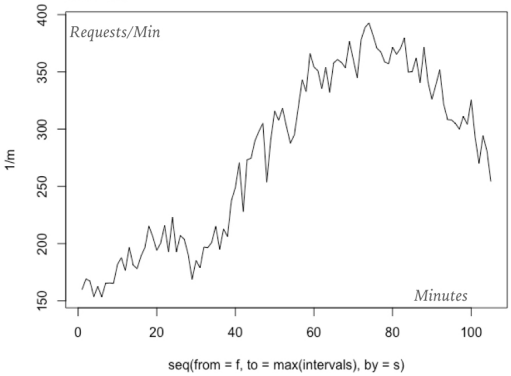
DAY-27

DAY-28

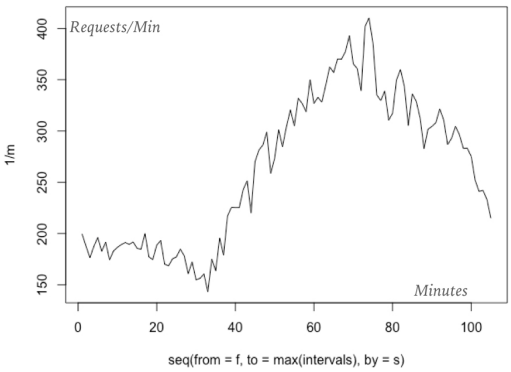
DAY-29



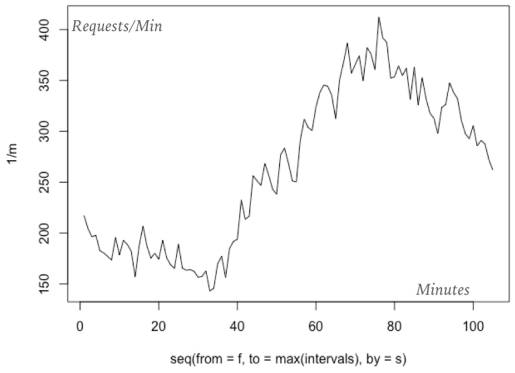
DAY-30



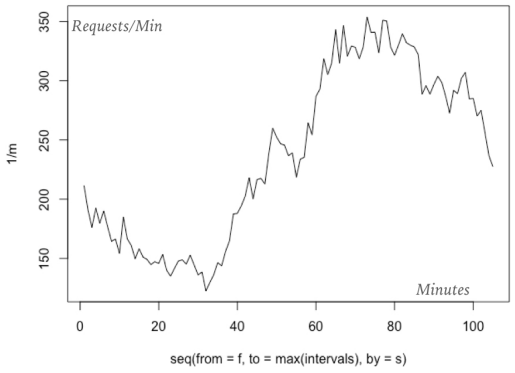
DAY-31



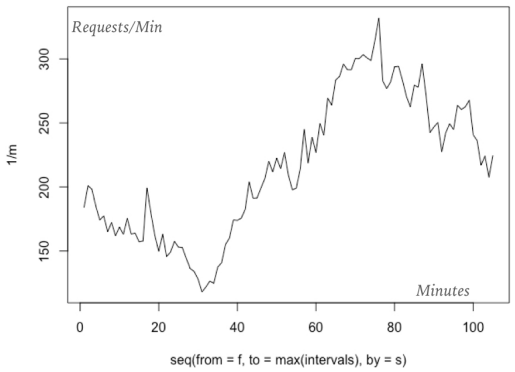
DAY-32



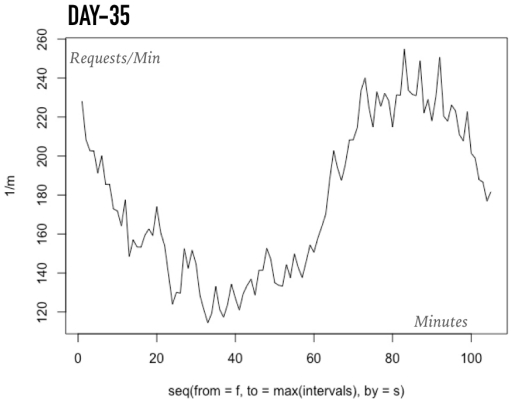
DAY-33



DAY-34

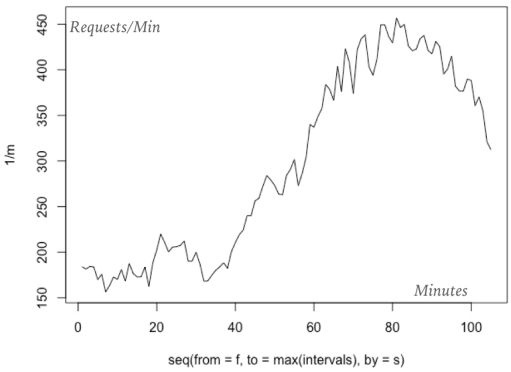


DAY-35

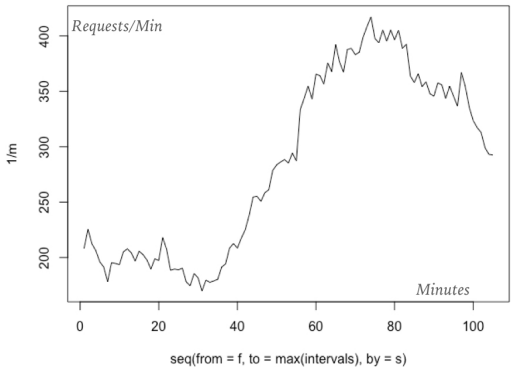


DAY-36

DAY-37

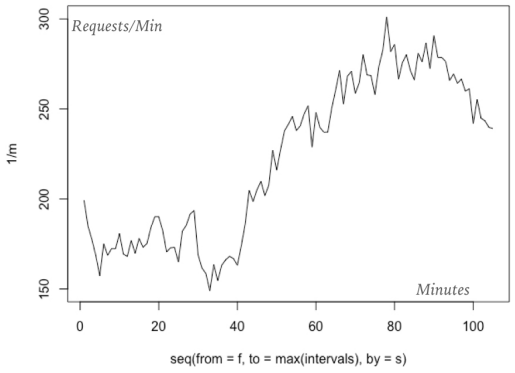


DAY-38

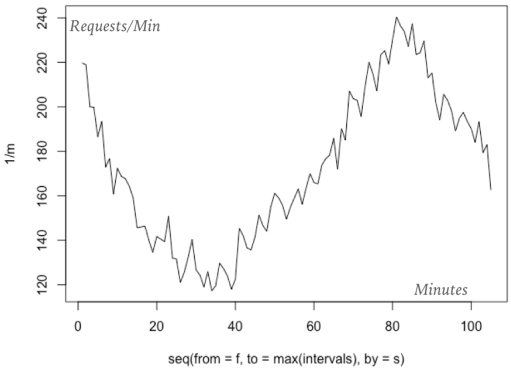


DAY-41

DAY-39

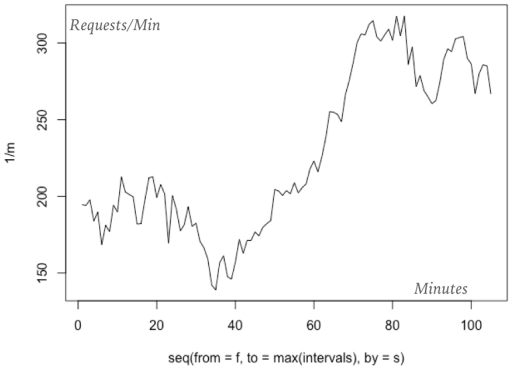


DAY-42



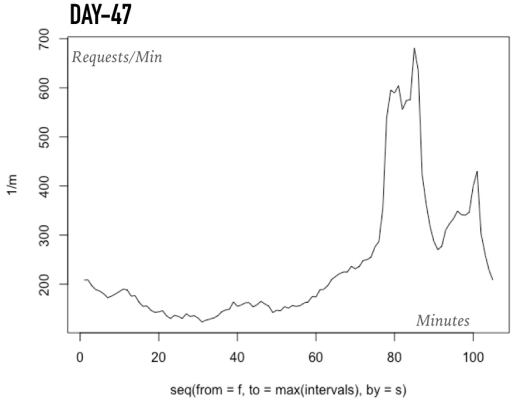
DAY-40

DAY-43



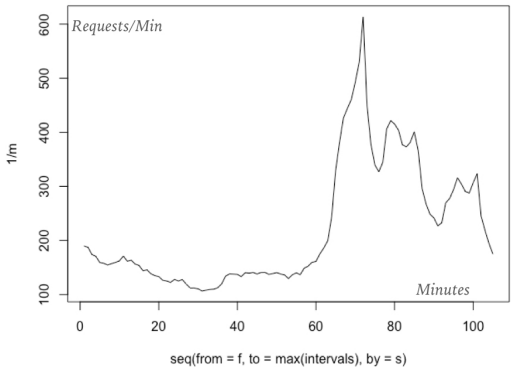
DAY-44

DAY-47

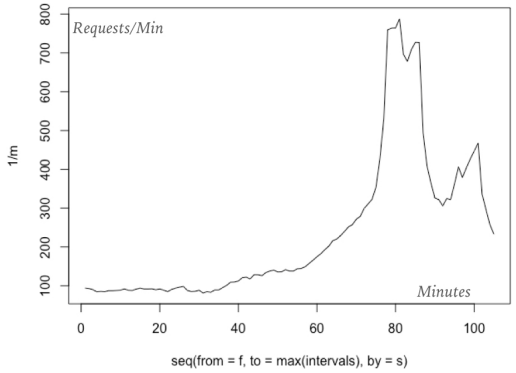


DAY-45

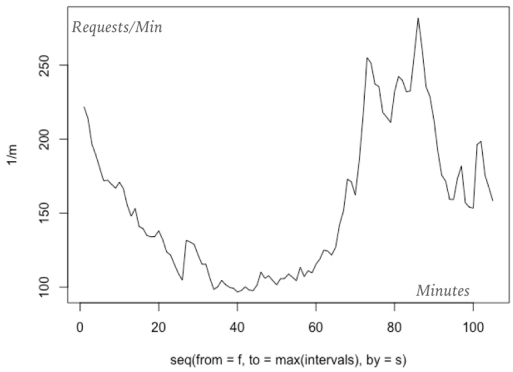
DAY-48



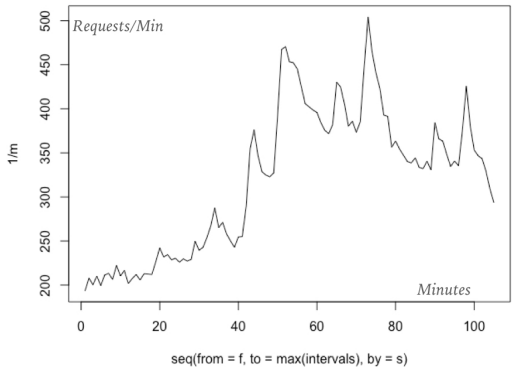
DAY-46



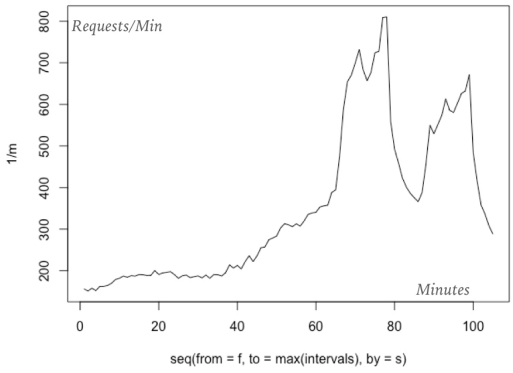
DAY-49



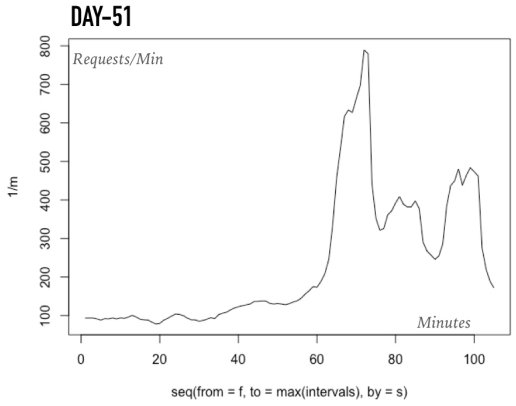
DAY-50



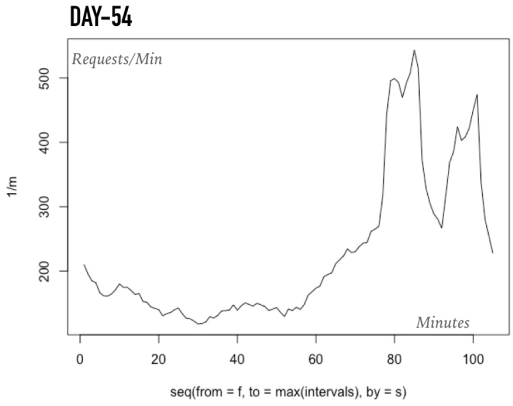
DAY-53



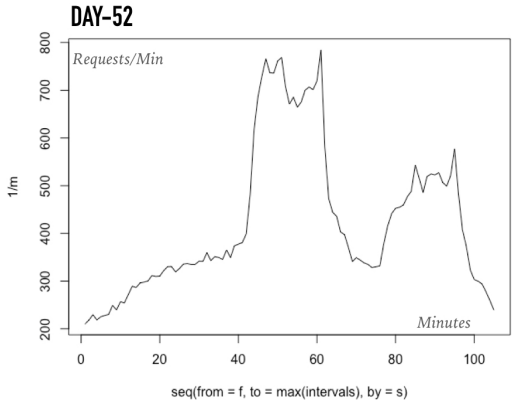
DAY-51



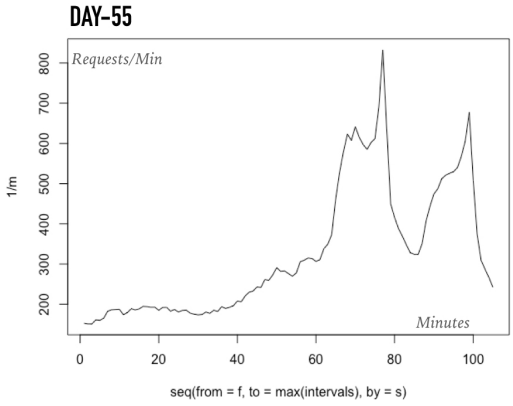
DAY-54



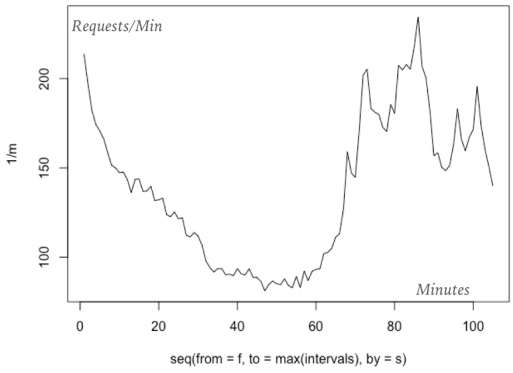
DAY-52



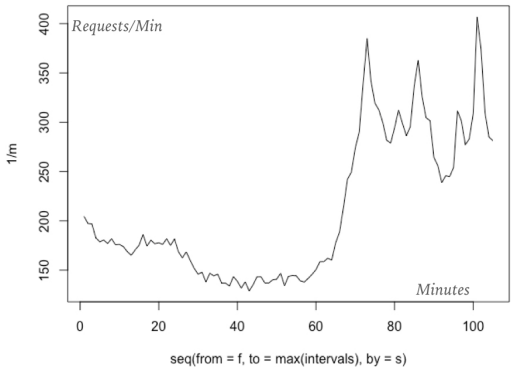
DAY-55



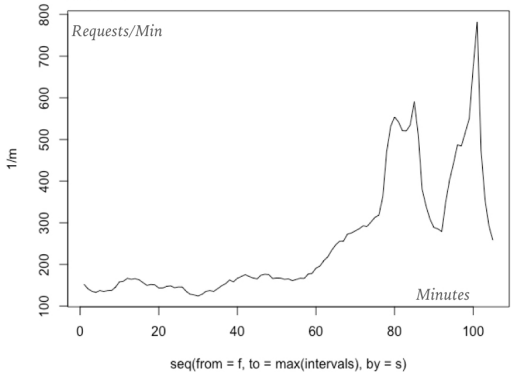
DAY-56



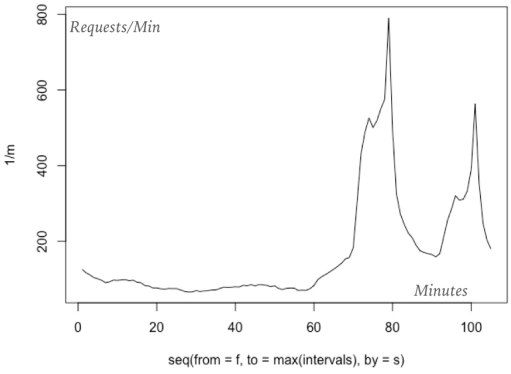
DAY-57



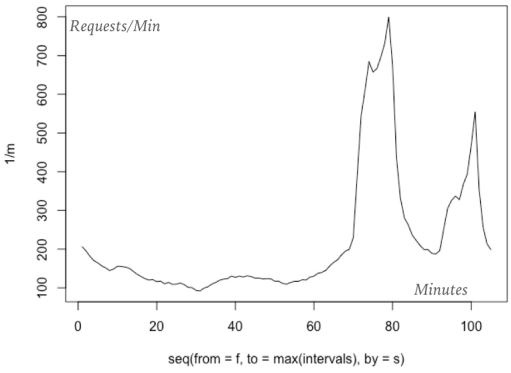
DAY-58



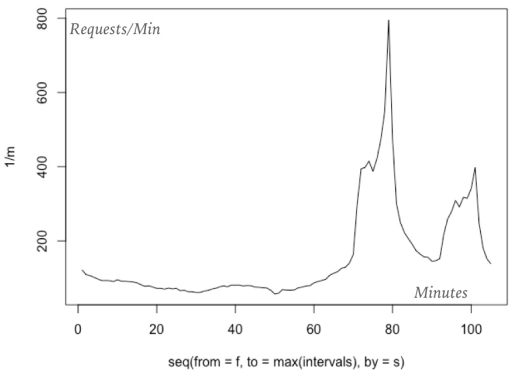
DAY-59



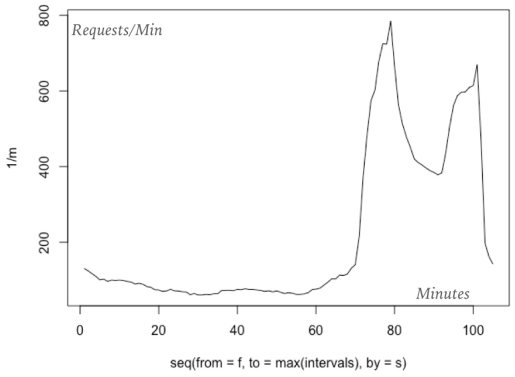
DAY-60



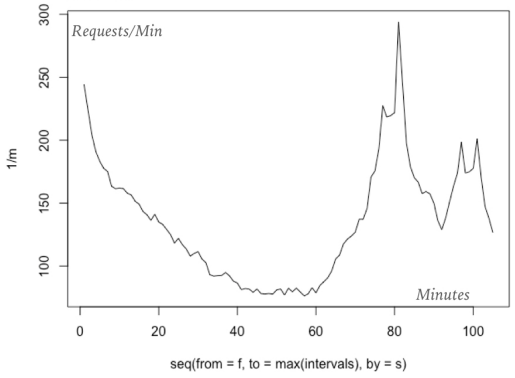
DAY-61



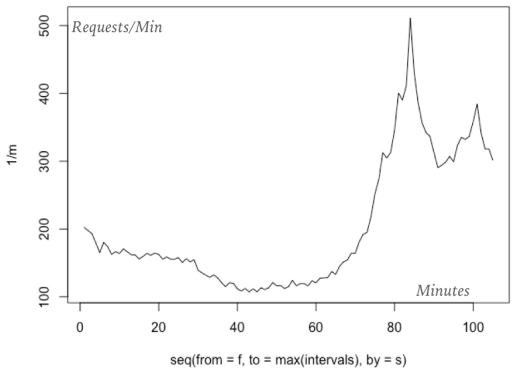
DAY-62



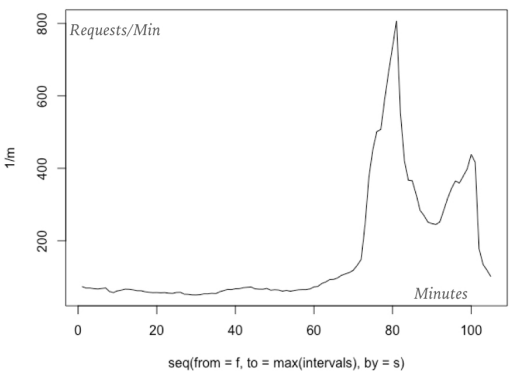
DAY-63



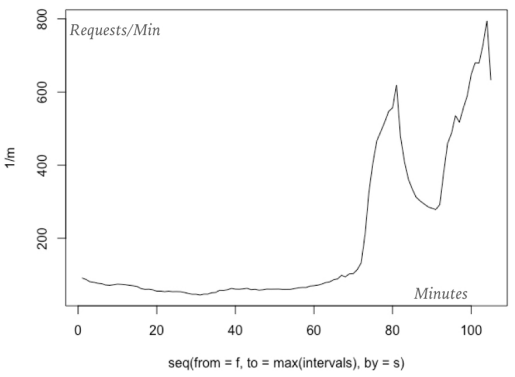
DAY-64



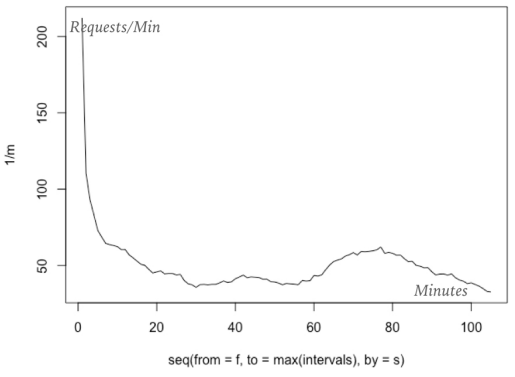
DAY-65



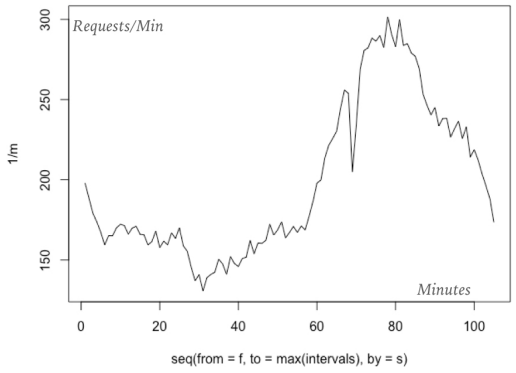
DAY-66



DAY-67

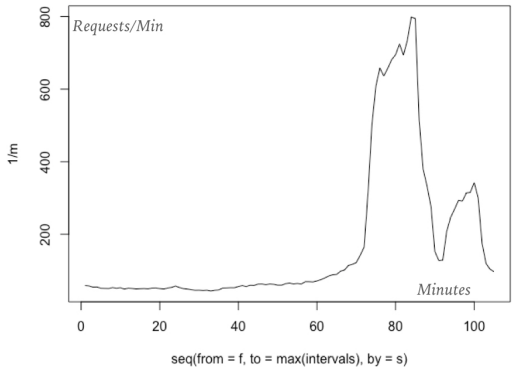


DAY-68

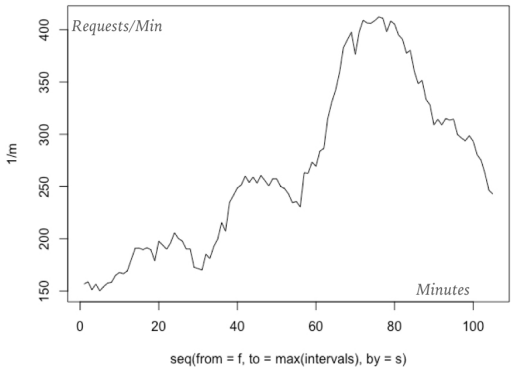


DAY-71

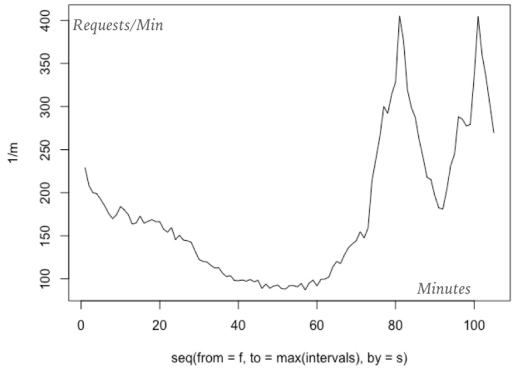
DAY-69



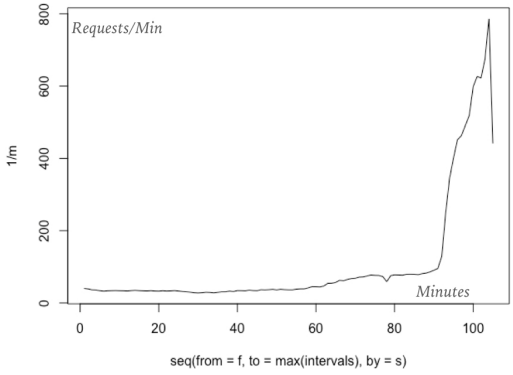
DAY-72



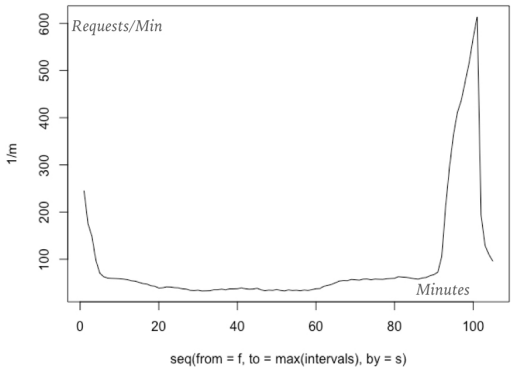
DAY-70



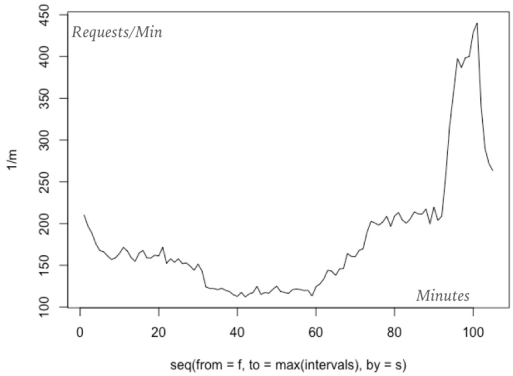
DAY-73



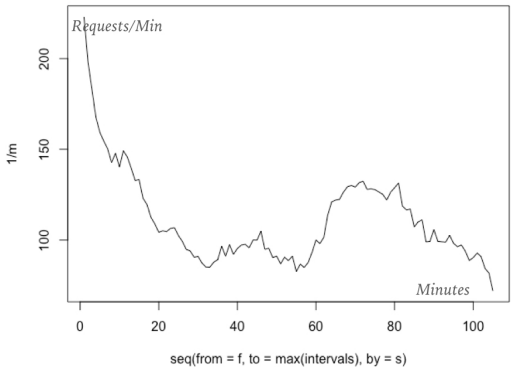
DAY-74



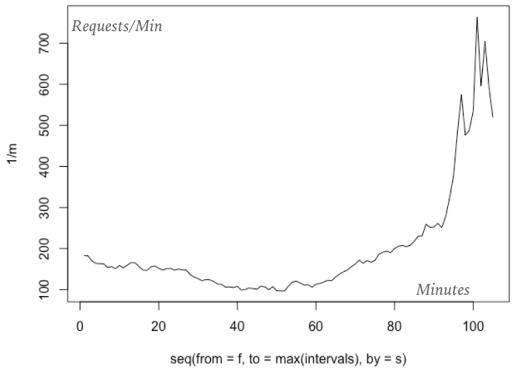
DAY-77



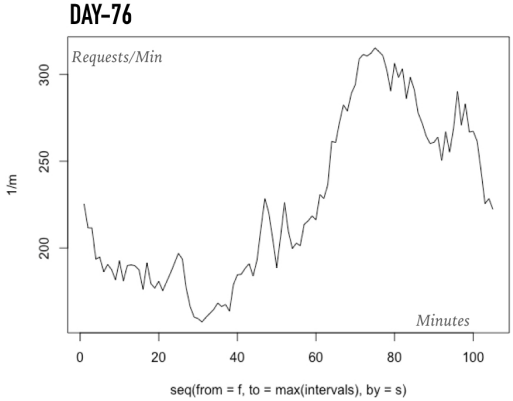
DAY-75



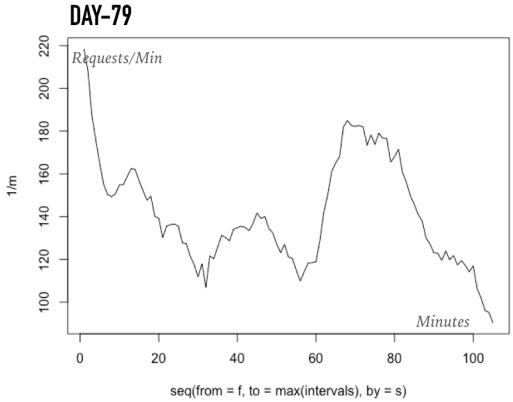
DAY-78



DAY-76

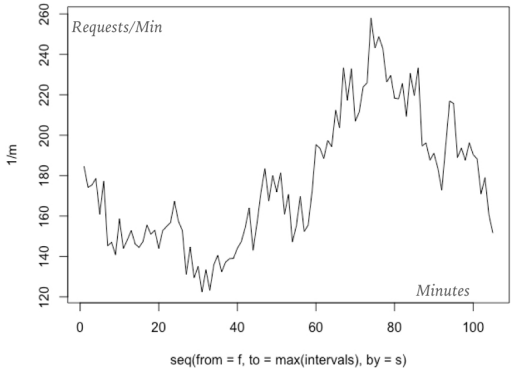


DAY-79

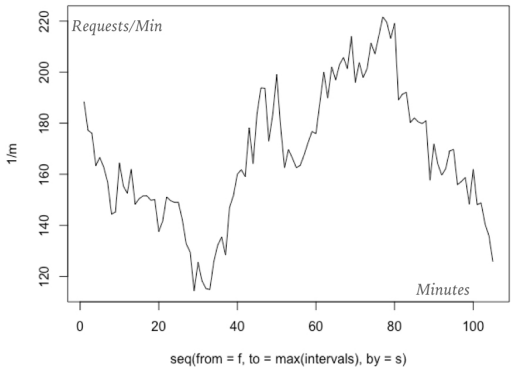


DAY-80

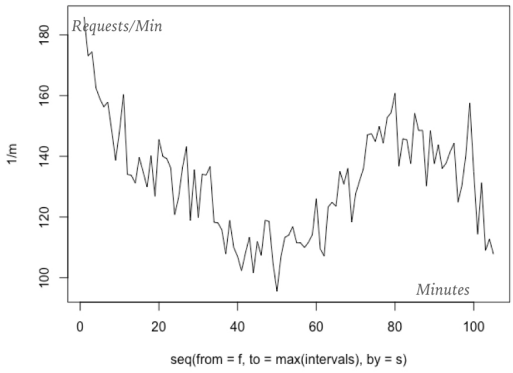
DAY-83



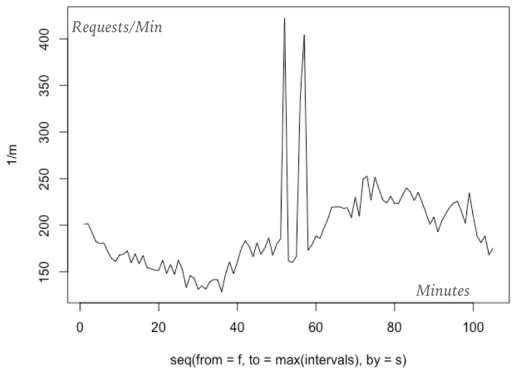
DAY-81



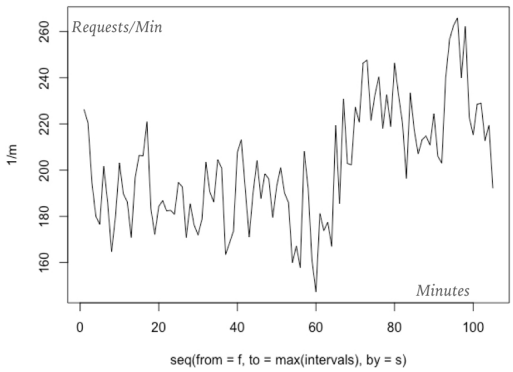
DAY-84



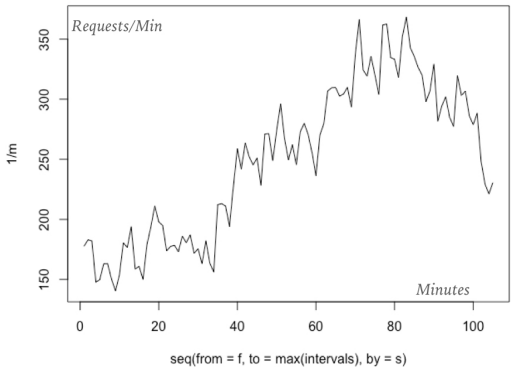
DAY-82



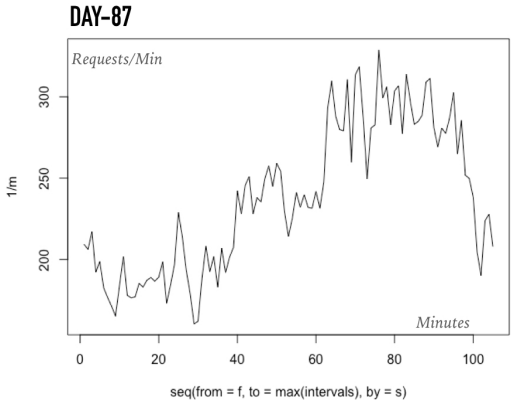
DAY-85



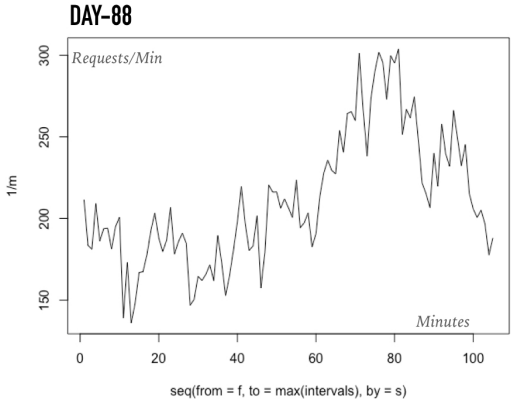
DAY-86



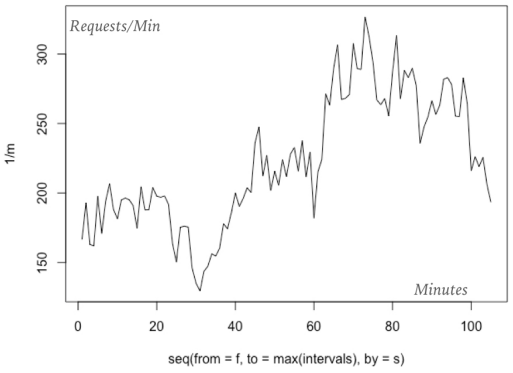
DAY-87



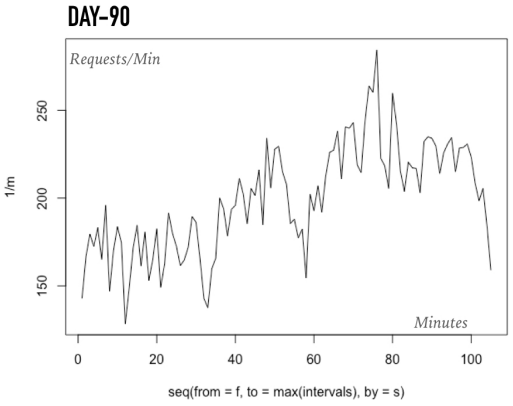
DAY-88



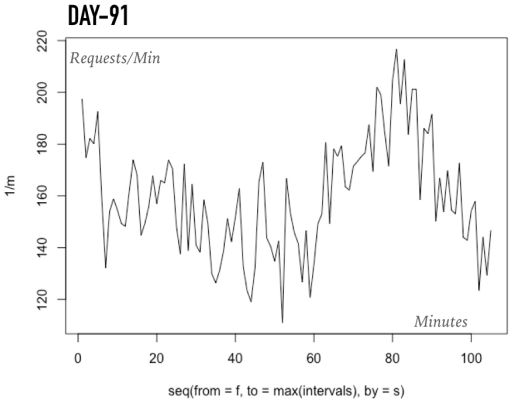
DAY-89

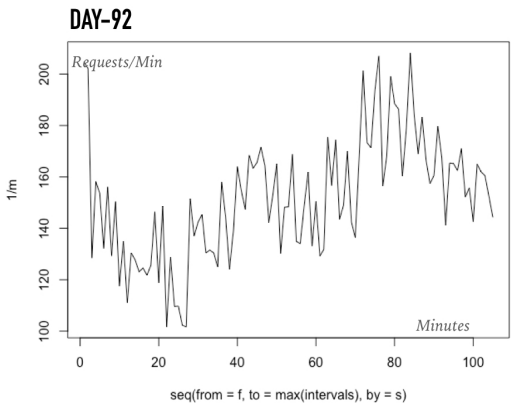


DAY-90



DAY-91




DAY-92
Requests/Min
Minutes
1/m
100
120
140
160
180
200
0
20
40
60
80
100
seq(from = f, to = max(intervals), by = s)

Bibliography

[1] Agnar Aamodt and Enric Plaza. Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. *AI Commun.*, 7(1):39–59, 1994. doi: 10.3233/AIC-1994-7104. URL https://doi.org/10.3233/AIC-1994-7104. 2.1.1, 7.2.1

[2] Hervé Abdi and Lynne J Williams. Principal Component Analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4):433–459, 2010. 4.2.2, 7.4, 8.2.3

[3] Ahmed Ali-Eldin, Maria Kihl, Johan Tordsson, and Erik Elmroth. Efficient Provisioning of Bursty Scientific Workloads on the Cloud Using Adaptive Elasticity Control. In *Proceedings of the 3rd Workshop on Scientific Cloud Computing*, ScienceCloud '12, pages 31–40, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1340-7. doi: 10.1145/2287036.2287044. URL http://doi.acm.org/10.1145/2287036.2287044. 2.4, 4

[4] Mehdi Amoui, Mazeiar Salehie, Siavash Mirarab, and Ladan Tahvildari. Adaptive Action Selection in Autonomic Software Using Reinforcement Learning. In *Fourth International Conference on Autonomic and Autonomous Systems, ICAS 2008, 16-21 March 2008, Gosier, Guadeloupe*, pages 175–181, 2008. doi: 10.1109/ICAS.2008.35. URL https://doi.org/10.1109/ICAS.2008.35. 3.4.2

[5] M. Arlitt and T. Jin. A Workload Characterization Study of the 1998 World Cup Web Site. *IEEE Network*, 14(3):30–37, May 2000. doi: 10.1109/65.844498. 6.1.1

[6] A. Bauer, N. Herbst, S. Spinner, A. Ali-Eldin, and S. Kounev. Chameleon: A Hybrid, Proactive Auto-Scaling Mechanism on a Level-Playing Field. *IEEE Transactions on Parallel and Distributed Systems*, 30(4):800–813, April 2019. doi: 10.1109/TPDS.2018.2870389. 2.4, 6.1.1, 8.4

[7] Michael Beetz and Drew V. McDermott. Improving Robot Plans During Their Execution. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems, University of Chicago, Chicago, Illinois, USA, June 13-15, 1994*, pages 7–12, 1994. URL http://www.aaai.org/Library/AIPS/1994/aips94-002.php. 2.4, 3.4.1, 3.4.3, 8.4

[8] Farinaz Behrooz, Rubiyah Yusof, and Uswah Khairuddin. Hybrid Nonlinear Controller Design for Air Conditioning System. In *12th Asian Control Conference, ASCC 2019, Kitakyushu-shi, Japan, June 9-12, 2019*, pages 793–798, 2019. URL http://ieeexplore.ieee.org/document/8764963. 2.1.1, 7.2.1

[9] Pascal Bercher. *Hybrid planning-from theory to practice*. PhD thesis, Universität Ulm,

2018. 2.2

[10] Andrew Berns and Sukumar Ghosh. Dissecting self-* properties. In *Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2009, San Francisco, California, USA, September 14-18, 2009*, pages 10–19, 2009. doi: 10.1109/SASO.2009.25. URL https://doi.org/10.1109/SASO.2009.25. 4.2.2, 8.2.1

[11] Piergiorgio Bertoli, Alessandro Cimatti, Marco Pistore, Marco Roveri, and Paolo Traverso. MBP: A Model Based Planner. In *Procedings of the IJCAI'01 Workshop on Planning under Uncertainty and Incomplete Information*, 2001. 2.4, 6.5.1

[12] L. F. Bertuccelli and J. P. How. Robust UAV Search for Environments with Imprecise Probability Maps. In *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 5680–5685, Dec 2005. doi: 10.1109/CDC.2005.1583068. 6.1.2

[13] Avrim Blum and Merrick L. Furst. Fast Planning Through Planning Graph Analysis. *Artificial Intelligence*, 90(1-2):281–300, 1997. doi: 10.1016/S0004-3702(96)00047-1. URL https://doi.org/10.1016/S0004-3702(96)00047-1. 2.1.2, 7.2.1

[14] Blai Bonet and Hector Geffner. Planning as heuristic search. *Artif. Intell.*, 129(1-2):5–33, 2001. doi: 10.1016/S0004-3702(01)00108-4. URL https://doi.org/10.1016/S0004-3702(01)00108-4. 1, 2.1.2

[15] Blai Bonet and Hector Geffner. Labeled RTDP: Improving the Convergence of Real-Time Dynamic Programming. In *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS 2003), June 9-13, 2003, Trento, Italy*, pages 12–21, 2003. URL http://www.aaai.org/Library/ICAPS/2003/icaps03-002.php. 2.4

[16] Blai Bonet and Hector Geffner. mGPT: A Probabilistic Planner Based on Heuristic Search. *Journal of Artificial Intelligence Research*, 24:933–944, 2005. doi: 10.1613/jair.1688. URL https://doi.org/10.1613/jair.1688. 6.5.1

[17] Blai Bonet and Hector Geffner. Action Selection for MDPs: Anytime AO* Versus UCT. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada.*, 2012. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/5136. 7.2.1

[18] Craig Boutilier, Richard Dearden, and Moisés Goldszmidt. Exploiting Structure in Policy Construction. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes*, pages 1104–1113, 1995. URL http://ijcai.org/Proceedings/95-2/Papers/012.pdf.

[19] Sebastian Brechtel, Tobias Gindele, and Rüdiger Dillmann. Probabilistic MDP-behavior planning for cars. In *14th International IEEE Conference on Intelligent Transportation Systems, ITSC 2011, Washington, DC, USA, October 5-7, 2011*, pages 1537–1542, 2011. doi: 10.1109/ITSC.2011.6082928. URL https://doi.org/10.1109/ITSC.2011.6082928. 7.2.1

[20] Edmund K. Burke, Graham Kendall, Jim Newall, Emma Hart, Peter Ross, and Sonia

Schulenburg. Hyper-Heuristics: An Emerging Direction in Modern Search Technology. In *Handbook of Metaheuristics*, pages 457–474. 2003. doi: 10.1007/0-306-48056-5_16. URL https://doi.org/10.1007/0-306-48056-5_16. 2.3, 8.2.4, 8.4

[21] Edmund K. Burke, Michel Gendreau, Matthew R. Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: a survey of the state of the art. *JORS*, 64(12): 1695–1724, 2013. doi: 10.1057/jors.2013.71. URL https://doi.org/10.1057/jors.2013.71. 1

[22] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Comp. Syst.*, 25(6):599–616, 2009. doi: 10.1016/j.future.2008.12.001. URL https://doi.org/10.1016/j.future.2008.12.001. 6.1.1

[23] Javier Cámara, David Garlan, Bradley R. Schmerl, and Ashutosh Pandey. Optimal planning for architecture-based self-adaptation via model checking of stochastic games. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing, Salamanca, Spain, April 13-17, 2015*, pages 428–435, 2015. doi: 10.1145/2695664.2695680. URL https://doi.org/10.1145/2695664.2695680. 1, 1.1

[24] Javier Cámara, Gabriel A. Moreno, and David Garlan. Reasoning about Human Participation in Self-Adaptive Systems. In *10th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2015, Florence, Italy, May 18-19, 2015*, pages 146–156, 2015. doi: 10.1109/SEAMS.2015.14. URL https://doi.org/10.1109/SEAMS.2015.14. 8.3.2

[25] Javier Cámara, Bradley R. Schmerl, Gabriel A. Moreno, and David Garlan. MOSAICO: Offline Synthesis of Adaptation Strategy Repertoires with Flexible Trade-offs. *Automated Software Engineering*, 25(3):595–626, 2018. doi: 10.1007/s10515-018-0234-9. URL https://doi.org/10.1007/s10515-018-0234-9. 1

[26] Taolue Chen, Vojtech Forejt, Marta Z. Kwiatkowska, David Parker, and Aistis Simaitis. PRISM-games: A Model Checker for Stochastic Multi-Player Games. In *Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference, TACAS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*, pages 185–191, 2013. doi: 10.1007/978-3-642-36742-7_13. URL https://doi.org/10.1007/978-3-642-36742-7_13. 7.2.1

[27] Shang-Wen Cheng, An-Cheng Huang, David Garlan, Bradley R. Schmerl, and Peter Steenkiste. Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure. In *1st International Conference on Autonomic Computing (ICAC 2004), 17-19 May 2004, New York, NY, USA*, pages 276–277, 2004. doi: 10.1109/ICAC.2004.46. URL http://doi.ieeecomputersociety.org/10.1109/ICAC.2004.46. 1, 1.1, 1.5, 2.1.1, 3.2, 6.6, 7.2.1, 8.1.2, 8.4

[28] Zack Coker, David Garlan, and Claire Le Goues. SASS: Self-Adaptation Using Stochastic Search. In *10th IEEE/ACM International Symposium on Software Engineering for Adaptive*

and Self-Managing Systems, SEAMS 2015, Florence, Italy, May 18-19, 2015, pages 168–174, 2015. doi: 10.1109/SEAMS.2015.16. URL https://doi.org/10.1109/SEAMS.2015.16. 1, 1.1, 6.6, 8.1.2

[29] OW2 Consortium et al. Rubis: Rice university bidding system. 2013. URL http://rubis.ow2.org. 1.1

[30] Padraig Cunningham and Sarah Jane Delany. k-Nearest Neighbour Classifiers. *Multiple Classifier Systems*, 34(8):1–17, 2007. 8.2.3

[31] Tushar Deshpande, Panagiotis Katsaros, Scott A. Smolka, and Scott D. Stoller. Stochastic Game-Based Analysis of the DNS Bandwidth Amplification Attack Using Probabilistic Model Checking. In *2014 Tenth European Dependable Computing Conference, Newcastle, United Kingdom, May 13-16, 2014*, pages 226–237, 2014. doi: 10.1109/EDCC.2014.37. URL https://doi.org/10.1109/EDCC.2014.37. 7.2.1

[32] M. Benjamin Dias, Dominique Locher, Ming Li, Wael El-Deredy, and Paulo J.G. Lisboa. The Value of Personalised Recommender Systems to e-Business: A Case Study. In *Proceedings of the 2008 ACM Conference on Recommender Systems*, RecSys '08, pages 291–294, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-093-7. doi: 10.1145/1454008.1454054. URL http://doi.acm.org/10.1145/1454008.1454054. 1.1

[33] S. Duttagupta, R. Virk, and M. Nambiar. Predicting Performance in the Presence of Software and Hardware Resource Bottlenecks. In *International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2014)*, pages 542–549, July 2014. doi: 10.1109/SPECTS.2014.6879991. 1.1

[34] D. Eskins and W. H. Sanders. The Multiple-Asymmetric-Utility System Model: A Framework for Modeling Cyber-Human Systems. In *2011 Eighth International Conference on Quantitative Evaluation of SysTems*, pages 233–242, Sep. 2011. doi: 10.1109/QEST.2011.38. 8.3.2

[35] Maria Fox. A Modular Architecture for Hybrid Planning with Theories. In *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, pages 1–2, 2014. doi: 10.1007/978-3-319-10428-7_1. URL https://doi.org/10.1007/978-3-319-10428-7_1. 2.2

[36] A. Gandhi, P. Dube, A. Karve, A. Kochut, and L. Zhang. Modeling the Impact of Workload on Cloud Resource Scaling. In *2014 IEEE 26th International Symposium on Computer Architecture and High Performance Computing*, pages 310–317, Oct 2014. doi: 10.1109/SBAC-PAD.2014.16. 1.1

[37] Anshul Gandhi, Mor Harchol-Balter, Ram Raghunathan, and Michael A. Kozuch. AutoScale: Dynamic, Robust Capacity Management for Multi-Tier Data Centers. *ACM Trans. Comput. Syst.*, 30(4):14:1–14:26, 2012. doi: 10.1145/2382553.2382556. URL https://doi.org/10.1145/2382553.2382556. 4.2.2, 6.1.1, 7.2.2

[38] Erann Gat. On Three-layer Architectures. *Artificial Intelligence and Mobile Robots*, 195: 210, 1998. 2.1.4

[39] Simos Gerasimou, Radu Calinescu, and Alec Banks. Efficient Runtime Quantitative Verification Using Caching, Lookahead, and Nearly-optimal Reconfiguration. In *9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2014, Proceedings, Hyderabad, India, June 2-3, 2014*, pages 115–124, 2014. doi: 10.1145/2593929.2593932. URL https://doi.org/10.1145/2593929.2593932. 1.1, 6.1.1, 8.2.1, 8.2.4

[40] Alfonso Gerevini, Alessandro Saetti, Ivan Serina, and Paolo Toninelli. LPG-TD: A Fully Automated Planner for PDDL2. 2 Domains. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS) International Planning Competition abstracts*. Citeseer, 2004. 2.1.2

[41] Arthur Gervais, Ghassan O. Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the Security and Performance of Proof of Work Blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 3–16, 2016. doi: 10.1145/2976749.2978341. URL https://doi.org/10.1145/2976749.2978341. 7.2.1

[42] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely Randomized Trees. *Machine learning*, 63(1):3–42, 2006. 6.2.1

[43] S. Ghahremani, H. Giese, and T. Vogel. Efficient Utility-Driven Self-Healing Employing Adaptation Rules for Large Dynamic Architectures. In *2017 IEEE International Conference on Autonomic Computing (ICAC)*, pages 59–68, July 2017. doi: 10.1109/ICAC.2017.35. 8.2.1

[44] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: theory and practice*. Elsevier, 2004. 1, 1.3, 4.1.2, 5.2, 7.2.1, 3, 8.2.1, 8.2.3

[45] Fred Glover and Manuel Laguna. Tabu Search. In *Handbook of combinatorial optimization*, pages 2093–2229. Springer, 1998. 2.3

[46] Jonathan Gratch and Steve A. Chien. Adaptive problem-solving for large-scale scheduling problems: A case study. *Journal of Artificial Intelligence Research*, 4:365–396, 1996. doi: 10.1613/jair.177. URL https://doi.org/10.1613/jair.177. 2.3

[47] Jonathan Gratch, Steve A. Chien, and Gerald DeJong. Learning Search Control Knowledge for Deep Space Network Scheduling. In *Machine Learning, Proceedings of the Tenth International Conference, University of Massachusetts, Amherst, MA, USA, June 27-29, 1993*, pages 135–142, 1993. doi: 10.1016/b978-1-55860-307-3.50024-1. URL https://doi.org/10.1016/b978-1-55860-307-3.50024-1. 2.3

[48] Mor Harchol-Balter. *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. Cambridge University Press, 2013. 1.1

[49] Jörg Hoffmann and Bernhard Nebel. The FF Planning System: Fast Plan Generation Through Heuristic Search. *J. Artif. Intell. Res.*, 14:253–302, 2001. doi: 10.1613/jair.855. URL https://doi.org/10.1613/jair.855. 1, 2.1.2

[50] Shigeru Imai. *Elastic Cloud Computing for QOS-aware Data Processing*. PhD thesis, Rensselaer Polytechnic Institute, 2018. 6.1.1

[51] Waheed Iqbal, Matthew N. Dailey, David Carrera, and Paul Janecek. Adaptive resource provisioning for read intensive multi-tier applications in the cloud. *Future Generation Comp. Syst.*, 27(6):871–879, 2011. doi: 10.1016/j.future.2010.10.016. URL https://doi.org/10.1016/j.future.2010.10.016. 2.4, 4

[52] M. A. Islam, S. Ren, A. H. Mahmud, and G. Quan. Online Energy Budgeting for Cost Minimization in Virtualized Data Center. *IEEE Transactions on Services Computing*, 9(3): 421–432, May 2016. doi: 10.1109/TSC.2015.2390231. 1.1

[53] Pooyan Jamshidi, Aakash Ahmad, and Claus Pahl. Autonomic resource provisioning for cloud-based software. In *9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2014, Proceedings, Hyderabad, India, June 2-3, 2014*, pages 95–104, 2014. doi: 10.1145/2593929.2593940. URL https://doi.org/10.1145/2593929.2593940. 1, 1.1, 6.6, 8.1.2

[54] Andreas Junghanns and Jonathan Schaeffer. Domain-Dependent Single-Agent Search Enhancements. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages*, pages 570–577, 1999. URL http://ijcai.org/Proceedings/99-1/Papers/082.pdf. 1

[55] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and Acting in Partially Observable Stochastic Domains. *Artificial Intelligence*, 101(1-2):99–134, 1998. doi: 10.1016/S0004-3702(98)00023-X. URL https://doi.org/10.1016/S0004-3702(98)00023-X. 1.1, 2.1.2, 5.2, 7.2.1

[56] Daniel Kahneman and Patrick Egan. *Thinking, Fast and Slow*, volume 1. Farrar, Straus and Giroux New York, 2011. 2.2, 8.2.4, 8.4

[57] Subbarao Kambhampati. A Comparative Analysis of Partial Order Planning and Task Reduction Planning. *SIGART Bulletin*, 6(1):16–25, 1995. doi: 10.1145/202187.202192. URL https://doi.org/10.1145/202187.202192. 2.2

[58] Subbarao Kambhampati, Amol Dattatraya Mali, and Biplav Srivastava. Hybrid Planning for Partially Hierarchical Domains. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 98, IAAI 98, July 26-30, 1998, Madison, Wisconsin, USA.*, pages 882–888, 1998. URL http://www.aaai.org/Library/AAAI/1998/aaai98-125.php. 2.2

[59] Donald L Keefer. Certainty Equivalents for Three-point Discrete-distribution Approximations. *Management science*, 40(6):760–773, 1994. 6.1.1, 6.1.2

[60] Thomas Keller and Malte Helmert. Trial-Based Heuristic Tree Search for Finite Horizon MDPs. In *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling, ICAPS 2013, Rome, Italy, June 10-14, 2013*, 2013. URL http://www.aaai.org/ocs/index.php/ICAPS/ICAPS13/paper/view/6026. 7.2.1

[61] J. O. Kephart and D. M. Chess. The Vision of Autonomic Computing. *IEEE Computer*, 36 (1):41–50, Jan 2003. doi: 10.1109/MC.2003.1160055. 1, 5.3

[62] Dongsun Kim and Sooyong Park. Reinforcement learning-based dynamic adaptation planning method for architecture-based self-managed software. In *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2009, Vancouver, BC, Canada, May 18-19, 2009*, pages 76–85, 2009. doi: 10.1109/SEAMS.2009.5069076. URL https://doi.org/10.1109/SEAMS.2009.5069076. 1

[63] Cody Kinneer, Zack Coker, Jiacheng Wang, David Garlan, and Claire Le Goues. Managing Uncertainty in Self-adaptive Systems with Plan Reuse and Stochastic Search. In *Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '18, pages 40–50, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5715-9. doi: 10.1145/3194133.3194145. URL http://doi.acm.org/10.1145/3194133.3194145. 1, 6.6

[64] Cristian Klein, Martina Maggio, Karl-Erik Årzén, and Francisco Hernández-Rodriguez. Brownout: Building More Robust Cloud Applications. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 700–711, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2756-5. doi: 10.1145/2568225.2568227. URL http://doi.acm.org/10.1145/2568225.2568227. 1.1

[65] John C. Knight. Safety Critical Systems: Challenges and Directions. In *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, pages 547–550, New York, NY, USA, 2002. ACM. ISBN 1-58113-472-X. doi: 10.1145/581339.581406. URL http://doi.acm.org/10.1145/581339.581406. 1

[66] Ron Kohavi. A Study of Cross-validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, pages 1137–1143, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc. ISBN 1-55860-363-8. URL http://dl.acm.org/citation.cfm?id=1643031.1643047. 4.2.2

[67] David Kortenkamp, Reid G. Simmons, and Davide Brugali. Robotic Systems Architectures and Programming. In *Springer Handbook of Robotics*, pages 283–306. 2016. doi: 10.1007/978-3-319-32552-1_12. URL https://doi.org/10.1007/978-3-319-32552-1_12. 2.1.4

[68] Jeff Kramer and Jeff Magee. Self-Managed Systems: an Architectural Challenge. In *International Conference on Software Engineering, ISCE 2007, Workshop on the Future of Software Engineering, FOSE 2007, May 23-25, 2007, Minneapolis, MN, USA*, pages 259–268, 2007. doi: 10.1109/FOSE.2007.19. URL https://doi.org/10.1109/FOSE.2007.19. 2.1.4

[69] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, pages 585–591, 2011. doi: 10.1007/978-3-642-22110-1_47. URL https://doi.org/10.1007/978-3-642-22110-1_47. 3.2, 4.2.2, 7.2.2, 8.2.3

[70] Amina Lamghari and Roussos Dimitrakopoulos. Hyper-heuristic Approaches for Strategic Mine Planning Under Uncertainty. *Computers Operations Research*, 11 2018. doi:

10.1016/j.cor.2018.11.010. 2.3

[71] Steven M LaValle. *Planning Algorithms*. Cambridge university press, 2006. 7.2.1, 8.2.1, 8.2.3

[72] Hui Li and Brian Williams. Hybrid Planning with Temporally Extended Goals for Sustainable Ocean Observing. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*, 2011. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/view/3667. 2.2

[73] Michael L. Littman, Thomas L. Dean, and Leslie Pack Kaelbling. On the Complexity of Solving Markov Decision Problems. *CoRR*, abs/1302.4971, 2013. URL http://arxiv.org/abs/1302.4971. 2.1.2, 7.2.1

[74] Eric Lloyd, Shihong Huang, and Emmanuelle Tognoli. Improving human-in-the-loop adaptive systems using brain-computer interaction. In *12th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS@ICSE 2017, Buenos Aires, Argentina, May 22-23, 2017*, pages 163–174, 2017. doi: 10.1109/SEAMS.2017.1. URL https://doi.org/10.1109/SEAMS.2017.1. 8.3.2

[75] Wyatt Lloyd, Michael J. Freedman, Michael Kaminsky, and David G. Andersen. Stronger Semantics for Low-latency Geo-replicated Storage. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, nsdi'13, pages 313–328, Berkeley, CA, USA, 2013. USENIX Association. URL http://dl.acm.org/citation.cfm?id=2482626.2482657. 1.1

[76] Freerk A Lootsma. *Fuzzy Logic for Planning and Decision Making*, volume 8. Springer Science & Business Media, 2013. 2.1.1, 7.2.1

[77] Frank D. Macías-Escrivá, Rodolfo E. Haber, Raúl M. del Toro, and Vicente Hernández. Self-adaptive systems: A survey of current approaches, research challenges and applications. *Expert Systems with Applications*, 40(18):7267–7279, 2013. doi: 10.1016/j.eswa.2013.07.033. URL https://doi.org/10.1016/j.eswa.2013.07.033. 1

[78] Mausam and Andrey Kolobov. *Planning with Markov Decision Processes: An AI Perspective*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012. doi: 10.2200/S00426ED1V01Y201206AIM017. URL https://doi.org/10.2200/S00426ED1V01Y201206AIM017. 1.3, 2.1.2, 3.2, 4.1.2, 7.2, 7.2.1, 8.2.2

[79] Mausam, Piergiorgio Bertoli, and Daniel S. Weld. A hybridized planner for stochastic domains. In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 1972–1978, 2007. URL http://ijcai.org/Proceedings/07/Papers/318.pdf. 2.4, 3, 3.4.3, 3.5, 6, 6.5.1, 8.1.1, 8.4

[80] Bruce L Miller. Finite State Continuous Time Markov Decision Processes with a Finite Planning Horizon. *SIAM Journal on Control*, 6(2):266–280, 1968. 2.1.2

[81] Tom M Mitchell. Machine Learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45(37):870–877,

1997. 4, 7.4, 8.2.3

[82] G. A. Moreno, J. CÃąmara, D. Garlan, and B. Schmerl. Efficient Decision-Making Under Uncertainty for Proactive Self-Adaptation. In *2016 IEEE International Conference on Autonomic Computing (ICAC)*, pages 147–156, July 2016. doi: 10.1109/ICAC.2016.59. 1.1, 1.1

[83] G. A. Moreno, O. Strichman, S. Chaki, and R. Vaisman. Decision-Making with Cross-Entropy for Self-Adaptation. In *2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 90–101, May 2017. doi: 10.1109/SEAMS.2017.7. 6.6

[84] Gabriel A. Moreno. *Adaptation Timing in Self-Adaptive Systems*. PhD thesis, Carnegie Mellon University, 2017. 6.1.2

[85] Gabriel A. Moreno, Javier Cámara, David Garlan, and Bradley R. Schmerl. Proactive self-adaptation under uncertainty: a probabilistic model checking approach. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015*, pages 1–12, 2015. doi: 10.1145/2786805.2786853. URL https://doi.org/10.1145/2786805.2786853. 1, 1.1, 1.1, 3.2, 4.2.2, 6.1.1, 6.6, 8.1.2, 8.2.1

[86] Gabriel A. Moreno, Javier Cámara, David Garlan, and Bradley Schmerl. Flexible and Efficient Decision-Making for Proactive Latency-Aware Self-Adaptation. *ACM Trans. Auton. Adapt. Syst.*, 13(1):3:1–3:36, April 2018. ISSN 1556-4665. doi: 10.1145/3149180. URL http://doi.acm.org/10.1145/3149180. 6.6, 8.1.2, 8.2.4

[87] Gabriel A. Moreno, Bradley Schmerl, and David Garlan. SWIM: An Exemplar for Evaluation and Comparison of Self-adaptation Approaches for Web Applications. In *Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '18, pages 137–143, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5715-9. doi: 10.1145/3194133.3194163. URL http://doi.acm.org/10.1145/3194133.3194163. 6.1.1, 6.6, 8.1.2

[88] Gabriel A. Moreno, Cody Kinneer, Ashutosh Pandey, and David Garlan. DARTSim: An Exemplar for Evaluation and Comparison of Self-adaptation Approaches for Smart Cyber-physical Systems. In *Proceedings of the 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '19, pages 181–187, Piscataway, NJ, USA, 2019. IEEE Press. doi: 10.1109/SEAMS.2019.00031. URL https://doi.org/10.1109/SEAMS.2019.00031. 6.3, 6.1.2, 6.6, 8.1.2

[89] Hala Mostafa and Victor R. Lesser. Offline Planning for Communication by Exploiting Structured Interactions in Decentralized MDPs. In *Proceedings of the 2009 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IAT 2009, Milan, Italy, 15-18 September 2009*, pages 193–200, 2009. doi: 10.1109/WI-IAT.2009.150. URL https://doi.org/10.1109/WI-IAT.2009.150. 7.2.1

[90] David J. Musliner, Edmund H. Durfee, and Kang G. Shin. World Modeling for the Dynamic Construction of Real-Time Control Plans. *Artificial Intelligence*, 74(1):83–127, 1995. doi: 10.1016/0004-3702(94)00008-O. URL https://doi.org/10.1016/0004-

3702(94)00008-O. 2.1.4

[91] Gethin Norman, David Parker, and Xueyi Zou. Verification and control of partially observable probabilistic systems. *Real-Time Systems*, 53(3):354–402, 2017. doi: 10.1007/s11241-017-9269-4. URL https://doi.org/10.1007/s11241-017-9269-4. 8.2.3

[92] A. Pandey, B. Schmerl, and D. Garlan. Instance-Based Learning for Hybrid Planning. In *2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, pages 64–69, Sep. 2017. doi: 10.1109/FAS-W.2017.122. 3.4.3

[93] Ashutosh Pandey. Prism planning specifications for the cloud-based system and the team of UAVs. http://reports-archive.adm.cs.cmu.edu/anon/isr2020/abstracts/20-100-Appendix.html. 6.1.1

[94] Ashutosh Pandey, Gabriel A. Moreno, Javier Cámara, and David Garlan. Hybrid Planning for Decision Making in Self-Adaptive Systems. In *10th IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2016, Augsburg, Germany, September 12-16, 2016*, pages 130–139, 2016. doi: 10.1109/SASO.2016.19. URL https://doi.org/10.1109/SASO.2016.19. 6.6, 8.1.2, 8.2.1

[95] Ashutosh Pandey, Ivan Ruchkin, Bradley R. Schmerl, and Javier Cámara. Towards a formal framework for hybrid planning in self-adaptation. In *12th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS@ICSE 2017, Buenos Aires, Argentina, May 22-23, 2017*, pages 109–115, 2017. doi: 10.1109/SEAMS.2017.14. URL https://doi.org/10.1109/SEAMS.2017.14.

[96] Joelle Pineau, Geoffrey J. Gordon, and Sebastian Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pages 1025–1032, 2003. URL http://ijcai.org/Proceedings/03/Papers/147.pdf. 2.1.2, 7.2.1, 7.2.1

[97] Barry Porter and Roberto Vito Rodrigues Filho. Losing Control: The Case for Emergent Software Systems Using Autonomous Assembly, Perception, and Learning. In *10th IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2016, Augsburg, Germany, September 12-16, 2016*, pages 40–49, 2016. doi: 10.1109/SASO.2016.10. URL https://doi.org/10.1109/SASO.2016.10. 1, 1.1, 3.4.2, 6.6, 8.1.2

[98] K. Qazi, Y. Li, and A. Sohn. Workload Prediction of Virtual Machines for Harnessing Data Center Resources. In *2014 IEEE 7th International Conference on Cloud Computing*, pages 522–529, June 2014. doi: 10.1109/CLOUD.2014.76. 1.1

[99] John R. Rice. The Algorithm Selection Problem. *Advances in Computers*, 15:65–118, 1976. doi: 10.1016/S0065-2458(08)60520-3. URL https://doi.org/10.1016/S0065-2458(08)60520-3. 2.3

[100] Stéphane Ross, Joelle Pineau, Sébastien Paquet, and Brahim Chaib-draa. Online Planning Algorithms for POMDPs. *CoRR*, abs/1401.3436, 2014. URL http://arxiv.org/abs/1401.3436. 2.1.2, 7.2.1

[101] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Malaysia; Pearson Education Limited,, 2016. 4.2.2, 8.2.3

[102] Mazeiar Salehie and Ladan Tahvildari. Autonomic computing: Emerging trends and open problems. In *Proceedings of the 2005 Workshop on Design and Evolution of Autonomic Application Software*, DEAS '05, pages 1–7, New York, NY, USA, 2005. ACM. ISBN 1-59593-039-6. doi: 10.1145/1083063.1083082. URL http://doi.acm.org/10.1145/1083063.1083082. 8.2.4

[103] Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software: Landscape and research challenges. *TAAS*, 4(2):14:1–14:42, 2009. doi: 10.1145/1516533.1516538. URL https://doi.org/10.1145/1516533.1516538. 1, 8.2.3, 8.2.4

[104] Bernd Schattenberg, Andreas Weigl, and Susanne Biundo. Hybrid Planning Using Flexible Strategies. In *KI 2005: Advances in Artificial Intelligence, 28th Annual German Conference on AI, KI 2005, Koblenz, Germany, September 11-14, 2005, Proceedings*, pages 249–263, 2005. doi: 10.1007/11551263_21. URL https://doi.org/10.1007/11551263_21. 2.2

[105] Burr Settles. Active learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1):1–114, 2012. 8.3.1

[106] Trey Smith and Reid G. Simmons. Heuristic Search Value Iteration for POMDPs. *CoRR*, abs/1207.4166, 2012. URL http://arxiv.org/abs/1207.4166. 2.1.2, 7.2.1

[107] S. Soltani, P. Martin, and K. Elgazzar. QuARAM Recommender: Case-Based Reasoning for IaaS Service Selection. In *2014 International Conference on Cloud and Autonomic Computing*, pages 220–226, Sep. 2014. doi: 10.1109/ICCAC.2014.26. 1, 1.1, 2.1.1, 3.2, 6.6, 7.2.1, 8.1.2

[108] R. Sukkerd, J. CÃąmara, D. Garlan, and R. Simmons. Multiscale Time Abstractions for Long-Range Planning under Uncertainty. In *2016 IEEE/ACM 2nd International Workshop on Software Engineering for Smart Cyber-Physical Systems (SEsCPS)*, pages 15–21, May 2016. doi: 10.1109/SEsCPS.2016.011. 3.2, 6.1.1, 8.2.1

[109] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT press, 2018. 2.1.3, 2.3

[110] Daniel Sykes, William Heaven, Jeff Magee, and Jeff Kramer. Plan-directed architectural change for autonomous systems. In *Proceedings of the 2007 Conference Specification and Verification of Component-Based Systems, SAVCBS 2007, Dubrovnik, Croatia, September 3-4, 2007*, pages 15–21, 2007. doi: 10.1145/1292316.1292318. URL https://doi.org/10.1145/1292316.1292318. 1, 3.4.2, 8.2.1

[111] A. Symington, S. Waharte, S. Julier, and N. Trigoni. Probabilistic target detection by camera-equipped UAVs. In *2010 IEEE International Conference on Robotics and Automation*, pages 4076–4081, May 2010. doi: 10.1109/ROBOT.2010.5509355. 6.1.2

[112] Hossein Tajalli, Joshua Garcia, George Edwards, and Nenad Medvidovic. PLASMA: A Plan-based Layered Architecture for Software Model-driven Adaptation. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages

467–476, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0116-9. doi: 10.1145/1858996.1859092. URL http://doi.acm.org/10.1145/1858996.1859092. 2.1.4, 8.2.1

[113] A. Tallavajhula, S. Choudhury, S. Scherer, and A. Kelly. List Prediction Applied to Motion Planning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 213–220, May 2016. doi: 10.1109/ICRA.2016.7487136. 2.4, 3.4.3, 8.4

[114] Florent Teichteil-Königsbuch, Charles Lesire, and Guillaume Infantes. A generic framework for anytime execution-driven planning in robotics. In *IEEE International Conference on Robotics and Automation, ICRA 2011, Shanghai, China, 9-13 May 2011*, pages 299–304, 2011. doi: 10.1109/ICRA.2011.5980289. URL https://doi.org/10.1109/ICRA.2011.5980289. 7.2.1

[115] Manuela M. Veloso. *Planning and Learning by Analogical Reasoning*, volume 886 of *Lecture Notes in Computer Science*. Springer, 1994. ISBN 3-540-58811-6. doi: 10.1007/3-540-58811-6. URL https://doi.org/10.1007/3-540-58811-6. 2.1.1, 7.2.1

[116] Michael J. Veth. Advanced Formation Flight Control. Technical report, Air Force Institute of Technology, 1994. 6.1.2

[117] David H. Wolpert and William G. Macready. Coevolutionary free lunches. *IEEE Trans. Evolutionary Computation*, 9(6):721–735, 2005. doi: 10.1109/TEVC.2005.856205. URL https://doi.org/10.1109/TEVC.2005.856205. 5, 2.3

[118] Minxian Xu and Rajkumar Buyya. Brownout Approach for Adaptive Management of Resources and Applications in Cloud Computing Systems: A Taxonomy and Future Directions. *ACM Comput. Surv.*, 52(1):8:1–8:27, 2019. doi: 10.1145/3234151. URL https://doi.org/10.1145/3234151. 1.1

[119] Nan Ye, Adhiraj Somani, David Hsu, and Wee Sun Lee. DESPOT: Online POMDP Planning with Regularization. *J. Artif. Intell. Res.*, 58:231–266, 2017. doi: 10.1613/jair.5328. URL https://doi.org/10.1613/jair.5328. 7.2.1

[120] Lu Yu and Richard R. Brooks. Applying POMDP to moving target optimization. In *Cyber Security and Information Intelligence, CSIIRW '13, Oak Ridge, TN, USA, January 8-10, 2013*, page 49, 2013. doi: 10.1145/2459976.2460032. URL https://doi.org/10.1145/2459976.2460032. 7.2.1

[121] J. Zhan, L. Wang, X. Li, W. Shi, C. Weng, W. Zhang, and X. Zang. Cost-Aware Cooperative Resource Provisioning for Heterogeneous Workloads in Data Centers. *IEEE Transactions on Computers*, 62(11):2155–2168, Nov 2013. doi: 10.1109/TC.2012.103. 6.1.1

[122] Shlomo Zilberstein. Using Anytime Algorithms in Intelligent Systems. *AI Magazine*, 17(3): 73–83, 1996. URL http://www.aaai.org/ojs/index.php/aimagazine/article/view/1232. 2.4, 7.2.1