EXECUTION RESOURCE ALLOCATION FOR DISTRIBUTED REAL-TIME CONTROLLERS

by

Haksun Li

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2004

Doctoral Committee:

Professor Edmund H. Durfee, Co-Chair
Professor Kang G. Shin, Co-Chair
Professor Martha E. Pollack
Professor Robert L. Smith

ACKNOWLEDGEMENTS

I am most grateful and am forever indebted to my advisor for his incessant guidance, support, and encouragement. Professor Ed Durfee has provided tremendous help in completing this dissertation. I am very fortunate to have had this enviable opportunity to learn from such a famous, bright, careful, diligent (often demanding), patient, amiable, and always-smiling scientist. His influences on me go beyond the acquisition of knowledge in computer science. His words and actions have significantly improved my way of thinking and changed my personality. For example, I am a more humble person than I used to be; I am also more patient with people and am more understanding. His teachings are now seeded and will continue to grow throughout my life.

I thank my co-chair, Professor Kang Shin, for teaching me about real-time systems. He has suggested helpful research directions as well as good reference materials to broaden my perspective. As a result, I can better understand the issues involved in my research. I thank also my committee members, Professor Martha Pollack and Professor Robert Smith, for their constructive suggestions and feedback.

During my graduate study, I have been priviledged to collaborate with other graduate students. Ella Atkins has introduced me to CIRCA. I am sorry that I irritated her from time to time (unintentionally). Dmitri Dolgov is a very good fellow and has given

me helpful comments on my work. We have collaborated together to implement the latest version of CIRCA system. I have also enjoyed discussing ideas and papers with other students, such as Brad Clement, Pradeep Pappachan, Chris Brooks, Jeff Cox, Thom Bartold, Jeremy Shapiro, and Mankit Sze. All these people have added colors to my rather plain graduate study experience.

My work was in part funded by Honeywell under the DARPA/AFRL Contract F30602-00-C-0017. I would like to express gratitude for this very generous financial support and acknowledge the detailed comments of the Honeywell colleagues, Robert Goldman, David Musliner, and Hakan Younes.

Of course, this dissertation would not be possible without the love and patience of my beloved parents, my dearest sister, and friends. I am very grateful for their encouragement and support during this challenging time.

Last but not least, I praise LORD God for loving and forgiving a corrupted person like me. I am forever grateful for him leading in my life all steps, the ones that I understand, as well as the ones that I do not understand. I have not a bigger desire than dedicating my life to Him.

TABLE OF CONTENTS

LIST OF TABLES

Table

LIST OF FIGURES

Figure

LIST OF APPENDICES

Chapter 1

Introduction

In this dissertation, we develop techniques for a real-time agent to reason about limited execution resources, whether it is acting alone or sharing the environment with other agents. We assume that by ignoring resource constraints this agent is able to generate a preferred plan that, if it can be executed faithfully, guarantees safety and achieves goals. Our objective is that, from this preferred plan, the agent can iteratively make selective modifications to find a *feasible* plan that it can actually put into operation. While this revised plan may not guarantee absolute safety or goal achievement, its chance of failure and chance of success should be close to those of the preferred plan.

In particular, when the agent lacks the resources required to follow the preferred plan exactly, using the methods in this thesis it can determine which parts of the plan it should spend its resource on and which parts it should sacrifice. The agent may also repair the plan with respect to the resource constraints to reduce resource consumption. Furthermore, this thesis describes how real-time agents in a multiagent environment may efficiently search for local resource allocation improvements to their plans that are globally beneficial.

1.1 Motivation

In a complex and dynamic environment, a real-time agent needs to monitor for aspects of the world state and respond appropriately to emerging hazards before certain deadlines. In general, if the agent allocates more of its resources (or some of its resources more frequently) to monitoring for some states (or state features), then it will be less

capable of tracking others successfully. Similarly, it cannot be assumed to execute all the resource-demanding actions equally well. Therefore, a resource-limited agent must carefully allocate its limited perceptual, computational, and actuator resources to maximize its performance as much as computationally feasible. The resource allocation problem is more intricate when the agent is situated in a multiagent environment where it needs to consider the interactions among agents.

For instance, an ideal car driver could theoretically know exactly what to do in all possible circumstances, obtain the most updated information about the environment, and carry out his reactions instantaneously. A real driver, on the other hand, cannot monitor for his surroundings and react instantaneously because his execution resources are constrained. This may be perhaps because of sensory limitations (e.g., the driver cannot look between the front, the rear-view mirror, and the dashboard fast enough), or actuator limitations (e.g., the driver cannot steer the car, apply the emergency brake, turn on the emergency flashers, and honk the horn all at the same time).

Computational agents for dynamic, real-time applications must also deal with these problems. For example, a driverless autonomous ground vehicle can get into complicated driving situations that demand more resources (e.g., sensors, CPU, speed) than it has. An absolutely safe driving plan is in general not practical but an autonomous vehicle can achieve acceptable performance if it judiciously prioritizes the usage of its limited resources. Other examples of autonomous robotic systems, some of which will be touched later in this thesis, could include unmanned combat aircraft and autonomous remote planetary rovers.

Agents in non-robotic applications also face the same challenges. For example, a recent trend in Wall Street is to move forward from automatic trading (where humans make trading decisions and computers execute them) to program trading (where computers both make and execute trading decisions). An autonomous program trading agent needs to constantly monitor the changing real-time markets, process all the market news and data, and react (practically) instantaneously.[1] To process all the potentially relevant data, however, is impractical. A limited real-time trading agent must prioritize the market situations to track and respond to so as to minimize risk and maximize profit. Autonomous agents operating in other decision-making domains, such as in power plants, intensive care units, or battlefield management systems, might need to make similar kinds of tradeoffs.

In a multiagent environment, not only does an agent (again, consider our driving example above) need to be prepared to react to events arising naturally in the world (e.g., rockslides) but it must also be ready for those due to the activities of other agents. For example, there are usually other cars sharing the road (unfortunately!). The agent must look out for events such as merging traffic, tailgating and cutting across lanes by other cars. The number of possibilities is large because of the many combinations of agent's actions and ways of interactions. Compared to exogenous events in the single agent case, some events in a multiagent environment happen only when other agents have chosen to do certain actions. Thus, some possibilities may or may not happen during execution (e.g., no one has actually tailgated the driver). The agent should not (and probably cannot) plan

[1] 2 seconds ago in Wall Street is history. A friend of the author impressed his boss negatively when market data from his feed program came in 0.5 second later than those from a bank.

for all these possible contingencies otherwise it may waste resources on those events that will never occur.

Furthermore, agents in a multiagent environment can exchange information. Before they engage in communication, the agents may already have some initial plans, especially when the agents are loosely coupled, have different or even conflicting goals, or are owned by different organizations. Continuing our example above, the driver may plan to stay on the leftmost lane all the time; an ambulance may need to overtake cars by using that lane on its way to hospital. When the agents construct their plans independently, they lack coordination. This consequently leads to poor local resource allocation for each of them. Intuitively, the agents should be able to improve the global performance as a group by making mutually-beneficial changes to their local plans. For instance, the driver could have made a small sacrifice by staying out of the leftmost lane to facilitate the ambulance.

Automated agents in worlds shared with other agents (computational or human) are confronted with similar problems. A driverless ground vehicle better considers its potential interactions with emergency vehicles such as ambulances when it is released on the road. These same concepts apply in the other domains previously mentioned, whether robotic (a squadron of unmanned combat aerial vehicles or a team of Mars rovers) or non-robotic (trading agents for a single investment firm that must coordinate international trades, or different electrical power plant/grid agents in different geographical areas that must collaborate to ensure an uninterrupted power supply and maximize the efficiency of power transmission).

These examples illustrate the broad applicability and the practical relevance of solving the fundamental problem of planning under resource constraints among loosely-coupled agents. To solve this problem, an agent needs to prioritize its use of local resources, to avoid wasting resources on things that will not occur, and when possible to work together with other agents to improve their plans to make the best use of their collective resources.

1.2 Problem Statement

This dissertation addresses the problem of constructing a *schedulable* real-time control plan [3] for a real-time agent given its limited execution resources. We define *execution resources* as those that an agent needs to implement and follow its plan. They include the perceptual (e.g., sensors), effectual (e.g., robot arms), and reasoning capabilities (e.g., CPU time) that the agent uses during execution. A real-time control plan consists of a set of periodic tasks together with their deadlines. The agent must schedule those tasks such that all of them are guaranteed to be completed before their respective deadlines.

We assume that the agent is situated in an environment that is dynamic (the world can change via exogenous events besides the agent's own actions) and uncertain (events and actions happen probabilistically). When the agent is sharing the world with other agents, we assume that the agents are loosely coupled and cooperative. We could possibly assume instead that there is a designated agent to centrally construct an effective or even optimal combination of control plans for the collection of agents, but it is debatable whether centrally constructing optimal joint control plans (policies) is feasible [122]. Therefore, we have chosen to emphasize applications where agents are loosely-

coupled to the extent that a divide-and-conquer approach, in which they formulate their own plans independently and then coordinate themselves, is sufficiently effective and much more efficient.

Forming a schedulable control plan for an agent in such an environment, however, may not be feasible because of its limited execution hardware. For example, there may not be enough sensors to support the frequencies and durations for all perceptual tasks. Or, the actuators cannot move fast enough from task to task to keep up with the deadlines. *Thus the control plan, which is generated offline before the agent is put into operation, must satisfy the inherent resource limitations of the agent's execution architecture so that it can in fact be implemented.*

Throughout this dissertation, we stay agnostic about the agent's planning algorithm. We only require that the agent is capable of generating a plan by deciding what action to take in a state. Depending on the method that the agent uses to decide on an action in a state, the preferred action choice may or may not be optimal. The agent may or may not reason about resource utilization during planning. Consequently, the plan that the agent initially produces may not satisfy its resource constraints. Its preferred (but not necessarily optimal) plan may not be schedulable.

Our primary objective is therefore to design computationally tractable algorithms that the agent can iteratively apply in a greedy manner to modify an existing unschedulable plan in order to generate a satisficing plan given the resource constraints. While this satisficing plan may have a lower utility because it is more restricted, each modification should heuristically decrease the plan utility minimally. In other words, the algorithms will not necessarily find the globally optimal plan. They also do not improve

upon the quality of the initial plan because they explore only the search space starting from the initial plans and do so in a myopic and greedy fashion. Instead, the algorithms transform an unschedulable plan to a schedulable plan so that the agent's performance degrades gracefully as its resource constraints become more severe relative to the resource needs of its preferred and unconstrained plan.

More specifically, we will show that finding the optimal schedulable plan is usually intractable. To transform an unschedulable plan to a schedulable plan, we answer the question of what information an agent needs, and how the agent applies the information to incrementally modify its initial, preferred, unconstrained plan toward satisfying its resource constraints. We analytically and empirically investigate the usefulness of the information, the cost to discover the information, and the computational complexities of the algorithms that take advantage of the information. In this thesis, there are four kinds of information we consider: the probabilistic temporal trajectory of an agent during execution, the post-planning cost-benefit analysis of the actions in the preferred plan, other cooperating agents' plans, and the better understanding of the interactions among agents after discovering other agents' plans.

Therefore, a solution to the problem of constructing a schedulable real-time control plan that this dissertation seeks will consist of a set of carefully characterized, analyzed, and evaluated data structures and algorithms for discovering, representing, and using new knowledge for scheduling limited agent resources. These algorithms and representations can be used in a variety of applications such as those mentioned in the last section to tractably formulate resource-efficient plans. To justify and validate our

solution, we have implemented and tested this more general solution in the Cooperative Intelligent Real-time Control Architecture.

1.3 Approach

In pursuit of the overall goal of making intelligent resource allocation decisions for a real-time agent, whether it is alone or in a shared environment, this dissertation presents methods that draw on tools from mathematics, operations research, simulation, as well as AI. We did not develop a new architecture for this research. Instead, we took advantage of existing systems that combine planning, scheduling, and plan execution to automate the process of designing and executing real-time control plans.

Specifically, we build on top of the Cooperative Intelligent Real-time Control Architecture (CIRCA) [87], and CIRCA-II [2], which select, schedule, and execute recognition-reactions assuming a resource-limited execution architecture.[2] The major new capabilities we have added to CIRCA enable the agent to: 1) model the execution trajectory of a real-time agent as a stochastic process, 2) efficiently search for improvements to reduce schedule utilization of its plan,[3] 3) reason about the activities of other agents and their influences on the agent, 4) propose and evaluate local plan improvements collaboratively with other agents to find a better global resource allocation.

In theory, an ideal agent with no resource constraints needs only to figure out what action to take in each of the states using traditional planning techniques. In practice, the resource requirements for such a policy or a control plan are usually overwhelming. Our strategy to make this unconstrained plan schedulable is to have the agent incrementally modify this plan using information.

[2] We will just call them CIRCA throughout this thesis without making a distinction.
[3] Schedule utilization is defined formally in Section 3.1.

There are two ways to modify a plan – removing and replacing actions. Using the removing action technique, an agent needs to prioritize the actions by how much they affect the utility if they are removed from the preferred plans. The agent should remove the actions that have the least impacts to the plan utility. As events in a complex and dynamic real-time domain happen only probabilistically, some events are more likely than others to happen. Thus, not all actions are equally likely to be needed. When the consequences of failing to perform the actions are equally bad, the agent should drop the actions that are least likely to be used until it can fit the rest in a schedule. Also, we briefly discuss at the end of Section 4.10 the extension of this approach to handle the more general case when the utilities of states need to be considered.

To reason about the probabilistic temporal execution trajectory of a real-time agent, we have designed a probabilistic planning framework that allows an agent to model the temporal dynamics of its actions and the exogenous environmental events as stochastic processes. More importantly, using this action and event representation, an agent can compute the probabilities of encountering different states (events) during run-time.

To calculate precisely these probabilities is very computationally expensive, if at all possible, so we have also provided a discrete approximation method. For stochastic processes that are Markovian, we describe how to compute these probabilities analytically. For non-Markovian processes, we combine this discrete approximation with tools from operations research and statistics to estimate the probabilities by simulating the stochastic processes.

The replacing action technique to modify a plan is in fact a generalization of the removing action technique. When an action is dropped, the associated task becomes unhandled. For all intents and purposes, the action is effectively replaced by a NOOP. Yet, instead of ignoring the unlikely tasks altogether, the agent may be better off by handling them by some less desirable actions.

We have developed an action replacement algorithm to let the agent replace actions with cheaper actions. The algorithm begins with the plan that the agent would construct given unlimited resources, and iteratively repairs the plan to minimally degrade its utility until the plan becomes schedulable. The action replacement algorithm is a hill-climbing algorithm so it considers one replacement at a time, ignoring the combinatorial effects of choosing different replacing actions. This reduces the time complexity from non-polynomial (exponentiation) to polynomial (multiplication).

Also, to avoid trying and comparing all combinations of action replacements, we appeal to the intuition that reusing actions that are already in the plan reduces the branching factor of the search. Specifically, some actions may be replaced by actions that are already in the plan and that can also acceptably (though not optimally) accomplish the same tasks. To choose the pair of replaced and replacing actions that is estimated to best improve the plan at each iteration, we have developed heuristics to prioritize both which actions to replace and, for each, which replacing actions to try.

This dissertation has also extended these concepts to systems of cooperating real-time agents. These cooperative agents collaboratively generate plans as well as make sure that they do not over-utilize their local resource capacities. Yet, when an agent is situated in a multiagent environment, due to the many possible ways of interactions among agents

and the uncertainty of what other agents will do, the agent's ability to compute its probabilistic execution path is impaired.

To effectively use the removing and replacing action techniques, a resource-limited agent in a multiagent environment needs to know enough about the plans of other agents. To address this problem, we have introduced the Convergence Protocol to improve the local (probabilistic) worldviews of the agents. The protocol exploits the fact that many of the events of concern are conditioned upon the action choices of other agents. That is, some events occur only when the agents interact in certain ways. Some amount of judicious communication among the agents can allow each to develop a more coherent view of the global activities. They can recognize which events are the most important to be prepared for.

Moreover, since communication is usually costly, the agents should not need to acquire the entire plans from other agents, because many details in those plans are irrelevant to their local planning. They exchange only the most pertinent parts of their plans about the ways they interact to reduce the number of messages sent.

Finally, an agent usually selects actions for states based in part on the action choices in other states that it has already made (the partially-constructed plan) and/or the choices it might make in the future (lookahead). The plan context, however, may change when the agent communicates with other agents because it becomes more aware of the global activities. Some actions have become invalidated. They may want to change those now suboptimal actions that they have announced and hence committed to other agents. Furthermore, as one agent changes its planned actions, it could trigger a chain reaction of changes to ripple through other agents' plans.

To improve resource allocation after a context change, we have developed the Propose-Evaluate protocol. This protocol lets agents iteratively improve their local plans in a distributed and controlled manner, exploiting the new knowledge available after a context change, e.g., after running the Convergence Protocol. Specifically, at each iteration, the worst-off agent computes a possible way of modifying its local plan. The agent proposes to other agents for evaluation the changes that are most likely to improve resource allocation. If the changes increase the global performance, they are adopted by all agents. Otherwise, the agent may propose another set of changes. Another agent repeats this process when it becomes the worst-off agent. This cycle continues until either all agents are schedulable, or all unschedulable agents have proposed, or they run out of time (give up).

To search for local improvements, an agent uses the action replacement algorithm which has been proven to be effective in improving an agent's local plan efficiently using post-planning information (in the single agent case). The appropriate post-planning information in the multiagent case is the context change knowledge, such as which actions have become suboptimal. We have therefore incorporated the action replacement algorithm into the Propose-Evaluate protocol to improve a local plan by reconsidering the actions that are now suboptimal.

In summary, to construct a schedulable plan under resource constraints, an agent first uses its initial knowledge to plan appropriate actions for the dynamic world. It will locally attempt to schedule these actions, including applying the action replacement algorithm. If it succeeds, then it need not gather information from other agents. But if it cannot schedule its needed actions and there are other agents in the world, the agent uses

the Convergence Protocol to discover and prune away unreachable states (and the actions it had planned for them). If there are unschedulable agents in the multiagent system, they can use the Propose-Evaluate Protocol to collaboratively search for local resource allocation improvements to increase the global performance. If the agent is still over-utilized, it resorts to dropping the actions least likely to be needed to reduce schedule utilization until it becomes schedulable.

1.4 Contributions

The contributions of this dissertation are directed toward the development of algorithms that can be used to automate the decision-making for a real-time artificial agent, whether or not it is in a shared environment. A realistic real-time agent has a limited architecture that may not be easily extended during execution (e.g., when a Mars rover is carrying out its mission on Mars), or may not be cost-effective to extend (e.g., taking down an automated trading system to perform an extensive upgrade). The agent, using these algorithms, can generate a real-time control plan that can be feasibly implemented on its execution hardware. Specifically, we have made the following advances.

Probabilistic Temporal Projection Model. We have developed a probabilistic model that a real-time agent can use to compute its temporal trajectory. There are other probabilistic planners but they generally do not satisfy the needs of planning for a real-time controller (Sections 2.6). For example, they cannot reason about event durations and periodic actions because they assume transition probabilities are stationary. They cannot model mutually-exclusive activities because multiple effects of executing an action are usually folded into transition probabilities.

In contrast, our model computes transition probabilities based upon the combinations of events, actions, and more importantly, their temporal characteristics such as durations and periods. It can also handle concurrently-enabled, mutually-exclusive, and time-dependent activities (transition probabilities can change with time). Therefore, the model represents a novel and useful tool for researchers in the broader community who develop artificial agents that need to project their temporal trajectories in time-critical applications.

A Bounded Risk Estimator. While it may be obvious that in a probabilistic world some sort of risk likelihood estimation is needed to decide which reactions in the plans are more important than others, it is not so intuitive as to which estimator should be used. Our work shows that ignoring actions based on their probabilities of being used, rather than ignoring hazardous events based on their probabilities of actually happening, has bounded decrements in utility. That is, the worst scenario can be estimated.

Using this probabilistic model, a real-time agent can make informed resource allocation decisions to generate a feasible plan. This feasible plan is generated by dropping the actions that handle the events that are least likely to be encountered. Our experiments show empirically that a reasonably good plan can very often be generated in this way. Therefore, the researchers should use, among other factors, this bounded estimator to estimate the risk or the worst consequences when their agents try to remove actions from plans to reduce resource consumption.

Reusing Actions to Reduce Resource Consumption. Furthermore, we have extended this technique into a more general action replacement algorithm. The existing resource allocation algorithms in the literature make restricting assumptions that are

inappropriate for real-time applications, such as that the transition probabilities are stationary, that explicit state enumeration is possible, that the time horizon is finite, and/or that the environments are static and non-deterministic (Sections 2.1 and 2.5).

Specifically, we have investigated the concept of a *reusing action* and shown that it can guide the search for action replacement. In general, there are many (combinatorial) ways to make a successful action replacement. Also, replacing actions in general does not necessarily lead to improved schedulability. Focusing on reusing actions, however, can lead the search to quickly identify possible replacements that are very likely to reduce resource consumption (benefit) with a small sacrifice to utility (cost). This technique allows researchers building resource-limited agents to reason about the cost-benefit tradeoffs of making resource allocation decisions in real-time domains. The agents can reduce resource consumption and still be able to be prepared for all hazardous events.

Resource Allocation in Multiagent Planning. For a real-time agent in a multiagent environment, our empirical results show that ignorance about other agents' activities can be very detrimental to the agent's local resource allocation. In addition to the traditional multiagent problems such as resolving conflicts among agents (e.g., contending for the same component) and coordinating activities (e.g., ensuring simultaneity), our results mean that multiagent researchers also have to confront the problem that, for agents with limited resources, many of the resources could be unnecessarily wasted.

To address this problem, we have developed the Convergence Protocol that agents can use to acquire information from other agents to make more informed resource allocation decisions. The Convergence Protocol is not a heuristic method for improving

the agents' performance, but instead is a principled way for them to selectively improve their local worldviews. Researchers building agents for multiagent environments, where the agents can interact in many possible ways, should incorporate our protocol into the agents in addition to their coordination mechanisms so that the agents can avoid wasting resources on events that provably can never occur.

Planning Using Context Change Information. As the planning context changes when the agents become more aware of the global activities, the agents can potentially improve their local plans. We have developed the Propose-Evaluate Protocol, which allows agents to collaboratively propose and evaluate in a distributed, controlled manner *only* the local changes that are most likely to be globally beneficial.

This protocol has two advantages over the previously developed techniques for finding satisficing plans (Sections 2.2, 2.3). First, it does not rely on the agents' ability to share/swap their responsibilities. In fact, it does not assume that agents can share responsibilities. The protocol reduces the agents' responsibilities by understanding the dependencies in the task sets. Second, our protocol can search for plan improvements efficiently by taking advantage of the plan context change. That is, while other generic algorithms apply to a wide variety of applications, they are not very efficient for planning problems. Our protocol, though, is specifically designed for the (still large) class of problems involving distributed planning, and it is more efficient because it can reason about domain knowledge that is inherent in planning problems.

When researchers build agents in multiagent environments, and when they foresee there will be room for the agents to improve their plans later on due to changes in plan context, e.g., changes in other agents' plans, changes in environmental settings, their

agents can apply the Propose-Evaluate Protocol. The protocol allows the agents to perform an efficient, informed, and focused search for plan improvements toward satisfying local resource constraints by reasoning about the context change information.

1.5 Outline

The rest of this dissertation is organized as follows. We begin in Chapter 2 with surveying the existing literature and algorithms related to the problem of planning under execution resource constraints we defined in Section 1.2. We compare the assumptions they make to our assumptions, and discuss how our problem is unique in terms of allocating local resources for real-time agents. We will particularly focus on the limitations of these algorithms, and explain why they are inadequate for real-time planning purposes.

In Chapter 3, we describe the architecture of our real-time artificial intelligence planning system. It is the testbed on which we implemented, tested, and evaluated all the techniques developed in this thesis. It also gives an example of why our problem of allocating limited execution resources is realistic and important to a real-time agent in practice. The subsequent chapters assume that the reader is familiar with the concepts and terminologies presented in this chapter.

In Chapter 4, we first describe the probabilistic modeling of the temporal dynamics of events and actions. Using the modeling, an agent can compute the probabilities of visiting all foreseeable states. We describe how to use the standard Markov chain technique to compute the probabilities if the agent's execution trajectory is Markovian. Otherwise, we use Monte Carlo simulation to estimate them. Then for an unschedulable agent, we present the removing action technique that drops the least

important or least likely used actions using the probability information. It can always make an unschedulable agent schedulable because it replaces the actions by only NOOPs, which consume no resources.

In Chapter 5, we generalize this technique and introduce the action replacement algorithm. We begin by analyzing the complexity of replacing actions to find an optimal plan to show that optimization is in general intractable.[4] We must therefore restrict the general action replacement algorithm. We study the concept of a reusing action. Using this concept we develop heuristics to make the action replacement algorithm an effective and efficient search. This addresses the problem of tractably replacing actions to reduce utilization. We also determine the factors that affect the performance of the action replacement algorithm.

In Chapter 6, we apply the concepts and algorithms developed for a single agent to the multiagent case. We describe the extension to our real-time planning architecture to support reasoning about the co-existence of other agents sharing the same environment. We describe how agents would each locally generate a plan to account for all foreseeable contingencies if there were no resource constraints. For unschedulable agents, we present the Convergence Protocol, which the agents can use to iteratively refine their unconstrained plans until either they satisfy their local resource constraints or there is no more information. We demonstrate that it is very important for resource-bounded agents to run the Convergence Protocol because ignorant agents may waste a lot of resources to prepare for events that will never occur.

In Chapter 7, we characterize the high-level computational complexity of finding the optimal changes in a multiagent environment. This problem is in general very

[4] There may be more than one optimal plan.

difficult. We describe our solution, the Propose-Evaluate Protocol, which lets agents collaboratively revise and improve their plans in a distributed, controlled, hill-climbing manner. We then describe how an agent may search for globally-beneficial local changes using the action replacement algorithm. Our empirical results justify that the protocol and the action replacement algorithm often lead to good improvements at a modest communication overhead.

In Chapter 8, we summarize the topics covered and our conclusions in this dissertation. We also discuss some possible directions about further development of CIRCA, the limitations and possible enhancements of the algorithms developed here, as well as the more general resource allocation problem.

Chapter 2

Related Work

In this chapter, we begin by surveying some existing work in resource allocation in both single agent planning (Section 2.1) and multiagent planning (Sections 2.2 and 2.3). We will also look at the resource allocation problem from the perspective of the real-time control community (Section 2.4). Then, in Section 2.5, we will discuss their limitations with respect to real-time failure avoidance. We will point out why these efforts are inadequate for real-time planning purposes, and where our work fills in the broader community of resource allocation research. As part of our solution involves probabilistic reasoning, we will also describe some research in probabilistic temporal projection and planning in Section 2.6.

2.1 Single Agent Resource Allocation

The resource allocation literature generally classifies resources into two categories: renewable (or reusable) and consumable resources. A renewable resource is used during the execution of an action, and is released afterward. It is never destroyed. Typical examples are bandwidth and machines in shop scheduling. Renewable resource constraints limit what actions can be executed concurrently.

In contrast, a consumable resource or some quantity of it vanishes after an action uses it. Consumable resources are further classified into discrete (consumed in whole units, e.g., inventory items like nuts and bolts), continuous (consumed in any quantity, e.g., fuel), non-monotonic (can be replenished) and monotonic (decreasing or increasing

only). Plans that demand more resources than available cannot be implemented. Consumable resource constraints limit what plans are valid.

Temporal planning with resources makes sure that concurrent actions do not take up more renewable resources than are available at any time for every such resource, and that the entire execution process does not require more consumable resources than available. There are a variety of planners that have been developed to address the resource allocation problem. They can roughly be classified into two categories: optimization and satisfaction.

2.1.1 Optimization

Optimization of some objective function(s) usually requires integrating the planning and scheduling processes. Otherwise, the systems may be incapable of considering the interactions between choosing actions in planning and allocating resources in scheduling. This is particularly undesirable when the resource-related objectives are to be optimized.

One major approach to maximizing the utility under resource constraints is to cast the problem into a constrained Markov Decision Process (CMDP). In this framework, the resource constraints are bounds that are expressed as inequalities in addition to a utility function. The utility function can be the expected total cost or the expected average cost. There is a long history of work to attempt to solve CMDP problems under various assumptions. The most popular methodology is Linear Programming (LP). Altman provides a survey on the various techniques for solving CMDP problems [1].

The other approach is based on Operations Research (OR). A resource allocation problem is cast into the Integer Programming (IP) or Mixed Integer Programming (MIP)

framework [93]. A cost function is to be minimized subject to a system of linear equations. The popular solution methods are Simplex [22] and interior-point methods [60]. The advantage of using IP in AI planning is the incorporation of numerical constraints and objectives into the planning domain. Bockmayr and Dimopoulos described domain-dependent IP models for specific problem domains [10]. Kautz and Walser invented ILP-PLAN that reasons about action costs and resources [63]. Vossen and colleagues described the state-change technique to develop good domain-independent IP formulations for AI planning [115]. IP has been an active research area and a lot of work can be found in the International Conference on Automated Planning and Scheduling (ICAPS) proceedings.

Other methods include the approach of Ephrati and colleagues [38]. They apply A* search in the plan space to find the lowest cost plan, which can lead to a significant decrease in total planning time. The heuristic evaluation function is computed by a deep lookahead that calculates the cost of complete plans for a set of subgoals, under the (generally false) assumption that they do not interact. Williamson's work in PYRRHUS uses plan quality information to find an optimal plan [121]. PYRRHUS iteratively prunes from the search space any partial plans that are guaranteed to have a lower utility than the current plan until no partial plans remain, at which point the optimal plan is found.

EXCALIBUR by Nareyek uses a planning model based on the structural constraint satisfaction formulation [89, 90]. Structural constraints allow an agent to modify not just the instantiations of constraints in the plan (as in traditional constraint satisfaction problems), but the entire structure of the plan. Local search techniques are used to explore the constraints that define the planning problem. For each type of

resource constraint violation, there are heuristics to resolve it to improve the plan quality with respect to the global objectives.

2.1.2 Satisfaction

There are two major approaches to finding satisficing plans under resource constraints: constraint satisfaction and constraint programming (CP) techniques. One of the earliest works that casts the planning problem into the constraint satisfaction problem (CSP) framework is the work of Kautz and Selman. They introduced the idea of a state-based model with explanation closure axioms defined on state variables and no variables that explicitly model actions [61, 62].

CPlan by van Beek and Chen also demonstrated that the CSP approach to planning has advantages including the succinctness of models, and the robustness and speed with which plans can be found [112]. CPlan introduces a purely declarative representation of domain knowledge and is thus independent of any algorithm.

Using constraint programming as in parcPLAN, the resource allocation problem is often cast as a resource feasibility problem (RFP) [36]. A RFP is to determine whether there exists a feasible assignment of values to the variables in the resource requests by actions such that the temporal constraints are satisfied. The plans generated are often in the form of a non-optimal sequence of actions, where each action is given a start time and an end time or duration, i.e., an interval. Propagation of the constraints will result in, for example, shifts of the intervals to avoid resource contention.

Another CP-based planner is the O-Plan Resource Utilization Manager [33, 110]. It creates two separate profiles to represent the optimistic and pessimistic usages of resources within an activity plan. These profiles allow the planner to ensure that there is a

feasible assignment of resources available within any plan state being considered. For example, the optimistic profile is computed under the assumption that the start times occur as early as possible and the end times as late as possible. Resource shortages are detected when the optimistic profile falls below zero.

The IxTeT [46, 69] planner creates an interval graph from the temporal constraint network to search for actions that may potentially violate any resource constraint. A resolver is called to address each violation. A resolver is minimal if none of its disjuncts implies another. IxTeT employs a minimization procedure which removes the stronger constraint to maintain the overall least commitment search strategy. Examples of other constraint based planners that integrate time and resources are ASPEN [18], RAX-PS [58], and TP4 [51].

2.2 Multiagent Resource Allocation

As we can see from the last section, for resource allocation research in the case of a single agent, the focus has been on reasoning about the relationship between resource constraints and tasks. On the other hand, the resource allocation research in the multiagent case has been focusing on reasoning about the relationship between the agents involved and the relationship between their interactions and task assignments. There are two main approaches for allocating resources in multiagent environments: task allocation and coordination. In this section, we look at the major work in these two areas.

2.2.1 Task Allocation

Distributed task allocation research makes a fundamental assumption that a task can be solved by more than one agent. When an agent is incapable of accomplishing a

task alone, such as because it is overloaded, it can offload the task to other agents. Applications include job dispatching among machines in a manufacturing plant [113] and allocation of computational jobs among processors in a network [77].

A representative work is the Contract Net Protocol [106]. In Contract Net, the manager, which is an agent who wants a task to be accomplished, announces the task to other agents. The contractors, which are the agents who are capable of solving it, appraise and bid on the task. The manager evaluates the bids and selects a winner. After the winner finishes, the manager receives the result. If the task is complicated and thus beyond the ability of any single agent, the manager can decompose it into subtasks and award them to multiple agents. An agent can act as a manager for some tasks and a contractor for others simultaneously. In this way, the agents can negotiate to assign tasks among themselves to avoid being overloaded.

Another major work is market-based mechanisms. One important class is auction theory. Auction theory analyzes protocols and agents' strategies in auctions. An auction consists of an auctioneer and potential bidders. The auctioneer wants to subcontract out a task at the lowest possible price while the bidders, who can solve the task, want to receive the highest possible payment for doing so. Some popular auction protocols are: first-price open-cry, first-price sealed-bid, and second-price sealed-bid [56, 81, 100, 117]

In the first-price open-cry auction, also called the English auction, each bidder is free to raise his bid, and everyone knows the bid. When no one wants to raise the bid anymore, the highest bidder wins the item at the price of his bid. In the first-price sealed-bid auction, each bidder submits one bid without knowing the others' bids. The highest bidder wins the item at the price of his bid. In the second-price sealed-bid auction, also

called Vickrey auction, it is a sealed-bid auction in which the highest bidder wins the item at the price of the second highest bid. It is proven that, using Vickrey auction, a bidder's dominant strategy is to bid his true valuation of the item [114]. All these protocols allocate the auctioned item Pareto efficiently to the bidder who values it the most when the auctioneer always sells the item.

Moreover, Wellman and colleagues have studied the use of price-based search techniques to coordinate a set of agents by allowing them to buy and sell goods in order to maximize local utility. The price of a good implicitly conveys to an agent non-local information about the global utility of obtaining that good. It has been used effectively to structure distributed search in many multi-agent domains including multi-commodity flows [120] and supply chain management [118]. Cheng and Wellman have used general equilibrium theory to design a tatonnement algorithm that uses market prices to efficiently allocate goods and resources among agents [16]. Moreover, agents may be able reach global optimal solutions faster by exchanging more information beyond prices. Sandholm and Suri [101] discuss the need for non-price attributes and explicit constraints in conjunction with market protocols.

2.2.2 Coordination

Coordination techniques assume that a task requires resources from multiple agents. That is, some subset of agents needs to perform actions in concert to complete a task. The difficulty lies in assigning the correct resources to each task in a task set so that all tasks get their required resources. An incorrect allocation of resources to one task may result in a shortage of resources for some other tasks. Applications are found in distributed sensor networks [55], disaster rescue [65], and hospital scheduling [30].

For static domains that have no ambiguity, e.g., task sets do not change, some of these problems can be cast as centralized Constraint Satisfaction Problems (CSP) [42]. Formally, a CSP consists of n variables $x_1, x_2, \ldots, x_n$, whose values are taken from finite, discrete domains $D_1, D_2, \ldots, D_n$, respectively, and a set of constraints on their values. A constraint is defined as a predicate on the Cartesian product of the domains. The predicate is true if and only if the value assignment of these variables satisfies these constraints. Solving a CSP is equivalent to finding an assignment of values to all variables such that all constraints are satisfied [111].

In terms of distributed resource allocation, the variables are tasks; the domains are actions that the agents may perform. The agents need to schedule their actions given their resource constraints so that the tasks are completed successfully. There are a variety of algorithms for solving CSPs, such as filtering algorithms [119], consistency algorithms [43], and backtracking algorithms, heuristic revision [83], and decomposition techniques [27, 44].

Liu and Sycara have presented the Constraint Partition and Coordination Reaction approach, which is a framework to decompose a CSP into a set of subproblems based on constraint types and constraint connectivity, identify their interaction characteristics, and construct effective coordination mechanisms. It has been applied to job-shop scheduling problems to improve schedules [75]. Chia and colleagues have allowed agents to communicate heuristic domain information about their local jobs in the airport ground service scheduling problem [17]. Modi and colleagues have mapped the resource allocation problems to the Dynamic Distributed Constraint Satisfaction Problem [84].

For dynamic and ambiguous domains, e.g., task sets may change and agents' decisions may become obsolete, there are no general strategies. There are, however, algorithms that work well for specific applications. The distributed sensor network application is one that has gained a lot of attention. In this domain, multiple agents must deploy their sensors concurrently and collaboratively to track moving targets. Soh and Tsatsoulis have used a case-based reasoning approach where agents choose different negotiation strategies depending on the circumstances [107]. Other technologies applied to this problem include the TAEMS modeling language [28] and Design-to-Criteria plan scheduler [116].

A seminal work is the Partial Global Planning (PGP) [34]. It is a framework to coordinate multiple cooperative agents so that each agent can better utilize its local resources to generate better plans in response to changes and unexpected shocks in the environment. PGP integrates techniques from task-sharing, result-sharing, organization theory and planning to allow flexible reasoning of the agents' roles and responsibilities dynamically. The agents may modify their plans, e.g. the order of the major tasks, and communication actions to achieve better performance as a team. The basic idea behind PGP is that each agent constructs its local view of activities (a plan) that it intends to pursue and the relationships among these activities. The agents exchange relevant information about their plans to develop sufficient yet partial awareness of the global (i.e., partially global) views of the tasks.

2.3 Multiagent Constraint Satisfaction/Optimization

Resource allocation problems can also be cast as constraint satisfaction/optimization problems. The agents collaboratively search for satisficing plans

with respect to their resource constraints, or the (approximately) optimal plans if they are overly constrained. There is substantial research in finding satisficing or optimal solutions for multiagent systems subject to (resource) constraints. They are in general classified into Distributed Constraint Satisfaction Problems (DCSP) and Distributed Constraint Optimization Problems (DCOP). Example applications include transmission path restoration for dedicated circuits in a communication network [19], communication network path assignments [94], job shop scheduling [75], meeting scheduling [76], and supply chain management [15].

2.3.1 Distributed Constraint Satisfaction

DCSP is a distributed version of the CSP described in the Section 2.2.2. Each agent owns a subset of the variables. The domains of and constraints between variables belong to the agents owning the variables. In a resource allocation problem, the agents are coordinated such that their distributed actions do not violate the overall inter-agent constraints. The most well known method for solving DCSPs is the Asynchronous Backtracking algorithm (ABT) [125]. ABT says that each agent communicates its tentative value assignment to neighboring agents.[5] An agent changes its assignment if its current value assignment is not consistent with the assignments of the higher-priority agents. If the agent fails to find a consistent value, it generates a nogood. The nogood is communicated to the higher priority agents so they change their values.

There are many extensions to the basic version of ABT. For example, ABT does not say how to determine the priorities of agents. In [124], there is the asynchronous weak-commitment search. It is a method for dynamically ordering agents so that a bad

[5] It assumes that each agent owns one variable.

assignment can be revised without an exhaustive search. To escape from local minima faster, [85] introduces the breakout algorithm, which weights the violated constraints to indicate the agents who are most likely to improve the current solution.

While ABT is complete (it is guaranteed to find a solution if there is one), it fails to provide useful information if no solution exists. In many applications, especially when the agents are over-constrained, we would like to have some partial solutions, or even better, the optimal solutions given the constraints. There are a number of ways to define optimality. In Distributed Maximal CSP, the objective is to minimize the maximal number of violated constraints over agents [53].

In Distributed Hierarchical CSP [54], each constraint is labeled a positive integer called importance value, which represents an importance of the constraint, and a constraint with a larger importance value is considered more important. Agents try to find variable values that minimize the maximum importance value of violated constraints over agents. In other words, each agent tries to satisfy as many most important constraints as possible.

2.3.2 Distributed Constraint Optimization

Another major formulation is Distributed Constraint Optimization. A DCOP consists of n variables x_1, x_2, ..., x_n, each assigned to an agent, where the values of the variables are taken from finite, discrete domains, D_1, D_2, ..., D_n, respectively. Only the agent who is assigned a variable has control of its value and knowledge of its domain. The goal is to choose values for variables such that an objective function is minimized or maximized. An objective function is a function mapping from the Cartesian product of the domains to positive integers. In other words, instead of returning satisfied or

unsatisfied as in DCSP, the function returns a valuation. Schiex, Fargier and Verfaillie provided a classification of objective functions in [102].

One straightforward approach to DCOP has been to use iterative thresholding. The basic idea is to convert an optimization problem into a series of satisfaction problem by setting a threshold *a priori* and applying a DCSP algorithm. If no satisfactory solution is found, the threshold is iteratively lowered until a solution is found, or until no satisfactory solution is possible. This iterative thresholding method can find solutions within a user-specified distance from the optimal but cannot guarantee optimality in general.

Hirayama and Yokoo introduced the Synchronous Branch and Bound algorithm [53]. It simulates branch and bound search to update/decrease upper bounds during search in a distributed environment. It requires agents to perform computation in a sequential manner in which only one agent executes at a time. The order of execution is determined by a priority ordering. Lemaitre and Verfaille described another synchronous algorithm based on greedy repair search [72], but this algorithm is incomplete and requires a central agent to collect global state.

Modi and colleagues introduced the Asynchronous Distributed Optimization algorithm [84]. This algorithm is asynchronous, uses linear space, and is guaranteed to find optimal solutions or solutions within a user-specified distance from the optimal. Experimental results show that it is significantly more efficient than synchronous methods. The speedups are shown to be partly due to its distributed search strategy and partly due to the asynchrony of the algorithm. The algorithm works with objective

functions that are associative, commutative, monotonic aggregation operator defined over a totally ordered set of valuations, with minimum and maximum elements.

2.4 Real-Time Scheduling

The resource allocation problem is also studied in the real-time community in the context of scheduling. The purpose of a real-time scheduling algorithm is to ensure that critical timing constraints, such as deadlines and response times, are met. Standard scheduling algorithms include rate monotonic [74], earliest deadline first, buddy, first-fit, and best fit scheduling [66]. These algorithms have variants that work on single- and multi- processors, online and offline. Some can handle precedence constraints among the tasks.

However, these algorithms are not designed for planning agents. They, by and large, assume that a task set, once given, is fixed. There are no causal relations among the tasks. For example, accomplishing a task does not introduce a new task. In contrast, in planning an agent acting in a state usually enters a new state, hence faces a new task. Also, when there are not sufficient resources to schedule all tasks, real-time scheduling algorithms either drop or delay (in the case of non-critical missions that do not result in catastrophe) some tasks. Yet, in planning, the agent could instead backtrack to search for a different task set or a different plan [32], or could relax some constraints.

In general, real-time systems research addresses timeliness of tasks to assure safety, reliability, quality of output, predictability of performance, etc. It does not consider what the source of those tasks is. Real-time researchers in general assume that they are given tasks that have certain performance requirements, but the motivations for those tasks are left unspecified. AI research, on the other hand, studies the interactions

between an agent and the environment. AI planning searches for a set of actions/tasks that, if taken in the appropriate states, allows the agent to achieve its goals and/or maintain safety by preempting all dangerous events.

2.5 Resource Allocation for Real-Time Agents

Despite these research efforts, these methodologies are insufficient to address the resource allocation problem for a real-time controller/agent whether it is acting alone or in a shared environment. We are going to discuss the weaknesses of the work mentioned in the previous sections, and describe what we need in this dissertation to address them.

2.5.1 A Single Real-Time Agent

Very often, a real-time application requires the controller to repeatedly attend to a number of tasks for an indefinite amount of time. In the terminology of AI planning, when the agent does not keep track of time stamps, it will need to continuously traverse a cyclic state diagram. Or alternatively, when the agent keeps track of time stamps, it needs to plan for an infinite time horizon.

Many mathematical models such as CMDP, LP, and IP assume stationary transition probabilities. Yet, this assumption does not hold in a real-time application when the probability of failure happening in a state depends on how long the agent has stayed in the state. Also, to guarantee safety, we are concerned with the worst-case performance analysis in a real-time application. Yet, the objectives of the mathematical models are usually the expected or total cost analysis.

For CP-based planners, traditionally, their primary objective is to minimize plan length. As Nareyek has pointed out, this is a rather curious approach as this property is

usually irrelevant [89]. These planners usually require as inputs the bounds for the plan length or the number of actions. If a satisficing plan cannot be found, these bounds are expanded. In terms of optimization, an optimal plan can only be found when it lies within the initial bound. Determining a good bound is in general a very difficult task, if at all possible. Also, encoding a real-time problem in terms of variables and constraints may itself be a daunting task. While some modern SAT solvers are able to handle the simple example in [91], the encoding of this problem is still overwhelming as it requires 129,600 variables according to the encoding of CPlan.

The temporal interval planners are also inappropriate for real-time planning and scheduling purposes. First, they assume deterministic environments, where actions can always be successfully executed. However in a real-time domain, uncertainty often arises because of incompleteness and incorrectness in an agent's knowledge of the properties of the environmental dynamics. Hence, actions are no longer guaranteed to achieve their effects. An action may have multiple (probabilistic) effects depending on the concurrent environmental events and/or asynchronous activities of other agents.

Second, these temporal planners assume static environments, in which the only changes to the world are made by the actions of systems. They cannot reason about changes beyond their own actions. Yet, in many real world applications, the domains, e.g., aircraft domains [2], are dynamic. The world can change via events outside of an agent's control. The agent's actions can get interrupted by other events or simply fail for unknown reasons (incomplete knowledge of the designers). Dangers and hazards can happen unpredictably at any time. It is not realistic to foretell what actions the agent needs to execute at specific times or time intervals. Moreover, these planners are risk-

neutral as they are not designed to handle hard real-time constraints and catastrophic failures.

2.5.2 Multiple Real-Time Agents

Most multiagent systems research does not assume real-time agents, so these issues described in the last section about resource allocation for a real-time agent apply to the multiagent case as well. In terms of allocating resources in multiagent systems, many research efforts, such as those in Section 2.2, assume that an overloaded agent can share its tasks with other agents. The goal is to distribute the (fixed) task sets so that as many tasks can be accomplished as possible. The focus is on assigning tasks rather than on allocating local resources of the agents.

On the other hand, the problem we address in this dissertation focuses on the how agents allocate their local resources to decide which of the tasks they want to work on. There are two major differences. First, we do not assume that tasks can be passed among agents. For each task, there is an agent who is responsible for it. Second, we consider the source of those tasks. There are complicated dependency relationships among the tasks of different agents. For example, an agent needs to do some tasks only if other agents perform other tasks. So, the task sets of the agents are not fixed. Therefore, instead of studying which agent is responsible for which tasks, we look at the resource allocation problem in terms of what and how much information the agents need to know about each other to better utilize their local resources.

An important consideration in multiagent planning centers around how much an agent knows ahead of time about the other agents. At one extreme, it might know nothing. For approaches such as negotiation [104], plan merging [45], multiagent MDPs [11] and

(Generic) Partial Global Planning [34], agents must exchange plan information to begin to model each other. At the other extreme, coordination approaches, such as Social Laws [105], assume (in the offline case) that agents know everything they need to know about each other right from the beginning. Any plan an agent makes that adheres to the social laws is assured to be coordinated with the other law-abiding agents.

In the middle are approaches where agents have some knowledge (for example, the organizational roles of others) ahead of time, but need to exchange information to acquire more situation-specific details [29]. Our work resides in this middle ground, because we assume that our agents know everything about what others might do under different eventualities. But, in contrast with work such as [20] where subsequent communication is to add details to enlarge an agent's view of how the world might unfold, we instead view the purpose of communication as helping agents rule out some of the choices that they had anticipated others might make.

Furthermore, while we might possibly be able to take advantage of some of the algorithms described Section 2.3, e.g., the (distributed) constraint satisfaction/ optimization techniques, that would be very inefficient. First, the optimization algorithms theoretically find the optimal solutions so they are usually intractable. We are instead interested in designing algorithms that agents can practically use to find good solutions. Second, the number of possible plans that an agent can have is exponentially large. It is usually (much) bigger than the cardinality of the domains of variables in the applications where the DCSP/DCOP algorithms have been applied effectively. It is therefore formidable to treat agents as variables and their plans as possible value assignments.

Third and more importantly, while these algorithms are generic enough to capture and work with any CSPs, they could be inefficient compared to specialized algorithms. They cannot take advantage of domain knowledge which is inherent in planning problems. They cannot reason about structures that plans may have, e.g., subplans. In short, these generic algorithms are not designed for planning purposes.

2.5.3 Where Our Work Fills

In summary, a lot of prior work in resource allocation assumes a deterministic, static, and metric-time environment. For the mathematical models that can handle probabilistic events, they make assumptions, such as stationary transition probabilities, that do not apply to real-time environments. A lot of work in multiagent resource allocation focuses on task allocation. Some constraint satisfaction/optimization algorithms may apply but can be inefficient.

In this dissertation, we solve the resource allocation problem for an agent situated in a real-time environment where dynamic, probabilistic, time-dependent, and time-critical activities are possible. In a multiagent environment, we answer the problem in terms of how much information an agent needs so it can make more informed resource allocation decisions. We design data structures and algorithms for discovering, representing, and reasoning about the new knowledge for the agent to improve its existing plan efficiently.

2.6 Probabilistic Temporal Projection

Our solution to the above research problem in this dissertation involves reasoning about the temporal trajectory of an agent in a real-time environment. For a real-time

agent, this trajectory is non-deterministic. Therefore, we have built in CIRCA a framework that allows us to model the probabilistic temporal projection of external events as well as the agent's actions. Although this section and subsections (Sections 2.6.1 – 2.6.3) are not directly relevant to resource allocation, we survey the existing work in temporal projection because it is related to the probabilistic action and event representation we will develop, which an agent uses to make informed resource allocation decisions.

2.6.1 Proving Propositions over Time

Temporal projection, i.e., predicting future states of the world, is crucial to planning. An agent must be able to reason about the consequences of its actions to achieve goals or to evaluate the quality of a plan. One of the earliest papers on the temporal projection problem computes the consequences of a set of propositions or fluents or observations given a set of cause-and-effect relations referred to as causal rules [25].

Kanazawa has extended the theory to reason about change and time by applying Markov theory and survival analysis [23, 24, 59]. He presents a representational network, called a time net, to express knowledge of cause-and-effect relations in terms of conditional probabilities. The nodes in a time net are propositions and events at all time points. The links are the dependencies and causal rule instances. The probability of any proposition being true at any time point can be computed using a Bayesian network algorithm.

Unfortunately, it is very inefficient to calculate the probabilities of all propositions at all time points. First, it relies on the Bayesian network solving algorithm,

which is NP-hard [21]. Second, a time net usually requires a very large number of time points because the distance between each one is very small. Often in planning, only the probabilities at a few crucial times are necessary. In our case, we need to compute only the likelihood of ever visiting a state, i.e., the probability of a proposition *ever* being true.

Hanks addresses the inefficiency problem by introducing the concept of a probability threshold [48-50]. His method is still able to calculate the probabilities for any proposition at any time point. Given a probability threshold τ, the estimate, E, of the probability of a proposition at time t, is guaranteed to lie on the same side of the threshold as the actual value $P(\varphi_t)$. In other words, if $E < \tau$, then $P(\varphi_t) < \tau$; if $E > \tau$, then $P(\varphi_t) > \tau$.

The efficiency of Hanks' algorithm comes from its ability to ignore evidence. It ignores the evidence that confirms the current hypothesis. It also ignores the evidence that is too weak to change the current estimate with respect to the threshold because of the evidence's unreliability or remoteness in time. The algorithm only looks at the evidence that moves the current estimate to the other side of the threshold.

Hanks' algorithm is not guaranteed to converge. The estimate can swing wildly from one side to the other as more evidence is examined. Moreover, it assumes that multiple pieces of evidence combined will not change the estimate with respect to the threshold, when each piece of evidence, taken alone, will not be strong enough to revise the estimate. That is, the algorithm cannot handle combined pieces of evidence. In terms of planning, it will need to build a scenario tree, a temporal trace of the plan's execution. As the projection algorithm can only work on one proposition at a time, it has to be run for each scenario separately.

Throughout this thread of research by Dean, Kanazawa, Hanks, and McDermott, the expressivity of their models falls short for real-time planning purposes. Events and actions are assumed to be instantaneous. Effects are realized right at the next time step or immediately. The models do not provide semantics about the durations of events and the frequencies of actions. Besides, as mutually-exclusive activities are not possible, they cannot express the complex temporal relations and the multiple effects of having these activities concurrently.

2.6.2 Representing Probabilistic Actions

Kushmeric and colleagues built BURIDAN [68]. One of their contributions is that they have defined a symbolic action representation and provided it probabilistic semantics, allowing multiple probabilistic effects for the same action depending on the context. Specifically, an action is a set of consequences $\{\langle t_1, \rho_1, e_1\rangle,...,\langle t_n, \rho_n, e_n\rangle\}$. For each i, t_i is a precondition, called a trigger, on which each effect e_i depends. ρ_i is the probability that effect e_i will occur given that trigger t_i is satisfied. The triggers must be mutually-exclusive and exhaustive. Kushmeric describes how to compute the probability that a state will hold after executing a sequence of actions. While the computation is straightforward, it is extremely inefficient and does not scale up well at all as admitted by its inventors.

Furthermore, this representation of actions does not model events explicitly. Although it is possible to account for concurrent actions and events by enumerating all possible preconditions and effects, it is a daunting task for a user to specify all foreseeable interactions between *each* action and *all combinations* of events. It is also inadequate for real-time planning because of the lack of the ability to specify the timing

characteristics of actions, such as start times, end times, frequencies, and durations. In fact, the only allowed temporal relations for BURIDAN are "before" and "after."

Yampratoom has developed a much richer model of actions and events in terms of describing their interrelated temporal relations [123]. In his model, actions and events are not instantaneous; they span over intervals of time. Events are considered as exogenous actions. All actions, hence events, have multiple probabilistic effects as in Kushmerick's state-space operators. Yet an action's preconditions and effects can have more general temporal relations to the action. It is also possible to add more relations to the vocabularies as needs required by a planner arise. The model supports concurrent actions whether they are independent, have synergic effects, or interfere with each other.

Simulation is used to predict the probabilistic temporal projection of a plan encompassing these complex temporal relations. Given a partially ordered plan, the initial states of the world, and probabilistic causal rules relating actions to their associated preconditions and effects (including actions performed by exogenous events that the planner does not control), the simulator will randomly generate a specific scenario with a complete ordering of the planned actions, forward chain through time, and return the states of the world over the course of the simulation. Simulation indeed provides a simple way to compute temporal projections for complex scenarios, which are otherwise difficult (if at all possible) to compute analytically.

Nonetheless, it is not clear how a planner can take advantage of the rich semantics of this action representation *during* planning. Yampratoom has only briefly touched on the subject of planning and suggested ways such as plan refinement based on simulation results and planning with statistics [78-80]. For our purposes, we would instead want

operators that make possible searching the state space during planning, even at the cost of having simpler semantics.

There are a number of other action representations that also attach probabilities to multiple effects of an action, such as sequential effect trees (STs) [73] and 2 stage temporal Bayesian networks [12, 13]. In [73], it is shown that, together with Kushmerick's state-space operators, these representations are expressively equivalent, meaning that a planning problem specified in one representation can be converted to an equivalent planning problem in any of the other representations with at most a polynomial factor increase in the size of the resulting representation and the number of steps needed to reach the goal with sufficient probability. Moreover, a lot of these probabilistic planning problems can be formulated as Markov Decision Processes (MDPs).

2.6.3 Probabilistic Planning

Planners based on Markov Decision Processes (MDPs) incorporate representations of uncertainty of how actions may move the world among states. MDP planners generate plans (or policies) about what action (if any) an agent should perform in each (group of) state(s) to maximize the agent's expected performance. MDP planning [26] is most straightforward under assumptions such as: that states are fully observable (the agent executing the plan can know exactly what state it is in); that event models are implicit (the uncertainties of how events beyond the agent's control affect the states it can reach via an action it takes are folded into the state transition probabilities associated with the action); and that a transition probability is stationary (dependent only upon the state and action chosen but not upon timing associated with the state). In the terminology of

Boutilier, Dean and Hanks [14], these assumptions hold for a stationary, fully observable MDP with implicit event models.

Nonetheless, in a real-time application, not surprisingly, a core concern is "how long" something will take. In general, the transition probabilities of events and actions in a state depend how long they have been enabled *before* entering the state. For example, the longer an agent delays its response to preempt a failure, the more likely that the failure event will happen. The stochastic process does not satisfy the Markov property. Moreover, MDP planners require specification of transition probabilities of multiple effects of actions. Because of the prohibitively large space of combinations of actions, events, and frequencies for a real-time controller, it is unrealistic that a user can have all these transition probabilities *a priori*.

Therefore, the existing tools for projecting execution trajectory and for probabilistic planning are inadequate for real-time planning purposes (Section 2.6.1). In this dissertation, the action and event representation that we will develop captures the probabilistic effects of concurrent actions and events as in Kushmeric's state-space operator (Section 2.6.2). It also takes into account their temporal properties, i.e., start times, end times, frequencies, and durations. More importantly, our representation does not assume *a priori* knowledge of transition probabilities or stationary transition probabilities (Section 2.6.3). Using this representation, an agent can compute the transition probabilities for each combination of concurrent actions and events.

Chapter 3

The Cooperative Intelligent Real-Time Control Architecture

The Cooperative Intelligent Real-time Control Architecture (CIRCA) has been developed to support reasoning about resource constraints and about actions with temporal probabilistic effects. CIRCA selects, schedules, and executes recognition-reactions assuming a resource-limited execution module. While our algorithms developed in this dissertation are general enough to work with any type of limited real-time agent that schedules for periodic recognition-reactions, CIRCA is the platform on which we implemented all the algorithms and is the testbed on which we ran all the experiments. In this chapter, we describe the architecture, its state representation, and its planning objective.

There are two main subsystems in CIRCA, the Artificial Intelligence Subsystem (AIS) and the Real-Time Subsystem (RTS). The architecture is shown in Figure 3-1. The RTS executes the real-time control plans (see below) pre-computed by the AIS.[6] Inside the AIS are the probabilistic state-space planner and the scheduler. The planner generates a set of recognition-reactions, formally called Test-Action-Pairs (TAPs), by searching through the state space to determine the appropriate actions for states and their frequencies. The scheduler schedules the TAPs according to the real-time execution constraints.

[6] The AIS executes concurrently with the RTS, and is generating future control plans while the RTS is executing the current control plan.

3.1 CIRCA Control Plan

Traditional AI plan execution systems implicitly model information flow to an agent. They usually assume that information is available when the agent needs it; the agent is passive. In contrast, CIRCA explicitly models the effort and time necessary to actively gather information from the environment to determine what state or state group it is in. It has to look for various hazards frequently enough to allow room for reactions if any hazard is detected.

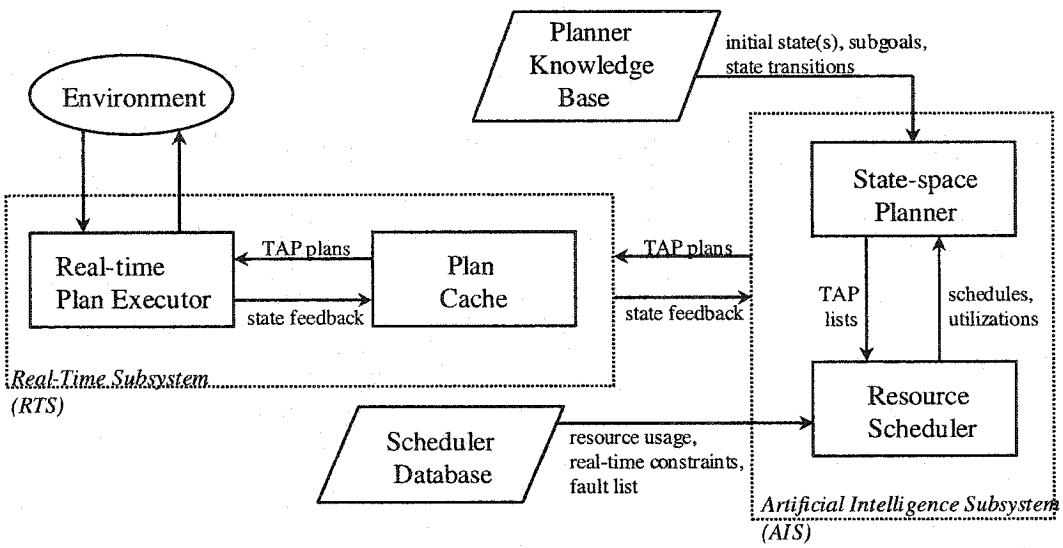


Figure 3-1: The CIRCA Architecture

The recognition test in a TAP is done by actively performing actions to collect data or monitoring for the relevant aspects of the world, e.g., looking for signs of hacking in a security system. A reaction is only executed if the world matches the state description in the corresponding recognition test. As the progress of the world is uncertain and dangerous events can happen sporadically, the RTS must check whether the reactions should be executed by continuously looping in a round-robin fashion over

all recognition tests frequently enough. If a recognition test is not satisfied, the RTS skips the reaction until the next round when it will decide again.

A CIRCA *control plan* or simply plan, composed of a scheduled set of TAPs, is therefore a cyclic (periodic) real-time schedule. Each TAP is considered a task unit. In this thesis, we will refer to a TAP as an action when it is not necessary to distinguish between a TAP (recognition and reaction) and its (re)action. An action thus has two parts conceptually. It has the primitives to gather information from the environment (recognition) and the primitives to influence/change the environment (reaction). The execution time of an action is thus the sum of the execution times to perform all these primitives. An example of such a plan is shown in Figure 3-2.[7]

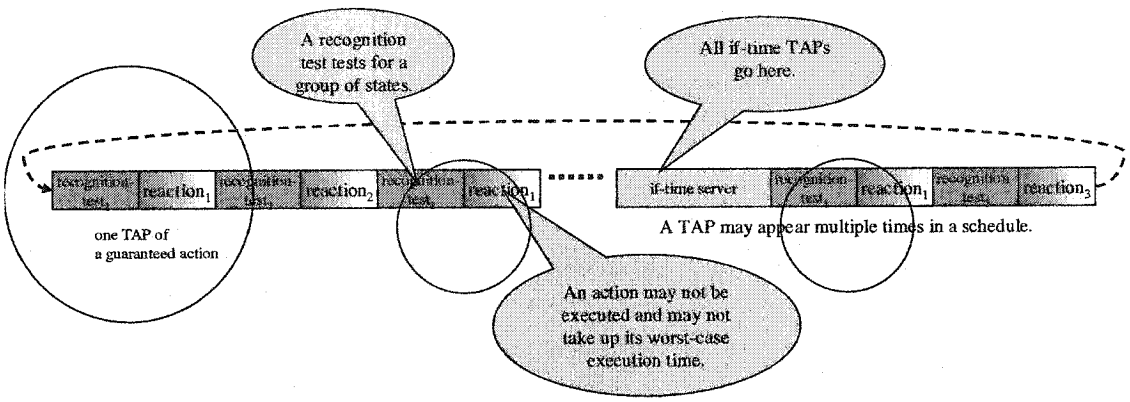


Figure 3-2: A CIRCA Control Plan

Each TAP in a schedule consumes or utilizes some resources of the agent. The *utilization* of a TAP is simply the sum of the testing and execution times of the TAP divided by its period.[8] When the sum of the utilizations of all TAPs exceeds 1, no schedule is possible. The agent is over-utilized because it has insufficient resources [66].

[7] The terminology in this figure will be defined in the next section.
[8] The period of a TAP is the frequency given to the TAP by the AIS during planning. We will go into detail in the next section.

Therefore, a CIRCA plan is different from a traditional AI plan. Unlike STRIPS-based planners [39], CIRCA does not generate a sequence of actions that leads the system from initial states to goal states. That kind of simple plans (without contingent plans) would only be appropriate for completely deterministic and static worlds, in which sensors are unnecessary and execution times are irrelevant. A CIRCA control plan is more similar to a MDP policy [13]. Both specify the actions to be taken in various states. Both assume uncertainty in actions and require sensors to find out the states that the systems are in. Yet, CIRCA further specifies the frequencies at which it needs to do sensing and information gathering to guarantee the timeliness of its actions to ensure all failures are preempted.

3.2 CIRCA State Diagram and Planning

The CIRCA state-space representation of the world is constructed from a set of state propositions, called state features, and actions and events, called *transitions*, included as part of the planner knowledge base (KB). A state consists of a set of state features which describe different aspects of the world. Each feature takes a value from its finite domain. A value of "undef" is a don't-care value. A state in a world model is created dynamically by applying a transition to its parent state.

There are two types of transitions. *Action transitions* are explicitly controlled by the plan executor in the RTS, and thus only occur when selected during planning. Events and natural processes outside the system's control are modeled as *temporal transitions*. They are further classified into either innocuous temporal transitions (labeled *tt*) or catastrophic temporal transitions leading to failure (labeled *ttf*). Each *ttf* is essentially a hazard. The agent is assumed to know what constitutes a failure for itself. For example,

the state labeled FAILURE in Figure 3-3 is a failure state. All transitions pointing to that state are *ttf*s. All other temporal transitions that are not hazards are innocuous.

When multiple transitions are applied or enabled in a state, only one of them will *fire*, meaning that the next state the agent reaches during execution is the child state of the fired transition. The multiple transitions in a state represent the concurrently-enabled, mutually-exclusive activities that can happen when the agent reaches that state. To date, CIRCA treats transitions independently, modeling concurrent activities only as a series of separate transitions. It does not model concurrent activities whose combined effects when taken in concert differ from their additive effects.

A temporal transition,[9] whether it is innocuous (*tt*) or catastrophic (*ttf*), is described by a precondition and a postcondition. A precondition is a set of features and their values. If they equal/match a subset of features in a state, then the transition is applicable in that state. A postcondition is the effect of a transition applied to a state, which is similar to the add and delete lists in STRIPS [39]. It specifies what values the features in the state change to. There is timing information associated with a temporal transition. It is described by a probability function. We will discuss this in Chapter 4.

An action transition, in addition to a precondition and a postcondition, is given a worst-case execution time (*wcet*). The worst-case execution time of an action is the longest possible time it takes for the action to complete its operation.

The primary responsibility of CIRCA is to avoid system failures. When there is a *ttf* in a state, CIRCA selects an action to preempt the failure. The planner will also choose its frequency, formally called a *period*, such that the action is tested (recognition) to decide whether it should be executed (reaction) frequently enough. The period is chosen

[9] When we refer to a temporal transition, we could also mean a temporal transition to failure.

so that the action is guaranteed to be executed (up to a certain probability threshold, ε) before the *ttf* fires. These actions are called *guaranteed actions*.

Alternatively, if no *ttf* is present, CIRCA selects the best action that gets the system closer to the goals. Those best actions are only executed when the guaranteed actions do not take up their worst-case execution times. In that case, the "if-time server" (Figure 3-2) is invoked and it picks one of those actions to perform [87]. They are thus are not guaranteed for real-time performance. They are called "*if-time*" actions, as opposed to "guaranteed" actions that preempt failures. During the entire plan execution, an if-time action may or may not be executed depending on whether the system has extra resources that are not used up by the guaranteed actions. If-time actions are assumed not to incur schedule utilization because they are only opportunistically done in situations (if they ever arise) with slack resources because guaranteed actions do not require their worst-case resource usages.

There is another type of action, called a *reliable action* that is also scheduled with a real-time guarantee and thus utilizes resources. However, a reliable action does not preempt any explicit failures. A reliable action is an action that the agent must guarantee execution of if it is planned in a state, e.g., necessary for goal achievement.

In Figure 3-3, ac_1 is a guaranteed action preempting a *ttf*; ac_2 is a reliable action; ac_3 is an if-time action. Note that while both ac_2 and ac_3 move the agent closer to the goal state, only ac_2 is guaranteed to be executed when its recognition test returns true. ac_3 may not be executed at all. Whether an action is reliable or if-time is determined by the agent's planning algorithm.

This thesis discusses only guaranteed and reliable actions because they are the actions that consume resources in a schedule. Our objective is that an agent can make informed decisions about what guaranteed and reliable actions it should select in the plan so that the actions can fit in a schedule.

A design goal of CIRCA's State-Space Planner is that it uses a compact representation of the information needed to generate real-time control plans. Specifically, CIRCA does not generate more than one state for each unique combination of features and values even if it can be reached via multiple paths from the initial states. That is, no two states in a state diagram have the same set of feature values. Cycles are allowed in a state diagram. For example, the two deadend states in Figure 3-3 form a cycle.

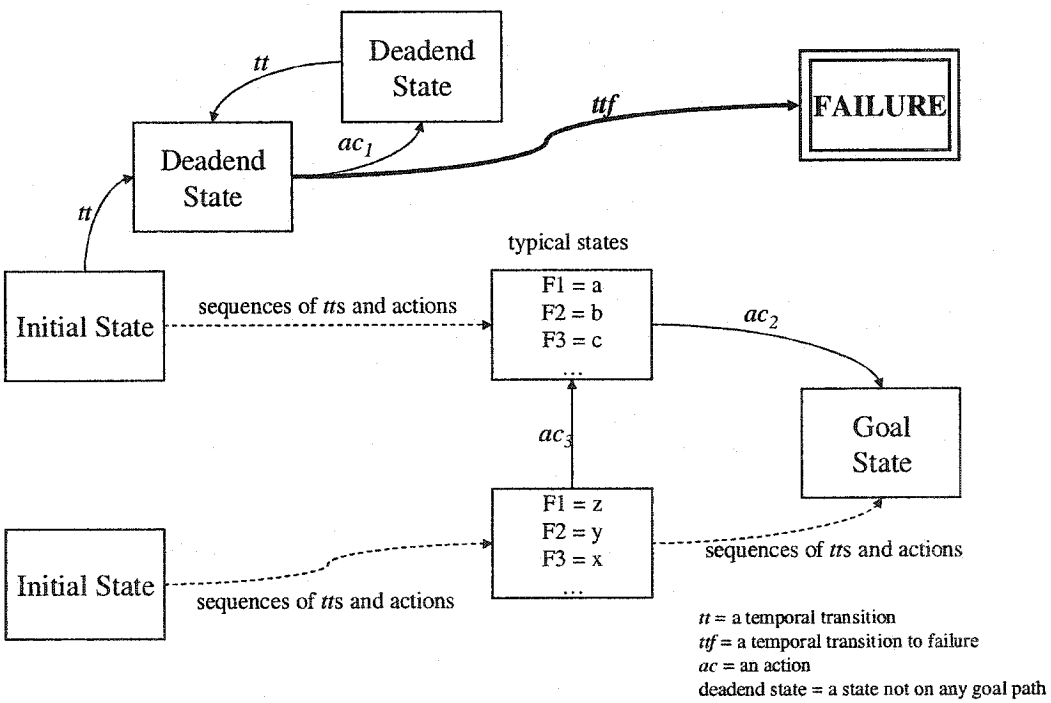


Figure 3-3: A Typical CIRCA State Diagram for Planning

Essentially, the RTS does not keep track of history. It executes a single action for a group of situations with the same features and values (which are represented by only one state in the state diagram), regardless of how and when that state is reached. It is precisely this characteristic that allows CIRCA to formulate a cyclic schedule in which actions are planned for states so that undesirable transitions cannot occur no matter when those states are encountered.

Unlike many other probabilistic planners, especially MDP planners, CIRCA does not assign rewards to states. CIRCA is similar to a classical AI planner, e.g., UCPOP [95], where the objective is goal achievement, safety maintenance, or failure avoidance. Yet it is situated in a probabilistic setting, where goals can only be achieved probabilistically and safety can only be given a probabilistic guarantee. Moreover, failure states do not have different degrees of failure and goal states do not have different degrees of success. Thus, all that matters is the probability of reaching each.

CIRCA primarily attempts to minimize the probability of failure. Its utility function is therefore simply:

$$U = \overline{F} = (1 - F) \qquad \textbf{eq. 3-1}$$

F is the failure probability. $\overline{F}$ is the safety probability. For some other applications, CIRCA tries to achieve goals on top of maintaining safety by inserting reliable actions in the schedules (consuming schedule utilization) and/or if-time actions in the if-time server (consuming no schedule utilization). In these cases, the utility function is defined using the Cobb-Douglas function [7]:

$$U = \overline{F}^{\alpha} G^{(1-\alpha)} \qquad \textbf{eq. 3-2}$$

G is the probability of reaching a goal. α ranges from 0 to 1. This utility function is concave, and thus satisfies most of the common properties of a utility function, such as diminishing return for each of the factors, $\overline{F}$ and G. Moreover, the marginal utility of one factor depends on the value of the other. For instance, the marginal utility is higher if the goal probability (G) can be increased at a bigger safety probability ($\overline{F}$). For CIRCA, the primary task is to maintain safety and hence α is big. When α equals 1, eq. 3-2 degenerates to eq. 3-1. In this case, we maximize only safety and are indifferent about reaching a goal. Please note that this does not mean that the goal probability is 0.

Ideally, when an agent is modifying its unschedulable plan to make it schedulable, the agent should account for the changes in both the failure and goal probabilities to try to minimize the decrements in utility. In this thesis, however, our heuristics consider only the changes in failure probability. In other words, we try to make to an unschedulable plan the changes that are expected to be the least detrimental to the agent's safety heuristically. So, our algorithms do not try to maximize the utility function. Neither does it guarantee that the final safety probability is the optimal.

Therefore, our primary measure of the quality of the schedulable plan produced is its safety probability. Since as a side effect our algorithms can change the goal probability, we will also compare the goal probability of the schedulable plan to the goal probability of the initial plan. In general, we desire the failure probability as small as possible and the goal probability as close to the initial goal probability as possible.

Chapter 4

Resource Allocation for a Real-Time Controller

For a real-time agent, we assume that its primary task is to preempt hazards leading to failures. Not all hazards, hence failures, are equally likely to occur. In cases where it is impossible to guarantee real-time performance to all preempting actions due to insufficient resources, our strategy to generate a schedulable plan is to find the most important subset of the actions that is schedulable, such that the utility of the final plan is maintained to the extent possible. In other words, we try to remove from the preferred plan those actions that have the least impact to the utility until what is left is schedulable.

To determine which actions to drop and which to schedule, we prioritize them by their probabilities of actually being executed during run-time. The intuition is that if all failures are equally bad, ignoring the hazards corresponding to situations which the agent is least likely to encounter does the least harm. This is a greedy strategy that minimally raises the failure probability for each action dropped. This strategy does not usually find the optimal plans because it does not consider all possible combinations of actions to be dropped.

If some failures are worse than others, our strategy can be easily modified to ignore the least catastrophic hazards weighted by their probabilities. This extension is described in Section 4.10. In either case, to reason about probabilistic guarantee, we need a probability model of the interactions between the exogenous (environmental) events and actions of an agent.

However, determining the likelihood of a real-time agent (such as a driver) encountering a particular situation (state) can be challenging because the likelihood is

dependent not only on the actions the agent performs (what to do if a driver is being tailgated), but also on its choices of how frequently it checks whether to perform them (how often the driver looks in the mirrors). By definition, a dynamic environment is one in which the state can change via events outside of the agent's control. In general, the sooner an agent detects and responds to a situation, the less opportunity there is for the environmental dynamics to intervene, and the higher the chance the agent is going to preempt failure. In the context of events that can lead to catastrophes, this simply means that the agent should (re)act fast enough to prevent disastrous events from happening.

The probability of encountering a state depends on the complex temporal coupling between the plan of an agent and the exogenous events. The probabilistic temporal projection of a real-time agent – what states the agent is in over time – is thus a complex stochastic process, which can even be non-Markovian.

This chapter begins with how (Sections 4.1 – 4.4) we model the interactions between external events and an agent's actions and why (Sections 4.5, 0). Then we describe how we compute the probabilistic temporal trajectory of the agent (Section 4.7) if the stochastic process that the agent undergoes is Markovian. In Sections 4.8 and 4.9, we describe how an agent approximates its probabilistic temporal trajectory using simulation if the stochastic process is non-Markovian. Using the probability information, the agent makes resource allocation decisions in case of insufficient resources to schedule all actions (Sections 4.10 – 4.12). Finally, we conclude this chapter by evaluating how well our strategy works in some randomly generated sample domains (Section 4.13).

4.1 Real-Time Execution as a Continuous Stochastic Process

Depending on the combination of the temporal and action transitions applicable in a state *and* the period of the action chosen (CIRCA chooses at most one action to take in a state), the transition probabilities for the temporal transitions and action vary. We model the probabilistic temporal effect of a transition by a probability function. A probability function can take the form of a cumulative probability function, a probability density function, or what we call a probability rate function (Section 4.3). It describes the probability of a transition happening as a function of the time since it was enabled, independently of any other transitions. These transitions are called temporal transitions because the transition probability of a transition firing in a state (transitioning out of the state via that transition) changes with how long the transition has been enabled.

Specifically, let T be the random variable denoting the time that a transition occurs after it was enabled.[10] $f(t)$ is the probability density function of T. t is the transition time, ranging from 0 to infinity. Equivalently, we can use $F(t)$, the cumulative probability function. Many realistic processes can be (and should be!) modeled this way. We provide a few examples.

Suppose a machine has two states: up or down. The transition probability of going from the up state to the down state can be described by a probability density function because the time the machine goes down is non-deterministic. The same is true for the other transition of going from the down state to the up state. The repair time is also non-deterministic.[11] Let $X(t)$ be the state of the machine at time t. Then $\{X(t), t \geq 0\}$ is a

[10] This chapter has a number of notations. The important symbols are summarized in Appendix B.

[11] Although we may have an estimate of when the machine will be repaired, the estimate is not the time *at* which the repair is guaranteed to be completed. The finish time usually lies on both sides of the estimate,

continuous-time stochastic process with state space {up, down} and the two transitions with time-varying transition probabilities.

A multitasking computer can execute a maximum of K programs simultaneously. Let $X(t)$ be the number of jobs running on such a multitasking computer at time t. Then $\{X(t), t \geq 0\}$ is a continuous-time stochastic process with state space {0, 1, 2, ... , K}. The transitions to go from one state to another can be described by probability density functions, because the time a new job arrives and the time a job is finished are non-deterministic.

After a fighter jet fires a missile at a bomber plane, the probability of the bomber getting hit changes over time. The bomber has zero probability of getting hit before a certain time because it takes some time for the missile to travel to the bomber. Then the probability of getting hit rises. After a while, the probability goes to zero again because the missile probably has missed if the bomber is not down by then.

4.1.1 The CIRCA Probability Model

Here we describe how we model the execution of a real-time agent as a continuous-time stochastic process. We assume that all transitions are mutually independent. After numbering the transitions (*ac/tt/ttf*) in a state, we denote the i-th transition by tt_i (it could be a temporal transition or an action transition). A user specifies its probability density function $f_i(t)$ (or the cumulative probability function $F_i(t)$, if we assume that all Fs are differentiable). t is the transition time, ranging from 0 to infinity. Although the user needs to specify either the density function or the cumulative

---

centering on it. In this case, the probability density function of the repair transition looks similar to a bell curve, called a normal or Gaussian distribution.

probability function, we assume that both are readily available for computation. We also denote $(1-F)$ by $\overline{F}$.

Given this modeling of the set of applicable transitions in a state *s*, we can apply basic stochastic process theory. In the remainder of this section, we will compute the probabilistic distribution of the duration the agent stays in *s*, and the expected duration, called the sojourn time of *s*. More importantly, we compute also the transition probabilities of the transitions in *s*.

Let T be the transition time of state *s*, i.e., the earliest time that any transition fires in the state. T is a random variable. Let T_i be the firing time of tt_i, i.e., the time that the agent leaves the state via tt_i. T_i is also a random variable. The transition time out of the state is the minimum of all transition times, i.e., $T = \min\{T_1, T_2, ..., T_n\}$.[12] The cumulative probability function for the sojourn time of state *s*, $F^s_{sojourn}(t)$, is:

$$\begin{aligned} F^s{}_{sojourn}(t) &= P(T > t) \\ &= P(\min\{T_1, T_2, ..., T_n\} > t) \\ &= P(T_1 > t, T_2 > t, ..., T_n > t) \\ &= P(T_1 > t)P(T_2 > t)...P(T_n > t) \\ &= (1 - F_1(t))(1 - F_2(t))...(1 - F_n(t)) \\ &= \overline{F}_1(t)\overline{F}_2(t)...\overline{F}_n(t) \end{aligned} \qquad \textbf{eq. 4-1}$$

$P(.)$ is the operator that indicates the probability of an event. Thus, $P(T > t)$ is the probability that the transition time T is greater than t. From the third step to the forth step, the independence assumption is used.

The expected sojourn time of state *s*, i.e., the expected amount of time that the agent spends in *s*, is simply:

[12] As all T_i 's are random variables, T is also a random variable.

$$w_s = \int_0^\infty t f^s{}_{sojourn}(t)dt \qquad \text{eq. 4-2}$$

$f^s{}_{sojourn}(t)$ is the probability density function of the sojourn time, which is the derivative of $F^s{}_{sojourn}(t)$.

Let us assume for now that all the clocks associated with the transitions are reset to zero after the agent transitions into a new state (the clock of a transition in a state starts counting from the time that the agent enters the state). The transition probability of tt_i, relative to all other applicable temporal transitions in a state, is the probability that its firing time, T_i, is equal to the minimum, T, of all transition times. Thus, the transition probability is:

$$\begin{aligned}
&P(T_i = T) = P(T_i = \min\{T_1,...,T_n\}) \\
&= \int_0^\infty P(T_i = \min\{T_1,...,T_{i-1},T_{i+1},...,T_n\} \mid T_i = t)P(T_i = t)dt \\
&= \int_0^\infty P(T_1 > t,...,T_{i-1} > t, T_{i+1} > t,...,T_n > t)P(T_i = t)dt \\
&= \int_0^\infty P(T_1 > t)...P(T_{i-1} > t)P(T_{i+1} > t)...P(T_n > t)P(T_i = t)dt \\
&= \int_0^\infty \overline{F}_1(t)...\overline{F}_{i-1}(t)\overline{F}_{i+1}(t)...\overline{F}_n(t)f_i(t)dt
\end{aligned} \qquad \text{eq. 4-3}$$

With transition probabilities computed using eq. 4-3 (and sojourn times computed using eq. 4-2), our framework and assumptions satisfy the continuous-time semi-Markov Chain (SMC) property. A stochastic process has the Markov property at every transition epoch S_n, i.e., the evolution of the process from time $t = S_n$ onward depends on the history of the process up to time S_n only via the state it is in at time S_n. Intuitively, it

means that the properties of a stochastic process an agent undergoes at time t in state s depends only on the history since the agent entered states s. The history before the agent entered s is irrelevant to what happens at time t and after. We postpone discussing the case when the stochastic process is not semi-Markov until Sections 4.8 and 4.9.

4.2 The Problem of Discretization

In most cases, it is very difficult to compute transition probabilities using eq. 4-3, because it is an integral of a product of a number of functions from time 0 to infinity. Moreover, it requires a user to specify complete knowledge of a transition at each time point. This is an intimidating task if at all possible! An arbitrary continuous function cannot be represented by a finite machine because, in principle, an infinitely many arbitrary numbers must be encoded. Therefore, we need an efficient way to specify the probability functions in a knowledge base and, more importantly, to compute transition probabilities.

4.2.1 An Example of Using Discretization

Some sort of approximation is mandatory. An obvious alternative is to use probability mass functions instead of probability density functions, i.e., using a discrete approximation. The discrete counterpart of eq. 4-3 using probability mass functions is:

$$P(T_i = T) = \sum_{\xi=1}^{\infty} \overline{F}_1(\xi)...\overline{F}_{i-1}(\xi)\overline{F}_{i+1}(\xi)...\overline{F}_n(\xi)P_i(\xi) \qquad \text{eq. 4-4}$$

$P_i(\xi)$ is the probability mass of tt_i firing in time interval ξ, which ranges from 1 to infinity. $\overline{F}_j(\xi)$ is the probability that tt_j has not fired before and will not fire in time

interval ξ. Unfortunately, eq. 4-4 suffers from what we call the "missing probability" or "simultaneity probability" problem, as illustrated by the following example. Let us assume there are two *tt*s in a state and their probability mass functions are as follows.

	time interval 1	time interval 2
tt_1	0.3	0.7
tt_2	0.2	0.8

According to eq. 4-4, the transition probability of tt_1 is 0.3*(1-0.2) + 0.7*(1-0.2-0.8) = 0.24 and that of tt_2 is 0.2*(1-0.3) + 0.8*(1-0.3-0.7) = 0.14. The sum is only (0.24+0.14) = 0.38. However, since either of them is guaranteed to fire in either time interval 1 or 2, the sum of the transition probabilities should be 1.

This missing probability problem is caused by the fact that simultaneity becomes problematic when a discrete approximation is used. Given a particular discrete representation, such as a probability mass function, the probability of more than one transition firing in the same discrete time interval is not zero (if none of the probability masses is zero). This problem is non-existent if we use eq. 4-3, the continuous formulation, because the probability of more than one transition firing simultaneously at a time point is zero.

In fact, we can compute the missing probability for the example above. The probability for both tt_1 and tt_2 firing "simultaneously" in time interval 1 is 0.3*0.2 = 0.06, and the probability for both tt_1 and tt_2 firing "simultaneously" in time interval 2 is 0.7*0.8 = 0.56. If we account for all scenarios, then the probabilities, 0.24, 0.14, 0.06, 0.56 sum up to 1.

Given CIRCA's state space representation (Figure 3-3), if we ignore missing probability, it is equivalent to saying that there is a certain probability, which equals the

missing probability, that the system stays in a state. This is certainly unacceptable. For the example above, it means that the agent has a 62% chance of staying in the state, while in truth any of the two transitions will certainly fire.

An alternative is to distribute this missing probability to the transition probabilities, such as distributing it proportionally. For instance, we may add (0.06 + 0.56) * (0.24 / (0.24 + 0.14) = 0.392 to the transition probability of tt_1. Similarly, we add (0.06 + 0.56) * (0.14 / (0.24 + 0.14) = 0.228 to the transition probability of tt_2. However, depending on what we assume about the underlying continuous processes that give rise to the discrete probabilities, distributing the missing probability proportionally may not make sense, as we will see.

4.2.2 Losing Information in Discretization

Before we describe how CIRCA currently handles the missing probability problem, we need to realize that as long as we are doing discretization, some information is lost. It is impossible to compute the "actual" transition probabilities based on incomplete data. For instance, consider where there are two *tt*s that both have probabilities of 1 to occur in a unit interval.[13] A good guess of the transition probabilities is 0.5 for each of the transitions. However, depending on the actual underlying continuous processes, the transition probabilities can vary considerably. We show two extreme cases below.

Figure 4-1 shows a set of probability density functions of two *tt*s. Both transitions have probability 1 to fire before time 1. In this case, the transition probability of tt_1 is 0, while that of tt_2 is 1. On the other hand, Figure 4-2 shows another set of probability

[13] The author just uses 1 to illustrate an obvious example, but other numbers work just as well.

density functions. In this case, the transition probability of tt_1 is 1, while that of tt_2 is 0. The reader can easily construct a variety of other scenarios that give any transition probabilities to both *tt*s by shifting the functions.

Although any methods that operate on discretized data are to some extent estimations, some are better than others. In the previous example, if we had chosen a discretization for which the probability density functions do not fluctuate too much within intervals, we would have gotten a better estimation. In fact, if intervals of length 0.5 instead of unit intervals had been used, we could have accurately computed the transition probabilities simply using eq. 4-4. In general, the finer the discretization is, the more accurate the results we can compute (because there is more information).

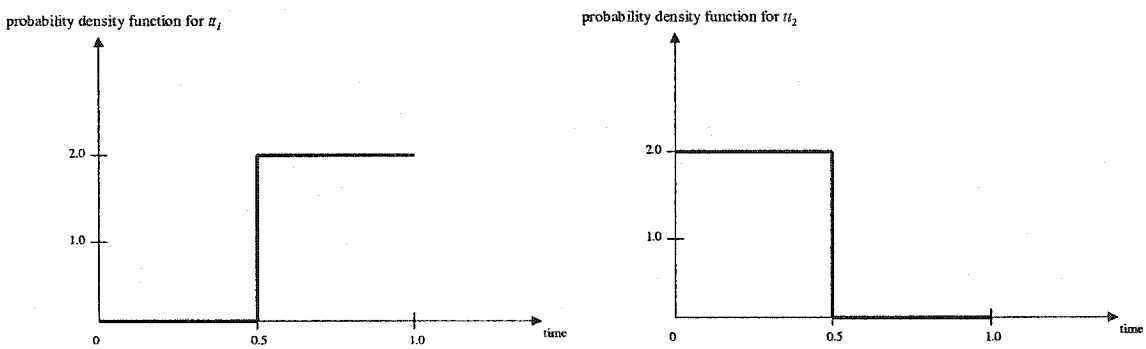


Figure 4-1: The Continuous Probability Density Functions of tt_1 and tt_2 (E.g., 1)

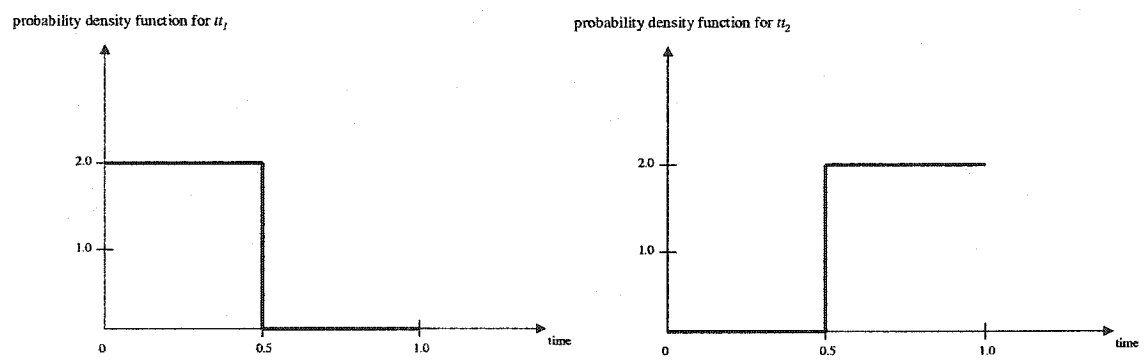


Figure 4-2: The Continuous Probability Density Functions of tt_1 and tt_2 (E.g., 2)

In the following, we first describe an alternative way to specify temporal probabilities (Section 4.3). Then we describe our heuristic to compute transition probabilities (Section 4.4). The justification is given in Sections 4.5 and 4.6.

4.3 Probability Rate Function

To address this discretization problem, we *estimate* the relative likelihood of each transition when multiple of them fire in the same time interval. To do so, we represent their time-dependent probabilities by what we call probability rate functions. For a temporal transition *tt*, a user specifies a *probability rate function* $\tilde{r}(h)$, $h \geq 1$,[14] over the h-th time interval $[t_h, t_{h+1})$. $\tilde{r}(h)$ is the probability that transition *tt* fires in the h-th time interval, given that the transition has not fired before t_h in any of the previous h-1 time intervals. In other words, it is the probability *per time interval*, hence the name "rate."

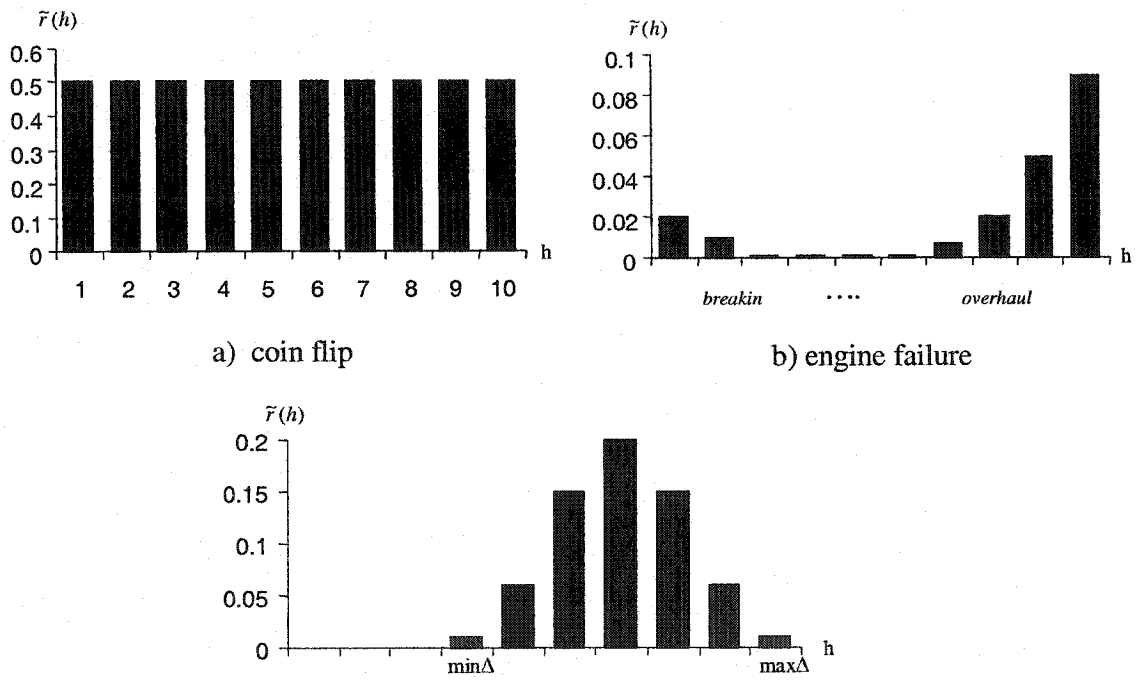
For example, if a fair coin is flipped once per second, the probability rate function for this transition from "heads" to "tails" has a constant value of 0.5 (50%) over each second, regardless of how much time has passed, given that the state is still "heads" after the flips so far. Figure 4-3a shows the probability rate function for this coin flip example. Figure 4-3b illustrates when an engine is first put into service. Its "failure rate" at first decreases during a break-in period. Afterward, the failure rate is very small during the normal operation period. When the engine nears the end of its life, the failure rate increases until the engine is considered "unsafe" and must be retired.

Figure 4-3c shows a probability rate function for a temporal transition to failure, e.g., getting shot down by a missile as a function of time since the missile was fired. The

[14] The $\tilde{r}$ notation in a discrete domain distinguishes itself from its continuous counterpart r in Appendix A.

transition has a minimal delay (*min*Δ) after which it is possible and a maximal delay (*max*Δ) after which the transition cannot occur (c.f., the third example in Section 4.1). CIRCA schedules an action to preempt this *ttf* to be executed before *min*Δ whenever the agent reaches the state in which the *ttf* is enabled.

Compared to probability density functions, it is often more intuitive and natural for engineers to use probability rate functions in some domains. For the example in Figure 4-3b, people are probably going to be better at saying, "Well, if you have an engine that is 5 years old and hasn't broken down yet, then I'd say you have an x% chance that it will break down within the next year."



a) coin flip

b) engine failure

c) a preemptible ttf

Figure 4-3: Discrete Probability Rate Functions

A probability rate has also been called a hazard rate since Barlow [6] first used this term. In [57] it is referred to as the conditional failure density function. Probability hazard functions are most commonly used in reliability theory, reliability applications,

and survival analysis [70, 92]. Their uses in biomedical research applications can be found in [71]. In [31] their engineering applications are discussed.

We do not want to call them probability hazard functions or probability hazard rate functions because of the stigma associated with hazards. In our case, while they can portray the temporal probabilities of transitions to failures (hazards) for an agent, they also model other transitions and actions that can potentially be beneficial. Therefore, we simply call them probability rate functions.

Actions, like temporal transitions, can also be described by probability rate functions. A guaranteed action is scheduled with a period such that it can preempt the corresponding hazard whenever the hazard is detected. Whenever the hazard appears, the action will always be examined and executed before the *ttf* has a chance to fire. In other words, if there is hazard in a state, the probability of the action being executed in the state increases until, in the last interval, it is guaranteed to be executed with probability 1 to preempt the *ttf*. Eq. 4-5 and Figure 4-4 show the probability rate function we adopt for guaranteed actions.

$$\tilde{r}(h) = \begin{cases} \dfrac{1}{(period - h + 1)} & ,1 \le h \le period \\ 0 & ,h > period \end{cases} \qquad \text{eq. 4-5}$$

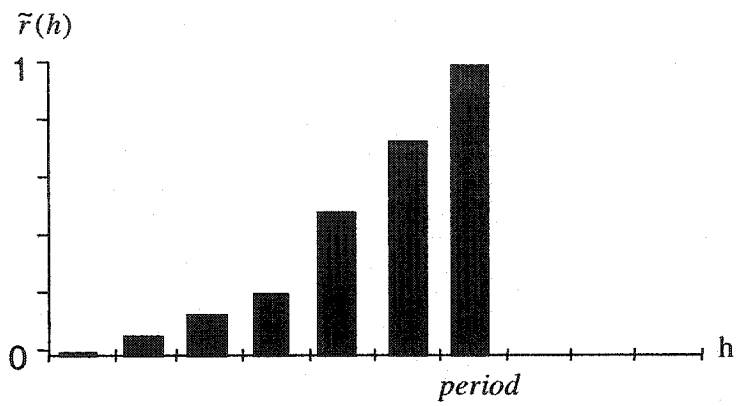


Figure 4-4: The Probability Rate Function of an Action Transition

As we can see from eq. 4-3 and eq. 4-5, the smaller the period is, the more frequently the action (or TAP) monitors for the aspects of the world for events that herald a transition to be avoided, and the faster the (re)action is executed to preempt the *ttf*. Our explicit action and event temporal representation allows CIRCA to select actions *as well as to determine their periods* so that the probabilities of failure transitions occurring are below some pre-specified threshold.

4.4 Computing Transition Probability

We are now ready to discuss the heuristic that we use to *estimate* transition probabilities from probability rate functions. The reader is reminded that calculating transition probabilities in a continuous domain using eq. 4-3 is extremely computationally expensive, if at all possible. On the other hand, there is the missing probability problem in a discrete domain. Thus, we need a heuristic to estimate the relative likelihoods of various transitions if they are to fire in the same time interval.

A probability rate function, like a probability mass/density/cumulative function, captures the probability of a transition occurring over time independently of other transitions. To model the dependency among a set of concurrent events and actions that match a state s – to calculate their transition probabilities – we need to compute the dependent probability rate function $\tilde{r}_i(h,s)$ for each state-specific transition[15] $trans_i$ (either a temporal transition or an action) in the state in each time interval $[t_h, t_{h+1})$, where h ranges from 1 to infinity. A *dependent probability rate function* $\tilde{r}_i(h,s)$ of a transition in state s describes the probabilistic temporal dynamic of a transition in that state when

[15] A state-specific transition is an applied temporal transition (event or action) in a state.

there are other concurrent applicable transitions. If there are no other concurrent transitions in the state, then we have:

$$\tilde{r}_i(h,s) = \tilde{r}_i(h) \qquad \textbf{eq. 4-6}$$

The reader should distinguish between an independent probability rate function $\tilde{r}_i(h)$ and a state-specific dependent probability rate function $\tilde{r}_i(h,s)$. We compute $\tilde{r}_i(h,s)$ by eq. 4-7:[16]

$$\tilde{r}_i(h,s) = \frac{\ln\left(1-\tilde{r}_i(h)\right)\left(1-P_{NONE}(h,s)\right)}{\sum_{\forall trans_j \in trans(s)} \ln\left(1-\tilde{r}_j(h)\right)} \qquad \textbf{eq. 4-7}$$

$trans(s)$ is the set of applicable transitions $\{trans_j\}$ in state s. $P_{NONE}(h,s)$ is the probability that no transition fires in the h-th time interval in state s. It is given by:

$$P_{NONE}(h,s) = \prod_{\forall trans_j \in trans(s)} \left(1-\tilde{r}_j(h)\right) \qquad \textbf{eq. 4-8}$$

The logarithm heuristic in eq. 4-7 is saying that the dependent probability rate in the h-th time interval $[t_h, t_{h+1})$ of a transition in state s is proportional to the relative logarithm of one minus its likelihood among those of all transitions, given that one of the transitions in the state must fire in the interval. Eq. 4-8 assumes that the transitions are mutually independent.

We can construct the cumulative probability function $F_i(h,s)$ from the dependent probability rates for a state-specific transition *trans*$_i$:

$$F_i(h,s) = \sum_{\eta=1}^{h} \tilde{r}_i(\eta,s)\,\overline{F}(\eta,s) \qquad \textbf{eq. 4-9}$$

[16] Eq. 4-7 is the only heuristic in this section. All other equations are basic probability theory.

$\overline{F}(\lambda, s)$ is the probability that the stochastic process is still in state s after t_λ. It equals the probability that no transition has fired before t_λ. It is computed recursively using eq. 4-10.

$$\begin{aligned}
&\overline{F}(h,s) = \overline{F}(h-1,s) P_{NONE}(h-1,s) \\
&= \prod_{\eta=0}^{h-1} P_{NONE}(\eta, s) \\
&P_{NONE}(0,s) = 1 \\
&\overline{F}(0,s) = 1
\end{aligned} \qquad \textbf{eq. 4-10}$$

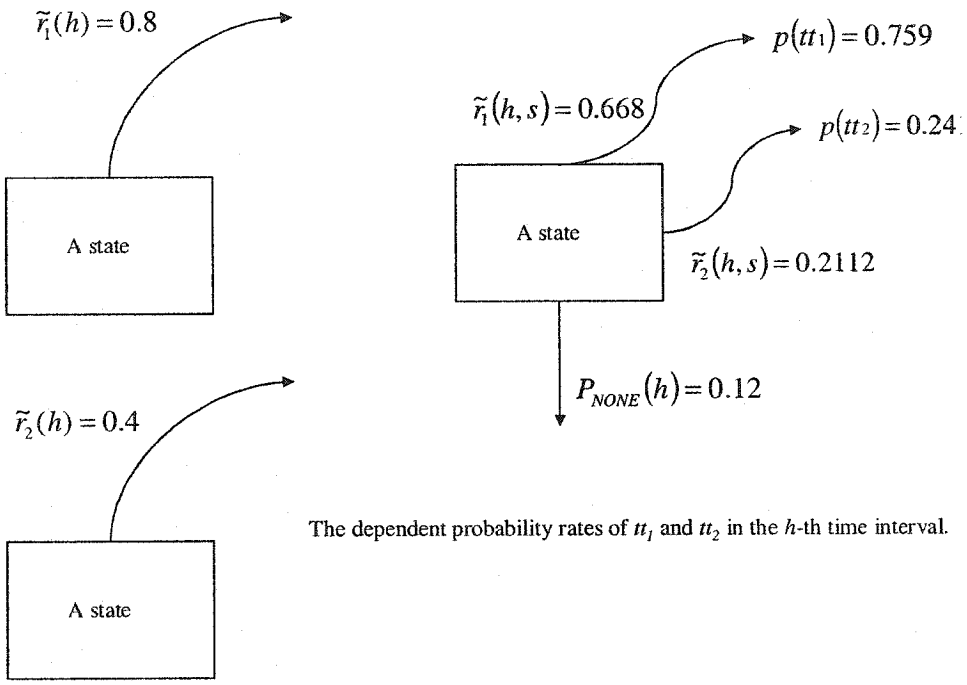
The transition probability for a state-specific transition in state s is simply:

$$p(trans_i, s) = F_i(\infty, s) = \sum_{\eta=0}^{\infty} \tilde{r}_i(\eta, s) \overline{F}(\eta, s) \qquad \textbf{eq. 4-11}$$

It is not necessary to compute a transition probability from the initial time interval up to infinity. A user needs only to sum up the terms up to a "converged" time interval, which is defined as the time when either the likelihood of still being in state s is below a desired threshold ε, i.e., $\overline{F}(\eta, s) < \varepsilon$, or after which all the probability rates are negligible, i.e., $\tilde{r}_i(\eta, s) < \varepsilon$.

As a simple example, let tt_1 and tt_2 be two temporal transitions which have independent probability rates of 0.8 and 0.4 respectively at each time step (constant probability rate functions). By eq. 4-7 and eq. 4-8, the dependent probability rates in any time interval are 0.668 for tt_1 and 0.2112 for tt_2, as shown in Figure 4-5. Eq. 4-11 tells us that the transition probabilities are 0.759 for tt_1 and 0.241 for tt_2. Note that the transition probabilities sum up to 0.759 + 0.241 = 1. Even though the transitions could fire in the same time interval, we estimate how likely it is that one will fire before the other *within*

that time interval. We now provide more insight to eq. 4-7, and justify the way we compute transition probabilities.



The dependent probability rates of tt_1 and tt_2 in the h-th time interval.

The independent probability rates of tt_1 and tt_2 in the h-th time interval.

Figure 4-5: From Independent Probability Rates to Dependent Probability Rates

4.5 Heuristic Justification: Convergence

As we have mentioned in Section 4.2, lacking the actual continuous probability density/rate functions of the underlying processes for a real-time agent, it is impossible to compute accurately the "true" transition probabilities. All methods that work with discrete, hence incomplete, data are approximations. Nonetheless, we would desire this property: as we get more information by having a finer discretization, the estimated transition probabilities converge to the values computed using eq. 4-3 as if the continuous functions were available. The intuition is that a finer discretization gives a better picture

of the actual continuous probability functions. The precision of the approximation should improve due to additional information.

We have proven in Appendix A that the transition probabilities calculated using the logarithm heuristic in Section 4.4 converge to the values calculated using eq. 4-3 as the length of the time intervals goes to zero. Basically, what the heuristic does is to approximate the integral by piecewise constant probability rate functions. If the continuous probability rate functions are indeed piecewise constant, the logarithm heuristic always computes the transition probabilities accurately. Otherwise, as the time intervals shrink, the piecewise constant functions better approximate the continuous functions. Consequently, the precision improves. We illustrate this with a numerical example in this section.

A state has temporal transitions tt_1 of constant probability rate 0.1, tt_2 of rate 0.6, and a guaranteed action with a period of 5. The two temporal transitions clearly have piecewise constant probability rates. The action, however, cannot be described by any piecewise constant probability rate function. In fact, the probability rate function of the action is a monotonic increasing function, so no segment of it is constant. The action gets more likely to be executed over time, because a guaranteed action is sure to fire by its period/deadline.

We will show that, nonetheless, as we get finer discretization, the results computed by the logarithm heuristic get closer to the values computed in the continuous domain. For this particular simple example, we can determine the continuous probability rate function of the transitions and compute the true transition probabilities. Then we compare them to the discretized results.

For a guaranteed action transition, the probabilities of it firing at any times before $t = P$, where P is the period of the action, are equal. We thus have the following.

- The probability density function of an action is $\begin{cases} \frac{1}{P}, & \text{if } 0 \le t \le P. \\ 0, & \text{if } t > P \end{cases}$
- The cumulative probability function is $\begin{cases} \frac{t}{P}, & \text{if } 0 \le t \le P. \\ 0, & \text{if } t > P \end{cases}$
- The survival function is $\begin{cases} \frac{P-t}{P}, & \text{if } 0 \le t \le P. \\ 0, & \text{if } t > P \end{cases}$

We can derive the continuous probability rate function of an action from these equations. The probability of an action firing in the time interval (t_1, t_2) given that it has not fired before t_1 is given in eq. 4-12. As t_2 moves toward P, keeping the length of the time interval (the difference between t_2 and t_1) fixed, the probability of the action firing increases over time, as we would expect. When t_2 goes to P, the probability rate equals 1. Eq. 4-12 is the continuous version of eq. 4-5.

$$P(t_1 \le T \le t_2 \mid t_1 < T) = \frac{P(t_1 \le T \le t_2)}{P(t_1 \le T)} = \frac{t_2 - t_1}{P - t_1} \qquad \textbf{eq. 4-12}$$

The probability rate functions for the two temporal transitions are just constant functions. Applying eq. 4-3 to the probability rate functions of the action and the two temporal transitions, we compute their transition probabilities.

Eq. 4-13 shows the derivations of the transition probability of a temporal transition when there are n concurrent temporal transitions with constant probability rates and an action with a period of P. Let T_i be the transition time of tt_i; and T_{ac} be the action transition time. T is the minimum transition time of all transitions.

$$
\begin{aligned}
P(T_i = T) &= \int_0^P P(T_1 > t,\ldots,T_{i-1} > t, T_{i+1} > t,\ldots,T_n > t, T_{ac} > t)\,dP\{T_i = t\} \\
&= \int_0^P \left(e^{-\lambda_1 t}\ldots e^{-\lambda_{i-1} t} e^{-\lambda_{i+1} t}\ldots e^{-\lambda_n t}\right)\left(\frac{P-t}{P}\right)\lambda_i e^{-\lambda_i t}\,dt \\
&= \int_0^P e^{-(\lambda_1+\ldots+\lambda_{i-1}+\lambda_{i+1}+\ldots\lambda_n)t}\,\lambda_i e^{-\lambda_i t}\left(\frac{P-t}{P}\right)dt \\
&= \lambda_i\left[\int_0^P e^{-\Lambda t}\,dt - \frac{1}{P}\int_0^P t e^{-\Lambda t}\,dt\right] \\
&= \frac{\lambda_i}{\Lambda}\left(1-e^{-\Lambda P}\right) + \frac{\lambda_i}{\Lambda}e^{-P\Lambda} - \frac{\lambda_i}{P\Lambda^2}\left(1-e^{-\Lambda P}\right)
\end{aligned}
\qquad \text{eq. 4-13}
$$

$$, \text{where } \Lambda = \sum_{i=1}^{n}\lambda_i$$

The upper limit is P rather than ∞ because some transition will certainly fire before or at $t = P$. As $P \to \infty$ (an infinite period means that the probability of the action firing in a finite time approaches zero, i.e., non-existent), eq. 4-13 becomes $\frac{\lambda_i}{\Lambda}$.

The probability of the action being the first transition to fire is:

$$
\begin{aligned}
P(T_{ac} = T) &= \int_0^P P(T_1 > t,\ldots,T_n > t)\,dP\{T_{ac} = t\} \\
&= \int_0^P \left(e^{-\lambda_1 t}\ldots e^{-\lambda_n t}\right)\frac{1}{P}\,dt \\
&= \int_0^P e^{-(\lambda_1+\ldots+\lambda_n)t}\,\frac{1}{P}\,dt \\
&= \frac{1-e^{-\Lambda P}}{P\Lambda}
\end{aligned}
\qquad \text{eq. 4-14}
$$

$$, \text{where } \Lambda = \sum_{i=1}^{n}\lambda_i$$

We have done a sanity check to verify that the sum of all probabilities of any transition being the first one to fire indeed equals 1. By eq. 4-13 and eq. 4-14, the transition probabilities of the action, tt_1, and tt_2 are 0.194578, 0.083059, and 0.722361 respectively. The transition probabilities computed by the logarithm heuristic in the discrete domain with various levels of discretization are shown in Table 4-1, Table 4-2, and Table 4-3.

Table 4-1: Transition Probabilities of the Action

number of time intervals	time each interval represents (in sec)	discrete approximation	true value computed using the continuous formulation	error (in %)
5	1	0.199963	0.194578	2.767282
10	0.5	0.195877	0.194578	0.667834
100	0.05	0.194591	0.194578	0.006697

Table 4-2: Transition Probabilities of tt_1

number of time intervals	time each interval represents (in sec)	discrete approximation	true value computed using the continuous formulation	error (in %)
5	1	0.082506	0.083059	0.665788
10	0.5	0.082927	0.083059	0.158578
100	0.05	0.083060	0.083059	0.001147

Table 4-3: Transition Probabilities for tt_2

number of time intervals	time each interval represents (in sec)	discrete approximation	true value computed using the continuous formulation	error (in %)
5	1	0.717531	0.722361	0.668575
10	0.5	0.721195	0.722361	0.161380
100	0.05	0.722349	0.722361	0.001659

As suggested by the data, the more we discretize the time line, the more closely the discrete approximation matches the continuous-time model, even though the action transition does not have a piecewise constant rate function. Techniques for solving a general problem in a continuous-time domain using the formulation in Section 4.1.1 can be extremely complicated and costly as seen from the derivations of eq. 4-12, eq. 4-13, and eq. 4-14. On the contrary, our discrete approximation provides a simple, yet effective and powerful method to compute transition probabilities from arbitrary probability rate functions.

4.6 Heuristic Justification: Piecewise Constant Rate Model

Given the piecewise constant rate model, it is most natural to derive the logarithm heuristic as shown in Appendix A. Essentially, we are approximating the integral in eq. 4-3 by piecewise constant probability rate functions. If the underlying processes of transitions indeed have piecewise constant probability rate functions, then our heuristic always computes the precise transition probabilities, *regardless of the length of the time intervals*. Otherwise, the "flatter" the rates of transitions over the intervals are, the more accurate the logarithm heuristic is.[17]

As we now know the types of probability rate functions that the logarithm heuristic works well with, when choosing a particular level of discretization, a user needs only to choose one such that within each interval the continuous probability rate functions are relatively constant/flat. The logarithm heuristic will work well while other heuristics, even if they exist, might still require further discretization until the length of

[17] By rates, we mean the continuous rates r of the underlying process here. The discrete rates $\tilde{r}_{a,b}$ in an interval are always constant by definition. See Appendix A.

the intervals reduces close to zero. In other words, the logarithm heuristic will likely converge to true values faster than other heuristics.

To illustrate this, we have compared the logarithm heuristic to another way. The ratio heuristic estimates state-specific dependent probability rates using the ratios of independent rates. Specifically, it replaces eq. 4-7 with the following.[18]

$$\tilde{r}_i(h,s) = \frac{\tilde{r}_i(h)\left(1 - P_{NONE}(h,s)\right)}{\sum_{\forall trans_j \in trans(s)} \tilde{r}_j(h)} \qquad \textbf{eq. 4-15}$$

We repeat the example that has two temporal and one action transitions in the last section using this ratio heuristic. Table 4-4, Table 4-5, and Table 4-6 show the results. Comparing these results to those in Table 4-1, Table 4-2, and Table 4-3, we see that the logarithm heuristic converges much faster. Also, the logarithm heuristic generates much better estimates when the time line is discretized into only a few intervals, e.g., 5 or 10.

The piecewise constant rate assumption is reasonable. A lot of realistic events do not have probability rate functions that fluctuate wildly in an interval, especially a short one. For example, if we suppose the probability (rate) to fail for a machine is 0.2 today, it is unreasonable that the failure rate will jump to 0.9 tomorrow and go back down to 0.4 the day after (c.f. the first example in Section 4.1). The job arrival rates for a multitasking computer may vary throughout the day, but the rate for each hour is fairly stable and does not change drastically minute by minute (c.f. the second example in Section 4.1).

[18] We do not know whether the ratio heuristic will converge in general. Neither do we know what types of probability rate functions that it works well with.

Table 4-4: Transition Probabilities of the Action Using the Ratio Heuristic

number of time intervals	discrete approximation	true value computed using the continuous formulation	error (in %)	error (in %) using the logarithm heuristic
5	0.240965	0.194578	23.83960	2.767282
10	0.217268	0.194578	11.66108	0.667834
100	0.196785	0.194578	1.134158	0.006697

Table 4-5: Transition Probabilities of tt_1 Using the Ratio Heuristic

number of time intervals	discrete approximation	true value computed using the continuous formulation	error (in %)	error (in %) using the logarithm heuristic
5	0.108434	0.083059	30.550122	0.665788
10	0.095896	0.083059	15.455622	0.158578
100	0.084345	0.083059	1.5488834	0.001147

Table 4-6: Transition Probabilities for tt_2 Using the Ratio Heuristic

number of time intervals	discrete approximation	true value computed using the continuous formulation	error (in %)	error (in %) using the logarithm heuristic
5	0.650602	0.722361	9.933987	0.668575
10	0.686836	0.722361	4.917926	0.161380
100	0.718870	0.722361	0.483319	0.001659

4.6.1 Applications of the Piecewise Constant Rate Model

Researchers in other areas, e.g., [86, 97], have long adopted the piecewise constant rate model. Often in the literature, it is called the piecewise constant exponential

model [96] or piecewise constant hazard rate[19] model [86]. Essentially, what this means is that the time span during which the process under investigation is observed is split into several intervals, or pieces (hence the name "piecewise"), and for each part a different constant (or baseline hazard) is estimated. The influences of the covariates usually are assumed to be the same in each interval. There are a number of pieces of statistical software that are available to estimate the rates in each interval, such as STPIECE [108] and STATA from Stata Corporation. These rates are the inputs in a CIRCA knowledge base.

Not only is the piecewise constant rate model very popular in other areas, but a lot of important processes well studied and widely applied are not just piecewise constant – they are wholly constant probability rate functions. All processes that have constant probability rate functions have exponential distributions. They are better known as the continuous-time Markov processes.

The familiar Poisson processes, which model things like shocks to an engineering system, earthquakes in a geological system, biological stimuli in a neural system, accidents in a given city, claims on an insurance company, demands on an inventory system, and failures in a manufacturing system, assume constant rates [67]. The Queuing Model that studies queues in grocery stores, banks, department stores, amusement parks, movie theaters, modern communication systems (e.g., emails), manufacturing systems, and supply chain management likewise assume constant rates. Other examples include Birth-and-Death processes.

Therefore, the modeling of temporal dynamics of events and actions in terms of probability rate functions is built upon an established model in statistics, reliability theory,

[19] The reader is reminded what other people call "hazard rate" is "probability rate" in this thesis.

and survival analysis, with software that allows us to estimate rates from experimental data. We took advantage of these existing works in the piecewise constant rate model to build our probabilistic temporal dynamics representation using probability rate functions for planning purpose. One top of this, we have devised a discrete approximation method, such as the logarithm heuristic, which a simple yet powerful way to estimate transition probabilities of any continuous-time processes. With transition probabilities, we can compute the temporal projection of an agent in a real-time environment, namely, the probabilities of it visiting each state during execution.

4.7 State Probability

The temporal trajectory of a real-time agent is a stochastic process. Under the assumption that the clocks associated with all transitions reset to zero after transitioning into a new state, the stochastic process is a semi-Markov chain (SMC). A semi-Markov chain can be characterized by sojourn times (eq. 4-2) and stationary transition probabilities (eq. 4-3) of all states. Although stationary transition probabilities and sojourn times are not enough if we are interested in the transient and occupancy-time analysis of a semi-Markov chain, they are adequate for our purpose of computing the probabilistic temporal trajectory, which an agent uses to make resource allocation decisions. In fact, we do not even need sojourn times.

State probability of a state is the probability that an agent will encounter the state during execution, not the probability of being in the state at any arbitrary time. In terms of a state diagram, it is the probability that the agent will visit that state at least once from one of the initial states. State probability is often called the first passage probability in mathematics or operations research.

4.7.1 Computing State Probabilities

Every semi-Markov chain (SMC) has an embedded discrete time Markov chain (DTMC) with the same state space and the same transition probabilities. Let S_n be the n-th epoch or the n-th transition from a state to another of a semi-Markov chain and $X(S_n)$ be the state at time S_n, the stochastic process $\{X(S_n), n \geq 0\}$ is a time homogeneous discrete time Markov Chain. This is called the embedded DTMC of the SMC [67].

Moreover, the first passage probabilities of the states in the DTMC are exactly the same as those in the SMC [52]. We can therefore compute the probabilities of states in a stochastic process by computing the probabilities of states in the embedded discrete time Markov chain. The transition probabilities alone characterize the Markov chain. We have already discussed in Section 4.4 how we compute transition probabilities.

With transition probabilities, we can construct a state transition matrix M. $M[i][j]$ is the transition probability from node i to node j (we use "node" and "state" interchangeably in this section). If node i is an absorbing node, then $M[i][i]=1$. We compute state probabilities from a transition matrix using a theorem in [64].

Given a transition matrix, we delete all the rows and columns corresponding to the recurrent states. A recurrent state is a state that has a probability of 1 to revisit that state again if the process leaves the state. The set of recurrent nodes usually form "absorbing" cycles where once the process gets into the cycles, it never gets out but simply traverses the cycles infinitely.

We rearrange the indices of the nodes such that the absorbing nodes have the smallest indices as in Figure 4-6:

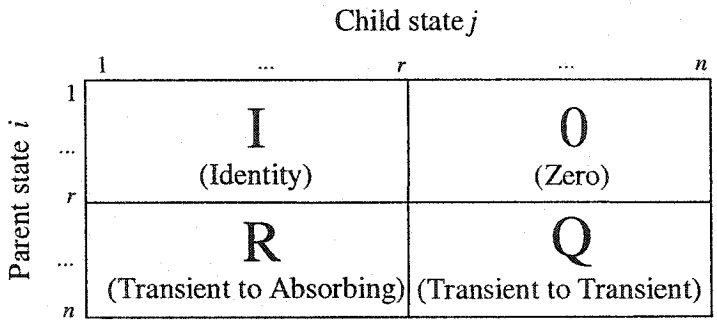


Figure 4-6: A Transition Matrix

***n*: total number of nodes; *r*: number of absorbing nodes**

Then, the probabilities of going from any nodes in the transient set (transient nodes are the ones which have outgoing transitions, i.e., *tt*s, *ttf*s, actions) to any nodes in the absorbing set (absorbing nodes are the ones which have no outgoing transitions) are given by the matrix P in eq. 4-16.

$$P = NR, \quad where \quad N = (1-Q)^{-1} \qquad \textbf{eq. 4-16}$$

P is a $(n-r)*r$ matrix, where n is the total number of nodes, r the number of absorbing nodes and $(n-r)$ the number of transient nodes. Both matrices, R and Q, are readily available from the transition matrix M. To compute the state probabilities of all absorbing nodes in a state diagram, we only need to apply eq. 4-16 once.

To compute the state probability of a transient node, we have to make it into an absorbing node by truncating all its outgoing transitions. As the state probability of a node, i.e., the probability of ever visiting the node, depends only on the paths from the initial states to the node, revisiting the same node does not contribute to the state probability. All the paths coming out from the node will not further affect its state

probability even if the paths subsequently enter into the node again, i.e., cycles. Thus, for each state probability computation of a transient node *i*, we need to construct a different transition matrix by setting $M[i][j]=0, \forall j \neq i$ and $M[i][i]=1$.

4.7.2 Time Complexity of Computing State Probabilities

The time complexity of computing *P* in eq. 4-16 is approximately $O(n^3)$ because of the matrix inversion.[20] On the surface, it appears that the complexity to compute the probabilities of all states in a state diagram is $O(n^4)$ in the worst case. However, significant reduction in the complexity can be obtained. First, we need to apply eq. 4-16 only once for all absorbing states. It is only the transient states that require different transition matrices. The time complexity is thus $O((n-r+1)*n^3)$. If the majority of the states are absorbing, it is closer to $O(n^3)$. Second, it is unnecessary to construct a transition matrix including all nodes in the state diagram. Eq. 4-16 is intended to work only with a transition matrix with all recurrent nodes removed.

Third, for a transient node, we only need to construct the transition matrix including only its ancestors because any other nodes do not contribute to its state probability. The matrix size is often much smaller than *n*. The procedure of identifying the ancestors that are not recurrent nodes and constructing the transition matrix can be done in $O(n)$ by traversing the state diagram from the transient node *backward* to the initial states, marking the ancestor nodes along the path.

It is easy to verify that no marked nodes are recurrent. The reasoning is as follows: the transient node is made absorbing by removing all its outgoing transitions, so it cannot

[20] Strassen showed it is $O(n^{\lg 7}) \approx O(n^{2.80735})$ in [109].

be recurrent. For any other marked node, there is always a path from the node to the transient node. It thus cannot be part of an absorbing cycle. So, neither can it be recurrent. Note that a matrix so constructed corresponds to a connected subgraph in the state diagram. The inverse in eq. 4-16 thus always exists.

Finally, most realistic environments cannot transition from every state to every other state, so the transition matrix for a transient node is a sparse matrix. There are a lot of efficient matrix inversion algorithms for various sparse matrix patterns. For example, if a matrix pattern is tridiagonal, the time complexity of inverting a matrix is only $O(\tilde{n})$. We use $\tilde{n}$ to denote the number of ancestors to distinguish it from the total number of nodes in the state diagram. The space complexity is also $O(\tilde{n})$. A substantial collection of routines for sparse matrix calculations is available from IMSL (IMSL Math/Library Users Manual) as the Yale Sparse Matrix Package [35]. A starting point is [41], which gives an introduction to fast sparse matrix inversion algorithms.

In summary, it is very superficial to say that the time complexity for state probability computation is $O(n^4)$. Very often, it is not the case. If a state diagram happens to be of the "right" type, e.g., acyclic, it could be as efficient as linear time to calculate the probabilities for *all* states [4]!

4.8 Dependent Temporal Transition

Section 4.7 computes state probabilities analytically using the standard technique in Markov chain theory. We can do this because we have been assuming that the clocks of all transitions reset to zero after an agent enters a new state. The stochastic process that the agent undergoes is thus a semi-Markov chain. However, some applications involve non-Markovian stochastic processes. In these cases, we can no longer apply eq. 4-16 to

compute state probabilities. In this section and the next, we describe an extension to our theory and how we compute state probabilities when the stochastic processes are non-Markovian.

When a temporal transition persists across a sequence of states, its rate function in a later state must consider the time spent in the prior states. Musliner calls such a temporal transition a *dependent temporal transition* (*dtt*), because the probability rate function in a state differs depending on how long the transition has been triggered before the process enters the state [87]. We refer to those that are not dependent temporal transitions as *independent temporal transitions.*

An example of a *dtt* is shown in Figure 4-7. In this example, an aircraft flies from FIX1 to FIX2 and then to FIX3. The faulty engine may fail to cause the plane to crash at any point during the course. The plane needs to do an emergency landing (ac_1) when it begins to crash (ttf_1).

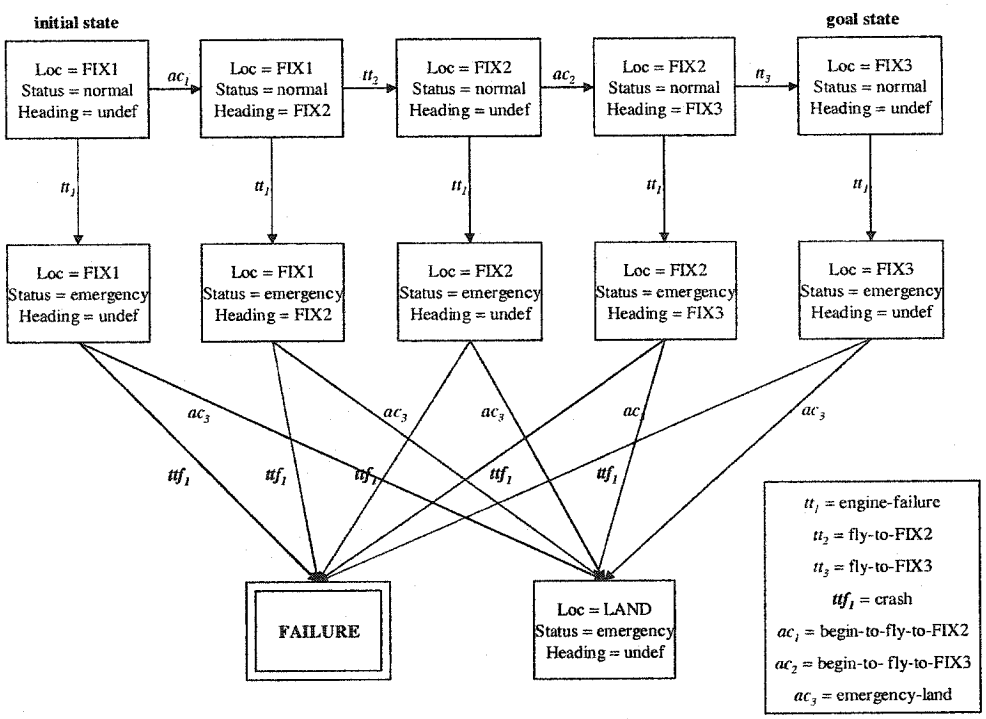


Figure 4-7: A Dependent Temporal Transition (tt_1)

Suppose that the probability rate of engine-failure, tt_1, increases with time. Then the probability of tt_1 occurring is higher in FIX2 than in FIX1, because tt_1 is already "enabled" in FIX1 before the flight enters FIX2. Similarly, the probability is higher in FIX3 than in FIX2. If the plane ever gets to FIX3, the probability of the engine failing is higher than when the plane initially starts out in FIX1.

4.8.1 Specifying a Dependent Temporal Transition in a State

Therefore, we cannot simply use its (continuous) probability rate function $r(h)$ to compute the transition probabilities of tt_1 in all states. For each state s, the probability rate function of tt_1 should be "delayed" or "shifted" by the amount of time, D, that the transition has been enabled before entering the state as in eq. 4-17. The relation between a delayed probability rate function in a particular state and the unmodified/raw probability rate function is shown in Figure 4-8.

$$r_s(h) = r(h+D) \qquad \textbf{eq. 4-17}$$

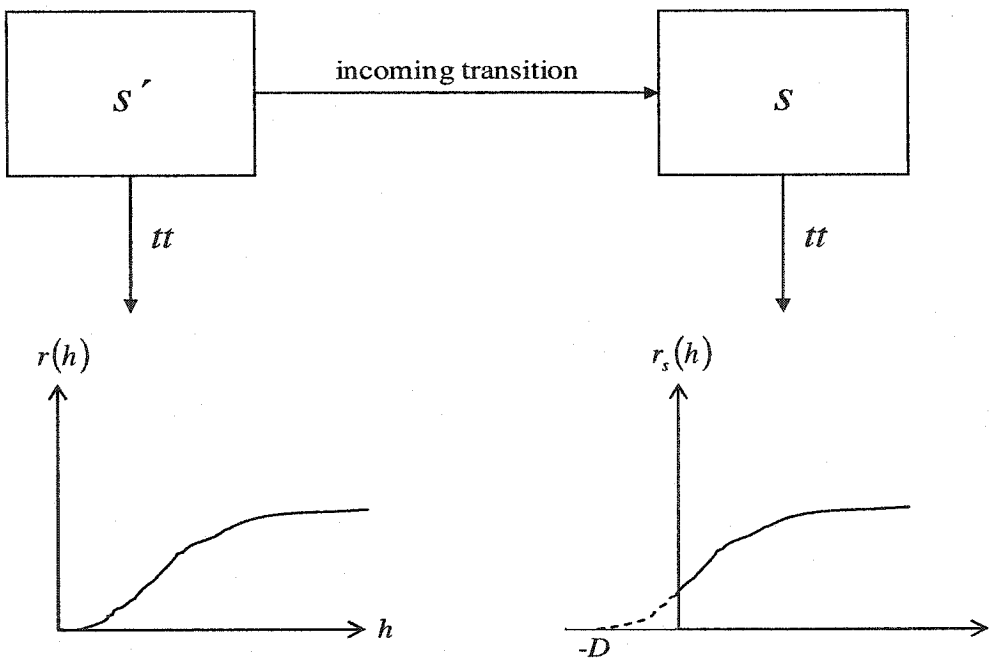


Figure 4-8: A Shifted Probability Rate Function with a Delay = D

For a dependent temporal transition that has a constant probability rate function, the delay is irrelevant. Thus,

$$\tilde{r}_s(h) = \tilde{r}(h) \qquad \text{eq. 4-18}$$

Recall that state probabilities depend only on transition probabilities, which in turn depend only on probability rate functions of the transitions. Dependent temporal transitions having constant probability rate functions in a state thus have the same transition probabilities regardless of the delays. Rare as they sound, those constant rate temporal transitions actually form a large class of stochastic processes. They have exponential distributions and are commonly known as continuous-time Markov processes. Many realistic phenomena can be modeled as such. We have listed some of their important applications in Section 4.6.1.

4.8.2 Complications of Non-Markovian Processes

In general, probability rate functions are not constant so the stochastic processes are not semi-Markovian. For any state, there are often multiple paths from the initial states to that state, and the process can spend a random amount of time on any of the paths. For the example in Figure 4-8, the delay of *tt* in *s* is the transition time of the "incoming transition," which is a random variable. Consequently, the transition probabilities of the transitions in the state are different for each possible delay. We typically do not know the delay for a dependent temporal transition in a state during planning.

One would hope that it is possible to compute some sort of "average" probability rate function for a *dtt*, or equivalently, an "average" transition probability, so that we

might use these averages to compute the state probabilities. We worked out a way to compute the "average" probability rate function for a dependent temporal transition, which is defined as the sum of all $r_s(h) = r(h + D)$, weighted by the probability of each delay, D, for all delays. While the state probabilities computed using eq. 4-16 from these weighted average transition probabilities are quite accurate sometimes, they could also lead to large errors. The values computed using the weighted average method could be twice as big as the true values. We are not able to provide an upper bound to the errors for using this method.

The fundamental problem is that there is no single value that we can assign to a transition for all transitions that allows any analytical method, not just eq. 4-16, to compute the accurate probabilities for all states in a state diagram. Some information is lost. We illustrate this using the example in Figure 4-9.

It is not too hard to see that the probability of state s_3 is 1. Starting from s_1, the initial state, there are only 4 possible paths. They are:

- $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_6$
- $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_4 \rightarrow s_5$
- $s_1 \rightarrow s_4 \rightarrow s_3 \rightarrow s_6$
- $s_1 \rightarrow s_4 \rightarrow s_3 \rightarrow s_4 \rightarrow s_5$

All these paths pass through s_3, so s_3 has a state probability of 1. Depending on how s_4 is reached, the transition probability of the *dtt* in s_4 can either be 1 or 0. If the process reaches s_4 from s_1, then the transition probability of the *dtt* in s_4 is 1 because at least 500 time steps have been spent in s_1 before entering s_4. The delay for the *dtt* in s_4 is at least 500, which makes its probability rate 1 in the very first time step, versus a

probability rate 0 of tt_4 in s_4. If the process reaches s_4 from s_3, the *dtt* transition probability is 0 because tt_4 is guaranteed to fire before the 500-th time interval, before which the *dtt* in s_4 has a probability rate of 0 throughout. In other words, the transition probability of the *dtt* in s_4 depends on the history before the process enters s_4.

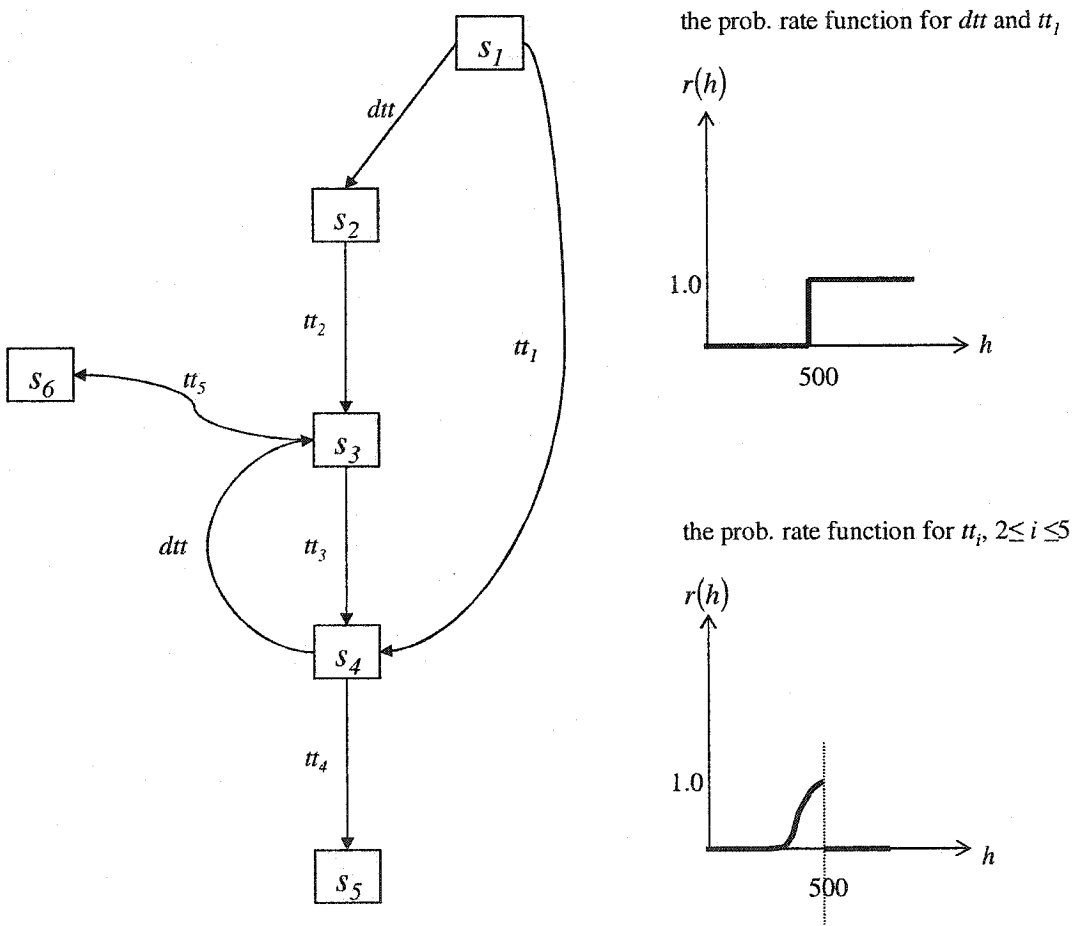


Figure 4-9: *DTT* Anomaly

Moreover, there is no value we can assign to this transition probability such that we can accurately compute the probabilities of all states in the state diagram. In order to give s_3 a state probability 1, it is necessary to assign 1 to the transition probability of the *dtt* in s_4. Otherwise, for any other value, there is a chance that s_3 may not be reached if the process travels along the $tt_1 \rightarrow tt_4$ path. Unfortunately, by assigning 1 to the transition

probability, s_5 will never be reached. Each time the process enters into s_4, it will certainly go to s_3. Then it may either settle in s_6, or loop between $s_3 \rightarrow s_4 \rightarrow s_3$ until it settles in s_6. In other words, regardless of what value we assign to the transition probability of the *dtt* in s_4, we cannot accurately compute the probabilities of *all* states in the state diagram without keeping track of all path- and time- dependent histories.

Not only do we have to keep track of the paths by which the process enters a state, but, even worse, we must also enumerate the scenarios for each possible time spent in traversing these paths. Obviously, this is impractical even for a small problem. Note that the key here is "all states." We can usually compute accurate probabilities for some states, but not all of them at the same time. For the example above, we can calculate the accurate state probability for s_3 by giving 1 to the transition probability of the *dtt* in s_4. What we cannot do is to compute accurately the state probabilities of s_3 and s_5 with a single value assigned to the *dtt* transition probability.

This result should not be surprising. After all, if we could compute some sort of "average" or "summary" transition probabilities that describe a stochastic process, we would not need the history. The stochastic process from the transition epoch onward thus depends only on the state alone, which is ultimately the Markov property. Clearly, for a stochastic process that violates the Markov property, this kind of "average" or "summary" information does not exist.

4.9 State Probability Computation by Simulation

The reader is reminded that all the work developed so far to compute state probabilities is to let a real-time agent to allocate its limited resources by reasoning about its execution trajectory (Section 4.10). Unless we would like to keep track of all possible

delays and paths into all states, there is no analytical method that would accurately compute the probabilities for all states in a state diagram when there are *non-constant* probability rate functions that are dependent temporal transitions. We therefore resort to using simulation. The most significant advantage of simulation is that, in principle, it is applicable to systems of arbitrary complexity. There is a large body of literature that studies simulations, especially discrete event simulations [40, 99, 103].

A standard way of doing simulation is Monte Carlo simulation. It estimates a quantity θ by obtaining an output of the relevant system X, a random variable whose expected value is θ. A second independent simulation run provides a new and independent random variable having mean θ. This continues until we have amassed a total of k runs. Hence, we have k independent random variables X_1, X_2, ..., X_k, all of which are identically distributed with mean θ. The average of these k values is given by eq. 4-19. $\overline{X}$ is the maximum likelihood estimator of θ.

$$\overline{X} = \sum_{i=1}^{k} \frac{X_i}{k} \qquad \textbf{eq. 4-19}$$

In terms of CIRCA, we need to generate sample paths from initial states to absorbing states, marking the states visited. After a number of runs, for each state, the number of visits divided by the total number of runs is the *estimated* or *sample* state probability of the state.

There are some sample path construction algorithms in the literature, such as Algorithm 4.17 in [103]. That algorithm assumes a semi-Markovian framework. If the stochastic process is semi-Markovian, we would use eq. 4-16 instead. There is no need for simulation. Moreover, these sample path construction algorithms do not usually

specify how to handle cases when two transitions in a state fire simultaneously in the same time interval, i.e., handling the missing probability problem in Section 4.2.

In order to have consistent results produced by the analytical method (eq. 4-16) and simulation when there are only constant probability rate functions, we need to generate sample paths according to the transition probabilities computed using the logarithm heuristic (eq. 4-7). The algorithm is shown in Algorithm 4-1.

Inputs:

a state diagram

Outputs:

a sequence of states marked as visited

Algorithm:

1. Pick an initial state s in accordance to the initial state probabilities.
2. Set time (step) $t = 0$.
3. For each transition in s, compute its dependent probability rate in this time step, t, as in eq. 4-7.
4. Determine if there is any transition firing in this time step according to the dependent probability rates proportionally.
5. If no transition fires, increment t by 1. Loop back to step 3.
6. Otherwise, let s' be the child of the firing transition.
7. If s' is an absorbing state, quit.
8. If there is a temporal transition that is in both s' and s, shift the probability rate function of this dependent temporal transition in s' by t plus the delay already in s, as in eq. 4-17.
9. Set s' to s.
10. Loop back to step 2.

Algorithm 4-1: CIRCA Sample Path Generation Algorithm

In step 3, the sum of the dependent probability rates for all transitions plus the probability that no transition fires in this time step must be 1. We pick a firing transition randomly according to their dependent probability rates proportionally. For example, if $trans_1$ has a rate of 0.8 and $trans_2$ 0.4, then $trans_1$ should be selected twice as likely as $trans_2$. We attach a local clock t (line 2) to each state. If there is a dependent temporal

transition in state s', we can shift its probability rate function by the local clock of its parent s plus the already shifted amount of this transition in s. Since its probability rate function in s may already be shifted, we need to accumulate the delays along the entire chain of the dependent temporal transitions. The delay is reset to 0 once the *dtt* chain ends.

Moreover, it is possible that the simulation runs into an infinite loop, e.g., cycles. In this case, we need some other stopping criteria besides line 7. Currently, we stop the run after it has generated more than N states. N is a multiple of the total number of states in the state diagram. Essentially, we truncate the simulation by having a finite horizon. This finite horizon is fixed in terms of the number of states, but varies in terms of the global time (sum of the local clocks in line 2 in all states). The truncation does not affect the state probabilities in case of an absorbing cycle, but it introduces errors when there is a very tiny probability for the process to escape the cycle.

A reasonable question to ask at this point is how many runs we need to get a reasonable estimate of a state probability. Ross [99] suggests that for an acceptable value d for the standard deviation of the estimator, we need $k > 30$ runs, such that:

$$k > \frac{S^2}{d^2} \qquad \textbf{eq. 4-20}$$

There is nothing magical about 30. In fact, for our purpose where the probability of a state tends to be small, we usually need a lot more than 30 runs (could be in order of 10^5 to 10^6 depending on the precision desired). S^2 is the sample variance and is defined in eq. 4-21. It is possible to compute S^2 recursively so that we do not need to recompute from scratch each time a new datum value is generated.

We can also compute the (approximate) confidence interval for a state probability estimate. Let $x = \overline{X}$ be the estimator and $s = S = \sqrt{S^2}$ the sample standard deviation. Then the interval $x \pm z_{\alpha/2} s/\sqrt{k}$ is an (approximate) $100(1-\alpha)$ percent confidence interval. z is the unit normal random variable we can look up from the normal distribution table, e.g., $z_{0.025} = 1.96$.

$$S^2 = \frac{\sum_{i=1}^{k} (X_i - \overline{X})}{k-1} \qquad \text{eq. 4-21}$$

Although it is relatively easy to implement the simulation algorithm reported here, we would not want to use it if the analytical method is applicable. Simulation can at best give imprecise approximations. Sometimes, we need to impose a finite horizon to avoid infinite loops. It is also subject to sampling variability. It is often slow compared to the analytical approach in eq. 4-16. For example, when a state has a low state probability, e.g., 0.0001, most of the sample paths will not visit it. Consequently, it takes a lot of runs to compute for states with low state probabilities. In contrast, for acyclic state diagrams, it could take as little as linear time to compute state probabilities analytically as well as precisely.[21] Therefore, CIRCA uses simulation only when there are dependent temporal transitions having non-constant probability rate functions.

4.10 The Unlikely State Strategy

The CIRCA planner, during each planning loop, selects a state and "expands" it by applying all enabled temporal transitions, of which the preconditions are met in the

[21] However, eq. 4-16 requires significantly more work in implementation to do the sparse matrix inverse speedup.

state. Depending on the possible consequences of these transitions, the planner may select an action for the state. The period of the action is also determined such that the transition probabilities of the transitions to failure are less than a user-specified probability threshold, ε. The transition probabilities of the action and temporal transitions are computed. The successor states of these transitions are added to the state diagram. The planner continues the planning loop until all states have been expanded.

This *preferred, unconstrained plan* is passed to the scheduler, which tries to schedule the actions according to the resource constraints of the Real-Time Subsystem. If the set of planned actions is schedulable, the AIS is done. Otherwise, the state probabilities of the states are computed. The actions planned for the least likely states are removed in increasing order of their state probabilities until the remaining set of actions is schedulable. The intuition is that if an agent has a very low probability of reaching a state, then ignoring the hazards or temporal transitions to failure (*ttf*s) in this state does the least harm (assuming all failures are equally bad). We call this the *unlikely state (cutoff) strategy*. The algorithm is shown in Algorithm 4-2.

Inputs:

an initial, preferred, unschedulable plan

Outputs:

a schedulable plan

Algorithm:

```
1. while (the plan is unschedulable) {
2.       compute the probabilities of the states;
3.       remove the action in the state having the smallest state probability;
4. }
5.
6. return the schedulable plan;
```

Algorithm 4-2: The Unlikely State Strategy

After removing the actions in the unlikely states, these states become unplanned for in the final plan. If one of the low probability "ignored" states is actually reached during execution, CIRCA has mechanisms to detect it and replan or retrieve a contingency plan, as are discussed in [5].

If not all failure states are equally bad, our strategy can be easily modified to accommodate domains with varying degrees of failure. For instance, instead of ignoring the least likely states, we could ignore the least catastrophic states weighted by their state probabilities. Specifically, let p be the probability of a state, and u the (dis)utility of entering its descendant failure state. We can order the states by $p*u$, and iteratively remove the ones with the lowest expected utilities in a similar manner. The agent retains only those actions that are expected to prevent the most significant failures.

4.11 Justification of the Unlikely State Strategy

Removing an action transition from a state diagram changes the probabilities of the states. We will show in the following that the unlikely state strategy is a greedy strategy that minimally raises the failure probability at each iteration (Algorithm 4-2). That is, the agent's failure probability, F, increases by no more than the probability of the state the action was planned for. Specifically, we will prove that eq. 4-22 is true for a dropped action *ac*. ΔF is the increment in failure probability. $P(s)$ is the probability of s, the state *ac* was planned for.

$$\Delta F \leq P(s) \qquad \textbf{eq. 4-22}$$

ΔF has two components. The first component, ΔF_1, is that the agent is now unable to preempt the local *ttf* in state *s*.[22] It is simply the probability of the failure state reached by the *ttf*. Thus,

$$\Delta F_1 = P(s)F_{ttf}(\infty, s) \qquad \text{eq. 4-23}$$

$F_{ttf}(\infty, s)$ is the transition probability of the *ttf* that would otherwise be preempted by the action. It is computed by eq. 4-11. In the example in Figure 4-10, after the agent removes the action in state *s*, the transition probability of the *ttf* in *s* becomes 0.1. Thus, $F_{ttf}(\infty, s) = 0.1$. $\Delta F_1 = P(s) \times 0.1$.

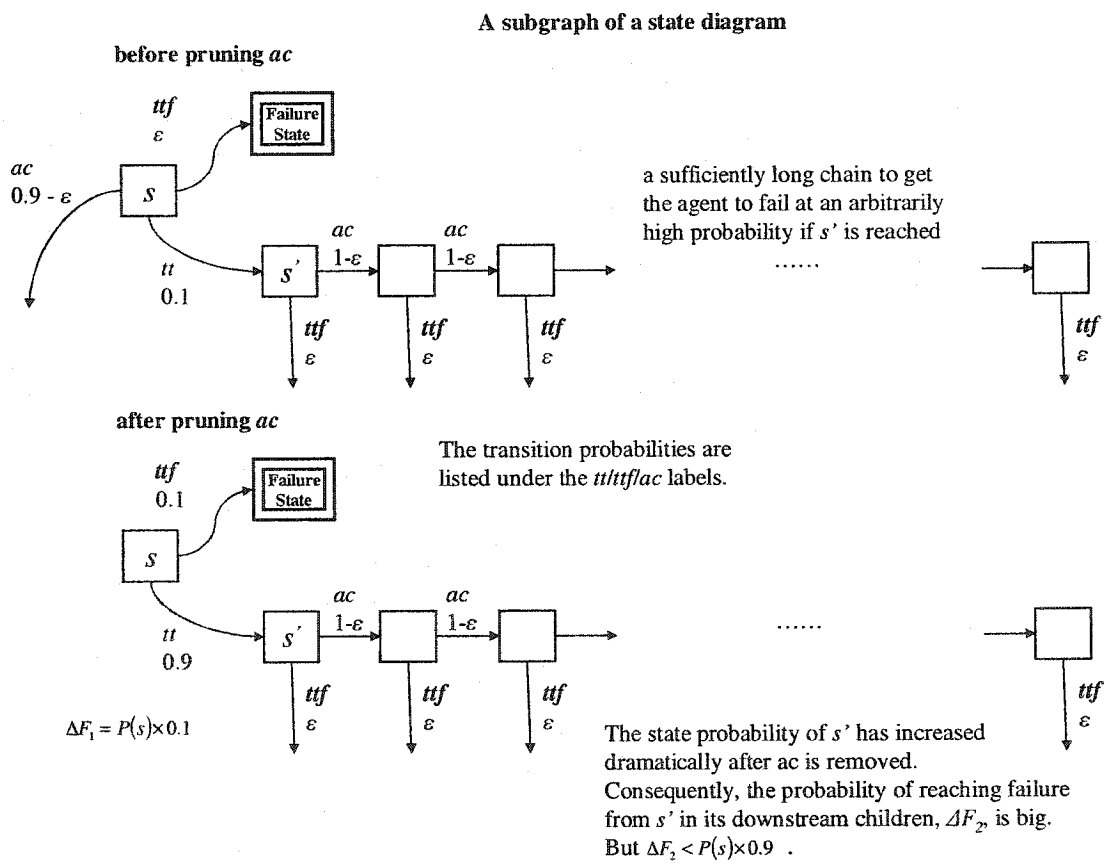


Figure 4-10: Change in the Failure Probability after Removing an Unlikely Action

[22] The result developed here is also valid when there is more than one *ttf* in a state.

After an action planned for a state, s, is removed, the transition probabilities in that state change. Consequently, the probabilities of *other* states change. By the definition of state probability, the paths coming out of a state, including the action, does not affect the state probability. So, $P(s)$ stays the same.

The second component, ΔF_2, comes from the changes in the state probabilities. As a result, the probability of reaching failures changes as well even if the transition probabilities to failure remain the same. This second increment is bounded by:

$$\Delta F_2 \le P(s)\left(1 - F_{ttf}(\infty, s)\right) \qquad \textbf{eq. 4-24}$$

The right hand side is the worst case when the downstream child states of s will inevitably lead the agent to failure, as in Figure 4-10. Thus, we have:

$$\begin{aligned} \Delta F &= \Delta F_1 + \Delta F_2 \\ &\le P(s)F_{ttf}(\infty, s) + P(s)\left(1 - F_{ttf}(\infty, s)\right) \\ &= P(s)\left[F_{ttf}(\infty, s) + \left(1 - F_{ttf}(\infty, s)\right)\right] \\ &= P(s) \end{aligned} \qquad \textbf{eq. 4-25}$$

In our earlier work, we at one point considered another strategy that ignores unlikely threats instead of unlikely states. It orders the actions by the probabilities of the temporal transitions to failure that the actions are preempting. The probability of a threat $P(ttf)$ equals the probability of the state the *ttf* is in multiplied by the transition probability of the *ttf* when no action is planned for the state:

$$P(ttf) = P(s)F_{ttf}(\infty, s) \qquad \textbf{eq. 4-26}$$

However, in light of eq. 4-25, this strategy is inferior to the unlikely state strategy. It does not provide an upper bound to the increment in failure probability each time an

action is removed. For the example in Figure 4-10, if the *ttf* in state s has a very low transition probability, e.g., 1^{-100}, then the action in s will be removed according to the unlikely threat strategy. Unfortunately, if s is the initial state, then removing *ac* will inevitably lead the system to failure. Although ΔF_1 is very tiny, ΔF_2 nears 1. We would therefore prefer the unlikely state strategy that bounds the increments because predictability of the strategy performance is very important.

Intuitively if a user specifies a smaller transition probability to failure, ε, the actions will have shorter periods.[23] The failure probability F before pruning is smaller. Scheduling is more difficult. The unlikely state strategy will drop more actions, which raises F more (a bigger ΔF). On the other hand, if the user specifies a larger ε, the failure probability F before pruning is bigger. Scheduling is easier. The unlikely state strategy will not drop as many actions, which consequently does not raise F as much (a smaller ΔF). Therefore, the user needs to decide, in case of insufficient resources, whether it is better to generate an initial plan with a bigger failure probability but smaller increments later on, or the reverse.

We do not have a good theory on the relations between ε, ΔF and F. In practice, the user can do a number of rounds of trial-and-error to estimate their relations, and come up with a plan that has an acceptable level of safety. Optimality, nevertheless, is very difficult. It may not even be worth it if considerable efforts are needed to go from "acceptable" to "optimal." Often, we want a plan that is not of theoretical interest, but one that is accessible and practical.

[23] CIRCA plans an action to preempt a *ttf* such that its transition probability is less than a pre-specified threshold, ε.

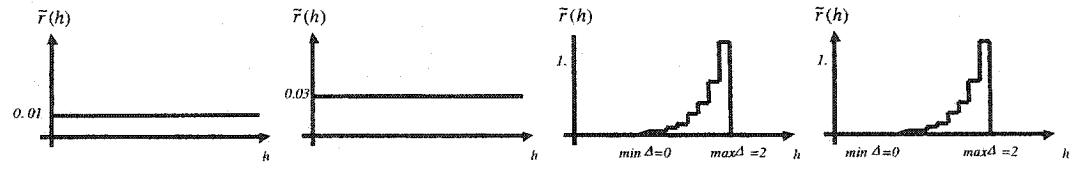
4.12 Demonstration

To illustrate the advantage of having a probabilistic worldview in the case of a resource shortage, we present a demonstration in a simulated domain of an autonomous aircraft. In this example, we are concerned only with safety, so this aircraft's utility function is $U = \overline{F} = (1 - F)$ (eq. 3-1). There are 11 non-failure states that the aircraft *may* encounter. It requires 7 actions, ac_1 to ac_7, to avoid all failures that could arise. These 7 actions must be scheduled frequently enough such that the aircraft detects an emergency soon enough to have enough time to execute the corresponding reaction. Therefore, the aircraft has to determine which action to take in each state and how frequently to examine and execute each action.

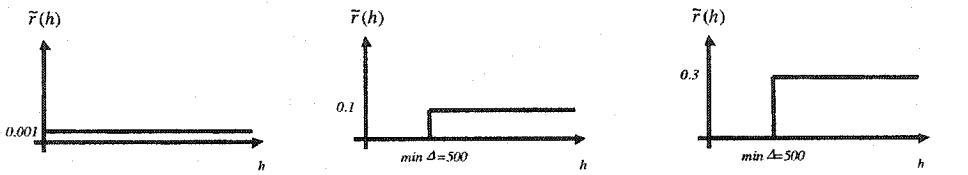
Unfortunately, in this example, the aircraft has insufficient resources to schedule all 7 actions. It finds out from feedback from the scheduler that it can only guarantee real-time performance for any 5 actions. The question becomes which 2 actions the planner should drop such that the utility is maximized, or equivalently the failure probability is minimized.

In general, finding the optimal, schedulable subset of actions is a very difficult combinatorial problem. The unlikely state strategy proposed in this thesis allows us to order the states based on their encounter likelihoods and ignore unlikely situations. The probability rate functions of the temporal transitions and actions of this aircraft example are shown in Figure 4-11 (the *ttf*s are in bold). It does not show the pre- and post-conditions as they should be evident in Figure 4-12.

Temporal transitions:



tt_1: radar_threat tt_2: IR_threat tt_3: evade_radar_missile tt_4: evade_IR_missile



tt_5: swoop **ttf_1: radar_threat_kills_you** **ttf_2: IR_threat_kills_you**

Actions:

ac_1: blow_chaff, $wcet = 50$
ac_2: begin_radar_evasive, $wcet = 50$
ac_3: resume_normal_path, $wcet = 50$
ac_4: deploy_flare_sequence, $wcet = 50$
ac_5: begin_IR_evasive, $wcet = 50$
ac_6: fly_to_destination, $wcet = 50$
ac_7: climb, $wcet = 5$

Figure 4-11: The Temporal Transitions and Actions for an Autonomous Aircraft

The ideal state diagram, if there were no resource constraints, is shown in Figure 4-12. In this case, the utility is 1; the failure probability is 0; the goal probability is 1.

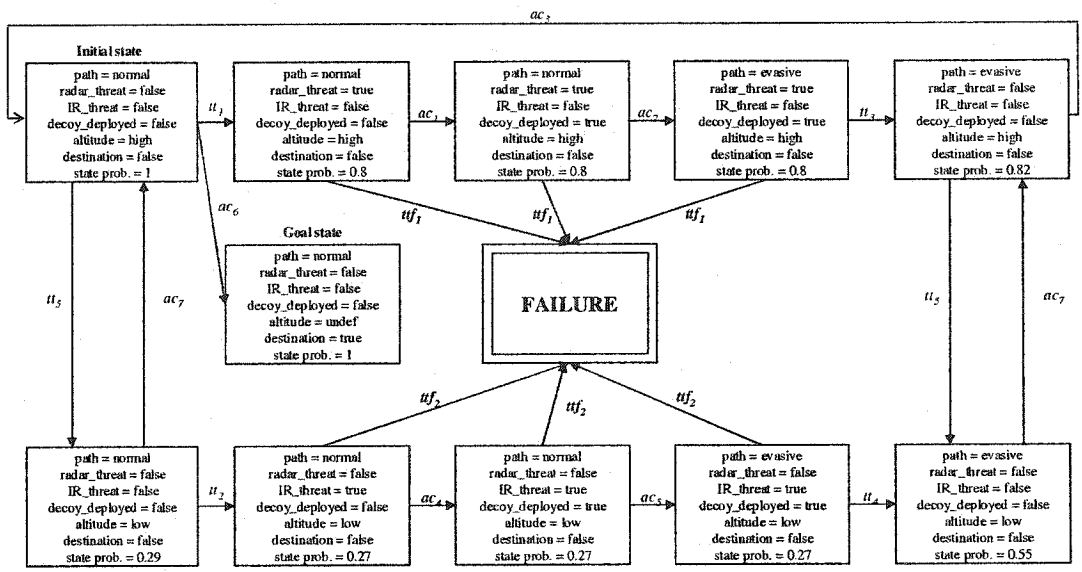


Figure 4-12: The State Diagram with All 7 Required Actions to Guarantee 100% Safety

In the case of insufficient resources such that the aircraft can schedule only 5 actions, without calculating the state probabilities, the aircraft may be led to think that the low altitude path is more probable than the high altitude path, because tt_2 (the bottom path) has probability rate 0.03 while tt_1 (the top path) has probability rate 0.01. If the user decides to ignore the radar threats (tt_1 and ttf_1), hence dropping ac_1 and ac_2, the plan generated is shown in Figure 4-13. The failure probability is 0.8 and the utility is only (1 – 0.8) = 0.2.

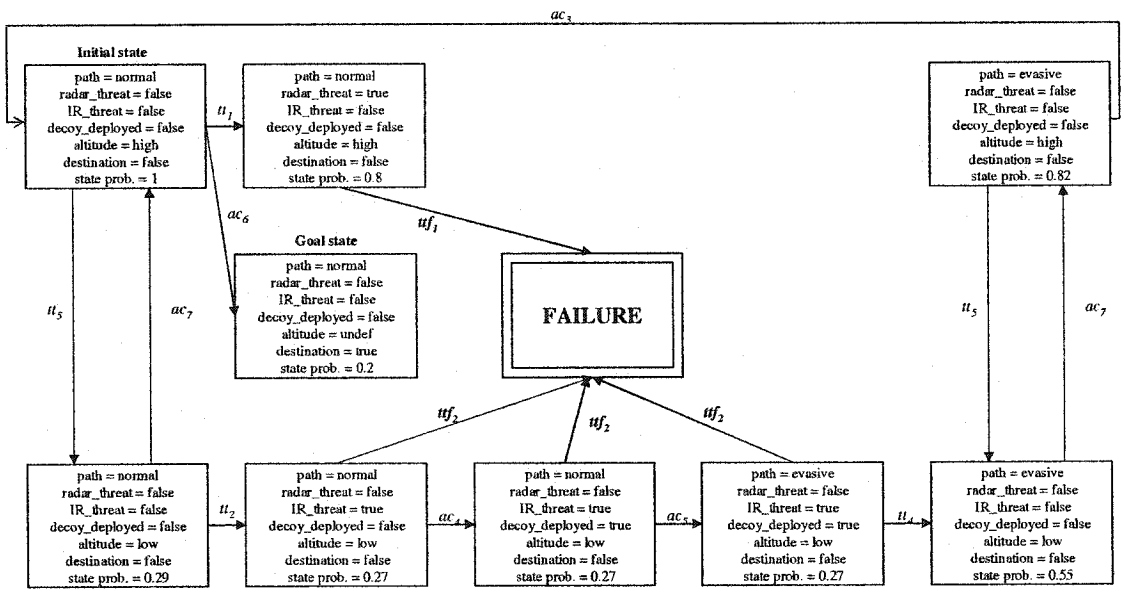


Figure 4-13: The State Diagram Ignoring the Radar Threat; Utility = 0.2

With state probabilities, the agent can make a more informed decision. It knows that the probabilities of ever reaching the states in the high altitude path are 0.8 while the probabilities of ever reaching those in the low altitude path are only 0.27. It will instead ignore the IR threats (tt_2 and ttf_2). ac_4 and ac_5 are dropped from the preferred plan. The utility in this case is (1 – 0.27) = 0.73. The optimal plan, *under the resource constraints*, is shown is Figure 4-14.

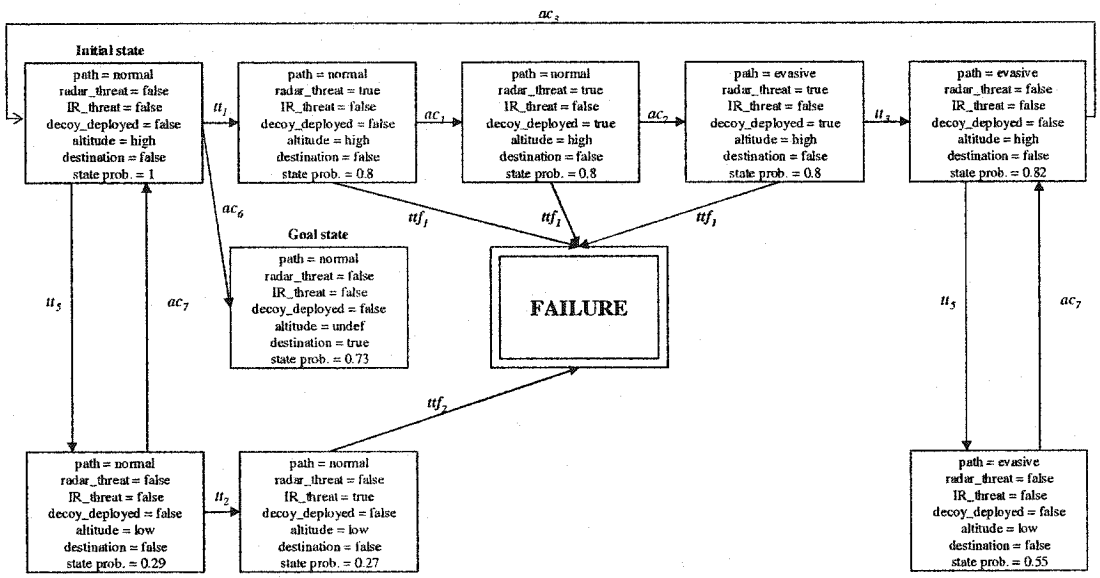


Figure 4-14: The State Diagram Ignoring the IR Threat; Utility = 0.73

4.13 Evaluation

We have evaluated how well the unlikely state strategy works in random domains. To this end, we have generated 500 random sample domains. Each domain has one knowledge base (Section 3.2). We then apply both the unlikely state strategy and the "random" strategy to the domain to generate a schedulable plan. We compare the utilities of the plans constructed by each of the strategies. The random strategy produces a schedulable plan by removing random actions in the initial plan until it becomes

schedulable. The random strategy produces a number of candidates and picks the best one. The plans produced by using the random strategy are the benchmark against which we measure the effectiveness of our strategy.

4.13.1 CIRCA Random Knowledge Base Generator

To generate a random KB, we use our random KB generator. It takes 12 parameters as inputs. We give each parameter random values within a range. Most of these parameters are self-explanatory but some parameters require explanations. A goal description is not a state; rather, like a precondition or a postcondition, it is a set of feature-value pairs. A state is a goal state if and only if its features match those in a goal description. The fewer precondition features there are in a transition (action/*tt*/*ttf*) and the less specific the transition is, the more states it can be applied to. The number of postcondition features tells how different a child state is from its parent state. The 12 parameters and their ranges are shown in Table 4-7.

Table 4-7: KB Parameters and Their Values

number of features	5 – 10
number of initial states	1 – 5
number of goal descriptions	1 – 10
number of features in goal descriptions	1 – 10
number of *tt*s	1 – 20
number of preconditions features in a *tt*	2 – 6
number of postcondition features in a *tt*	1 – 5
number of *ttf*s	1 – 15
number of preconditions features in a *ttf*	2 – 6
number of random actions	1 – 15
number of precondition features in an action	2 – 6
number of postcondition features in an action	1 – 5

To build a random KB, the generator first creates a set of binary (T/F) features. It then generates the initial states, goal descriptions, actions, and *tt*s and *ttf*s that manipulate these features based on the parameters. A state, a goal description, or a precondition of an action/*tt*/*ttf* is generated by randomly picking a subset of features from the feature set and values. The postcondition of a transition is generated by picking a subset of features in the precondition and inverting the values. In other words, the actions and temporal transitions flip a random number of the binary features.

To guarantee that each foreseeable *ttf* is possibly but not necessarily preemptible,[24] in addition to each random action, we add to a KB a preempting action for each *ttf*. The idea behind this is to ensure that when the planner fails to find a plan, it is because of schedulability of actions, rather than inability to avoid failures. The total number of actions is thus equal to the number of random actions plus the number of *ttf*s. As actions and *ttf*s are generated randomly, it could be possible that multiple actions have the same pre- and post- conditions. So, the number of distinct actions is accordingly smaller. The author has also repeated the experiments without inserting preempting actions. The results are similar.

4.13.2 Experiment Results

Our experiments show that the average utility, measured by the average safety probability of plans produced by using the unlikely state strategy, is 0.812 (stdev 0.231). On the other hand, the average utility of plans produced by using the random strategy is only 0.642 (stdev 0.352). That is, an agent can produce a plan having a utility on average 26.48% higher. The random strategy represents a lower bound of how good a schedulable

[24] If an agent does not have the ability (action) to preempt a *ttf*, then the *ttf* is not possibly preemptible.

plan on average can be when it is produced by removing enough actions until the initial plan becomes schedulable. We would also like to compare the unlikely state strategy to an upper bound.

To this purpose, we have generated 200 small random samples. These samples are small because they have only from 2 to 8 actions in their plans. We limit these experiments to only small samples because it is in general very time consuming to find the optimal plans for them using our enumeration method. For each sample, we enumerate and compare all possible schedulable subsets of actions, except those combinations that are themselves subsets of other schedulable subsets. Those combinations will not have higher utilities than some other schedulable subsets that we compare. The subset having the highest utility among all these enumerations is the optimal plan for this particular sample. We measure how close a heuristic final plan (produced by using the unlikely state strategy) is to the corresponding optimal plan for a KB by:

$$optimality = \frac{U_h}{U_{opt}} \qquad \textbf{eq. 4-27}$$

U_h is the utility of a heuristic plan, while U_{opt} is the utility of the corresponding optimal plan. For the 200 samples, the average optimality is 0.89, with a standard deviation of 0.20. In other words, the optimal plans are, on average, only 11% better than the heuristic plans. In fact, for 58.33% of the samples, using the unlikely state strategy does produce plans having utilities equal to the optimal utilities. For those samples that the unlikely state strategy does not find the optimal plans, the average optimality is 0.77, with a standard deviation of 0.24.

We note that the unlikely state strategy is a hill-climbing algorithm that tries to remove the actions that are the least significant to the agent's utility, as measured by how likely the actions are to be used, or the probabilities of the states that the actions are in. Thus, it in general will not find the optimal plans. In retrospect of the experiments, we found that using the unlikely state strategy often produces lower quality plans when it fails to remove the "insignificant" actions before other more significant ones.

Specifically, this strategy does not consider the probabilities of the hazards actually happening. For example, a hazard can have a very small probability to occur even though it is in a very likely to be visited state. The hazard could be preempted by the *tt*s in the state in addition to the action. Removing this action therefore makes little sacrifice to the agent's utility. Unfortunately, the unlikely state strategy may not find these insignificant actions when they are planned in the states having big state probabilities. On the other hand, the unlikely threat strategy (Section 4.11), which orders actions by the probabilities of their *ttf*s actually happening in ascending order, may produce better schedulable plans. Despite this, as we have shown in Section 4.11, the unlikely state strategy has the advantage over the unlikely threat strategy that the increment in failure probability is bounded each time an action is removed.

Moreover, while our empirical results justify that the unlikely state strategy is reasonably useful, we can devise better strategies by combining both state probability and other information. For example, instead of ignoring the least likely state, we could ignore the one that has a slightly higher state probability but much bigger utilization. Ignoring this other state, though incrementing the failure probability more, could allow much easier scheduling for the rest of the actions and ultimately gives a schedulable plan

having a higher utility. All these other strategies, however, should incorporate the state probability information developed in this dissertation.

4.14 Summary

To address the resource allocation problem for a real-time agent, we have in this chapter developed and evaluated the unlikely state strategy. The resource limited agent uses this heuristic to generate a feasible plan by iteratively dropping the least likely used actions. In other words, the agent ignores the *ttf*s by the probabilities of the states they are in. Each time an action is dropped, the decrement in utility is bounded by the probability of the state that the action, and hence the *ttf*, are in. Our empirical results show that the utilities of the schedulable plans produced by using this heuristic are on average 26.48% higher than the utilities of the plans produced by randomly removing actions.

To compute state probabilities, we have developed a probabilistic framework which models the temporal dynamics of actions and events in a real-time environment, where concurrently-enabled, mutually-exclusive, and time-dependent (e.g., periodic) activities are possible. To facilitate the computation of transition probabilities, we provide a discrete approximation using the probability rate functions to specify the temporal dynamics. With transition probabilities, we compute state probabilities analytically when the stochastic process of an agent is Markovian. Otherwise, we draw on tools from Operations Research and Statistics to simulate the stochastic process to estimate state probabilities.

Chapter 5

Improving Resource Allocation by Action Replacement

The unlikely state strategy reduces the schedule utilization of an overly constrained agent by dropping the least likely to be used actions. Each time an action is dropped, the failure probability increases (Section 4.11). This strategy can always make a plan schedulable, but as a result, some *ttf*s become unhandled. Essentially, what the unlikely state strategy does is to replace the actions by NOOPs. This may be overly aggressive because the agent might be able to replace the actions by some cheaper actions to reduce schedule utilization but at the same time preempt the *ttf*s. In this chapter, we generalize this action replacement technique to allow an unschedulable agent to replace actions by not only NOOPs, but also other eligible actions.

Our overall solution to solve the resource allocation problem for an agent, at the conceptual level, is to employ a hill-climbing algorithm to iteratively improve its preferred (unschedulable) plan until it becomes schedulable. The only operator needed to modify a plan is replacement: it replaces an action (or a NOOP) in a state by another action (or a NOOP). Multiple applications of this operator allow the agent to jump from any plan to any other plan in the search space of possible plans.

In other words, we append an improvement step to the agent's planning process. Comparing this approach to the opposite approach that tries to enhance the planning algorithm of an agent (by, e.g., adding the capability to reason about action utilization), our approach stays agnostic about whatever planning algorithm that the agent chooses to use. Moreover, a real-time agent, whose planning and allocating resource processes are

difficult to capture in one mathematical framework as they usually involve worst-case analysis, can use our approach to decouple these two processes.

More importantly, repairing an unconstrained plan to meet resource constraints enables the agent to improve its plan by heuristically exploiting new information that is available only after producing the initial plan. In principle, we could do an uninformed search by trying all possible combinations of replacements to find the optimal resource-satisficing plan. Yet, as we will show, the complexity of modifying a plan in this way is the same as planning from scratch. It is intractable.

Using post-planning information, we can make the search informed to avoid examining all possible action replacements. A single-agent CIRCA uses two types of post-planning information. In Chapter 4, the agent uses the probability information to remove unlikely actions. In this chapter, when an agent produces its initial, preferred plan, it will have found a set of actions that preempt all *ttf*s. By analyzing the action costs in this plan, the agent replaces actions by cheaper ones that can still preempt the *ttf*s. Additionally, in a multiagent environment, the agent exploits the new knowledge it obtains by communicating with other agents (Chapter 7).

We assume that there is a monotonic non-decreasing relationship between the utilization of a plan and the utility. That is, as the resource utilization decreases, the utility may also decrease (but never increase). This is justifiable for a rational agent that always attempts to select actions leading to a higher utility but having a lower utilization. Moreover, the replacing actions are supposedly less desirable than the replaced actions. Otherwise, the rational agent would have selected them in the first place. Thus, each time that the agent makes a replacement, it reduces utilization at the cost of lowering utility.

This assumption is true only for a perfectly rational agent who is able to generate optimal plans. As we will see from our experiments in Section 5.5.2, this assumption is sometimes violated in practice.

Here we develop the action replacement technique to improve resource allocation of an agent. We develop heuristics to prioritize both which actions to replace and, for each, which replacing actions to try. We also examine some constraints to keep the search space tractable in terms of controlling the size. The objective is to reduce schedule utilization but at the same time maintain the utility as close to that of the initial, preferred plan as possible.

In the following, we begin by analyzing the complexity of replacing actions to find an optimal plan to show that optimization in general is intractable (Section 5.1). We must therefore restrict the general action replacement technique. We introduce the CIRCA resource allocation algorithm in Section 5.2. We illustrate by an example in Section 5.3 the concept of a reusing action and how it helps replace actions by cheaper alternatives. In Section 5.4, using this concept we develop the reusing action strategy to address the problem of tractably replacing actions to reduce utilization. In Section 5.5, we evaluate the reusing action strategy by applying it to random domains, and we determine the factors that affect its performance.

5.1 Complexity of Improving a Plan by Action Replacement

There are as many possible combinations of action replacements as there are possible plans. Consequently, it is in general very difficult to generate an optimal plan by iteratively improving an unconstrained plan. Let s be the number of states in a plan; t be the number of action candidates that do not introduce new reachable states from each

state (including the null action). Theoretically, the number of state-action pairs that the agent can change in the plan, i.e., the number of possible combinations of action replacements, is:

$$|P| = t^s \qquad \textbf{eq. 5-1}$$

Eq. 5-1 is the same as the number of possible plans when the agent is planning using only the same subset of actions used for replacements. In any non-trivial domain, the exponent s tends to be big. Unless there is some domain knowledge that enables the agent to identify the best plan that maximizes utility, exhaustively exploring this search space to find an optimal plan under resource constraints is prohibitive.

Now, eq. 5-1 assumes that no new states are introduced into the initial state diagram, or equivalently, that the actions that replace other actions in a plan do not lead the agent into new reachable states. Otherwise, the exponent factor could be closer to the number of representable states, s', instead of the number of states in the original plan. Presumably, the set of representable states is (much) bigger. The search space would increase exponentially by $\frac{s'}{s}$, that is,[25]

$$|P| = t^{s'} = \left(t^s\right)^{\frac{s'}{s}}, \qquad \frac{s'}{s} \geq 1 \qquad \textbf{eq. 5-2}$$

Eq. 5-2 is the same as the number of possible plans! Although we need only one simple replacement operator to explore the search space, the size is huge. Moreover, not only is it very expensive for an agent to enumerate all possible modifications to its plan, but evaluating each new plan could itself be a complicated process. To evaluate a plan,

[25] For simplicity, we ignore the fact that there are now more actions in a state for the agent to choose from.

the agent needs to compute the probabilities of reaching states and schedule the actions. Scheduling actions is in general an NP-hard problem [82]. Depending on the structure of the state diagram, the complexity of computing the probabilities ranges from $O(n)$ to $O(n^4)$, where n is the number of states (Section 4.7.2). Clearly, it is not generally practical to do this evaluation on all possible revisions.

5.2 The Action Replacement Algorithm

As finding an optimal plan under resource constraints is in general intractable, our solution compromises optimality for tractability by using an iterative hill-climbing algorithm that might only find a locally optimal plan. The search begins with the most preferred plan that the agent can come up by ignoring resource constraints, and then makes incremental improvements to the plan. It replaces actions in some states by other actions to reduce schedule utilization without overly harming its utility.

To avoid considering all possible action replacements in each step of the hill-climbing search, the algorithm employs heuristics. First, the algorithm uses post-planning information to limit the scope of replacing actions that it selects. That is, it tries to reduce the branching factor, t, in eq. 5-1. Despite this, the agent still needs to examine all combinations of replacing actions in all states. The time complexity class remains exponentiation.

Thus, secondly, for each to-be-replaced action, *ac*, the algorithm, again using post-planning information, picks the most promising action, *ac'*, in terms of producing the best final schedulable plan. This action can be a NOOP or *ac* itself. The algorithm tries to replace *ac* by *ac'*. If the plan is improved, the replacement is adopted and hill-climbing iterates. Otherwise, the algorithm tries the next promising action until the last

available alternative is tried. Each action replacement decision is considered only once. Essentially, it is a greedy algorithm that considers one replacement at a time, ignoring the combinatorial effects of choosing different replacing actions. The complexity reduces from exponentiation to multiplication.

$$|P| = ts \qquad \text{eq. 5-3}$$

The algorithm is shown in Algorithm 5-1. It encapsulates the three fundamental questions about the replacement operator. Line 1 asks "Which action to replace next?" Line 3 asks "Which action to replace it with?" The third question "What constraints should we place on the search?" is related to the second one. There are complications about replacing an action as we will discuss shortly. We need to limit the search for replacing actions to keep the algorithm tractable. There are no best answers to these questions. The answers depend on the post-planning information the algorithm exploits.

Input:

an unschedulable plan

Output:

an improved plan

Algorithm:

```
1. order the actions in the plan using an action ordering heuristic;
2. for (each action, ac, to be replaced) {
3.       select a replacing action, ac', using a replacement heuristic;
4.       evaluate the replacement;
5.       if (the plan can be improved) {
6.            commit the replacement; //in the agent's plan and state diagram
7.       }
8. }
```

Algorithm 5-1: The Action Replacement Algorithm

In this chapter, we focus on heuristics that replace actions by cheaper actions that are already selected in the plan. A replacing action must preempt the *ttf*s that the replaced action is preempting in a state. Therefore, we do not allow a replacing action to be a NOOP. We call this particular usage of Algorithm 5-1 the *reusing action strategy* because the idea behind it (i.e., the strategy used) is to exploit the possibility of reusing actions that are already in a plan. There is no guarantee, however, that the resultant plan will be schedulable.

We have already developed, in Chapter 4, the unlikely state strategy. The unlikely state strategy is simply another usage of the action replacement algorithm. Its replacement heuristic is simple: it only replaces an action by a NOOP. In contrast to the reusing action strategy, the unlikely state strategy ignores *ttf*s instead of handling them by less desirable actions. So, while the unlikely state strategy always makes a plan schedulable, we argue that it should only be used as a last resort.

We of course can combine both the post-planning action cost and probability information. In fact, the CIRCA resource allocation algorithm is built with the reusing action strategy and the unlikely state strategy. The algorithm is shown in Algorithm 5-2.

Input:
an unschedulable plan
Output:
a schedulable plan
Algorithm: 1. apply the reusing action strategy; 2. if (the plan is still unschedulable) { 3. apply the unlikely state strategy; 4. }

Algorithm 5-2: The CIRCA Resource Allocation Algorithm

Please note that the algorithm is simply another usage of Algorithm 5-1, though it has more complicated heuristics. It orders the actions to be replaced in such a way that the actions that can be replaced by cheaper actions in the plan have priority over those that do not. The replacement heuristic says that if an action can be replaced, it is replaced by applying the reusing action strategy. Otherwise, it is replaced by a NOOP, i.e., being dropped. We are now going to discuss the reusing action strategy.

5.3 Example of Reusing Actions

To avoid searching for and evaluating all possible action replacements, we want to identify the pairs of replaced and replacing actions that are most likely to reduce schedule utilization. Conceivably, there can be many cheaper alternatives for an action. For example, any other actions in the knowledge base that have smaller utilizations and preempt the same *ttf* are possible candidates. Fortunately, we can narrow down the choices to reduce the branching factor of the search. We restrict our algorithm to consider only actions that are already included in the plan.

We study the concept of reusing actions. Specifically, for a real-time system like CIRCA, as all actions are scheduled on the same shared execution platform, planning/reusing an action that is already in the plan has a minimal marginal cost. For simplicity, we can think of the marginal cost as zero. After the agent produces an initial plan that includes the actions that preempt all *ttf*s, it can analyze the state-action pairs to find out whether some of the preemptions can be accomplished by other planned actions. The agent can reuse those other actions for the tasks to reduce schedule utilization. We illustrate this concept using the following example.

Suppose there is an unmanned vehicle cruising in downtown. In our simplified example shown in Figure 5-1, the agent may encounter five different scenarios: jay-walking kids appearing suddenly, cars cutting lane, red lights, yellow lights, and one way streets (right turn only). As long as the agent stays in downtown, it may run into any or all of these situations periodically. In terms of state diagram, as long as the agent is looping in the downtown region (DOWNTOWN = T), it may visit any or all the dangerous states one or multiple times. Thus, the agent must continuously monitor its surrounding, determine its responses for, and react to these events.

To assure complete safety, this vehicle has to be prepared for (and allocate resources to the actions for) all these contingent events, whether or not they actually do happen. The responses to the different events are shown in Figure 5-1. We summarize the plan, i.e., the state-action mapping, below.

- When (there is an unexpected object, e.g., a pedestrian or a car), REDUCE-SPEED to avoid collision.
- When (there is a red light), STOP to avoid a ticket.
- When (there is a yellow light), SPEED-UP to avoid a ticket.[26]
- When (there is a one-way street, right turn only), TURN-RIGHT.

For the sake of simplicity, let us assume that all TAPs have the same utilization. For example, the agent may watch out for and test each situation (and react if appropriate) every 6 seconds; each TAP has a worst-case execution time of 2 seconds. The utilization is 0.33 for each TAP; the total utilization is 1.33 so the plan is unschedulable.

[26] For reckless drivers, 'green' means 'go'; 'yellow' means 'go faster'.

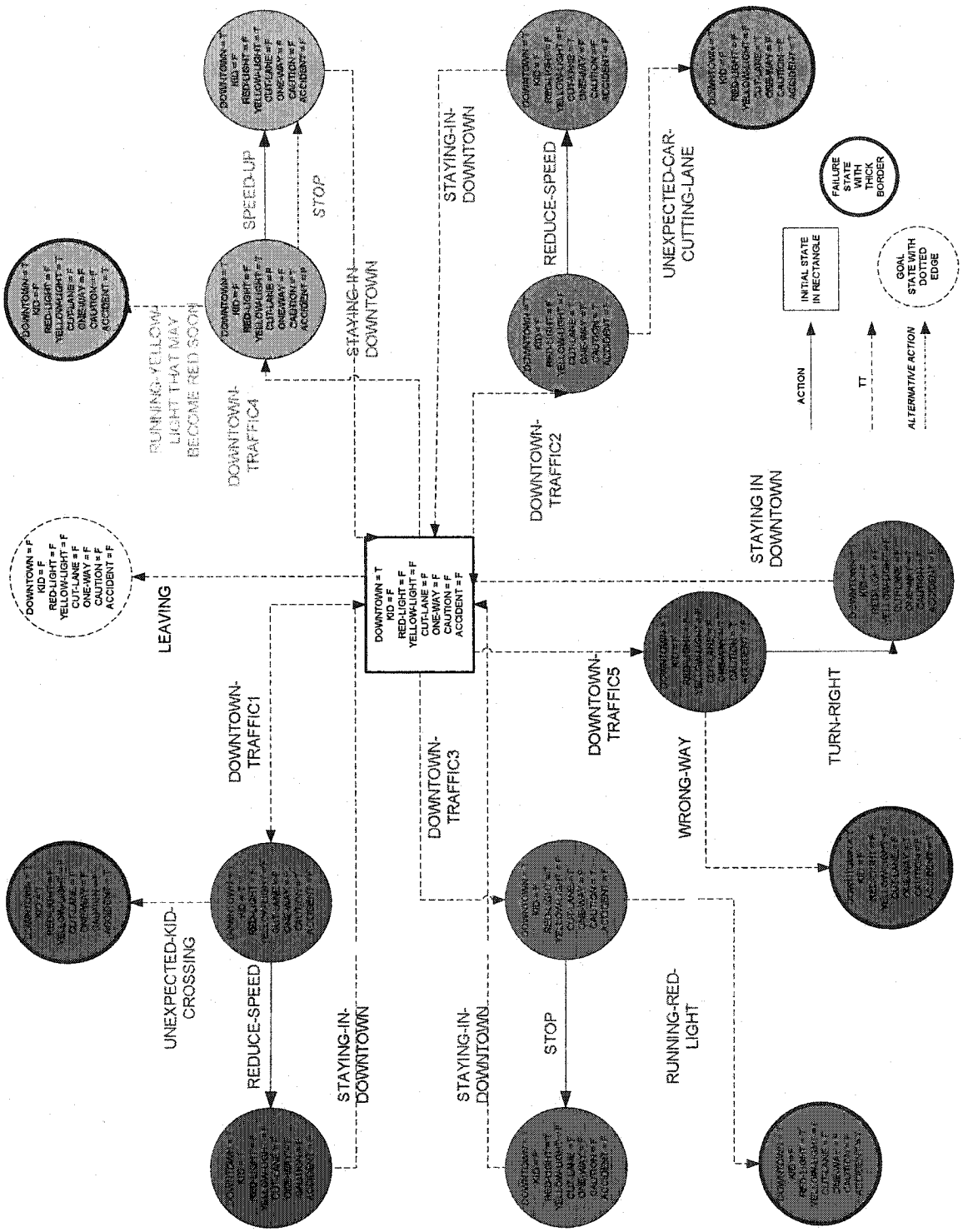


Figure 5-1: A Cruising Car around Town

Unfortunately for this vehicle (but fortunately for the residents), it has only enough resources to schedule 3 distinct actions (TAPs). In the original plan where the vehicle selects its most preferred action in each state, there are 4 TAPs in total. This plan cannot be executed due to over-utilization.

To reduce utilization, this vehicle can try to replace some actions by other actions that are also in the plan. After analyzing the preferred plan, the vehicle finds out that it could have used STOP to avoid running the yellow light that may become red soon.

Presumably, SPEED-UP is the preferred option; STOP is not, otherwise, the agent would have selected STOP initially. For example, the agent may like to run through yellow lights because it wants to spend less time waiting in traffic (this context is not shown in the state diagram in Figure 5-1; the agent's planning algorithm would need to understand its preference). Although STOP is not its preferred action, replacing SPEED-UP by STOP can reduce its resource requirement by 1 TAP. The agent now needs only to schedule 3 TAPs having a total utilization of 0.999; hence its resource constraint is satisfied. The schedulable plan after replacement is:

- When (there is a one-way street, right turn only), TURN-RIGHT.
- When (there is an unexpected object), REDUCE-SPEED.
- When (there is a light which is not green), STOP.

Essentially, reusing an action amounts to combining the recognition parts of multiple state-action mappings into the recognition part of TAP of the reusing action. In general, each of the state-action-mappings that the agent identifies during its planning process has a different period. When one or multiple state-action mappings, having different actions, could instead use the action of another state-action mapping, the agent

may combine them into one TAP of that action. This combined or replacing TAP will have the minimum of the periods, or equivalently the maximum of the frequencies, of those mappings. The TAP's worst-case execution time remains the same.

Consequently, reusing an existing action in a state is free when the state does not require the TAP of the action to have a shorter period to preempt the failure transition in the state. Otherwise the replacing TAP will have a greater utilization than before, but the utilizations of the dropped state-action mappings are reclaimed.

A secondary effect of reusing an action is that the time required to do the recognition test of the TAP may be reduced, and hence the utilization of the TAP is reduced. Rather than matching to a particular state, the replacing TAP's recognition only needs to classify the current state that the agent is in as to whether it is among the states that trigger the reaction or not. The test can thus be represented as a decision tree (constructed with ID3 [87]) where the number of state features to check to decide whether the action should be taken or not is never greater than the number of features to check for a specific state.

The recognition time of the TAP's test is never longer than for any single initial state-action mapping. So, the utilization of the replacing TAP is not greater than the utilization of any initial state-action mapping. In the extreme case where all states require the same action, the agent needs not even spend any resources to do the recognition for information gathering. It does not need to know which state it is in and can simply execute the same action.

For the example in Figure 5-1, it takes the same amount of time for the vehicle to check whether a light is red, yellow, or green. As the agent is checking to see whether the

light is red every 6 seconds and is checking *separately* whether the light is yellow every 6 seconds, it can simply check whether the light is (not) green every 6 seconds. That is,

(... RED-LIGHT = T ..., STOP)[27]

(... YELLOW-LIGHT = T ..., SPEED-UP)

After reusing actions, these two state-action mappings become:

(... RED-LIGHT = T ..., STOP)

(... YELLOW-LIGHT = T ..., STOP)

By combining the recognitions, we have:

(... RED-LIGHT = T or YELLOW-LIGHT = T ..., STOP), or equivalently,

(... LIGHT != GREEN ..., STOP)

So, instead of spending the resources on two TAPs, the agent now needs to schedule only one.

CIRCA distinguishes between preferred actions and reusing actions of a TAP. An action in a state is a *preferred action* if it is the "best" action in that state according to whatever planning algorithm the agent uses. For example, both REDUCE-SPEED actions in the example are preferred actions in the two states. Otherwise, an action is a *reusing action* if it is chosen to reduce utilization because the agent has already planned the same action in other states as a preferred action. The TAP is already scheduled. For example, the STOP action responding to the yellow light is a reusing action of the STOP TAP.

[27] The ellipses denote other features.

5.4 The Reusing Action Strategy

When an over-utilizing agent is searching for modifications to the initial, unconstrained plan to make it schedulable, it should try to make as minimal a decrement to the utility as computationally feasible. Yet, as we have shown in Section 5.1, repairing a plan using the replacement operator to make it optimal is intractable.

We use the reusing action strategy to make the search tractable. The particular type of post-planning information we exploit comes from the intuition in Section 5.3 that reusing actions is cheap. When all actions are scheduled on the same shared execution platform, e.g., the CIRCA RTS (Chapter 3), replacing the preferred actions for some tasks by other actions that are already in the plan can have a *marginal* cost of zero if the replacing actions have longer periods in the new states. For example, a new action could simplify the recognition test for its TAP by requiring that fewer state features be observed because the TAP applies in a broader set of states. We demonstrated this in the last section using the example in Figure 5-1. Reusing an action could sometimes incur additional cost if it has a shorter period for the new state.

Specifically, when an agent has enough resources, it could choose different preferred actions in different states. Otherwise, the agent examines the set of state-action pairs to see if there is an action in the set that satisfies the needs of multiple states that have different preferred actions. If this action incurs a smaller cost than the total cost of the different actions together, the agent can replace the actions with the single action to reduce utilization.

The reader may wonder why we limit the search for replacing actions to those actions that are already in the plan rather than any cheaper actions. First, it reduces the

branching factor t in eq. 5-3. More importantly, a TAP usually applies in multiple states. Replacing each of the actions with another planned action does not incur any marginal cost. The sum of the marginal costs of all replacing actions is zero. The schedule utilization will always decrease after replacements.

On the other hand, if we replace the actions of the TAP by cheaper actions that are not already in the plan, the marginal cost of each replacing action is not zero. Although each replacing action may have a smaller cost than the replaced action, the sum of the costs of the replacing actions may exceed the cost of the replaced TAP. The schedule utilization may increase after replacement. An illustrating example follows shortly.

Conceptually, this strategy or particular usage of Algorithm 5-1 exploits the fact that, after initial planning, the set of preferred actions that preempt all foreseeable *ttf*s is found. The agent then tries to find a maximal subset that is schedulable and still accounts for all the contingencies. In the remainder of this section, we will discuss, for the reusing action strategy, various heuristics that select replaced and replacing actions and constraints that limit the size of a search space.

5.4.1 Heuristics for Deciding Which Actions to Replace

As Algorithm 5-1 considers each replacement decision in a state only once and does not backtrack (line 2), the ordering of actions to be considered is important (line 1). For instance, let us consider the example in Figure 5-2. In contrast to the simplified example in Figure 5-1 where all TAPs have the same utilization, now the TAPs have different utilizations (costs) as shown in the parentheses. The preferred plan in Figure 5-2 is the same as the one in Figure 5-1. The total utilization of the preferred plan is 0.2

(TURN-RIGHT) + 0.4 (REDUCE-SPEED) + 0.1 (STOP) + 0.4 (SPEED-UP) = 1.1. This plan is not schedulable because the utilization exceeds 1. Also, the actions in this example have different alternative actions than the ones in Figure 5-1. There are two possible action replacements in Figure 5-2.

- REDUCE-SPEED (0.4) → STOP (0.1) (2 copies)
- STOP (0.1) → TURN-RIGHT (0.2)

The order in which these two replacements are considered is important here. If REDUCE-SPEED is considered first, then the other replacement will not further decrease the schedule utilization. After the replacement, there are three STOP actions in the plan. Replacing one STOP by TURN-RIGHT has no effect. On the other hand, if STOP is considered first, then the other replacement cannot be done. After the replacement, STOP will not be in the plan. The agent can no longer replace REDUCE-SPEED by reusing STOP. Therefore, depending on the action ordering heuristic used (line 1 in Algorithm 5-1), the final plan and hence the final schedule utilization will be different.

The motivation behind replacing actions by cheaper alternatives is to reduce the utilization of an overloaded plan. We usually want to replace large utilization actions by small utilization actions. Thus, we want to give priority to actions having larger utilizations. In the example above, the ordering is: REDUCE-SPEED (0.4), STOP (0.1). The two REDUCE-SPEEDs can be replaced by STOP. After the replacement, there are three STOP actions in the plan. The new utilization is 0.2 (TURN-RIGHT) + 0.1 (STOP) + 0.4 (SPEED-UP) = 0.7. The plan becomes schedulable.

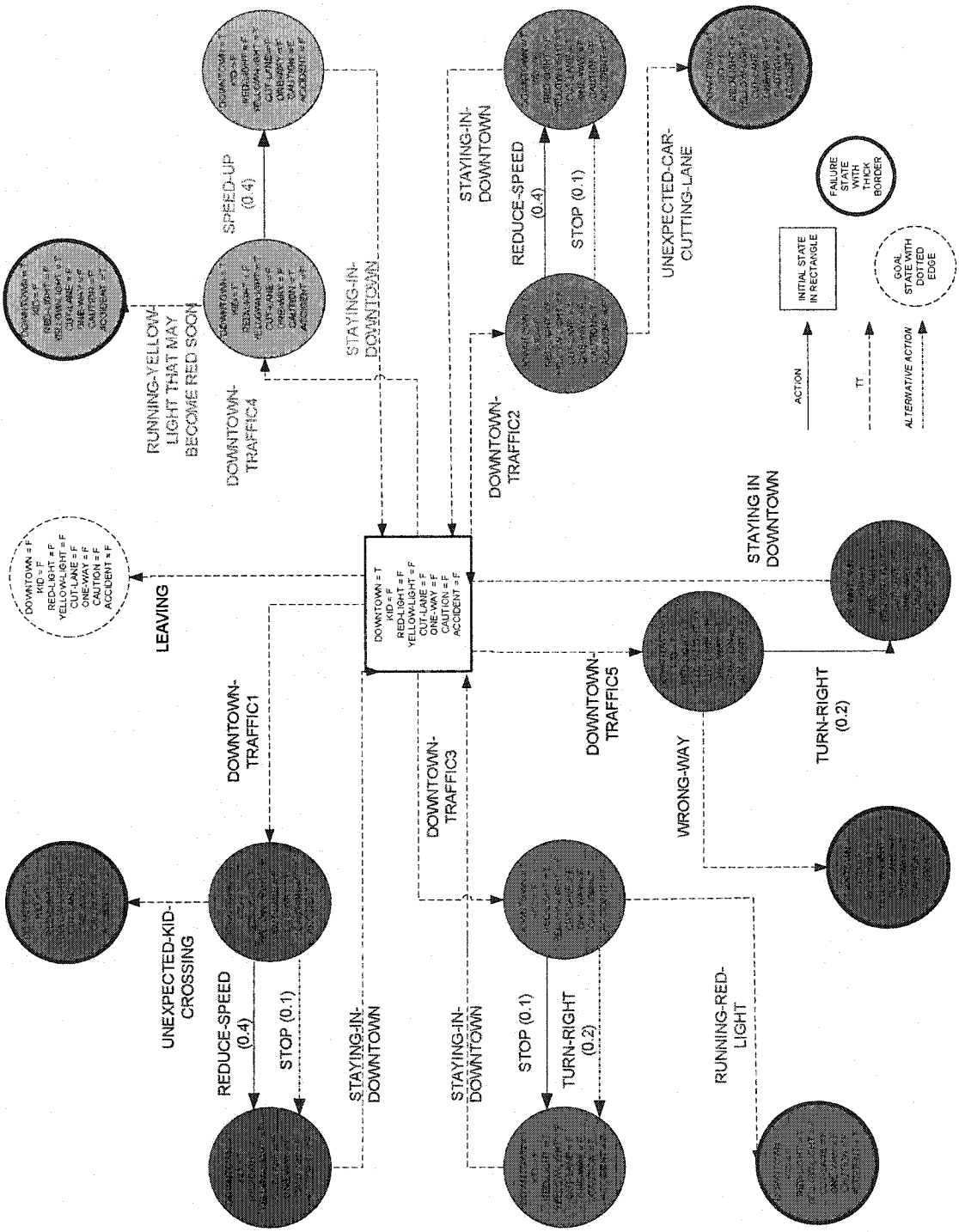


Figure 5-2: Selecting Actions to Replace (E.g., 1)

In this example, the TAP REDUCE-SPEED is completely removed because each REDUCE-SPEED action has an alternative action, namely, STOP. In general however, not all actions of a TAP necessarily have eligible alternatives. A slightly modified example is shown in Figure 5-3. One of the REDUCE-SPEED actions cannot be replaced. The only available alternative action in that state is SWERVING. SWERVING is not a planned action, so it cannot be reused. Although it has a smaller cost (0.2) than REDUCE-SPEED (0.4), replacing this REDUCE-SPEED by SWERVING will actually increase the schedule utilization by 0.2.

The previous ordering by utilizations thus turns out to be a bad ordering for this example. Not only does making the REDUCE-SPEED replacement not reduce schedule utilization, but it also prevents the other replacement from happening. Although replacing some actions of a TAP may decrease the utilization somewhat (the decrement is 0 in this simplified example where we ignore action period, c.f., Section 5.4.3), the decrement is always less than the cost of the TAP. Also, the fewer TAPs there are in a plan, the smaller the utilization of the plan is. We thus in general want to remove all copies of actions of a TAP to remove the TAP entirely from the schedule when possible.

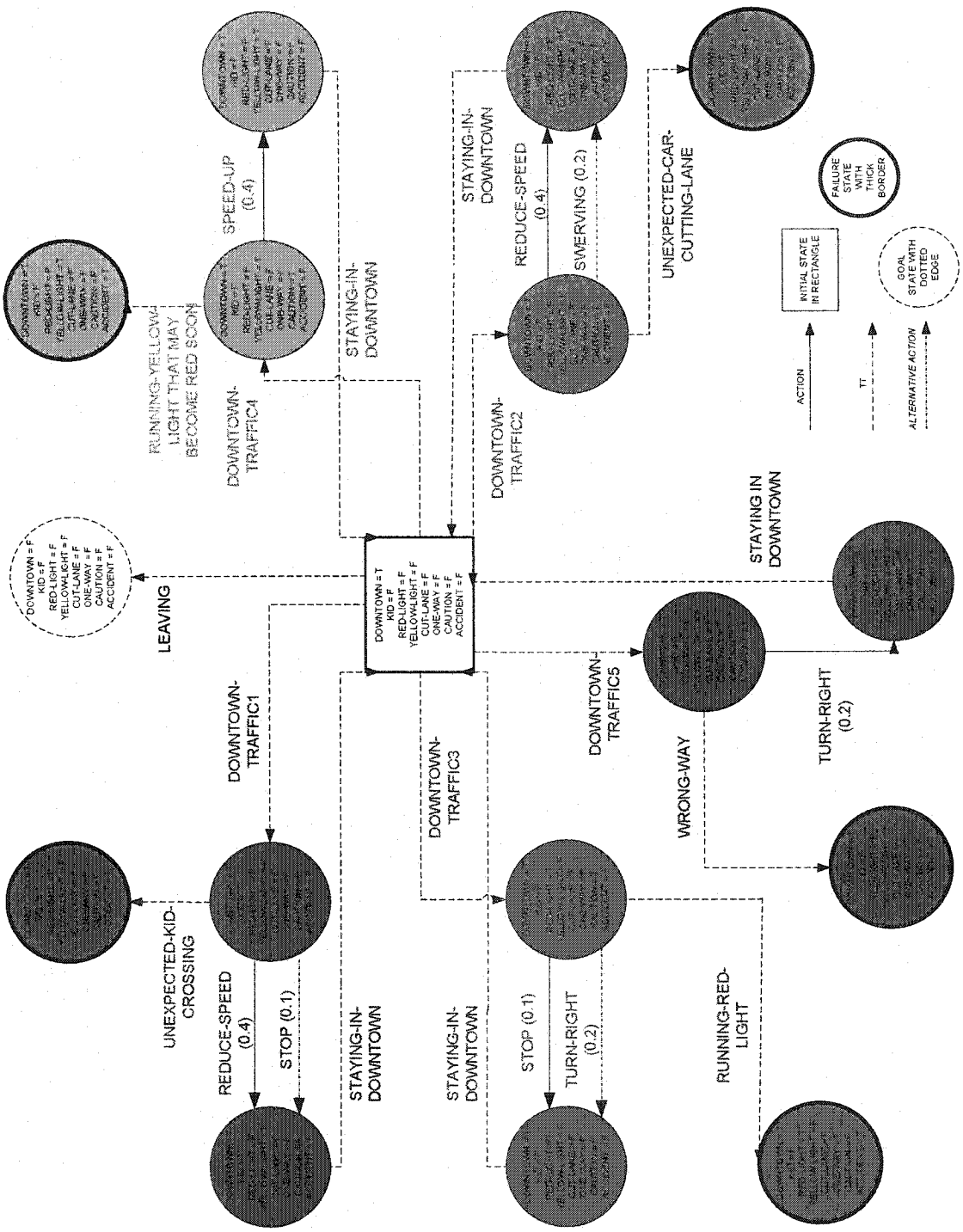


Figure 5-3: Selecting Actions to Replace (E.g., 2)

We define the *reference count* of a TAP as the number of states the TAP appears in.[28] In general, it is easier to remove a TAP that has a smaller reference count. Put another way, it is very hard to remove a TAP entirely when it appears in many states. The chances of replacing all copies of a TAP with a smaller reference count are better than for a TAP with a larger reference count. So, we want to try to replace first those actions that have fewer copies. In the example in Figure 5-3, the best ordering is: STOP (1 copy), REDUCE-SPEED (2 copies). The new utilization is 0.2 (TURN-RIGHT) + 0.4 (REDUCE-SPEED) + 0.4 (SPEED-UP) = 1.0. The plan becomes schedulable.

Therefore, to select an action to replace, the agent has to consider these two factors of an action: utilization and reference count. We summarize our two alternative action ordering heuristics in the following.

1. Order the actions in descending order of utilizations. This heuristic tries to decrease the schedule utilization in a greedy manner by replacing large utilization actions with small utilization actions.
2. Order the actions in ascending order of reference counts. This heuristic tries to remove as many TAPs as possible on the basis that a schedule having fewer TAPs has a smaller utilization. It is also easier to remove TAPs that appear in only a few states, e.g., one state.

5.4.2 Heuristics for Deciding Which Actions to Reuse

After picking an action to replace, we consider what alternative actions in the state can substitute for the action. A replacing action must be already in the plan and must preempt the *ttf*s in the state. Thus, in terms of failure avoidance, all eligible actions are

[28] The term 'reference count' is borrowed from C++ smart pointer. It keeps track of how many times an object is being referenced, i.e., being pointed to.

equally good because they all preempt the *ttf*s. On the other hand, by the assumption of monotonic relationship between utility and utilization for a perfectly rational agent, there is a cost for making each replacement.

For CIRCA, the cost is a decrement in goal probability because a less preferred action leads more indirectly and/or less likely to the goal states. Ideally, we want to make a replacement such that the goal probability decreases minimally. Unfortunately, as we have mentioned, computing state probabilities can be very expensive operation. It would not be practical to carry out this computation to evaluate every alternative action for every replacement, especially in a large domain having many states. We must resort to using heuristics.

As the motivation of the reusing action strategy is to reduce schedule utilization as much as possible, we in general want to replace a large utilization action by a small utilization action. For the example in Figure 5-4, SPEED-UP can be replaced by either STOP (0.1) or TURN-RIGHT (0.6). When all other factors are equal, we should choose STOP, which has a smaller cost.

Additionally, when choosing a replacing action, we consider whether the replacement decision still makes sense if the agent needs to apply the unlikely state strategy to make the plan schedulable. A replacement is bad if the preferred actions of a TAP are dropped, but the reusing actions still stay in the schedule. The TAP could have been dropped entirely and there would have been fewer TAPs in the schedule if the replacement had not been done. The agent should have replaced the action with another candidate.

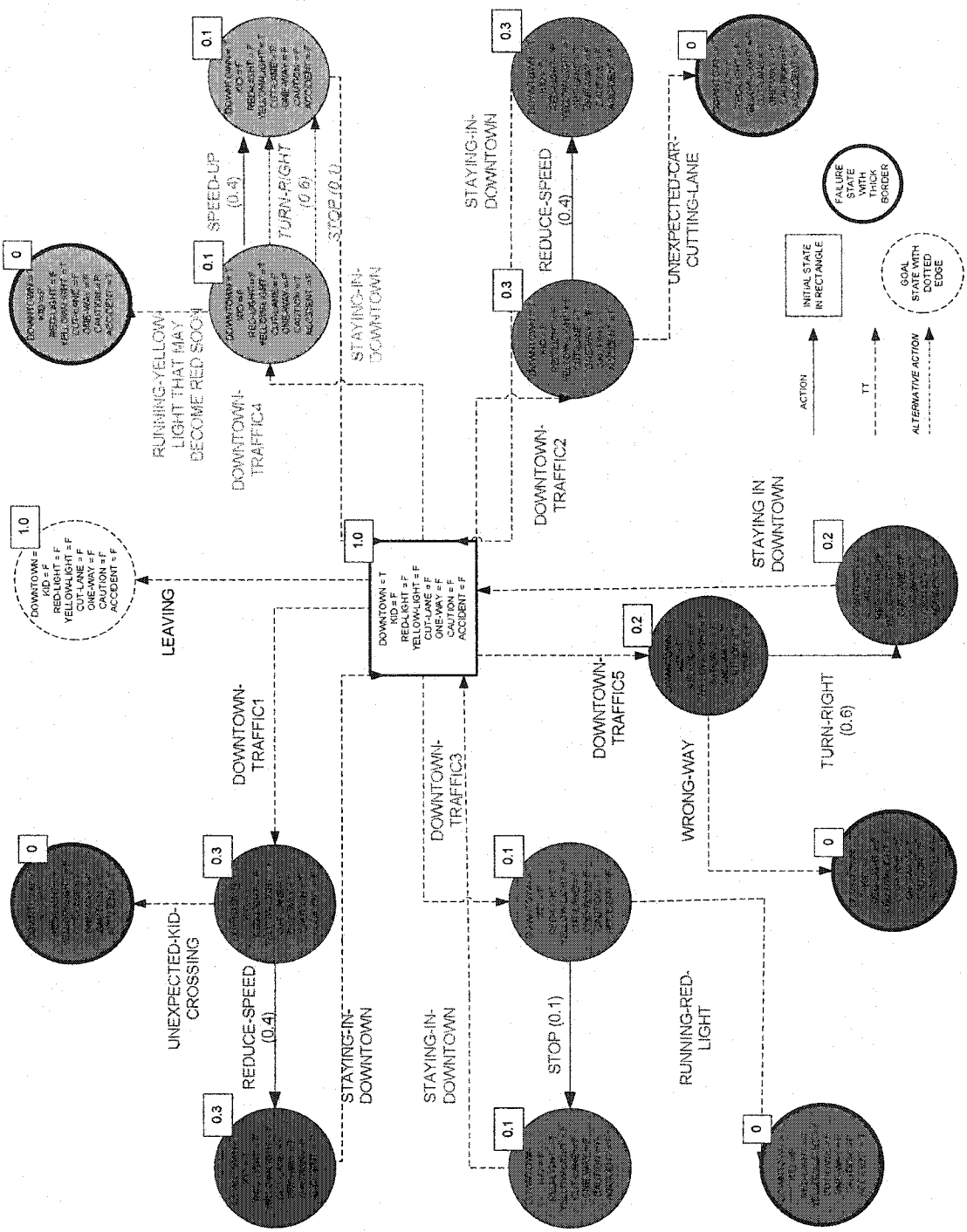


Figure 5-4: Selecting a Replacing Action

Let us consider again the example in Figure 5-4. So far in our previous examples, we have ignored state probability information. The probabilities of the agent visiting different states are shown in the upper right hand corners of the states in Figure 5-4. Note that the action TURN-RIGHT has a bigger cost, 0.6, in this example. So, after replacing SPEED-UP by STOP, the plan consists of TURN-RIGHT (0.6), REDUCE-SPEED (0.4), and STOP (0.1). The utilization is 0.6 + 0.4 + 0.1 = 1.1. The plan remains unschedulable. The agent will apply the unlikely state strategy to drop STOP, the least likely used action.

The replacement turns out to be a bad decision. The agent could have been schedulable after dropping the preferred STOP action, hence the TAP, because the TAP had only one action in the plan. Now, the agent needs to drop also the reusing action STOP to make the plan schedulable. If the agent could foresee that it is over-utilizing the resources too much and would remain unschedulable after applying the reusing action strategy, it would have chosen TURN-RIGHT instead even though it has a bigger utilization. In this case, the plan would become schedulable after dropping the only STOP action. The utilization of the plan is 0.6 (TURN-RIGHT) + 0.4 (REDUCE-SPEED) = 1.0. Therefore, a replacing action should never be an action that does not go into the final schedule.

In general, it is very important to choose a replacing action that is most likely to stay in the final schedule or equivalently, the least likely to be dropped. Otherwise, as we demonstrated by the above example and will prove in Section 5.5.1, the agent may unfortunately produce a worse plan by using the reusing action strategy. This is confirmed by our experiments in Section 5.5.2. The experiments in Section 5.5.4 further illustrate the importance of reusing actions that will stay in the plan.

Yet, the agent cannot know which actions will be dropped until it applies the unlikely state strategy. To estimate how likely it is that a TAP will stay in the final schedule, we can use either 1) the TAP probability (the highest probability of the state that the TAP appears in), or 2) the reference count of the TAP. Despite this effort to avoid picking replacing actions that will get dropped, it may still happen. For example when in a state there is only one eligible alternative action, the agent will make the replacement to reduce utilization regardless of how low a probability this action has. Thus, we have these three alternative replacement heuristics to choose a replacing action.

1. Choose a replacing action having the largest state probability. Actions in states having large probabilities tend to stay in the plan because the agent drops actions in ascending order of state probabilities.
2. Choose a replacing action having the biggest reference count. All copies of a TAP need to be removed before the TAP is removed from the schedule.
3. Choose a replacing action having the smallest utilization. We try to reduce the schedule utilization as much as possible.

5.4.3 Constraints on Replacing Actions

It might seem rather straightforward for an agent to choose a replacing action. The agent simply needs to compare t actions (c.f. eq. 5-3) using one of the three replacement heuristics above. This simplistic picture hides the important details that we are going to discuss in this section. First, the child state of a replacing action may not already be in the state diagram of the original plan. The agent needs to expand this new state by applying its planning algorithm as if it were an initial state. This could lead the agent into an

entirely new portion of the state space that it has not explored before. The agent must plan actions for any new states having *ttf*s found in the planning process.

Conceptually, if no new state arises, the agent replaces an action by only one action. Otherwise, if new states arise, the agent may replace an action by multiple actions. For the example in Figure 5-5 where new states are allowed, a new situation is introduced if the agent replaces SPEED-UP by TURN-RIGHT. Another kid may appear after performing TURN-RIGHT. The agent can STOP to avoid hitting the kid. Note that SPEED-UP cannot be replaced by TURN-RIGHT alone. SPEED-UP is replaced by multiple (two) actions – TURN-RIGHT and STOP. Both reusing actions are already in the plan.

The agent has two options to handle new states. Either it can run its planning algorithm without restriction as if it were an independent process, or it can restrict the planning algorithm to choose only actions that are already in the current plan. In the former case, while the marginal cost of reusing an action is zero, the total cost of the new actions added to the schedule to handle new states may exceed the reduction from making the replacement. For instance, a slightly modified version of Figure 5-5 is shown in Figure 5-6. The agent avoids hitting the kid by SWERVING. Yet, this new action, SWERVING, increases the schedule utilization by 0.5, which is bigger than the reduction by replacing SPEED-UP (0.4) with TURN-RIGHT. Note that TURN-RIGHT is already planned, so its marginal cost is 0.

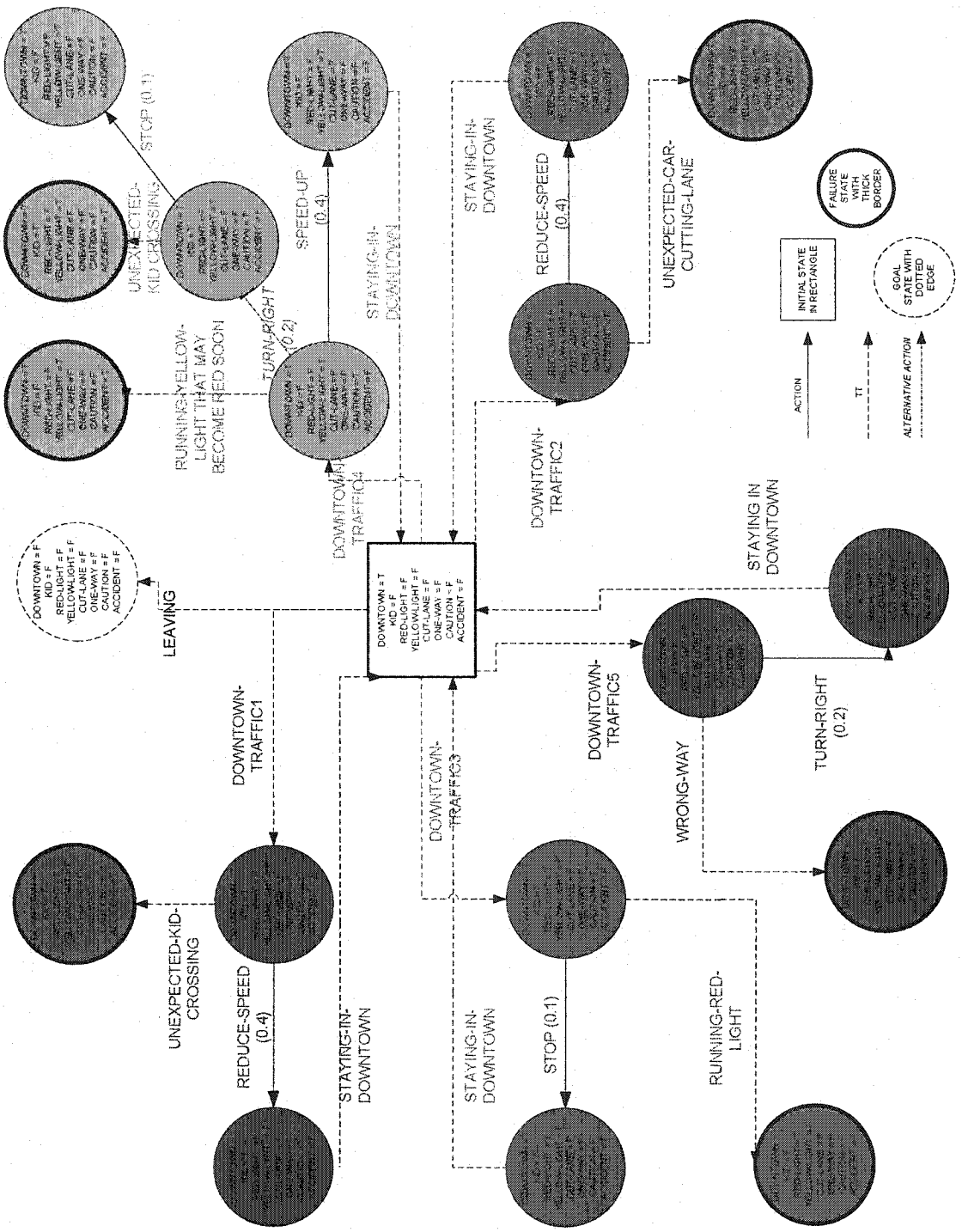


Figure 5-5: Replacing an Action by Multiple Actions

In the latter case, when some new states cannot be handled by only the actions in the plan, the replacing action is invalid because it introduces a failure that cannot be preempted. For the example in Figure 5-6, if the agent is not allowed to introduce the new action SWERVING that is not already in the plan, it cannot preempt the new *ttf* UNEXPECTED-KID-CROSSING. The agent eliminates this replacing action and considers another candidate (if any).

We summarize the possible options that the agent can make regarding introducing new actions.

1. Allow no new states. In other words, the agent chooses only actions whose child states are already in the state diagram. Consequently, there is no need to introduce new actions. This option limits the agent to replacing an action by one other action.
2. Allow new states but no new actions. The agent allows new states to be introduced as a result of replacing actions, but it handles the new states by only actions that are already planned. This option enables the agent to replace an action by multiple actions.
3. Allow new states and new actions. The agent essentially treats the child state of the replacing action as an initial state and runs its planning algorithm without restriction. The total cost of new actions may out-weigh the reduction in utilization, however. We do not consider this option in the thesis as we focus only on those replacing actions having a marginal cost of zero because they do not increase schedule utilization after action replacement.

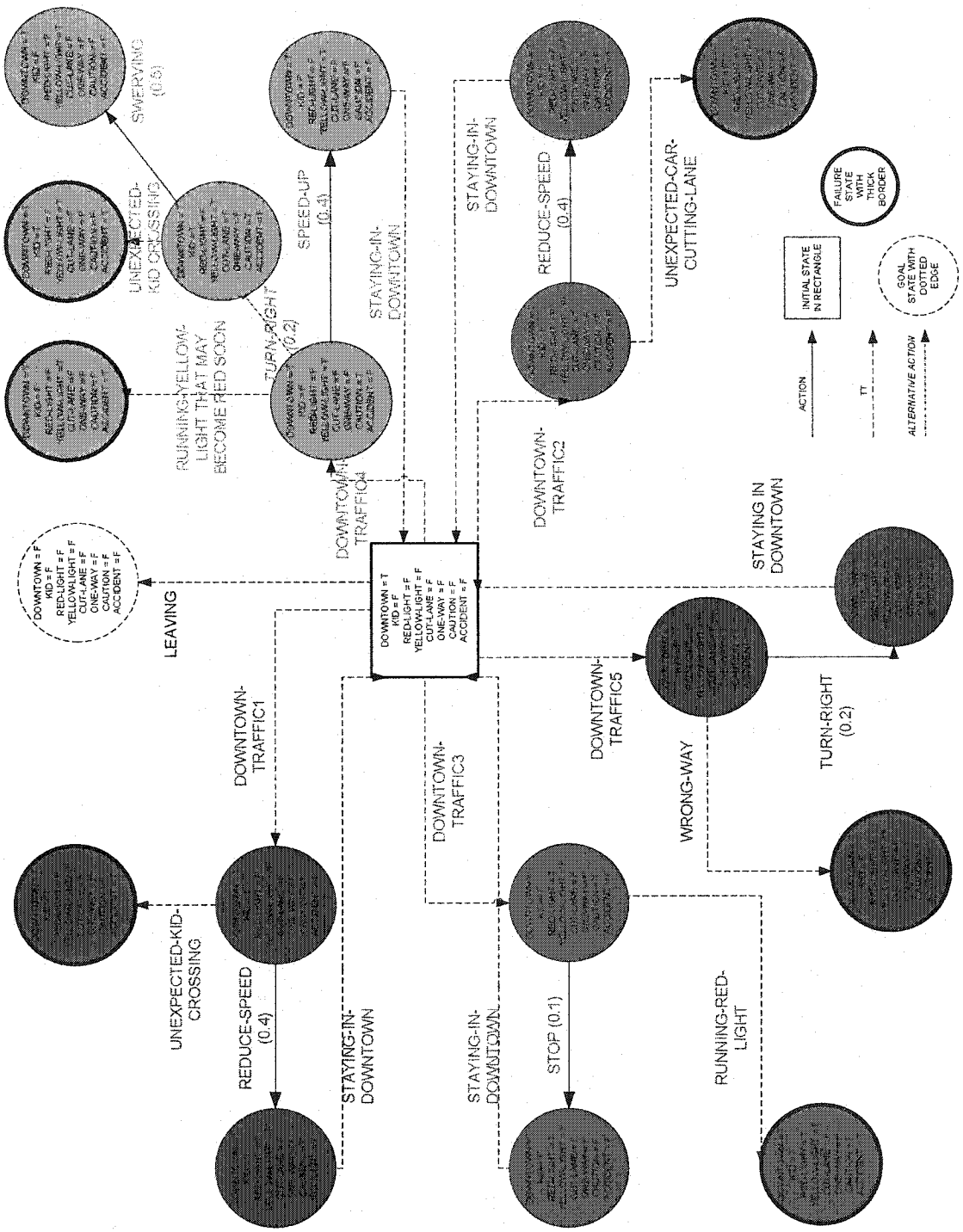


Figure 5-6: Replacing Action Can Introduce New States and New Actions

Finally, we have been assuming that the marginal cost of reusing an action is zero. Yet, in reality, the utilization of a TAP equals the utilization of the action of the TAP having the shortest period (highest frequency). A replacing action may occasionally incur additional cost if it has a shorter period for the new state than *all* periods of the same action in other states. When choosing a replacing action, the agent may limit its choices to only those actions that do not increase schedule utilization. We have two options regarding replacing actions that may incur additional costs.

1. Allow all eligible alternative actions that preempt failures.
2. Allow only those alternative actions that do not increase schedule utilization.

5.4.4 Possible Combinations of Heuristics and Constraints

Based on the above discussion, to implement the reusing action strategy using Algorithm 5-1, we have two heuristics to pick replaced actions, three heuristics to choose replacing actions, two options to handle new states, and two options to deal with possible increase in schedule utilization. In total, we can implement $2\times3\times2\times2=24$ variants of the strategy.

The two sets of heuristics search different parts of the search space. There are no dominant heuristics, but they work well in different domains. For instance, all six illustrating examples (Figure 5-1 – Figure 5-6) above require different heuristics to take the biggest advantage of the reusing action strategy. We will compare them and analyze the factors affecting the performance.

On the other hand, the two sets of constraints change the size of the search space. In general, if the agent allows for a bigger search space, e.g., allows new states and/or allows all eligible actions, it can find more action replacements. For example, the agent

may fail to replace an action by only one other action but can replace it by multiple actions.

While allowing a bigger search space may make finding replacements more successful, there are two disadvantages. First, it is more computationally expensive to explore a more complete search space. The agent needs to examine more combinations of actions for each replacement. The other downside is that, as there are more replacements, there is also a bigger decrement in the goal probability.

5.5 Evaluations and Analyses

In this section, our primary goal is to evaluate the reusing action strategy to see how well it transforms an unschedulable plan to a schedulable one but at the same time maintains the utility. To do so, we have used the random KB generator described in Section 4.13.1 to generate 318 random samples of CIRCA planning problems. Each sample demands more resources than an agent has. The agent applies the CIRCA resource allocation algorithm (Algorithm 5-2) to these samples to search for schedulable plans. Also, for each sample we produce a benchmark plan by skipping the reusing action strategy in Algorithm 5-2. We then compare the plans produced by using the reusing action strategy to those plans produced by not using it. We justify the reusing action strategy by showing that we can often produce (much) better plans by applying the strategy.

The utility of a plan is measured by the safety probability. We want to find out how much safer the plans are if an over-constrained agent has used the reusing action strategy to handle some tasks with less preferred actions instead of ignoring altogether the least likely to be needed tasks. Note that by using the reusing action strategy alone

will not necessarily produce a schedulable plan. Our baseline algorithm to produce a schedulable plan is the unlikely state strategy. Thus, it is important to understand that we are essentially measuring the *marginal* improvement of applying the reusing action strategy.

Also, we want to find out the cost of applying the reusing action strategy to justify if it is worthwhile. The cost can be computational and non-computational. Usually, the goal probability decreases per each replacement because the replacing action is less desirable. Computation cost includes how many actions, on average, an agent needs to examine for each problem instance and how many replacements it can find.

We will not compare all 24 possible combinations of the heuristics and options. Instead, we evaluate the heuristics or options in a category within that category. Specifically, the agent will apply the CIRCA resource allocation algorithm with the following implementations of the reusing action strategy to each sample.

H0. Not using the reusing action strategy;

H1. Order by utilizations; replacing actions have the highest state probabilities; allow new states; allow schedule utilization increase;

H2. Same as H1;

H3. Order by reference counts; replacing actions have the highest state probabilities; allow new states; allow schedule utilization increase;

H4. Same as H1;

H5. Replacing actions have the largest reference counts; order by utilizations; allow new states; allow schedule utilization increase;

H6. Replacing actions have the smallest utilizations; order by utilizations; allow new states; allow schedule utilization increase;

H7. Same as H1;

H8. No new states; order by utilizations; replacing actions have the highest state probabilities; allow schedule utilization increase;

H9. Same as H1;

H10. Do not allow schedule utilization increase; order by utilizations; replacing actions have the highest state probabilities; allow new states.

There are only 7 distinct combinations. The first group (H0 – H1) shows the gain in performance by using the reusing action strategy over not using it. The second group (H2 – H3) compares the ordering heuristics that select the next best replaced actions. The third group (H4 – H6) compares the replacement heuristics that select replacing actions. The fourth group (H7 – H8) evaluates the effects of opening up more of the search space by allowing the agent to add new states. The fifth group (H9 – H10) evaluates the effects of giving more action choices to the agent.

5.5.1 The Reusing Action Strategy Helps but not Always

We now describe our results for group one. The intention of this group of experiments is to show the performance of using the reusing action strategy over not using it. Out of the 318 samples, H1 (using the strategy) performs better in 248 of them. H0 (not using the strategy) performs better in the other 70 cases. In other words, using the reusing action strategy produces better plans 78% of the time.

It is expected that using the reusing action strategy should usually produce better plans. Let us denote T as the set of TAPs that would be dropped if we skip using the

reusing action strategy in Algorithm 5-2 (line 1), i.e., only the unlikely state strategy is applied (line 3). When using the reusing action strategy reduces the size of T, fewer actions are removed from the schedule. Consequently, the agent preempts more *ttf*s and the utility is thus bigger.

There are two main reasons why an agent sometimes produces a worse plan using the reusing action strategy. The first reason is when the reduction in utilization due to action replacement is not significant enough to reduce the size of T and when the agent has selected a replacing action, *ac*, which is in T. Two possible scenarios can happen. First, *ac* is removed when its TAP in T is removed. The agent could have removed T to make itself schedulable. Only the *ttf*s preempted by T were ignored. Now with action replacement, the *ttf* that *ac* preempts is also ignored. Second, *ac* is not removed, e.g., because the state *ac* is in has a high probability. Then, the TAP of *ac* in T cannot be removed. The agent will thus need to remove other actions not already in T to make itself schedulable. More *ttf*s become not preempted. In both cases, the failure probability increases more and the utility decreases.

The second reason why the reusing action strategy is not always beneficial is because action replacement may increase the probabilities of states whose actions are dropped by the unlikely state strategy. The probabilities of the *ttf*s that are preempted by these actions increase. Consequently, these hazards become more likely to happen. For example, after applying the reusing action strategy, the state diagram changes from being a tree to being a cycle. The agent will thus visit every state on the cycle during execution. When a *ttf* in one of these states is not preempted, it will fire eventually when the agent

traverses the cycle long enough. In this extreme example, action replacement changes the state diagram so dramatically that even ignoring one hazard will guarantee system failure.

5.5.2 The Performance of the Reusing Action Strategy

In this section, we describe the performance gain using the reusing action strategy. The experiments support our claim that the reusing action strategy is effective in (1) reducing the schedule utilization of an over-constrained agent, (2) increasing plan quality, and (3) finding replacing actions at a relatively low cost. Specifically, (1) 64% of the agents become schedulable. (2) The safety probability increases by 21.5% whereas the ideal improvement is 27.23% if there were no resource constraints. (3) The average success rate of finding replacing actions is 72%. On the other hand, the average final goal probability is only a little lower than if the reusing action strategy had not been used. We are now going into discuss the details of these results.

Table 5-1 shows the statistics in the 248 cases where the reusing strategy improves the plan quality. The initial utilizations of the samples range from 1.008 to 4.193. The average is 1.634 and the standard deviation is 0.5. By applying the reusing action strategy, we can reduce the (average) utilization of the samples from 1.634 to 0.902. Out of the 248 samples, 159 or 64% of them become schedulable by reusing actions that are already in the plans. Being schedulable, those agents can maintain their failure probabilities at 0.

The average failure probability of the plans of the agents who use the reusing action strategy is lower than that of the agents who do not, i.e., 0.045 vs. 0.214. In terms of the utility, namely the safety probability, the average safety probability rises by about 21.5% from 0.786 to 0.955. The ideal safety probability is 1.0 if resources were unlimited

so the maximum improvement possible is 27.23%. That is, the gain in performance has gotten a lot closer to the optimal plans under no constraints.

Table 5-1: H1 vs. H0 in the Samples that H1 Out-Performs H0

	H1	H0
initial utilization (avg)	1.634	1.634
initial utilization (stdev)	0.501	0.501
utilization after replacement (avg)	0.902	n/a
utilization after replacement (stdev)	0.402	n/a
schedulable after replacement	159	70
schedulable after replacement (%)	64.113	100
failure probability (avg)	0.045	0.214
failure probability (stdev)	0.130	0.290
safety probability/utility (avg)	0.955	0.786
goal probability (avg)	0.453	0.483
goal probability (stdev)	0.389	0.381

In terms of costs, each time an action replacement is made, the goal probability may decrease. After applying the reusing action strategy, the goal probability drops by 0.116 on average for our samples. The standard deviation is 0.267. In other words, the average utilization decreases by 0.732 at the cost of a 0.116 decrement in average goal probability. If a plan remains unschedulable after action replacement, the goal probability will drop further when the unlikely actions are removed by the unlikely state strategy. The drop is on average 0.431 in our experiments. The final average goal probability is 0.453.

Fortunately, as shown in Table 5-1, even when the reusing action strategy is not used, i.e., when only the unlikely state strategy is used to remove unlikely actions, the final average goal probability is not significantly bigger (0.483 vs. 0.453). The final average goal probabilities of using and not using the reusing action strategy are very similar. In other words, the sum of the drops of goal probability from action replacement

and action removal ends up about the same as the drop when only the unlikely state strategy is applied. Thus for our samples, using the reusing action strategy does not sacrifice the goal probability by too much.

We note that in 36 out of the 248 cases, the goal probabilities increase after action replacement. These plans have lower failure probabilities and bigger goal probabilities than the preferred plans that CIRCA had initially found. This violates the monotonic relationship between utilization and utility set forth in the beginning of this chapter. The reason is that CIRCA is not a perfectly rational agent. Our planner uses heuristics to find just good enough plans. It does not guarantee finding the best plans. In fact, without enumerating and comparing all possible plans, there are always some plans that any planning heuristic may overlook.

In terms of computational cost, it usually depends on how many actions are planned in the state diagram before replacement. The more actions are planned, the more actions the reusing action strategy needs to try to replace. Each time an action is replaced, Algorithm 5-1 has to select a replaced action, select a replacing action, test, and actually make the replacement in the agent's plan and state diagram. While these are not complex operations, they are neither trivial. The summary is listed in Table 5-2.

Table 5-2: H1 Computational Cost

	number of actions planned b4 replacement	number of actions tried	number of replaced actions	number of replacing actions per replacement
min	8	4	1	0
max	193	93	88	7
avg	37.310	22.040	15.766	1.457
std	31.313	16.634	16.308	1.235

The action selected to be replaced may or may not be successfully replaced. In our experiments, the average rate of successfully replacing actions is about 15.8/22.0 = 72%. For each action, the agent may select on average 1.5 replacing actions to replace it.

5.5.3 Comparing the Action Ordering Heuristics

As shown in Section 5.5.1, there is no dominant strategy to improve the resource allocation of a plan. The reusing action strategy is proven to be effective most of the time, but not always. Likewise, as we will show in the subsequent sections, there is also no dominant implementation of the reusing action strategy. We have implemented 6 different variants of the reusing action strategy to study the various properties of heuristics and constraints.

In this section, we compare the two action ordering heuristics to select the next action to replace. One heuristic orders the actions by utilizations in descending order (H1/H2). The rationale is to decrease the schedule utilization in a greedy manner. The other heuristic orders the action by reference counts in ascending order (H3). The rationale is to remove as many TAPs as possible (Section 5.4.1).

Our main result in this section is that the performances of both heuristics are very comparable (they produce plans having the same utility in 74.35% of the samples). Whether H1 will out perform H3 for a particular problem instance depends on how closely the problem matches the assumption of H1, and vice versa. We will illustrate this by walking through an example.

The performance statistics of these two implementations applied on the 318 samples are shown in Table 5-3. Note that the statistics of H1 in Table 5-3 are slightly different from those in Table 5-1. Table 5-1 intends to show the performance gain of

using the reusing action strategy so it covers only the 248 cases where using the strategy produces better plans. Table 5-3 intends to compare the different implementations of the reusing action strategy so it covers all 318 cases, including the 70 cases that using the strategy produces worse plans.

Our experiments show that H1 performs only marginally better than H3 in terms of utility, the number of schedulable agents, and the number of better plans. The differences are very little and statistically insignificant. The performances of H1 and H3 are very similar. In fact, 74.35% of the times, these two implementations produce plans having the same utility (safety probability).

Table 5-3: H1 vs. H3

	H1	H3
initial utilization (avg)	1.621	1.621
initial utilization (stdev)	0.498	0.498
utilization after replacement (avg)	0.911	0.970
utilization after replacement (stdev)	0.424	0.435
schedulable after replacement (%)	55.66	51.42
failure probability (avg)	0.087	0.093
failure probability (stdev)	0.200	0.213
safety probability/utility (avg)	0.913	0.907
goal probability (avg)	0.447	0.452
goal probability (stdev)	0.396	0.392
better plans (%)	14.620	10.850

Whether H1 or H3 will work better in a domain will depend on whose assumption is more closely satisfied in the domain. Specifically, H1 assumes that the larger utilization actions are easier to remove. H3 assumes that TAPs having bigger reference counts are harder to remove from the schedule because all actions of the TAPs must be removed. H1 and H3 have comparable performances when both of their assumptions are satisfied, or equivalently, their orderings are the same.

Put another way, for H1 to out perform H3 in a domain, H1's assumption must be more appropriate. That is, those actions having big reference counts can be replaced by reusing other actions (in contrast to H3's assumption) and they have big utilizations. We illustrate this with an example from an actual sample in our experiments. Table 5-4 shows the schedule of a plan before the agent applies the reusing action strategy. pRC stands for the number of preferred actions a TAP has; rRC stands for the number of reusing actions a TAP has. All rRCs are zero before action replacement.

Table 5-4: Schedule before Action Replacement Using H3; Utilization = 1.344

TAP	pRC	rRC	TAPProb	Period	WCET	Utilization
AC_PREEMPT2	3	0	0.798283	198	50	0.252525
AC_PREEMPT4	1	0	0.200000	248	10	0.040323
AC_PREEMPT5	2	0	0.798283	208	40	0.192308
AC_PREEMPT6	2	0	0.200000	190	58	0.305263
AC_PREEMPT7	2	0	0.200000	197	51	0.258883
AC_PREEMPT9	2	0	0.198025	207	41	0.198068
AC7	1	0	0.400000	499	14	0.028056
AC9	1	0	0.001717	248	17	0.068548

H3 orders the actions in ascending order of reference count (a bracket [] indicates that the action is successfully replaced): AC_PREEMPT4, [AC7], [AC9], AC_PREEMPT9, [AC_PREEMPT6], AC_PREEMPT7, AC_PREEMPT5, AC_PREEMPT2.

There are 3 actions that are successfully replaced. 1 action is dropped as a result of the replacements. The schedule after replacement is shown in Table 5-5. TAPs AC7 and AC9 are removed.

Table 5-5: Schedule after Action Replacement Using H3; Utilization = 1.215

TAP	pRC	rRC	TAPProb	Period	WCET	Utilization
AC_PREEMPT2	3	1	0.400000	198	50	0.252525
AC_PREEMPT4	1	0	0.200000	248	10	0.040323
AC_PREEMPT5	2	1	0.400000	208	40	0.192308
AC_PREEMPT6	1	1	0.200000	190	58	0.305263
AC_PREEMPT7	2	0	0.200000	197	51	0.258883
AC_PREEMPT9	1	0	0.198025	248	41	0.165323

H3 assumes that it is hard to remove AC_PREEMPT6 because it has a bigger reference count (2 > 1) in the plan. So, AC_PREEMPT6 is examined relatively late by the heuristic. As a result, it is used as a reusing action to replace another action (so, its rRC becomes 1), which makes it even more difficult to remove when it is examined.

Yet, in this example, H3 is wrong. AC_PREEMPT6 can in fact be replaced by other actions. Moreover, it has the biggest utilization in the schedule, so replacing it reduces the schedule utilization most. Therefore, H1 can make the agent schedulable by removing AC_PREEMPT6 in addition to AC7 and AC9. H1 thus out performs H3.

H1 orders the actions in descending order of utilization: [AC_PREEMPT6], AC_PREEMPT7, [AC_PREEMPT2], [AC_PREEMPT9], AC_PREEMPT5, [AC9], [AC7].

There are 5 replaced actions. In addition, 3 actions are dropped as a result of the replacements. So, H1 performs much better than H3 in this example. The new schedule is:

Table 5-6: Schedule after Action Replacement Using H1; Utilization = 0.942

TAP	pRC	rRC	TAPProb	Period	WCET	Utilization
AC_PREEMPT2	1	2	0.800000	198	50	0.252525
AC_PREEMPT4	1	0	0.200000	248	10	0.040323
AC_PREEMPT5	2	2	0.800000	208	40	0.192308
AC_PREEMPT7	2	0	0.200000	197	51	0.258883
AC_PREEMPT9	0	1	0.199740	207	41	0.198068

5.5.4 Comparing the Replacement Heuristics

In this section, we compare the three replacement heuristics. To replace an action, the first heuristic picks a replacing action having the biggest TAP probability (H1/H4); the second heuristic picks a replacing action having the biggest reference count (H5); the third heuristic picks a replacing action having the smallest utilization (H6). The rationale behind H1 and H5 is to pick a replacing action that is most likely to stay in the schedule if the agent needs to drop actions using the unlikely state strategy. The rationale behind H6 is to reduce schedule utilization as much as possible (Section 5.4.2).

Our main result in this section is that the reusing action strategy tends to produce better plans when an agent selects replacing actions that stay in the final schedulable plan. This is confirmed by our experiments in which H1 and H5 out perform H6. Moreover, as H1 performs better than H5, it is confirmed that state probability is a better estimator than reference count to predict which action candidate is more likely to stay in the final schedule.

The performance statistics of these three implementations applied on the 318 samples are shown in Table 5-7. Note that the statistics of H1 in Table 5-7 are the same as those in Table 5-3 (except "better plans"). The performances of H1 and H5 are similar

though H1 performs somewhat better than H5. In contrast, H6 on average performs worse than both H1 and H5 because it reduces schedule utilization less, produces fewer schedulable plans, and produces plans having bigger failure probabilities.

Table 5-7: H1 vs. H5 vs. H6

	H1	H5	H6
initial utilization (avg)	1.621	1.621	1.621
initial utilization (stdev)	0.498	0.498	0.498
utilization after replacement (avg)	0.911	0.961	1.429
utilization after replacement (stdev)	0.424	0.409	0.504
schedulable after replacement (%)	55.66	50.72	13.30
failure probability (avg)	0.087	0.116	0.185
failure probability (stdev)	0.200	0.246	0.301
safety probability/utility (avg)	0.913	0.884	0.815
goal probability (avg)	0.447	0.452	0.507
goal probability (stdev)	0.396	0.401	0.382
better plans (%)	20.450	7.58	8.330

We have explained in Sections 5.4.2 and 5.5.2 and illustrated by the example in Figure 5-4 how a worse plan may result when a replacing action is dropped by the unlikely state strategy. H6 replaces an action by another action having the smallest utilization. When the replacing action is in a low probability state, it will be dropped if the agent is still over-utilized. H6 makes no effort to select replacing actions that will stay in the plan after action replacement. On the other hand, H1 and H5 try to estimate how likely a replacing action is to stay in the plan by state probabilities and reference counts. They select actions that are estimated to be least likely to be dropped. So, H1 and H5 perform better than H6 on average.

Moreover, reusing an action having a smaller utilization does not reduce the schedule utilization more than any other action as far as the reusing action strategy goes. As we have been assuming, the marginal cost of the replacing action is zero because it is

already in the plan. Reusing any action is equally good as long as it preempts the *ttf*s. So, the agent does not gain anything by reusing a smaller utilization action. Reusing a smaller utilization action helps only when the agent is severely over-utilized and starts dropping low probability actions. Presumably, the agent drops fewer actions if the schedule utilization is smaller.

As H1 works better than H5, we conclude that state probability is on average a better estimator than reference count to predict whether an action will stay in the plan. By construction of the unlikely state strategy, an action in a state having a bigger probability is less likely to be removed. On the other hand, even if a TAP may have a big reference count, all its actions may be dropped if they all are in low probability states. Despite this, H1 does not always work better than H5. H1 uses the probability of actions before action replacement, but the probabilities can change after action replacement. Again, there is no dominant implementation of the reusing action strategy.

5.5.5 Introducing New States

In this section, we evaluate the effect of allowing new states to be added to the state diagram as a result of reusing actions. We compare H1/H7 (allowing new states) to H8 (allowing no new states). H8 is constrained to replace an action by only one other action. By introducing new states, H1 relaxes this constraint and may replace an action by a combination of actions (Section 5.4.3).

Our main result in this section is that allowing an agent to use multiple actions to replace an action will increase the success rate of finding a replacement (by 12.44% in our samples). The agent thus in general produces a better plan because of a larger search

space. The computational cost increase modestly. Table 5-8 shows the performance comparisons.

Table 5-8: H1 vs. H8

	H1	H8
initial utilization (avg)	1.621	1.621
initial utilization (stdev)	0.498	0.498
utilization after replacement (avg)	0.911	1.092
utilization after replacement (stdev)	0.424	0.491
schedulable after replacement (%)	55.66	37.38
failure probability (avg)	0.087	0.148
failure probability (stdev)	0.200	0.270
safety probability/utility (avg)	0.913	0.852
goal probability (avg)	0.447	0.450
goal probability (stdev)	0.396	0.401
better plans (%)	35.920	8.250

It is not surprising that H1 performs better than H8 because it can try more combinations of actions to replace an action. The rate of successfully replacing actions for H1 is 69.34% while that for H8 is only 56.90%. H1 makes 18.28% more agents schedulable and the average failure probability of the plans it produces is 41.22% smaller.

On the other hand, since H1 may add new states and expand them, its computational cost is higher. As in Section 5.5.2, we measure computational cost by the number of tried actions, replaced actions, and replacing actions. They constitute the main activities in Algorithm 5-1. Table 5-9 shows the computational cost comparisons.

H1 tries more actions, hence more computations, because new actions are added to the plan as new states are introduced. The agent needs to try to replace those newly added actions as well. In our experiments, the agents on average examine 18.70% more actions. As H1 has a higher success rate, it replaces 49.91% more actions. As H1 may

replace an action by multiple actions, for each action replacement H1 uses 190.24% more replacing actions

Table 5-9: H1 vs. H8 Computational Cost

	H1	H8
number of actions planned b4 replacement (avg)	35.976	35.976
number of actions planned b4 replacement (stdev)	28.692	28.692
number of actions tried (avg)	20.862	16.961
number of actions tried (stdev)	15.993	11.243
number of replaced actions (avg)	14.466	9.650
number of replaced actions (stdev)	15.676	11.282
number of replacing actions per replacement (avg)	1.457	0.502
number of replacing actions per replacement (stdev)	1.217	0.398
success rate (%)	69.341	56.895

It is expected that H1 reuses more than one action in each replacement (1.5 actions per replacement in our experiments). It is worth of pointing out why the number of replacing actions for H8 is not 1 (0.5 action per replacement in our experiments). The number of replacing actions per replacement is computed as the number of actions used to replace one action from the state diagram. When only the replaced action is removed from the plan, this ratio is 1. Sometimes, more actions are subsequently removed from the state diagram as a result of removing the replaced action (c.f. the illustrating example in Section 5.5.3). Those actions belong to the states that are detached from the state diagram when the child state of the replaced action becomes detached. Thus, more than one action may be removed from the state diagram in each action replacement. The ratio is thus less than 1.

Despite the fact that H1 can sometimes out perform H8 by reusing multiple actions to replace one action (35.92%), in our experiments H1 produces identical plans as H8 does 55.83% of the time. In other words, most of time, H1 reuses only a single action to replace an action. As allowing no new states essentially limits an agent to replace an action by only one other action (but the reverse is not true), H8 approximates H1. Thus, if we want to reduce computational cost, we can use implementation H8.

5.5.6 Increasing TAP Utilization

In this section, we evaluate the effect of allowing an agent to choose an action that increases the utilization of the corresponding TAP. We compare H1/H9 (allowing utilization increase) to H10 (allowing not utilization increase). H10 is constrained to limit the replacing action choices to those actions whose periods are not shorter than the periods of their TAPs. H1 lifts this constraint and lets the agent to choose any eligible actions that preempt the *ttf*s (Section 5.4.3).

Our main result in this section is that whether we allow actions to have shorter periods than their TAPs has only a modest effect on the final plan utility. While it makes finding a replacement more successful (10.33%), it does not improve a plan significantly. Yet, it does increase the computational cost because of a larger search space. We would thus usually want an agent to skip considering those actions to avoid extra computational cost. Table 5-10 shows the performance comparisons.

Table 5-10: H1 vs. H10

	H1	H10
initial utilization (avg)	1.621	1.621
initial utilization (stdev)	0.498	0.498
utilization after replacement (avg)	0.911	1.028
utilization after replacement (stdev)	0.424	0.400
schedulable after replacement (%)	55.66	45.24
failure probability (avg)	0.087	0.092
failure probability (stdev)	0.200	0.218
safety probability/utility (avg)	0.913	0.908
goal probability (avg)	0.447	0.489
goal probability (stdev)	0.396	0.397
better plans (%)	22.619	11.111

The performance of H1 and that of H10 are comparable, though H1 performs somewhat better than H10 because it has more action candidates to choose from to replace an action. The rate of successfully replacing actions for H1 is 69.34% while that for H10 is 59.010%. H1 makes 10.42% more agents schedulable and the average failure probability of the plans it produces is 0.018% smaller. In other words, this constraint of limiting the replacing actions to those having longer periods than their TAP periods is not as restricting as the constraint of not allowing new states in Section 5.5.5.

Table 5-11: H1 vs. H10 Computational Cost

	H1	H10
number of actions planned b4 replacement (avg)	35.976	35.976
number of actions planned b4 replacement (stdev)	28.692	28.692
number of actions tried (avg)	20.862	17.155
number of actions tried (stdev)	15.993	13.346
number of replaced actions (avg)	14.466	10.123
number of replaced actions (stdev)	15.676	13.520
number of replacing actions per replacement (avg)	1.457	0.946
number of replacing actions per replacement (stdev)	1.217	1.094
success rate (%)	69.341	59.010

At the tradeoff of having a worse performance, H10 has also a lower computational cost. In our experiments, H10 on average examines 21.61% fewer actions, replaces 30.02% fewer actions, and reuses 35.07% fewer replacing actions in each replacement.

5.6 Summary

In this chapter, we have generalized the unlikely state strategy to the action replacement algorithm. In general, there are many (combinatorial) ways to make a successful action replacement. Also, replacing actions does not guarantee decreasing schedule utilization. To make the algorithm computationally tractable, we have studied the concept of a reusing action. When all actions are scheduled on the same platform, reusing an action to handle a task has a zero marginal cost. Also, as each replacement has a marginal cost of zero, the total marginal cost of all replacements remains zero. The schedule utilization is thus guaranteed to decrease. Using the reusing action information, an agent can quickly find successful action replacements without trying and comparing many combinations of possible replacing actions.

This reusing action strategy can be implemented in various ways. There is no dominating implementation. We have studied the various heuristics for an agent to select an action to replace, to reuse an action, and to limit the search space size. We have identified the circumstances where which of the heuristics and constraints are most appropriate. For instance, we found that it is very important for an agent to replace an action by reusing an action that is most likely to stay in the final schedule (after using the unlikely state strategy). Otherwise, not only does the corresponding failure become

unhandled, but the agent might actually produce a worse off plan. Our experiments show that, 78% of the time, an agent benefits from using the reusing action strategy. The utility of a schedulable plan increases, on average, by 21%.

Chapter 6

Resource Allocation for Distributed Real-Time Controllers

In this chapter and the next, we extend the concepts developed for a single agent, such as probabilistic guarantee, state probability, replacing action, and reusing action, to a system of cooperating agents. We will show that the general action replacement algorithm (Algorithm 5-1) is also effective to improve the local plan of an agent in a multiagent environment despite the complex interactions among agents. That is, we will study how an agent in a loosely coupled and cooperative multiagent system (c.f. Section 1.2), may make its plan schedulable by applying the unlikely state strategy (Chapter 6), and how the agent may improve its local resource allocation by replacing suboptimal actions (Chapter 7).

In a multiagent environment, not only does an agent need to be prepared to react to events arising naturally in the world but it must also be ready for those events due to the activities of other agents. Ideally, the agent would want to guarantee real-time performance to all these (re)actions. Otherwise, it can use the unlikely state strategy to drop the least likely used actions to make its plan schedulable. To effectively apply the unlikely state strategy, the agent must be able to compute state probabilities reasonably well. This in turn requires the agent to have a reasonably accurate worldview of the temporal dynamics of events, its actions, and other agents' actions.

Each of the agents in the world, however, may not have a sufficiently complete picture of the global activities because it may not know about the plans of other agents. There are uncertainties about what the other agents might do. More importantly, the uncertainties prevent the agents from "envisioning" a similar future progression of the

world, so they may not agree on their expected interactions. Despite its local computation effort, an agent may spend resources on guaranteeing actions that it will never need because the agent has projected a future that will never happen. The agent needs to expand and update its local worldview with *some* knowledge of the global activities.

This dissertation views the problem of how an agent decides how much it needs to know about the plans of other agents as a type of multiagent planning problem. Although an agent might not be able to change the actions chosen by other agents, knowledge about those choices can update its local worldview for planning. With a more accurate worldview, the agent can compute more accurately the state probabilities. Consequently, it can make more informed resource allocation decisions. Therefore, this chapter is not about coordinating multiple agents to work together to achieve a common objective. Rather our problem is how much an agent, facing the uncertainties about other agents' activities may influence the world, needs to know about them to better allocate its local resources.

Specifically, we distinguish between two types of events. A natural event is one that may occur depending on the state of the world regardless of the activities of the agents. We call it an *unconditional event*, because it is independent of the agents' actions. In contrast, the other events occur only because other agents explicitly choose to take some actions. We call them *conditional events*.

We exploit the fact that other agents' actions are not just non-deterministic, unpredictable events. Instead, selective communication among agents can allow them to develop a more coherent view of the global activities. They can recognize which events are most important to prepare for, as well as which events are assuredly not going to

occur. Using our terminologies, an agent should differentiate between unconditional and conditional events. While the agent must expend resources to anticipate those unconditional events, it can avoid spending resources on the conditional ones if it knows that other agents will not cause them to happen. Moreover, having a more informed model of the plans of other agents, an agent can more accurately compute state probabilities.

Our solution to solving the limited resource allocation problem for a group of agents is therefore to have them first construct the worst-case resource consumption plans. These plans inevitably include states that are reachable conditional on other agents executing certain actions. The agents can then incrementally exchange details about their plans to each form a more precise (but not necessarily identical) view of the future. After the agents learn which conditional events will not happen because other agents have not planned them, they can remove the corresponding actions from their plans. The agents also update their state diagrams to incorporate the additional information so they can more accurately compute state probabilities. The agents continue doing so until they satisfy their resource constraints or until further information has no value.

In the remainder of this chapter, we describe the multiagent extension of CIRCA (Section 6.1). Then we describe how agents construct the worst-case scenario (unconstrained) plans (6.2). The agents use the Convergence Protocol developed in Sections 6.3 to 6.7 to iteratively refine their initial, unconstrained plans to improve the resource allocation decisions. We conclude this chapter by evaluating how well our solution works in sample domains (Section 6.8) and investigate the factors that affect the performance (Section 6.9).

6.1 CIRCA in a Multiagent Environment

A CIRCA agent in a multiagent environment builds on the single agent architecture (Chapter 3). It is augmented with the ability to distinguish between private (local) features and public (shared) features. An agent's private features are those that no other agents are interested in but the agent itself, such as its current fuel level. Private features do not appear in the state diagrams of other agents.

Public features are those features that more than one agent is interested in. An agent includes in its feature set only the public features that it cares about. Different agents may have different sets of public features. In other words, the feature set of an agent includes all its private features and some public features. It is through manipulating the public features that agents impact each other.

For example, Figure 6-1 and Figure 6-2 show the state diagrams for a fighter and a bomber. In this mission, BOMBER is assigned the responsibility to destroy one of the two locations (LOC1 or LOC2) (Figure 6-1). After the bomber attacks, the enemy planes may retaliate. The fighter is responsible for protecting BOMBER, patrolling between the two locations (Figure 6-2). BOMBER needs to report its status at LOC2 if FIGHTER has made such request. In Figure 6-1 and Figure 6-2, COMM and ENEMY are public features shared by both BOMBER and FIGHTER, while HEADINGF and LOCF are private features that are accessible only to FIGHTER.

Furthermore, a CIRCA agent includes in its knowledge base some public temporal transitions. It also includes the public actions of other agents. These action transitions of other agents behave very similarly to temporal transitions, and hence are labeled *ttac*, from the perspective of the agent. Both the temporal transitions (*tt*s) in the

environment and the actions of other agents (*ttac*s) affect the public features that the agent cares about. Both are beyond the control of the agent. In contrast, private temporal transitions and private actions of other agents do not include in their postconditions any public features but include only private features. The agent thus does not care about them because they do not change any features that it is interested in.

For instance, B:BOMB-1 and B:BOMB-2 are public actions of BOMBER in Figure 6-2, while the action HEAD-TO-LOC1 is private for FIGHTER. The temporal transitions FLY-TO-LOC0, FLY-TO-LOC1, and FLY-TO-LOC2 are private for FIGHTER. In Figure 6-1, there are temporal transitions having the same names but they are private for BOMBER. There is no public temporal transition in this example.

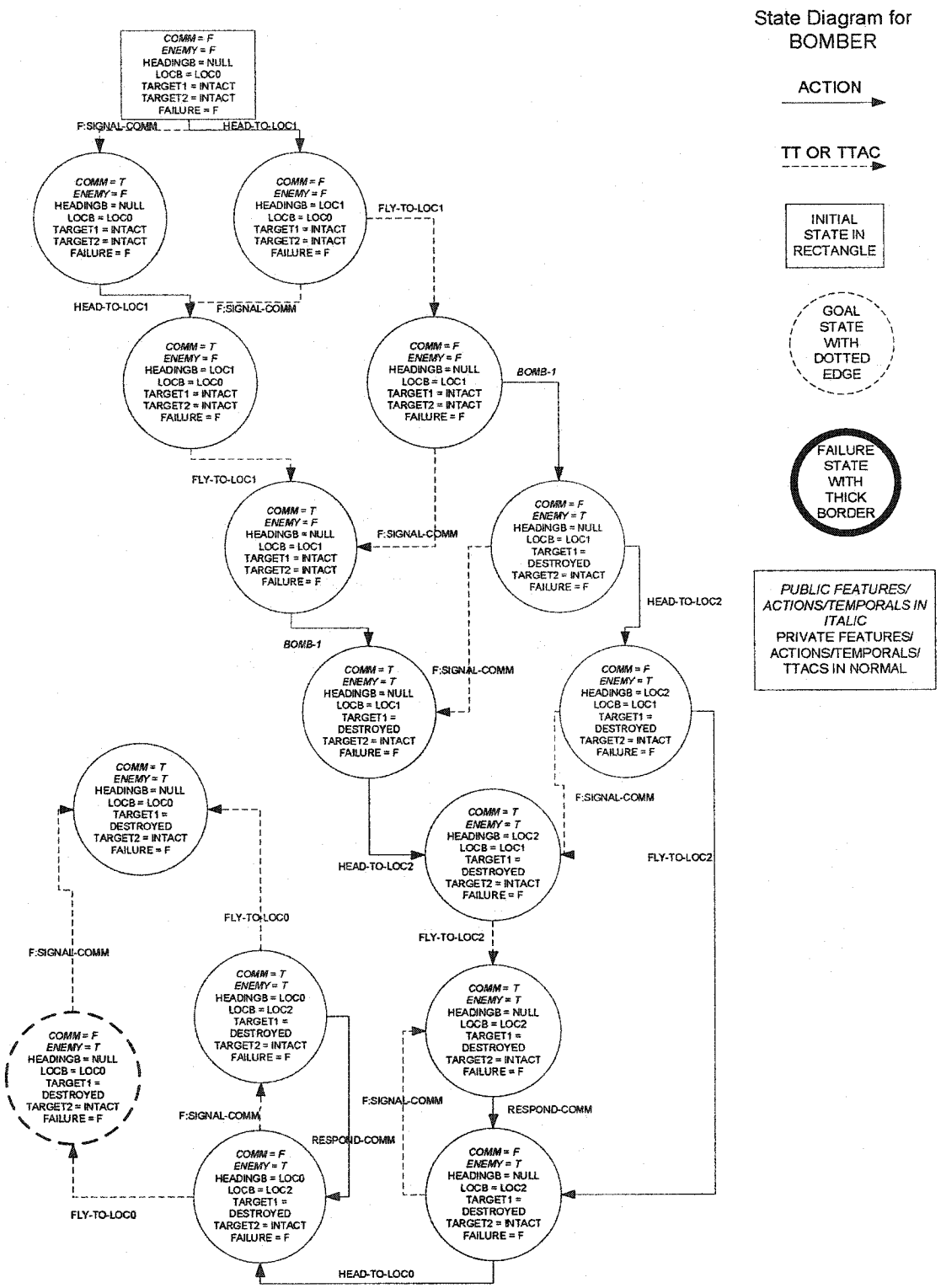


Figure 6-1: BOMBER State Diagram

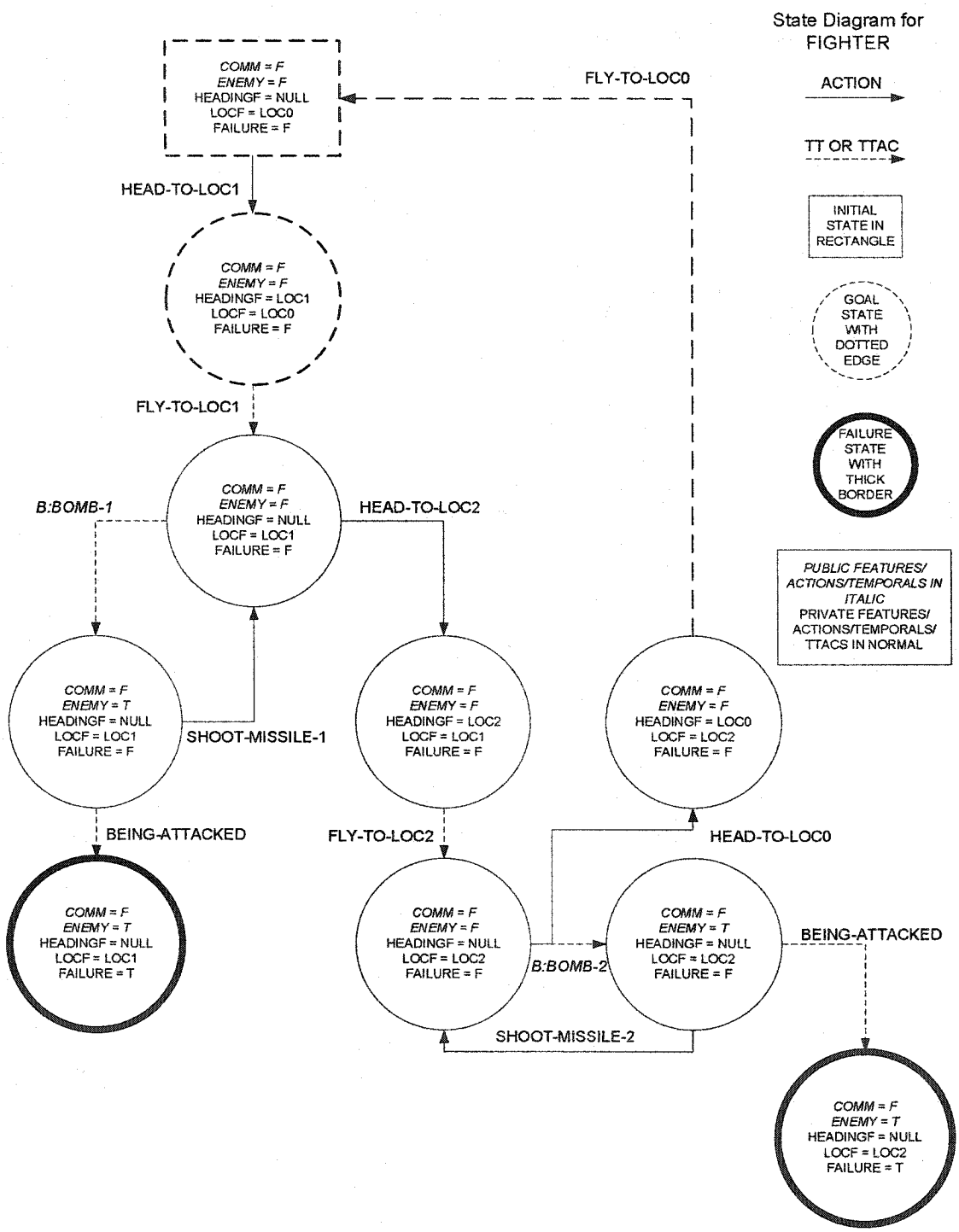


Figure 6-2: FIGHTER State Diagram

Differentiating between private and public features lets an agent model only the relevant aspects of the world. The agent does not need to know the entire plans of other agents, but only those parts affecting the public features. Since a state represented by an agent does not include the private features of other agents, it may correspond to a group of states represented by those agents. As the number of states is exponential in the number of features, this abstraction significantly reduces the planning complexity and state diagram sizes.

For instance, suppose there are 2 agents sharing 5 public binary features and each has 5 private binary features. Instead of planning explicitly for all of the $2^{15} = 32768$ states, each agent locally plans for at most $2^{10} = 1024$ local states, where each local state in this case corresponds to 32 possible global states. Some of these local states are not distinguishable from the perspective of the other agent because the public features have the same values (though the private features have different values). The number of distinguishable sets of states (or partial states) is at most 2^5. Using abstraction, all an agent needs to know about the other agent in its state diagram are those $2^5 = 32$ partial states instead of all 1024 states. Consequently, they can limit their information exchange to only those 32 states (having different public feature values)![29]

Figure 6-3 shows the state diagram of an agent with (left) and without (right) using this abstraction mechanism. The state diagram on the left shows only reachable states. The state diagram on the right shows all possible states in the state space. The highlighted states correspond to the (reachable) states in the state diagram on the left.

[29] Each partial state corresponds to a subset of the 1024 local states, and each local state in turn corresponds to a subset of those 32768 fully grounded states. When the context is clear, we will simply refer to a partial state as a state.

Clearly, the state diagram using the abstraction mechanism is more compact and more accessible to human.

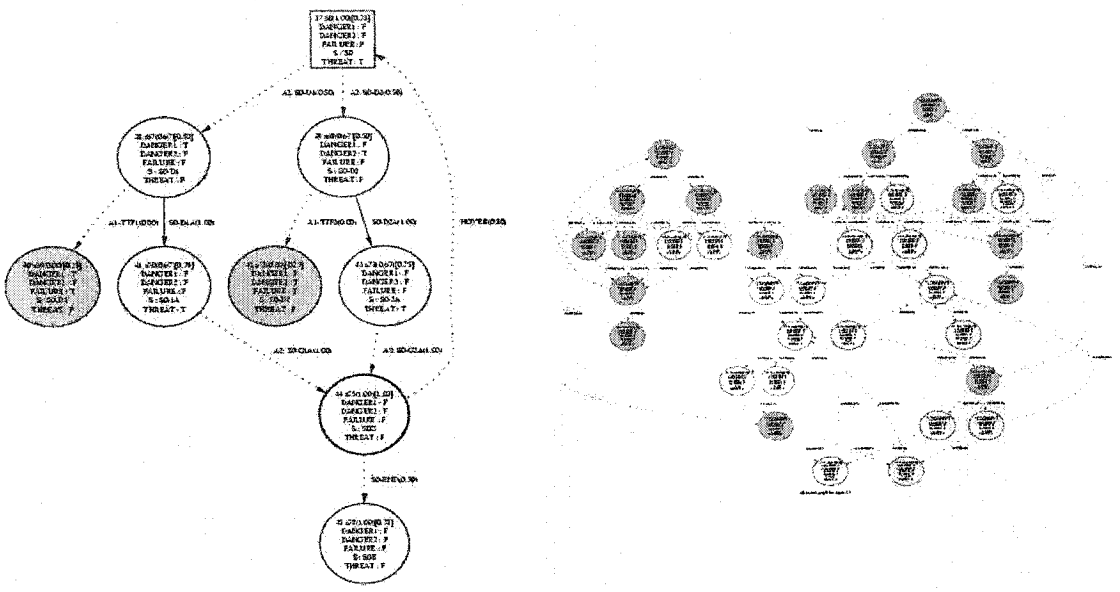


Figure 6-3: Public and Private Feature Abstraction Mechanism[30]

For a multiagent CIRCA system, we need to measure the utility for each individual agent as well as for the global performance. Similarly to the single agent case, the global utility function when failure avoidance is the only objective is:

$$U = \overline{F} = (1 - F) \qquad \textbf{eq. 6-1}$$

When the group of agents is considering both failure avoidance and goal achievement, the global utility function is:

$$U = (1 - F)^{\alpha} G^{(1-\alpha)} = \overline{F}^{\alpha} G^{(1-\alpha)} \qquad \textbf{eq. 6-2}$$

[30] The feature-values in these two state diagrams are not important. The point here is to compare the numbers of states with and without using the abstraction mechanism.

We need to define what is meant by success and failure for a group of agents. When we are dealing with a group of cooperative agents that are given a team mission (though each may have different responsibilities/goals), we adopt the traditional CIRCA safety principle. We define success of a multiagent system as when all agents in the team achieve their goals. We consider that the system fails when any of the agents fails. We make the simplifying assumption that each agent can succeed or fail independently of other agents. In other words, the failure probability of a multiagent plan is:

$$F = 1 - \prod_i (1 - F_i) \qquad \textbf{eq. 6-3}$$

F_i is the probability that agent i fails. The success probability is:

$$G = \prod_i G_i \qquad \textbf{eq. 6-4}$$

G_i is the probability that agent i succeeds.

6.2 Reachability Analysis

To succeed in a multiagent environment, a rational agent needs to envisage what actions other agents might take, and choose its own actions based on these predictions. Other agents cannot be expected to know what constitutes failure for the agent (which could in part be based on its private features), nor can they promise not to affect negatively the public features. To play on the safest side, the agent must consider and plan for all states that it foresees due to the transitions in the environment (unconditional events) as well as all possible actions executed by other agents (conditional events).

Regardless of whether a state is reachable by a sequence of temporal transitions (*tt*s) or a sequence of other agents' actions (*ttac*s) or a combination of both, the agent needs to expend resources to schedule an action to preempt any hazards in the state. We

call such an analysis a *reachability analysis*. A state diagram that includes all *ttacs* and all potentially reachable states generated in a reachability analysis is called a *reachability graph*. Figure 6-4 repeats the reachability graph of a fighter patrolling two nearby locations after any potential activities there by BOMBER (same as in Figure 6-2). FIGHTER includes in its planning, hence the state diagram, all public actions by BOMBER.

However, just because another agent is capable of taking an action does not mean that it will take that action, meaning that anticipating all possible actions on the part of other agents requires an agent to prepare for states that might never arise. We call those states *unreachable states*. In contrast, *reachable states* are those states that the agent *may* reach during execution.[31] Unreachable states are included in a reachability graph only because of ignorance. Using our terminologies, they correspond to some conditional events. Reachable states include some conditional and all unconditional events. The actions planned for the unreachable states are *unnecessary actions* and should be removed if they are recognized as such.

Forming control plans based on such a straightforward reachability analysis suffices only when an agent has enough resources to handle all contingencies. In this ideal case, it can disregard (and be proudly ignorant of) other agents' plans. Regardless of which of the known possible actions the other agents choose, and which potential states it may thus encounter, it can always execute the proper reactions fast enough to avoid failures. On the other hand, the agent may be unable to schedule all the actions for all the states that it thinks it may encounter in the reachability graph due to over-utilization of resources.

[31] The reachable states have state probabilities greater than 0 but not necessarily 1.

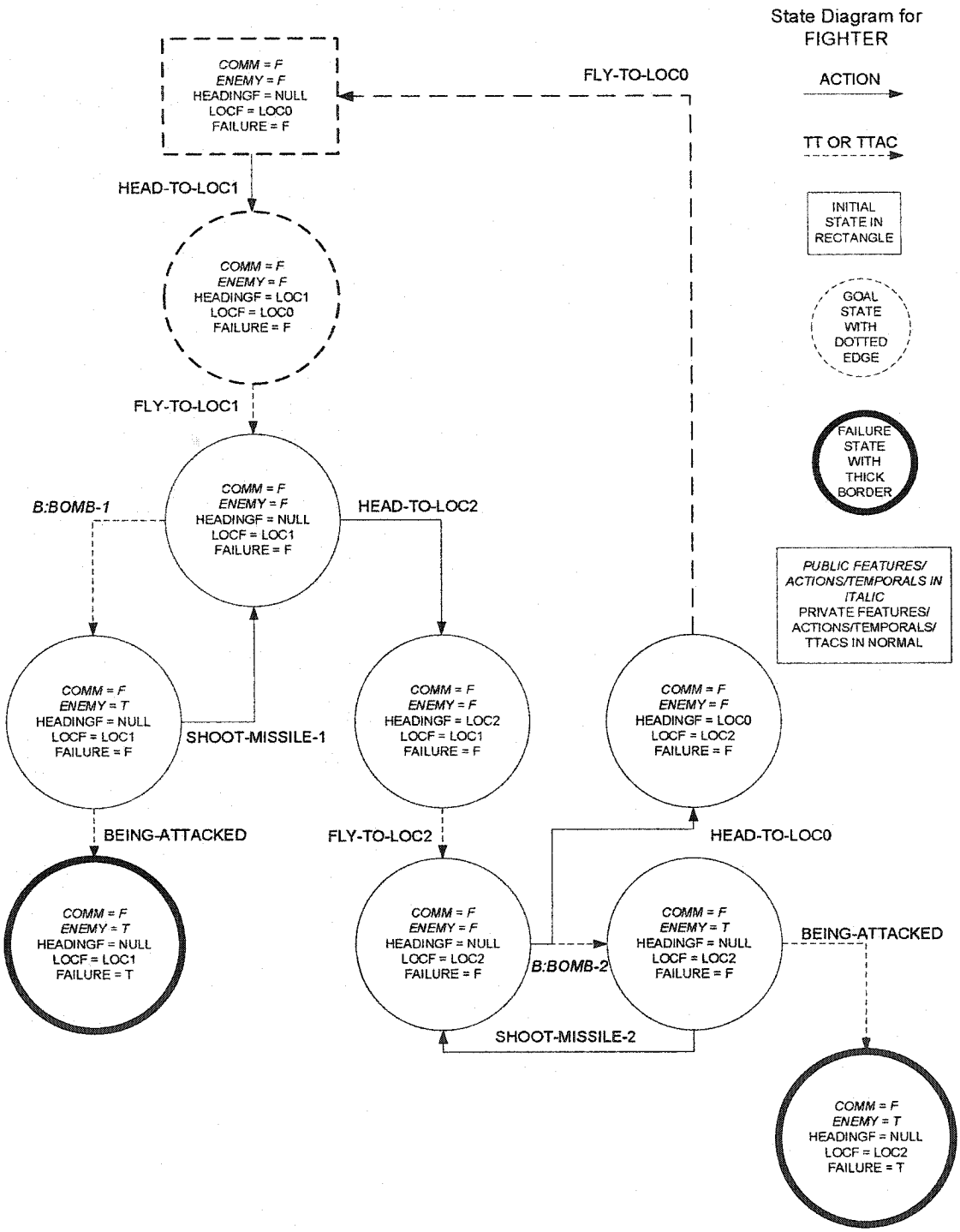


Figure 6-4: The Partial[32] Reachability Graph of FIGHTER[33]

[32] For clarity, this graph omits some B:BOMB-1 and B:BOMB-2 transitions. They do not affect the final plan.

6.3 The Convergence Protocol

We have developed the Convergence Protocol that allows agents to exchange information about the intersecting parts of their plans. By lowering the uncertainties in modeling the world events, the agents identify unreachable states in the state diagrams and eliminate the unnecessary actions from their initial plans. Consequently, their resource consumptions decrease. For instance, in Figure 6-4, when FIGHTER is unable to guarantee real-time performance to both SHOOT-MISSILE-1 and SHOOT-MISSILE-2, knowing which location BOMBER is not going to bomb removes the corresponding shoot-missile action from its plan.

We assume that the agents perform the protocol after they have locally formed their reachability graphs and have selected all the actions they would like to take (as in the worst-case scenario that they have to be prepared for all conditional and unconditional events). The protocol is described in Algorithm 6-1.

An *uncertain point* of interest in a reachability graph is a combination of a state and a set of mutually-exclusive *ttac*s. A set of *ttac*s in that state corresponds to the alternative action choices of another agent out of which it will choose one or none. As multiple agents can plan for a state, there could be multiple sets of *ttac*s. Thus, to choose an uncertain point, the agent picks a state as well as another agent planning for that state, i.e., a particular set of *ttac*s.

Each *ttac*, when planned by another agent, leads to a *realizable* branch in the state diagram that consumes certain resources. A branch from a state is a subgraph from a transition in that state. On the other hand, when a *ttac* is not planned by another agent, the

---

[33] All actions are either guaranteed (e.g., SHOOT-MISSILE-1) or reliable actions (e.g., HEAD-TO-LOC1). This holds true also for Figure 6-5.

branch is not realizable during execution. It can be safely pruned from the state diagram. All actions planned for the descendant states of the branch are subsequently removed (unless the states can be reached by other viable transition paths), freeing up resources and thus making scheduling easier.

```
Inquiring agent () {
1.    Choose the uncertain point that gives the biggest estimated utilization
      reduction; //*
2.    Ask the corresponding agent which action(s) it would take;
3.    Upon receiving an answer, update the state diagram and drop the unnecessary
      actions from the local plan;
4.    Loop until either the resource constraints are satisfied or all uncertain points
      are examined;
5.  }

Answering agent () {
6.    When (being asked by another agent about an uncertain point) {
7.        Identify the corresponding state(s) in the local state diagram;
8.        Reply with the action(s) (or none) planned for the state(s);
9.        Record the agent's name with the state(s);
10.   }
11.
12.   If (an action is removed from its state diagram/plan) { //**
13.       Inform all agents with names recorded with the state that the action is no
          longer planned for that state;
14.   }
15. }
```

Algorithm 6-1: The Convergence Protocol

The Convergence Protocol inquires about the states in descending order of the estimated resource consumption reduction *if* the unplanned actions (*ttacs*) of another agent in a state are removed. We postpone the discussion of picking the next uncertain point (line 1, labeled *) until Section 6.6.

6.3.1 Partial Convergence of Local Worldviews

If all features are public, i.e., there are no private features, then all agents have the same features. There is a one-to-one correspondence between the states in any two

agents' state diagrams. The answering agent can uniquely identify the state (if it exists) that the inquiring agent is asking about. As there can only be one planned action in a state for a CIRCA agent, it will reply with the single action (if planned) for the inquired state. All but one *ttac* in that state can be eliminated in the state diagram of the inquiring agent.

On the other hand, agents that distinguish between public and private features (Section 6.1) do not usually have the same set of features. Instead of sending all its state features, the inquiring agent sends a more abstract state description. A state description consists of only public features and their values. The answering agent may have more than one state in its local state diagram that matches the abstract description. It must reply with all the actions planned for those matching states. Accordingly, the inquiring agent may need to plan for more than one *ttac* in that state.

It is important to point out that an agent is asking which action another agent *would* execute if that other agent reaches a state (line 2). The answer can be found simply by looking up the state-action pairs in its reachability graph. If the agent is asking whether another agent *will* execute an action, that other agent may not be able to answer at all. The answer depends on its certainty about which states it will encounter during execution. In general, it cannot be certain of the reachability without the complete plans from all other agents. As the agents are refining their plans concurrently and asynchronously, there are no such complete plans before the multiagent planning process finishes.

The reachability graphs of agents can be very different. Some states may be in some graphs but not in others. In fact, agents typically do not even have the same sets of state features. Some agents may think that certain states are reachable while others may

think otherwise. For the example in Figure 6-4, FIGHTER thinks that enemies *may* appear at LOCF == LOC2, but BOMBER does not concur. In case of insufficient resources for agents to handle all states in their respective state spaces, the Convergence Protocol allows them to gradually and cooperatively discover which states are indeed reachable by exchanging only the relevant details of their plans. They are able to refine their individual plans by converging toward a set of commonly agreed-upon reachable states until their resource constraints are satisfied.

Note that they do not need to agree completely on the set of reachable states so all agents have identical state diagrams. The beauty of our protocol is that they only have to use it until they have enough information to schedule all their remaining actions after pruning *some* unnecessary ones. Even if an agent ends up allocating some resources to situations that cannot possibly arise due to incomplete knowledge, it is still acceptable. The agent has more than enough resources relative to all the demands it envisions. It can make all the guarantees it needs to make. If we require the agents to have identical state diagrams, other (more costly) protocols, such as [37], might work better.

6.3.2 Using the Convergence Protocol

The part of the protocol marked by (line 12, **) speeds up the pruning process of other agents by broadcasting to those who have previously inquired about (and therefore have shown interest in) a state if an action is no longer planned for that state. However, broadcasting information is expensive if there are a large number of agents. Even worse, some of those agents may no longer be interested in that state. For example, some may have already settled on their final plans; some may no longer consider that state reachable.

We therefore leave (**) as optional depending on how constrained a problem is and how costly it is to communicate. If the agents have very scarce resources, they probably want as much pruning information as possible. They should be kept updated of any pieces of useful information. On the other hand, if the agents are relatively rich in resources and only need to do a little pruning to satisfy their resource constraints, then the cost of broadcasting is harder to justify. Another alternative is to do (**) only at the beginning of the planning process when information tends to be more valuable.

Clearly, the Convergence Protocol terminates.[34] In the worst case, it terminates after all agents have examined all uncertain points in their reachability graphs. Moreover, if an agent starts with sufficient resources to prepare for all contingencies that are necessary, then it is guaranteed to find a plan that schedules all its actions. An agent's utility is not compromised if it can schedule all actions after running the protocol because the protocol always removes actions that are assured to be unnecessary. Its utility decreases only when it drops some necessary actions by raising the probability threshold.

Therefore, the order of inquiries an agent makes about the action choices of other agents is irrelevant to its utility. Similarly, the order of communication among agents is also irrelevant to the utility. Regardless of when an agent asks about a state, it will always get the most recent information about that state (by **) before it has to raise the probability threshold. In essence, what the agents ultimately end up finding as the definitely reachable states will be unaffected by the order in which states not in the intersection are pruned.

If an agent fails to schedule all remaining actions after completing the Convergence Protocol, it resorts to using the unlikely state strategy (Section 4.10) to

[34] This assumes a finite number of states in a domain.

remove the most unlikely (but possibly necessary) actions. In this case, the agent benefits from engaging in the partial plan exchange because it now has a more informed model of the plans of other agents, hence a more accurate worldview. Otherwise, it might remove more necessary actions in order to satisfy its resource constraints, yielding an even lower local utility.

6.4 Demonstration

We now continue our story in Figure 6-4. We demonstrate how our strategy lets agents cooperatively create a more coherent picture of the global activities. By reducing uncertainties in modeling other agents' plans in case of insufficient resources, agents may decrease their resource consumptions. Again, as mentioned in Section 6.1, a bomber (BOMBER) is given a mission to attack its choice of one of the two locations. After it attacks a location, enemy aircraft could arrive to retaliate. A fighter (FIGHTER) is assigned to patrol around the locations to repel the enemy aircraft whenever they arrive. Also, if FIGHTER requests a response from BOMBER, BOMBER has to report its status at LOC2. The complete state diagram (reachability graph) for BOMBER is repeated in Figure 6-5 (same as in Figure 6-1).

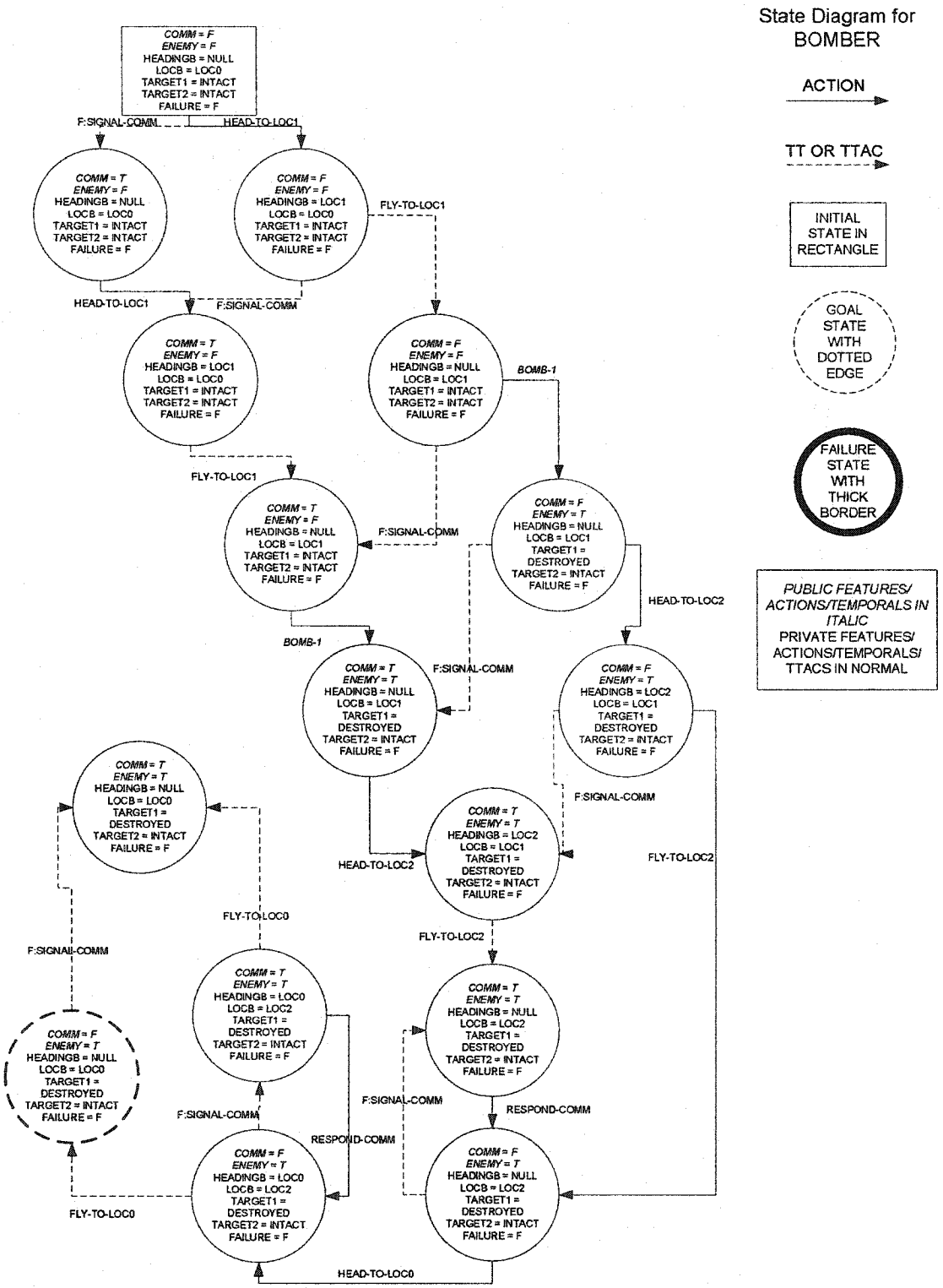


Figure 6-5: The Reachability Graph of BOMBER

Both FIGHTER (Figure 6-4) and BOMBER (Figure 6-5) have 5 actions to schedule if they do not know about the plan of the other agent. Suppose the resource constraints are simplified such that each agent can schedule only 4 TAPs. Both agents exceed their capacities. By running the Convergence Protocol, FIGHTER asks BOMBER what actions it plans to take when ((COMM == F) && (ENEMY == F)).[35] BOMBER replies that it is going to do BOMB-1. Then FIGHTER can safely remove the children of the *ttac* B1:BOMB-2, hence the action SHOOT-MISSILE-2, from its state diagram and plan. Likewise, BOMBER will discover that FIGHTER will not signal BOMBER to respond at LOC2. So, BOMBER can safely remove RESPOND-COMM from its plan.

As a result of communication using the Convergence Protocol, both of them have only 4 TAPs left for scheduling. Both agents in this case can satisfy their local resource constraints. Neither resorts to using the unlikely state strategy to drop actions. So, no agent's utility is compromised.

6.5 Circular Dependency of Actions

An attentive reader may wonder about this: an agent plans an action only because another agent plans another action. This other action is planned in turn because yet another agent plans another action, and so on. Will those actions be removed from the state diagram by the Convergence Protocol? When an agent plans ac_1 in a state that is reachable only because another agent takes action ac_2, we refer this as "ac_1 depends on ac_2," or $ac_1 \leftarrow ac_2$. We call ac_1 a *dependent action*. When ac_1 depends on ac_2, every path that goes from an initial state to the state having ac_1 contains ac_2 (or $ttac_2$ from the

[35] Agents communicate only the public features and actions.

perspective of the agent doing ac_1). Otherwise, we call an action an *independent action* if the state that it is in is reachable regardless of the plans of other agents.

It is easy to see that if the dependencies do not form a cycle (i.e., there is an independent action in the chain), then all actions in the chain will be removed by the Convergence Protocol after the independent action in the chain is removed. What is more interesting is the case when the dependencies form a cyclic chain (cycle) or an even more complicated cyclic group (cycles inside cycles). We define a cyclic group of dependent actions $\{ac_i \mid 1 \le i \le n\}$ as a set of actions (among agents) such that, for any i, there exists a j, such that $i \neq j$ and $ac_i \leftarrow ac_j$. n is the number of agents in the group.

We now show that it is not possible for agents to have such cyclic dependencies among their actions for progressive world state planners, such as CIRCA, which expand or search states starting from initial states (states that are guaranteed reachable). The intuition behind this is: the states associated with a group of cyclically dependent actions must be disconnected from the initial states. They will not be discovered or inserted into state diagrams during forward expansion. Consequently, the Convergence Protocol can always discover and remove *all* unreachable states. The proof is as follows.

Let ac be an action; $parent(ac)$ be the state having ac; $child(ac)$ be the child state of applying ac in $parent(ac)$. Let $A = \{ac_i \mid 1 \le i \le n\}$ be a cyclically dependent chain of actions. Denote $p_i = parent(ac_i)$. Note that these actions are planned by different agents. So, they are *ttac*s depending on the perspectives of the agents. We show that some states in $P = \{p_i\}_{i=1}^{n}$ are not reachable from the initial states.

We prove this by contradiction. Suppose all states in $P = \{p_i\}_{i=1}^{n}$ are reachable from the initial states (otherwise, the associated actions would not be planned in the first place, hence not in A). We denote the distance of $parent(ac_i)$ from the initial states by d_i. Let j be the index such that $d_j \leq d_i$ for any i. In other words, $parent(ac_j)$ in P is the closest to the initial states. By the definition of a cyclic group, there exists another action ac_k in A such that $ac_j \leftarrow ac_k$. As any path from the initial states to $parent(ac_j)$ contains ac_k, hence $parent(ac_k)$ and $child(ac_k)$, the distance of $parent(ac_k)$ is therefore smaller than d_j. This contradicts the proposition that d_j is the minimum. So, some states in $P = \{p_i\}_{i=1}^{n}$ are not reachable from the initial states. The agents cannot possibly plan the set of actions $A = \{ac_i \mid 1 \leq i \leq n\}$.

This shows an advantage of progressive world state planners. They only generate states that are potentially reachable from initial states. An unreachable state can be easily spotted by checking if there is a sequence of *tt*s, *ttac*s and actions that goes from the initial states to the state. The Convergence Protocol is designed to verify if these sequences are valid in the context of other agents' plans.

On the other hand, for those planners that explicitly or implicitly enumerate all possible states, e.g., MDP planners, it is difficult for them to identify a cycle of unreachable states. A cycle of unreachable states cannot be detected without collecting all the plans from all agents and analyzing the interacting path-dependent information, i.e., how an agent arrives at a state, as well as the dependence of one action of an agent on another. Those agents thus cannot take full advantage of the Convergence Protocol. Some

unreachable states may not be discovered. As CIRCA agents are progressive world state planners, the Convergence Protocol always removes all unreachable states.

Unfortunately, there is another type of dependency of actions that the Convergence Protocol fails to take into account. Actions in a state can preempt each other. Figure 6-6 shows such an example. According to the protocol, when agent 1 asks agent 2 whether it has planned $ttac_2$, agent 2 will reply yes. The same holds true when agent 2 asks agent 1 about $ttac_1$.

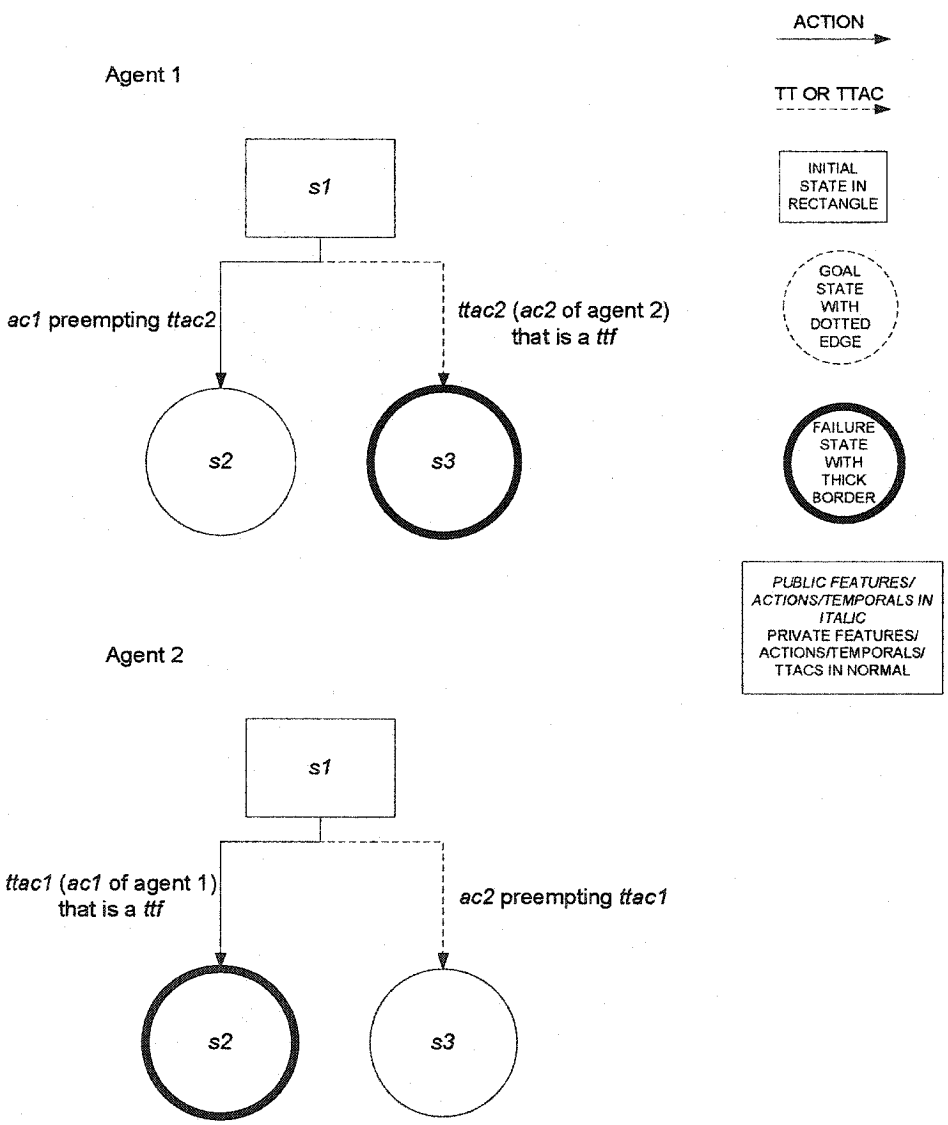


Figure 6-6: Cycle Anomaly

Although both states s_2 and s_3 are reachable (the Convergence Protocol removes only unreachable states), they are pointless. Actions ac_1 and ac_2 exist only to preempt each other to avoid failure, while at the same time (ignorantly) causing the other agent to fail. Therefore, both agents can benefit from taking out this pair of actions to reduce their resource consumptions. The Convergence Protocol, however, does not allow agents to negotiate in this way to make agreements. We will discuss how agents may change their plans to make better resource allocation decisions in Chapter 7.

6.6 Choice Function

This dissertation has been implicitly assuming that exchanging information costs something. Otherwise, agents would simply dump their entire plans or reachability graphs to other agents to share *all* information. In general, communication cost can be computational, e.g., (planning) time, power, or bandwidth, and/or non-computational, e.g., the risk of eavesdropping. Therefore, agents should reduce their communication costs by minimizing the number of inquires made. They somehow have to pick the "right" information to exchange.

The part of the Convergence Protocol (Algorithm 6-1) marked by (*) applies a heuristic *choice function* to choose the next best uncertain point an agent should inquire about. The more effective the choice function is, the sooner the protocol leads an agent into finding a satisfying plan, and the less computation and communication the agent does. In general, the states that are closer to the initial states and have more *ttac*s should be examined with priority. These states tend to have more downstream children states, hence more planned actions taking up resources. So, they tend to free up more resources *if* removed.

We have considered the following heuristics. They are listed in order of increasing complexity.

1. Random Choice Function: the agent inquires about a random state. This heuristic serves as a benchmark.
2. Sequential Choice Function: the agent inquires about the states in the order of their expansions during planning. If the state expansion is a breadth-first search, then the states closer to the initial states tend to be examined before others.
3. Distance Choice Function: the agent inquires about the states in ascending order of their distances to the initial states. The distance of a state is the minimal number of transitions that take the agent from any of the initial states to the state.
4. Load Choice Function: The agent inquires about the states in descending order of their numbers of actions per branch. The idea is to prune actions as fast as possible.
5. Utilization Choice Function: not all actions have the same utilizations. For example, an action with execution time 5 and period 10 has utilization 0.5 while an action with execution time 3 and period 5 has utilization 0.6. If we have a choice between the two, we would like to remove the one having a higher utilization to free up more resources. The agent thus inquires about the states in descending order of utilization per branch.

Each of these choice functions allows an agent to find satisfying plans at different converging speeds. Choosing the right choice function is important because it can speed up the planning process and dramatically reduce the costs as in the number of messages exchanged. Figure 6-7 shows a sample relation between utility and communication cost

using the simplest choice functions (repeated experiments). We will discuss the converging rates of all our choice functions in Section 6.8.1.

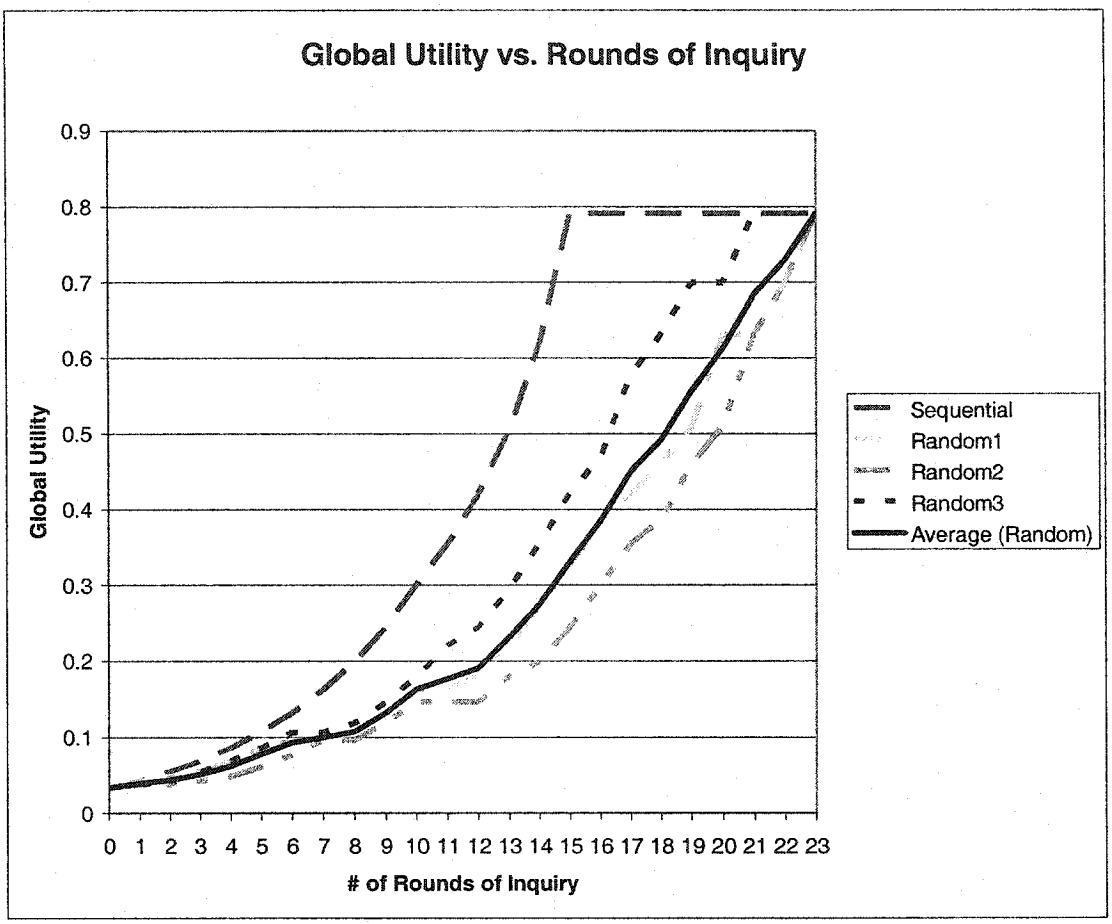


Figure 6-7: Varying Converging Speeds of Choice Functions

6.7 Anytime Property

If the agent's goal is only to find the best plans possible, and there are no costs to planning (e.g., time) and communication (e.g., bandwidth), then they should continue running the Convergence Protocol until either they have exhausted all uncertain points or they have found satisficing plans. Otherwise, they can stop when the cost of exchanging further information is too high or further information has little value. In this case, the anytime nature of the Convergence Protocol becomes particularly important. We note

that the agents never generate worse plans by exchanging messages, meaning that the agents never become worse off by decreasing uncertainties about the global activities. In the following, we illustrate the anytime nature of the Convergence Protocol by two examples.

Again, the utility of an individual agent i is (Section 3.2):

$$U = (1 - F_i)^{\alpha} G_i^{(1-\alpha)} = \overline{F}_i^{\alpha} G_i^{(1-\alpha)} \qquad \textbf{eq. 6-5}$$

The global utility for a group of CIRCA agents is (Section 6.1):

$$U = (1 - F)^{\alpha} (G)^{(1-\alpha)} = \prod_i (1 - F_i) \prod_i G_i \qquad \textbf{eq. 6-6}$$

The first small example involves two agents, A1 and A2. The reachability graph of A1 is shown in Figure 6-8. If A1 does not talk to A2, A1 needs to plan for 5 actions. They are, in descending order of their state probabilities, AS0, AS1, AS2, AS3, and AS4. AS0, AS1 and AS2 are conditional on A2 executing A2:P0, A2:P1 and A2:UNREMOVABLE-P2. In this example, A2 plans UNREMOVABLE-P2 but not P0 and P1. So, AS0 and AS1 are conditional and unnecessary actions. AS2 is conditional but cannot be pruned. AS3 and AS4 are necessary actions because they preempt unconditional events – the temporal transitions DANGER_S3 and DANGER_S4. The resource constraint for A1 is such that it can schedule only 3 actions.

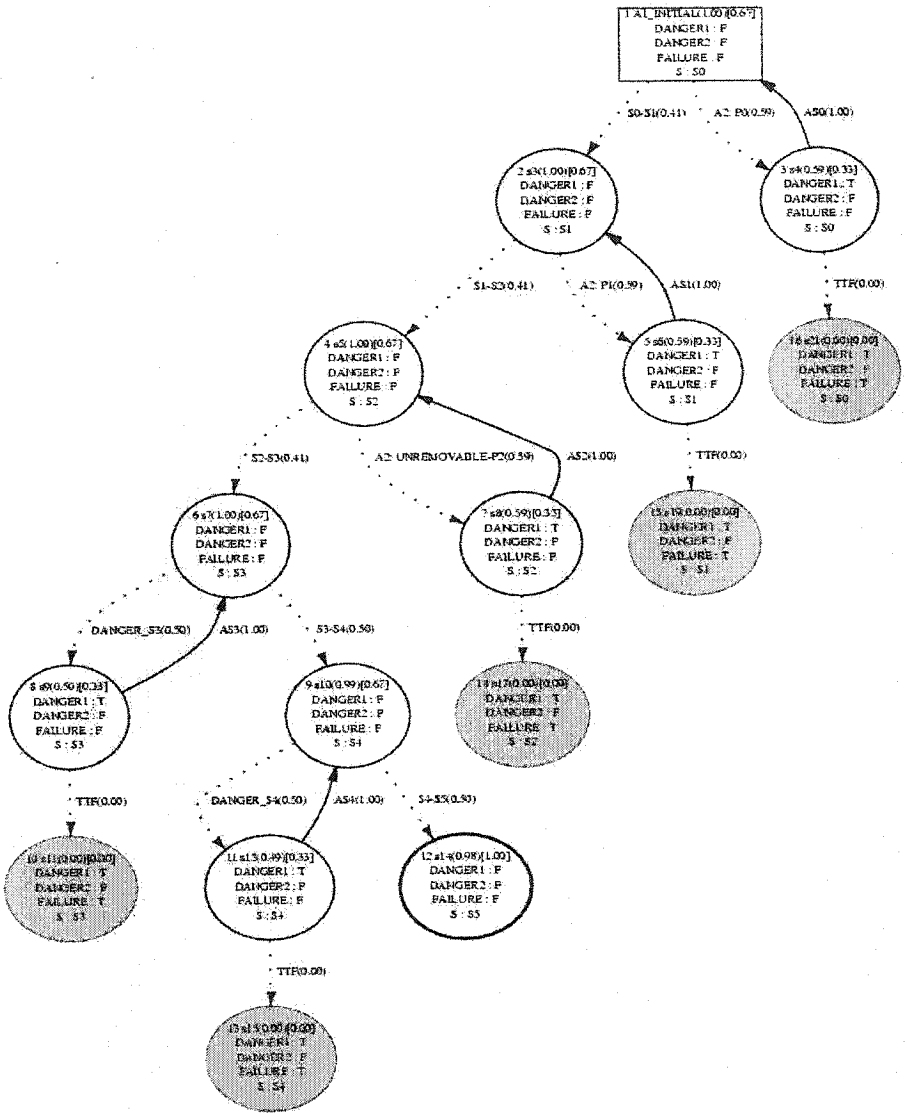


Figure 6-8: The Reachability Graph of Agent A1

Without running the Convergence Protocol, A1 removes two actions using the unlikely state strategy (Section 4.10). It needs to raise the probability threshold to remove the unlikely actions. The final schedule consists of {AS0, AS1, AS2}. The quality of this plan is very poor because two out of these three actions turn out to be unnecessary in light of A2's plan.

A1 can make more informed resource allocation decisions by running the Convergence Protocol. Using the sequential choice function, A1 finds out that A2 has not

planned A2:P0. So, it can safely remove AS0 from its initial plan. Similarly, after the second round of inquiry, A1 learns that A2 has not planned A2:P1. Action AS1 is removed. The schedulable plan, assuming using the unlikely state strategy as needed, after each round of communication is:

1. {AS1, AS2, AS3}
2. {AS2, AS3, AS4}

The anytime profile for this example is shown in Figure 6-9.

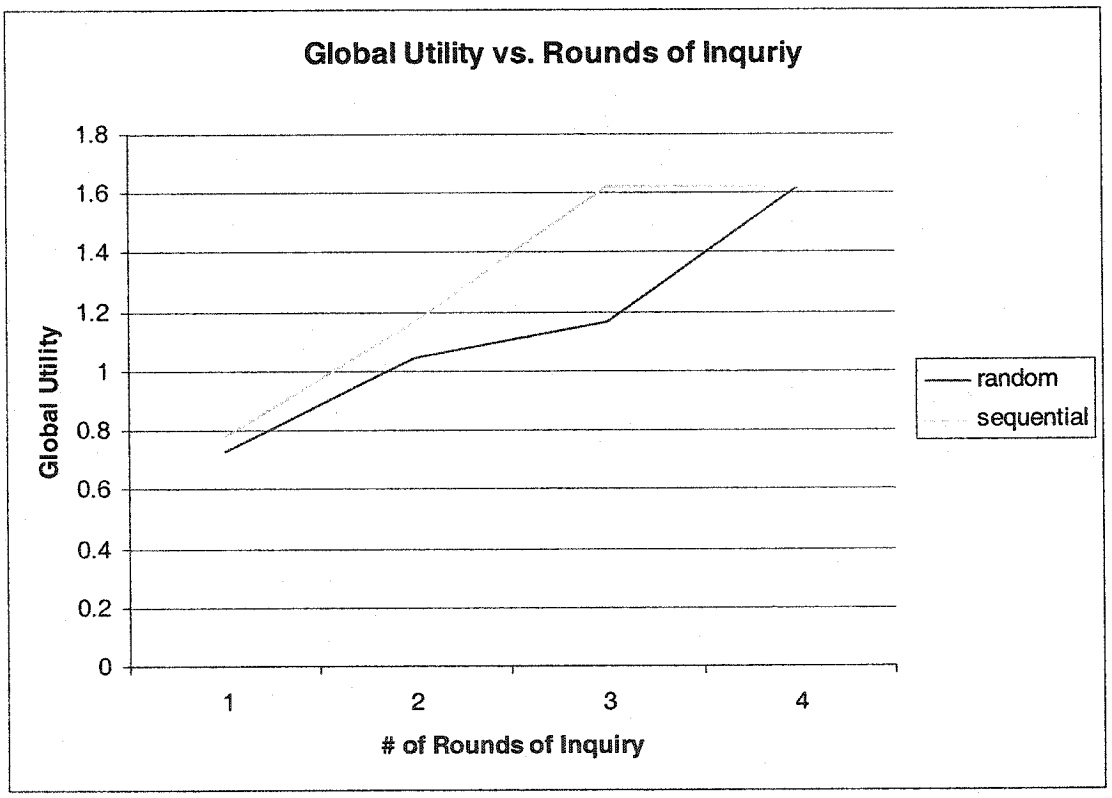


Figure 6-9: The Anytime Profile for the Team Consisting of A1 and A2 per Round of Communication

We have extended this small example by adding more states in a systematic way by duplicating the structure. For agent A1, there are now 14 unnecessary actions and 15 necessary actions. A1 does not know which are necessary and which are unnecessary

before running the Convergence Protocol. The resource constraint for A1 is such that it can schedule only 15 actions. 5 out of the 15 necessary actions are preempting temporal transitions to failure. So, they must be in the schedule regardless of A2's plan. They are similar to AS3 and AS4 in the previous example (Figure 6-8). A1 thus has to decide which 10 of the 14+(15-5)=24 actions go into the final schedulable plan. In the best case, A1 identifies all 14 necessary actions after sending 14 inquiries. In the worst case, it has to send 24 inquiries. The anytime profile for the global utility is shown in Figure 6-10.

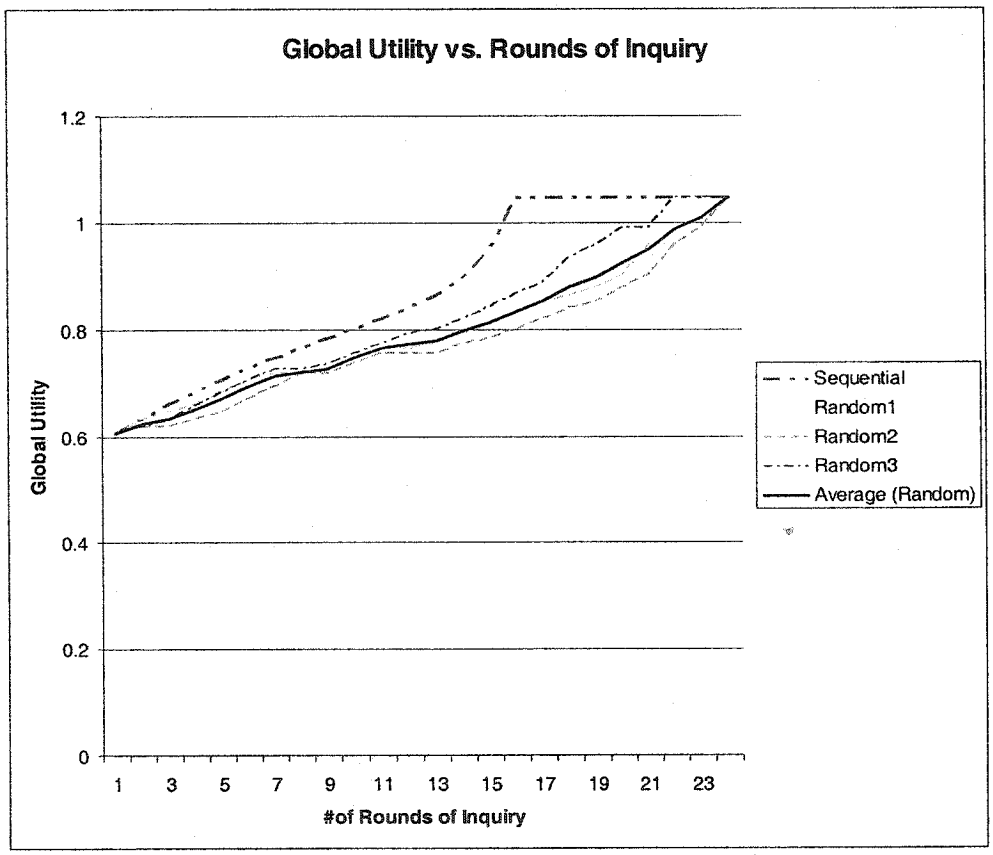


Figure 6-10: The Anytime Profile for the Team Consisting of A1 and A2 per Round of Communication Using Different Choice Functions

We show also the anytime profiles for the global failure probability and global goal probability per round of communication in Figure 6-11 and Figure 6-12. They are the factors in computing the global utility (Figure 6-10).

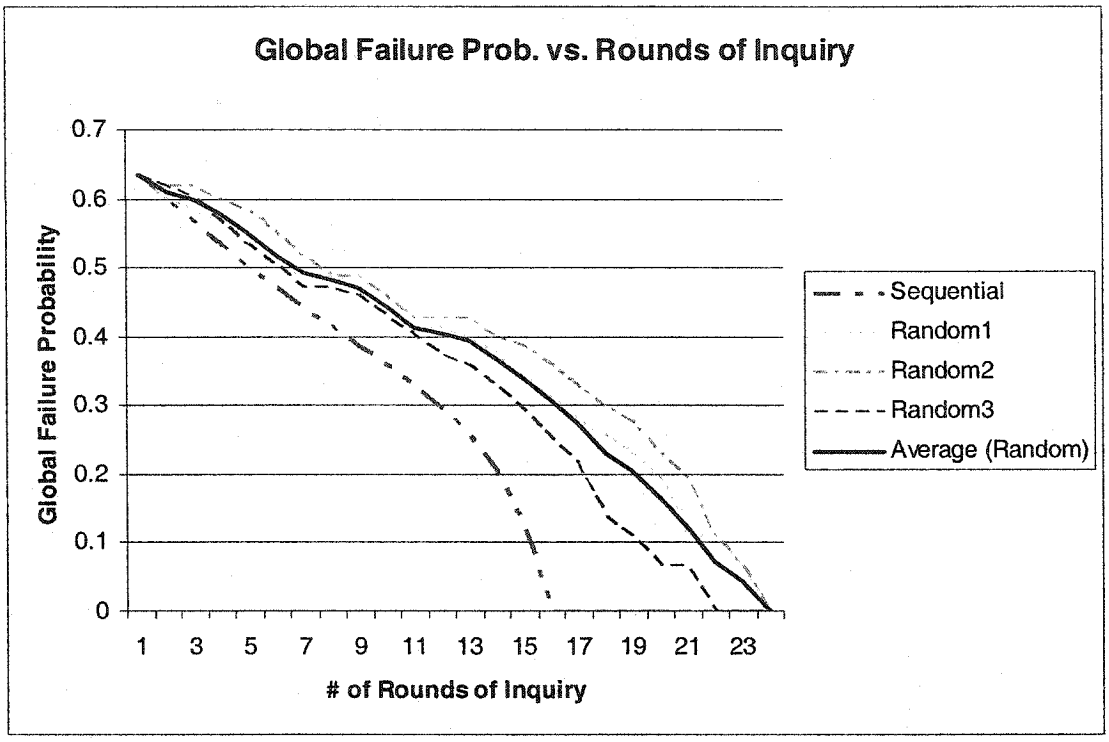


Figure 6-11: The Global Failure Probability for A1 and A2 per Round of Communication Using Different Choice Functions

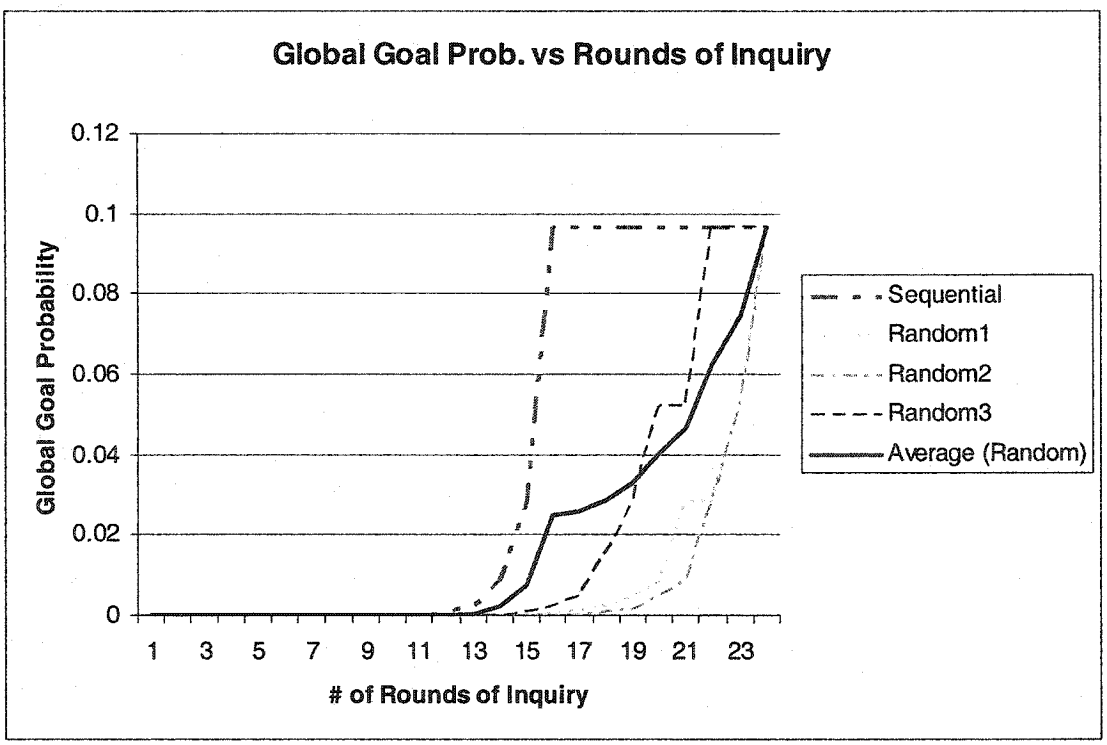


Figure 6-12: The Global Goal Probability for A1 and A2 per Round of Communication Using Different Choice Functions

6.8 Evaluation

To evaluate the merits of the Convergence Protocol, we have generated a set of random domains. Each domain consists of a random number of agents. Each agent is given a random knowledge base (KB). A KB is created by the KB generator described in Section 4.13. Here we break down the number of features into the number of private and public features. The percentage of public features, hence private features, in a domain is random. In our experiments, any public feature is shared by all agents. So, all knowledge bases for any given domain have the same number of public features. Likewise, all knowledge bases for any domain contain the same set of public temporal transitions and public actions.

In addition to the 12 parameters used to generate a KB for a single agent CIRCA, in a multiagent domain there are 2 more parameters. They are the number of agents and the number of public features, called the *connectivity*, in a domain. Connectivity measures how much the agents are coupled, i.e., how many public features the agents have in common. The 14 parameters used and their ranges are shown in Table 6-1.

We have generated 1626 agents (KBs) for 287 domains with which we performed our experiments. The number of agents that are able to schedule all their TAPs *before* running the Convergence Protocol is 202 (12.42%). The number of agents that are able to schedule all their TAPs *after* running the protocol is 704 (43.30%). In other words, 502 (30.87%) agents *become* able to schedule all their TAPs.

For those agents that still fail to schedule all remaining actions, they, nonetheless, drop fewer necessary actions (by applying the unlikely state strategy) than they would otherwise have needed to because they each now has a more accurate worldview. For our

experiments, the reduction in the number of necessary actions dropped is on average 59.55% with a standard deviation 39.85%. As a result, the agents' utilities are not as compromised as they would otherwise be without running the Convergence Protocol.

Table 6-1: KB Parameters and Their Values

number of agents	2 – 10
connectivity	1 – 7
number of private and public features	5 – 10
number of initial states	1 – 5
number of goal descriptions	1 – 5
number of features in goal descriptions	1 – 5
number of random actions	1 – 15
number of precondition features in an action	2 – 6
number of postcondition features in an action	1 – 5
number of *tt*s	1 – 20
number of precondition features in a *tt*	2 – 6
number of postcondition features in a *tt*	1 – 5
number of *ttf*s	1 – 15
number of precondition features in a *ttf*	2 – 6

Furthermore, we measure how effective the Convergence Protocol is by what we call action effectiveness and state effectiveness. *Action effectiveness* is the percentage of unnecessary actions removed by the protocol. Likewise, *state effectiveness* is the percentage of unreachable states included in an agent's reachability graph but removed by the protocol. The average state effectiveness is 53.74% and the standard deviation is 29.45%. Similarly, action effectiveness has an average of 51.74% and standard deviation of 35.84%. As our samples cover a large variety of domains, the standard deviations of these measures are big. We would therefore like to determine the most significant factors that contribute to the performance of the Convergence Protocol. We do this in Section 6.9.

The data suggest that more than half of the resources, at least in our random sample domains, are often wasted when an agent is ignorant of the plans of other agents. When the agents include in their planning all conceivable interactions based on all public actions of other agents, very often more than 50% of the states that they think they may encounter are in fact not reachable. Communication is therefore very important for resource-limited agents. Our protocol allows the agents to remove those states from their state diagrams by inquiring about just enough information from other agents.

6.8.1 Choice Function Efficiency

To study the convergence speeds and overheads of the various choice functions (Random, Sequential, Distance, Load, and Utilization) in Section 6.6, we define choice function efficiency and communication efficiency. *Choice Function Efficiency* is the convergence speed. The bigger it is, the faster an agent finds a satisficing plan. It is measured by the number of actions (states) removed per inquiry. Based on the same set of experimental domains, we have the statistics in Table 6-2.

Table 6-2: Choice Function Efficiencies

Choice Function Efficiency	Random (Action)	Random (State)	Sequential (Action)	Sequential (State)	Distance (Action)	Distance (State)	Load (Action)	Load (State)	Load (Action)	-Load (State)	Utilization (Action)	Utilization (State)
average	0.26	1.72	2.14	12.19	2.40	13.88	0.59	3.39	0.17	1.16	0.76	4.45
standard deviation	0.48	2.78	4.30	23.86	4.40	24.66	1.65	8.22	0.16	0.73	2.22	12.23

As we had expected, inquiry prioritized in terms of utilization (Utilization) is more efficient than in terms of number of actions (Load). Surprisingly, the efficiency of a choice function does not necessarily increase with its complexity. The Load and

Utilization choice functions are both worse than Sequential and Distance choice functions, but are still better than Random. The reason for the poorer performances of both Load and Utilization heuristics is that they are considering the number of actions and utilization *per branch*. Even if a state has many actions in its children, those actions could concentrate in one of the many branches of the state. The inquiry priority of the state, i.e., utilization per branch, is therefore "diluted" by those branches having fewer actions.

Moreover, distance is the best among the 5 heuristics. The data confirm our hypothesis that the states closer to the initial states should be inquired about with a higher priority. These states have a bigger total number of actions/utilization in their descendent states rather than actions/utilization per branch, because an ancestor of a state has at least as many actions as the state itself.

It is important to choose a good choice function. The author had once accidentally inverted the sign of the inquiry priority when implementing the Load heuristic. The performance for this inverted Load heuristic is also shown in the table (-Load). As the data show, a bad choice function can lead to performance worse than that of Random!

Although choice function efficiency tells us how fast an agent finds a satisficing plan by getting rid of unnecessary actions (states), it does not tell us how costly communication is using the Convergence Protocol. We would like to measure the overhead for an agent to communicate using the Convergence Protocol as a member in a group. We define *communication efficiency* as the number of actions (states) removed per message sent. The communication efficiencies for different choice functions are shown in Table 6-3.

As expected, the communication efficiency is much lower than choice function efficiency for a given choice function. Not only does an agent have to send one message per inquiry, but it also has to send messages to answer all inquiries from other agents and update them about any removed actions. Even if the agent itself has sufficient resources so that it never asks any questions, it may still answer a lot of inquiries made by other agents.

Table 6-3: Communication Efficiencies

Communication Efficiency	Random (Action)	Random (State)	Sequential (Action)	Sequential (State)	Distance (Action)	Distance (State)	Load (Action)	Load (State)	-Load (Action)	-Load (State)	Utilization (Action)	Utilization (State)
average	0.09	0.61	0.15	0.98	0.17	1.13	0.13	0.75	0.07	0.53	0.11	0.73
standard deviation	0.09	0.48	0.19	1.06	0.22	1.22	0.98	1.58	0.06	0.35	0.12	0.68

Moreover, the ordering of choice functions in terms of communication efficiency follows closely the ordering by choice function efficiency as shown in the tables. They are, in descending order, {Distance, Sequential}[36], {Utilization, Load}, {Random}, {-Load}. The correlation is because the fewer inquiries that are sent, the fewer messages that are sent. When picking and designing a choice function, a user can thus focus only on the factors, e.g., distance to initial states, which affect the choice function efficiency.

Both tables show that action efficiencies are lower than state efficiencies. This is because CIRCA often plans an action in more than one state. A TAP has multiple copies of the same action in multiple states. Only when all these states are pruned from the state diagram can the TAP be safely removed from the schedule (Section 3.1). Also, some pruned states have no associated actions. These states do not change the utilization of a

[36] We order the choice functions by groups because the performances of the choice functions inside a set, {...}, are comparable statistically.

plan. However, they give an agent a better idea of what states it may visit and how the world may evolve. The agent can also more accurately compute state probabilities. Accordingly, the agent can more effectively apply the unlikely state strategy to drop the least likely necessary actions in case of very stringent resource requirements. As discussed in Section 6.3, all choice functions give equal performance in terms of utility and generate equivalent final plans.

6.9 Predicting Effectiveness

The effectiveness of running the Convergence Protocol in general depends on many factors, such as the number of agents, the degree of coupling (connectivity), the number of states, the structures of state diagrams, and the resource requirements of the TAPs. We would like to predict whether it is worthwhile to apply the protocol in an unknown domain, and more importantly, to determine what factors contribute to effectiveness. The methodology we use to solve this problem is Ordinary Least Square (OLS) regression [47]. Regression analysis is a statistical technique that attempts to explain movements in one variable, the dependent variable, as a function of movements in a set of other variables, called the independent (or explanatory) variables. Linear regressions need to be linear in the coefficients, but they do not necessarily need to be linear in the variables.

6.9.1 Prediction Using KB Parameters

To start with, we want to determine the relations between the performance and the KB parameters. We have run an OLS regression taking effectiveness as the dependent variable against the 14 KB parameters in Table 6-1. The regression results are shown in

Table 6-4. The degree of fitness of the regression is only 37%. The estimation error has an average of 21.89% and a standard deviation of 16.45%.

Our experiments show that for the same set of parameters, a large variety of KBs with very different performance profiles can be produced. Nonetheless, we can still identify the significant parameters. We in general consider a parameter significant when the norm of its t-stats is above 2.0. The significant parameters, which are highlighted in bold, are (in descending order of significance): number of *tt*s, number of precondition features in a *tt*, connectivity, number of random actions, number of precondition features in a *ttf*, and number of postcondition features in a *tt*.

Table 6-4: OLS Regression on KB Parameters

	coefficient[37]	t-stats (significance)
number of agents	0.571	0.871
connectivity	**8.287**	**7.775**
number of private and public features	1.453	1.756
number of initial states	-1.120	-1.770
number of goal descriptions	-0.896	-1.468
number of features in goal descriptions	-0.574	-0.879
number of random actions	**1.803**	**4.461**
number of precondition features in an action	-0.859	-0.965
number of postcondition features in an action	1.161	1.269
number of *tt*s	**-2.192**	**-15.254**
number of precondition features in a *tt*	**11.913**	**11.049**
number of postcondition features in a *tt*	**-3.793**	**-3.116**
number of *ttf*s	-0.070	-0.250
number of precondition features in a *ttf*	**3.727**	**4.540**
constant[38]	-26.176	-3.181

[37] A positive coefficient means that the effectiveness rises with the parameter; a negative coefficient means the opposite.

These results are by and large no surprise. As we would expect, the more *tt*s there are or the fewer precondition features there are in *tt*s (the *tt*s are more applicable,), the less likely that a state can be removed by the Convergence Protocol. When a state is reachable via a *tt*, no inquiry can have it removed. An inquiry may remove *ttac*s, which are actions of other agents, but it never removes *tt*s, which are exogenous events in the environment. The action planned for this state is thus necessary, meaning that it preempts an inevitable *ttf*, regardless of other agents' actions. This result is further confirmed in the next regression which shows that the proportion of actions planned for conditional events in the plan of an agent dominates other variables in predicting how useful the protocol is (Section 6.9.2).

The other significant factors include connectivity, which measures how much the agents are coupled. Understandably, the more interactions the agents have, the more advantageous it is for them to utilize the protocol in case of insufficient resources. Similarly, the larger the number of actions, the more likely some of them can be removed. What is more surprising is perhaps that the number of agents does not play a major role in determining the effectiveness. The effectiveness values of the Convergence Protocol for groups of 2 agents up to 10 agents are all relatively the same.

The predictions made based merely on the KB parameters certainly have room for improvement, e.g., a higher degree of fitness, a smaller average error, and a smaller standard deviation. The analysis based on KB parameters alone, nevertheless, provides a cheap and quick estimation of how beneficial it is to run the protocol in case of insufficient resources.

---

[38] The constants are listed so the reader can construct the analytical expressions from the tables.

6.9.2 Prediction Using Run-Time Statistics

The metrics that can make a more accurate prediction about an unknown domain are those that characterize reachability graphs. In other words, one can make a better prediction about whether the Convergence Protocol will be helpful to a problem by gathering some run-time statistics. We have done some regressions on different sets of variables of reachability graphs to identify the significant ones. The regression results on effectiveness against the most significant parameters that characterize the domain space are listed in Table 6-5.

Table 6-5: OLS Regression on the Most Significant Reachability Graph Parameters

	coefficient	t-stats (significance)
% of actions planned for unconditional events	-50.850	-22.563
ratio of unconditional actions over all actions other agents may execute[39]	-47.298	-12.203
ratio of planned actions over all actions other agents may execute	40.960	9.231
number of planned actions in reachability graph	0.909	4.274
number of applied *ttac*s	-0.188	-3.917
number of states that have *ttac*s	0.113	2.238
constant	58.250	22.615

The degree of fitness is 69%. The error has an average of 14.16% and a standard deviation of 12.43%. Clearly, this regression identifies more relevant variables. Its accuracy improves drastically comparing to the last regression. It also confirms that once someone estimates the proportion of necessary actions in a domain, he can get quite a good idea of how worthwhile it is to perform the Convergence Protocol. This conclusion

[39] They are distinct actions, so they are TAPs if included in the final plans.

should be intuitive. After all, the goal of the protocol is to remove unnecessary actions. The more unnecessary actions there are, the more effective the protocol is.

The two ratios are intended to measure how likely other agents are to remove actions from their plans. The higher the ratio of unconditional actions is, the less likely other agents are to remove their actions. Thus, the less likely the agent can remove states by inquiring about other agents' plans. On the other hand, the higher the ratio of planned actions is, the more unnecessary actions there are in other agents' plans, the more beneficial it is to use the protocol. The same argument holds true for the parameter about the number of total actions in the agent's own reachability graph.

6.9.3 Prediction Using the Most Significant Factors

By combining the most significant in the last two regressions, i.e., the KB parameters and reachability graph properties, we are able to improve the prediction. The regression results using both sets of parameters are listed in Table 6-6.[40]

The degree of fitness of this regression is 75%. The error has an average of 12.35% and a standard deviation of 11.68%.

To test how well we can make predictions on unknown domains using our regression results, we applied the analytical expression from the third regression to a separate and independent set of 76 KBs. The outcome is very encouraging. The average error is only 11.27% with a standard deviation of 12.89%. In fact, other than a few skewing data, 62% of them have errors less than 10%, and 86% of them less than 20%. Therefore, quantitatively, we have obtained a statistically good heuristic that allows us to predict the effectiveness of running the Convergence Protocol. Qualitatively, we have

[40] Only the significant variables are shown.

identified the important parameters that determine how useful it is to apply the Convergence Protocol.

Table 6-6: OLS Regression on the Most Significant Factors

	coefficient	t-stats (significance)
% of actions planned for unconditional events	-48.264	-23.192
number of precondition features in a *tt*	7.195	10.159
number of *tt*s	-0.947	-9.046
ratio of unconditional actions over all actions other agents may execute	-32.385	-8.104
ratio of planned actions over all actions other agents may execute	34.823	8.070
number of postcondition features in a *tt*	-3.997	-5.249
number of total actions in reachability graph	1.519	4.852
number of states that have *ttac*s	0.179	3.642
number of precondition features in a *ttf*	-1.672	-2.951
number of random actions	-0.592	-2.247
number of applied *ttac*s	-0.126	-2.211
constant	54.007	11.338

6.10 Summary

In this chapter, we have shown that ignorance about other agents' activities can be very detrimental to the agent's local resource allocation. In our experiments, more than 50% of the resources may be wasted when the agents are ignorant of the plans of others. To address this problem, we have developed the Convergence Protocol.

The rationale behind the protocol is this: some events occur only when the agents interact in certain ways. An ignorant agent may waste resources on those events that will

never occur. Intuitively, the more the agent knows about other agents' activities, the better it can utilize its resources. Using the Convergence Protocol, the agents can cooperatively determine the set of states for which they need to react to by exchanging partial plans. Each agent can consequently have a more coherent view of the global activities.

Moreover, the agents do not need to acquire the entire plans from the other agents, because many details in those plans are irrelevant to the agents' local planning. We have developed various choice functions, which are heuristics the agents can use to prioritize what information are more important than others to acquire. Our experiments show the tradeoffs between complexity and efficiency of various choice functions. When communication is costly but computation time is not, we can adopt a choice function having a higher efficiency to minimize the overhead, and vice versa.

Chapter 7

Improving Resource Allocation among Collaborative Agents

In the last chapter, we have introduced the Convergence Protocol that helps an unschedulable agent eliminate unreachable states. As a result, the agent can better apply the unlikely state strategy to remove the unnecessary (zero probability) and low probability actions to reduce its utilization. One implicit assumption that the protocol makes is this: while the agent removes actions in states that it considers to be impossible or unlikely to reach, it does not change the actions planned for other states above the probability threshold. Each agent settles on its state-action mapping for all potentially reachable states before engaging in the Convergence Protocol.

Consequently, the global performance as measured by the global utility is bounded by the plan each agent initially constructs. While this assumption is valid for some kinds of planning algorithms, such as those that select actions for states independently of the action planned in other states, e.g., the previous generations of CIRCA that do not use the reusing action strategy [2, 87], it might not faithfully represent the predilections of other planners.

In general, an agent selects actions for states based in part on the action choices in other states that it has already made (the partially-constructed plan) and/or the choices it might make in the future (lookahead). For instance, plans are generally considered to be less costly when they involve fewer actions. Many AI planners, such as the operations research based approach of [10], ILP-PLAN [63], CPlan [112], and the approach of [98], try to reduce plan length, which is essentially the number of actions in a plan, as a criterion to optimize resource usage. In Section 5.4, we have introduced the reusing

action strategy to reduce plan length. When a single action's effect satisfies the requirements in multiple states, CIRCA may choose it over distinct actions for each of the states, even if the single action would not be the preferred choice for some (or all!) of those states when each would be considered separately.

When an agent chooses an action to achieve a goal or respond to a situation based on the other actions that are already in its plan (or that are expected to be added to its plan), we say that the decision is based on its *plan context*. While such contextualized decisions, e.g., reusing actions, often make more efficient use of limited resources, they are also much more susceptible to change. For example, if the agent learns that a particular event for which it had planned is assured of not happening, not only should the agent remove the action to respond to that event but it should also reconsider whether the remaining actions now are appropriate given the new context.

For example, a reaction to brake when approaching stopped traffic could be generalized to also brake for slow-moving traffic, or merging traffic, or obstructions in the road. While other responses might be better for these other events (perhaps coasting when facing slow-moving traffic, and swerving when encountering obstructions), a driver might be unable to afford to include these in the repertoire of responses without risking too long a delay of indecision when confronted with stopped traffic. However, if the driver discovers that it is unnecessary to be prepared to brake for stopped traffic (e.g., everyone agrees to pull onto the shoulder if the car breaks down), then not only should the driver remove that reaction from the plan (e.g., brake for a stopped car in the road), but he might reconsider whether he can afford to take the more appropriate "coasting" or "swerving" actions for the other situations.

While the above discussion about context change is for the multiagent case, it can happen in the single case as well. When an agent applies the unlikely state strategy, the state space gets smaller. The agent may be able to generate a better plan for the reduced state space. However, our experiments (not reported in this thesis) show that for most cases, a CIRCA agent benefits only marginally from repairing its plan after using the unlikely state strategy. One reason for this: in the single agent case, it is relatively easier to "foresee" the possible context changes and factor them in during planning. For example, the reusing action strategy avoids choosing the replacing actions that are likely to be dropped. Similarly, the unlikely state strategy drops actions that have the minimal impact to the plan.

In contrast, in a multiagent environment, as we have shown in Chapter 6, as many as 50% of the states in an agent's state diagram can turn out to be unreachable due to ignorance about other agents' plans. First, it is difficult for an agent to factor in planning something that it is ignorant of to make its action choices more context-insensitive. Second, removing 50% of the states from its state diagram indeed constitutes a major change in the plan context. Consequently, an agent in a multiagent environment is in a better position to reconsider some parts of its plan in the new context to improve the plan. Thus, our study of improving a plan using context change information will focus on the multiagent case.

We have shown in Chapter 5 that the action replacement algorithm (Algorithm 5-1) is very effective in improving a plan by exploiting post-planning information. We would thus hope that agents, after they become more aware of the global activities by

running the Convergence Protocol, can improve their plans by using (an extended version of) the action replacement algorithm to exploit the context change information.

However, things are exacerbated as agents interact closely. When an agent changes its local plan, chances are that other agents should be notified of the changes. This could trigger a chain reaction of changes to ripple through other agents' plans. Also, the agent often needs to collect feedback from other agents to evaluate the local changes it wants to make. Besides the huge plan space the agent may have locally (Section 5.1), we must control how the agents cooperate and how information is passed among them to make the search for plan improvements tractable and systematic.

In this chapter, we examine the problem of how a group of agents may search for local changes to their plans to improve global performance. We characterize the high-level computational complexity of the problem and show that finding the optimal changes in a multiagent environment is in general very difficult. In response to this, we show that the agents can improve their plans by reconsidering some context-sensitive actions that might have become suboptimal in light of additional knowledge of other agents.

Specifically, we present a distributed search protocol, which the agents can use to iteratively improve their plans in a controlled, hill-climbing manner. We combine this protocol with the action replacement algorithm (Algorithm 5-1) that exploits context change information. The agents performs an *efficient*, *informed*, and *focused* search for plan improvements by proposing and evaluating *only* the local changes that are most likely to be globally beneficial.

7.1 Example of Improving Multiagent Resource Allocation

To ground our discussion in terms of context change, we illustrate why an unschedulable agent might want to change its local plan after it has acquired new information from other agents. We consider the case where the costs of some actions change after running the Convergence Protocol.

For CIRCA, the cost of an action equals the utilization of schedule. In Chapter 5, when an agent fails to schedule all preferred actions, it will use the reusing action strategy to replace some actions with less desirable actions that are also in the plan. We have distinguished between preferred actions of a TAP (the "best" actions) and reusing actions (the replacing actions that are less desirable). The marginal costs of the reusing actions are assumed to be zero, because the agent has already scheduled the preferred actions of the same TAPs.

After the agent exchanges some of its plan information with other agents, some TAPs have all their preferred actions dropped because those actions were planned for unreachable states. The costs of scheduling the reusing actions belonging to these TAPs become non-zero because they are no longer reusing other actions in the plan. The reason for reusing these TAPs is now gone. The agent may produce a better plan by reconsidering the action choices in the states that have those reusing actions. The agent may replace them by better alternatives having lower costs (if any).

For example, suppose there are two security agents who are responsible for monitoring a computer system for signs of hacking by periodically scanning four ports and running tests on the gathered data. Assume agent 1 is less capable than agent 2. Agent 1 can only scan one port at a time and perform a quick/simple test. In contrast,

agent 2 can scan more than one port at a time and run a full test. The scanning activities must be so scheduled that hacking signs are detected fast enough to ensure predictive sufficiency [88] and to isolate the system before sensitive information leaks out. In other words, an agent must scan a port frequently enough to conclude that the system is safe in the interval between any two scans of the port.

Agent 1 is responsible for scanning port 1. Agents 1 and 2 together are responsible for scanning ports 2, 3, and 4, and can divide the responsibilities between them. If agent 2 scans only port 2, agent 1 cannot guarantee complete safety as it cannot scan ports 3 and 4, along with port 1, fast enough. Agent 1, given these tasks, would thus formulate a plan that would always shut down the network to the system to be on the safe side. If agent 2 instead scans both ports 2 and 3, then, ideally, agent 1 would want to scan port 4 if it had the resources. Unfortunately, agent 1 cannot schedule all these actions – SCAN1 (0.4), SHUT-DOWN (0.6), SCAN4 (0.3), and QUICK-TEST (0.2) – as the total utilization is $0.4 + 0.6 + 0.3 + 0.2 = 1.5 > 1$. The reachability graph for agent 1 is shown in Figure 7-1.

To reduce utilization, agent 1 applies the reusing action strategy. It reuses the action SHUT-DOWN in the shaded state. SHUT-DOWN was previously planned for the case where agent 2 scans only port 2. Now, due to the lack of resources, agent 1 plans also SHUT-DOWN even if agent 2 scans both ports 2 and 3. This decision is less desirable than SCAN4 if there were no resource constraints. In other words, agent 1 stays conservative. It shuts down the network to the system anyhow so no security might be compromised. This plan consists of SCAN1 (0.4), SHUT-DOWN (0.6), and QUICK-

TEST (0.2). The SHUT-DOWN action is reused in two states, and the utilization reduces to $0.4 + 0.6 + 0.2 = 1.2 > 1$.

Yet, agent 1 is still unschedulable. After running the Convergence Protocol, agent 1 knows for sure that agent 2 scans both ports 2 and 3. There was no need to plan for the situation where agent 2 scans only port 2. One of the SHUT-DOWN actions was planned for an unreachable state. As a result, there is no longer incentive to reuse SHUT-DOWN when agent 2 scans both ports 2 and 3. In light of this new information, agent 1 can now become schedulable simply by choosing the other action, SCAN4, instead. SCAN4 in this state *changes* to now have a lower cost than SHUT-DOWN in the new context because whereas SHUT-DOWN used to be free (when it was needed anyway), now it is not. The new plan consists of SCAN1 (0.4), SCAN4 (0.3), and QUICK-TEST (0.2). The utilization is $0.4 + 0.3 + 0.2 = 0.9 < 1$. Agent 1 becomes schedulable.

As this example shows, exploiting new information available after a context change usually involves modifying a local plan, e.g., replacing action SHUT-DOWN with action SCAN4. Theoretically, all agents can revise their plans in many ways. The space of possible revisions to the agents' state-action mappings is huge as will be discussed in the next section.

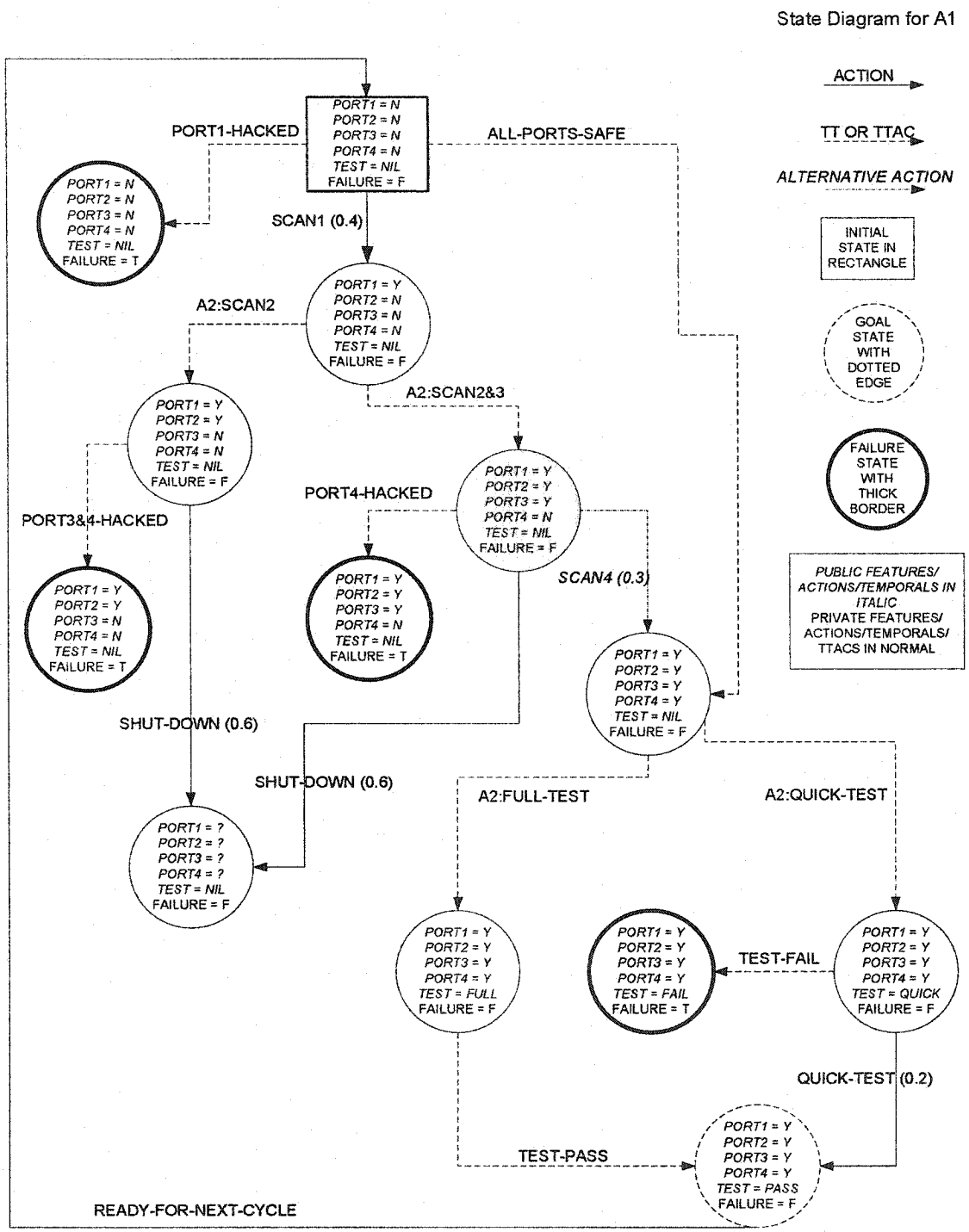


Figure 7-1: The Reachability Graph of Agent A1

7.2 Problems with Multiagent Plan Improvement

Ideally, after becoming more aware of the multiagent context by running the Convergence Protocol, unschedulable agents can modify their local plans so that their plans are satisficing with respect to their local resource constraints, or if there are no such plans, that the global utility of the multiagent system is increased as much as computation allows.

However, for an agent to reason about the relationship between local changes and the global utility, it would require significant amounts of communication to gather information from all other agents, assuming that the agents are even willing to disclose the (possibly private) information. In the case of CIRCA, an agent, after running the Convergence Protocol, knows which *ttac*s in its local state diagram have been chosen by other agents, but the agent still does not have enough data to reason about the global utility. The agent would also need to know what actions other agents have chosen in the non-intersecting parts of their state spaces (in the case of using the public and private features), other agents' utilizations, and how constrained they are. The global utility depends on the complete plans of all agents and their resource constraints. No single agent has all the information.

Therefore, we have chosen to give up on trying to increase the global utility directly. Instead, an unschedulable agent tries to maximize its local utility. The rationale is that there is approximately a monotonic relationship between local utility and global utility: the higher the local utility is, the higher the global utility usually is. Although searching for changes that maximize the local utility is itself another intractable problem, we have already shown that the action replacement algorithm (Algorithm 5-1) is effective

in improving a local plan. Thus, we will use this algorithm in the multiagent case as well. The post-planning information an agent exploits is the context change information.

There are two complications about assuming the monotonic relationship. First, it is not always true that the global utility increases with the local utility of an agent. For example, the agent may find a change in its local plan that reduces its utilization significantly, allowing it to schedule more actions locally and thereby lessen its need to ignore unlikely states. However, the change could lead to areas in the state space that cause other agents to schedule a lot more actions. While the local utility increases significantly, the global utility may actually decrease.

Nevertheless, in an application where the utilizations of plans are the main concern, the interests of the agents usually align. For instance, an agent would like to reduce the number of reachable states that require actions. It may try to avoid getting itself, hence other agents, into a portion of the state space with many reachable states. When the number of reachable states decreases, other agents' utilizations tend to decrease as well. Although an optimal local plan is not necessarily always part of an optimal combination of agent plans, the local utility of an agent and the global utility tend to move in the same direction.

The second problem about the monotonic relationship is that all (unschedulable) agents may want to revise their plans. Some of them may try multiple revisions before finding a successful local plan to improve the global utility. The changes made in one agent's plan may become invalidated by the changes other agents make. Each agent should always perform its local search using the most updated information about the global state of the multiagent system. Information flow among agents must be both

selective and timely. The search for local improvements by agents must be conducted in a well-controlled and distributed manner.

7.3 Iterative Refinement of Agents' Plans

Our solution to improve a combination of agent plans using the new knowledge after a context change, e.g., after running the Convergence Protocol, is this. A specific unschedulable agent computes a possible way of modifying its plan (local changes). We call such a possible modification a *proposal*. A proposal is expected to increase the agent's local utility as much as possible. The proposal is broadcast to all relevant agents for evaluation. If the new global utility is higher than the current global utility, the proposal is adopted, leading to a new set of agent plans. Otherwise, this proposal is rejected.

The agent repeats this propose, evaluate process until either it finds one that increases the global utility, or it has exhausted all promising proposals. Another agent may now propose. This cycle continues either until all agents are schedulable, or all unschedulable agents have had all their proposed changes rejected, or they run out of time (give up). This Propose-Evaluate Protocol is formally described in Algorithm 7-1.

Input:

a set of agents' plans and a context change

Output:

an improved set of agents' plans having a higher global utility

Algorithm:

```
1.  while (some unschedulable agents have not proposed &&
2.          there is still time for computation) {
3.
4.      select an unschedulable agent, A, who gets to propose plan changes;
5.
6.      agent A do { // this do-while loop is for agent A; other agents wait till signaled
7.          computes the next best proposal, P, using a proposal-ordering heuristic;
8.
9.          // agent A has no more proposal for evaluation
10.         // another agent may now propose
11.         if (no more promising proposal) {
12.             P = nil;
13.             break;
14.         }
15.
16.         broadcast P to all agents involved;
17.         //all agents including A run the Convergence Protocol to synchronize;
18.         wait for their feedback;
19.         compute the new global utility;
20.
21.         if (the global utility improves) {
22.             // the first acceptable proposal has been found
23.             break;
24.         } // else, find the next best proposal
25.
26.     } while (P is not accepted);
27.
28.     //all agents adopt P and make changes to their local plans accordingly;
29.
30.     // loop back to step 1; another agent may propose
31. }
```

Algorithm 7-1: The Propose-Evaluate Protocol

7.3.1 Selecting a Proposing Agent

There are two choice points in this protocol. First, the agents need to select a proposing agent (line 4). As it is a steepest ascent hill-climbing algorithm, a reasonable heuristic for selecting the next proposing agent would be to pick the agent that can increase the global utility the most for the same (unit) increase in local utility. Mathematically, this agent has the biggest rate of change of the global utility. That is, the partial derivative of the global utility with respect to its local utility is the biggest.

In terms of CIRCA, when it considers only safety (or failure), the local and global utilities given in Section 3.2 and 6.1 are reproduced in eq. 7-1 and eq. 7-2.

The local utility of an individual agent *i* is:

$$U_i = (1 - F_i) = \overline{F}_i \qquad \textbf{eq. 7-1}$$

The global utility for a group of CIRCA agents is:

$$U_g = (1 - F) = \prod_j (1 - F_j) = \prod_j \overline{F}_j \qquad \textbf{eq. 7-2}$$

The rate of change of the global utility for agent *i* is thus:

$$\frac{\partial U_g}{\partial U_i} = \frac{\partial \prod_j \overline{F}_j}{\partial \overline{F}_i} = \prod_{j \neq i} \overline{F}_j \qquad \textbf{eq. 7-3}$$

The rate of change of the global utility for agent *i* is bigger when the safety probabilities of *other* agents are bigger. Equivalently, the rate of change for agent *i* is inversely proportional to the safety probability of the agent. We thus select the agent that has the minimal safety probability, i.e., the maximal failure probability, to propose. In other words, the "worst off" agent is the first agent to try to make changes to its local

plan. After it proposes, the next agent that now becomes the worst off may repeat the propose-evaluate cycle, and so on.

7.3.2 Computing a Proposal

The second choice point involves how an agent computes the next proposal (line 7). Again, from the perspective of a hill-climbing search, the agent should make the changes that are expected to increase the global utility the most in its local plan. Using the monotonic relationship approximation between local and global utilities, the agent needs to order the proposals by how much they are expected to increase the local utility. The agent proposes and evaluates the proposals in the list one by one until it finds one that is acceptable to all agents.

However, having an agent produce, let alone compare, all proposals is unrealistic because of the large space of possible changes. Instead, the agent should only produce proposals on demand. That is, the agent produces only one proposal at a time. Only after this proposal is rejected should the agent produce another proposal.

Moreover, the agent should propose only the changes that are likely to increase local utility. We distinguish between two types of possible proposals. We call a proposal *promising* if it consists of only changes that are related to the context change. Otherwise, we call the proposal *unpromising*. There is no justification why a plan transformed by an unpromising proposal might be better than the current plan. The new information due to context change does not indicate in one way or the other why the changes are beneficial (or harmful) and can only stay agnostic about them.

There are (hopefully) significantly fewer promising proposals than there are possible proposals. There are in general an exponentially large number of possible

proposals. By evaluating only promising proposals, the search space size can be reduced (significantly). In other words, while theoretically an agent may propose and evaluate just about *any* plan, the agent practically should only propose and evaluate the promising proposals that it has justification about why those changes make sense. After the agent exhausts all the promising proposals (line 11), another agent may propose.

Therefore, to have an agent propose only promising proposals and generate proposals on demand, we introduce the concept of a *proposal-ordering heuristic*. A proposal-ordering heuristic uses context change information to generate proposals in descending order of their *expected* increments in the local utility. The heuristic tries to generate proposals in an order that approximates the optimal ordering of proposals as closely as possible. It is exactly the proposal-ordering heuristic that makes the protocol in Algorithm 7-1 an informed distributed search process. While in the worst case the agent may still need to generate and examine all proposals, it is hoped that, on average, the agent will find an acceptable one (a lot) sooner by using a proposal-ordering heuristic.

There are conceivably many ways to design a proposal-ordering heuristic because there are many different pieces of information that can be incorporated into a heuristic. If domain-specific knowledge is available, a heuristic can take advantage of that as well. We have considered a particular proposal-ordering heuristic that is appropriate for CIRCA-like systems that, in case of insufficient resources, reuse actions to reduce utilization.

7.4 Ordering Proposals

The intuition behind our proposal-ordering heuristic is that some replacing actions, such as SHUT-DOWN in Figure 7-1, may become invalidated after some states are

discovered to be unreachable. As explained in Sections 5.4.2, 5.5.1, 5.5.2, and 5.5.4, selecting a replacing action that will be removed from the schedule is a bad replacement decision. Selecting a replacing action belonging to a TAP whose preferred actions are removed is also a bad replacement decision. Yet an agent cannot avoid all the bad decisions because of ignorance about other agents' plans.

Our heuristic thus examines whether the replacing actions still make sense after the context change due to running the Convergence Protocol. When some replacing actions turn out to be undesirable, the agent may produce a new local plan by replacing them with other more desirable actions in the new context. The reusing action information gives justification to the new plan about why it is potentially better than the current plan.

A proposal-ordering heuristic does two things. First, it generates a proposal; second, it generates the next best proposal when a proposal is rejected. In the following, we first describe how a proposal is generated by replacing the invalidated reusing actions. Then we extend the action replacement algorithm (Algorithm 5-1) so that we can systematically search for the proposals that change those invalidated reusing actions.

7.4.1 Generating a Proposal

A proposal is a set of changes in an agent's plan. Each change is essentially an action replacement. In other words, a proposal is a set of action replacements. To make a change in a plan, an unschedulable agent needs to answer two questions: 1) what actions to replace (the replaced actions) and 2) what actions to replace them by (the replacing actions). We have already shown in Chapter 5 that the action replacement algorithm answers these two questions. To avoid examining all possible combinations of changes in

a plan (essentially being able to generate any new local plan), and to generate only promising proposals, we thus apply the action replacement algorithm to improve the plan.

To apply the action replacement algorithm, we need to design an action ordering heuristic and a replacement heuristic (Section 5.2). In Chapter 5, we use the post-planning action cost information. In this chapter, we can take advantage of the context change information available after running the Convergence Protocol. It is about the states that are found to be unreachable, and more importantly, their actions that are removed from the schedule.

Specifically, we define the *preference count* of a TAP as the number preferred actions the TAP has.[41] The preferred actions of a TAP are the "best" actions in the states. The other actions belonging to the TAP are reusing actions. Before running the Convergence Protocol, TAPs having a reusing action must have a positive preference count. After running the Convergence Protocol, some TAPs may have their preference counts drop to zero. Those TAPs are the ones whose preferred actions are planned for unreachable states. They are thus removed from the schedule. The reusing actions of the TAPs consequently become invalidated due to the change in the context that they were planned for. The marginal cost for each reusing action is no longer zero. Other actions may now be more desirable. Note though that the reusing actions may still be the preferred actions in the new context.

To improve the local plan, an agent thus can focus on the states that have reusing actions whose TAPs have preference counts that have become zero. We call those actions *suboptimal actions*. In fact, with only the information about what actions are removed

[41] Preference count counts only the preferred actions of a TAP; reference count (Chapter 5) counts both the preferred and reusing actions. So, for a TAP, the preference count is always less than or equal to the reference count.

from the schedule after running the Convergence Protocol, the agent does not know how to modify the states other than those states having suboptimal actions. The agent has no indication to justify why replanning for other states may yield a better plan. Moreover, when the number of states having suboptimal actions is significantly smaller than the number of reachable states, we cut down tremendously the search space of proposals.

Therefore, the action ordering heuristic is to order the actions by their TAPs' preference counts in ascending order. The replacement heuristic is to keep the actions if their preference counts remain positive. Otherwise, we can use one of the three heuristics in Section 5.4.2 to replace the actions whose counts are zero. For our implementation, we allow introducing new actions that are not already planned; we do not allow introducing new reachable states; we allow actions that increase schedule utilization.

Unlike the reusing action strategy developed in Chapter 5, we do not limit the replacing actions to only those actions that are already planned. That is, we are not reusing actions. In this chapter, we may replace an action whose cost has increased after a context change by any cheaper action that preempts the *ttf*s in the state. The advantage of this relaxation is that the agents can find more eligible replacing actions. The downside is that the schedule utilization may increase as a result of replacements because the marginal cost of adding new actions is not necessarily zero (Section 5.4). The utility may decrease. So, the agents need a filtering mechanism to skip proposing changes that do not increase local utilities. We call this particular usage of Algorithm 5-1 the *suboptimal action strategy* because we are replacing actions that are potentially suboptimal.

7.4.2 Generating the next Proposal

Our proposal-ordering heuristic incorporates this suboptimal action strategy to generate proposals. It also has a mechanism to generate the next proposal when a proposal is rejected by other agents (line 7 in Algorithm 7-1). Given the same set of suboptimal actions and their alternatives, the suboptimal action strategy always generates the same proposal. Yet, to generate a different proposal we need to systematically explore the different combinations of suboptimal actions and alternatives.

Our proposal-ordering heuristic is a chronological backtracking algorithm that uses the suboptimal action strategy. When a proposal is rejected, it removes the alternative, which is the action selected in the rejected proposal, in the domain of the state having the lowest priority. The domain of a state consists of the eligible action candidates to replace the suboptimal action. We assume that each domain has at least one candidate to replace the suboptimal action, otherwise, the state needs not to be considered.

As the domain of the state has changed, applying again the suboptimal action strategy will generate a different proposal having a different action replacement in that state. After all action candidates in a state have been tried, or equivalently, the domain of the state has become empty, it tries to make a replacement change in the state one level up. The proposal-ordering heuristic is shown in Algorithm 7-2. Note that the suboptimal action strategy is not guaranteed to generate a proposal that increases the local utility. The agent needs to filter out those ineligible proposals that do not increase its local utility (lines 6-11).

Input:

n suboptimal actions

Output:

a successful proposal

Algorithm:

```
1.  assign priorities (1 ... n) to the states having suboptimal actions;
2.  initialize the current priority, cp, to n, and backtrack level, bl, to n;
3.
4.  while (a successful proposal has not been found) {
5.      generate a proposal P using the suboptimal action strategy;
6.      if (P leads to a higher local utility) {
7.          // P is broadcast to all relevant agents for evaluation
8.          if (P is accepted) {
9.              return P;
10.         }
11.     }
12.
13.     // P leads to a lower local utility or is rejected
14.     // denote s as the state having priority = cp
15.     while (the domain of s has only one candidate left) {
16.         // try changing the state one level up
17.         cp--;
18.
19.         if (cp < bl) {
20.             bl--;
21.         }
22.         // all proposals exhausted
23.         if (bl == 0) {
24.             // hill-climbing fails; another agent may now propose
25.             return nil;
26.         }
27.
28.         // determine the backtracking state;
29.         update s = the state having priority cp;
30.     }
31.
32.     // backtrack
33.     eliminate the action candidate selected in the domain of s that is in P;
34.     reinitialize the domains of the states having priorities below (>) cp;
35.     reset cp = n;
36. }
```

Algorithm 7-2: The Proposal-ordering Heuristic Incorporating the Suboptimal Action Strategy

7.5 Evaluation

Our evaluation has two parts. First, we would like to justify the Propose-Evaluate Protocol that coordinates the distributed search for resource allocation improvement among multiple agents. Second, we want to assess the effectiveness of using the suboptimal action strategy to generate proposals. To do the evaluation, we have generated 250 random problem instances. Each sample consists of a random number of agents, among which at least one of them is unschedulable and have successfully replaced actions using the reusing action strategy. The samples are generated using the random KB generator described in Section 4.13.1. The KB parameters and their ranges are shown in Table 7-1.

Table 7-1: KB Parameters and Their Values

number of agents	2 – 8
connectivity	1 – 5
number of private and public features	5 – 8
number of initial states	1 – 5
number of goal descriptions	1 – 5
number of features in goal descriptions	1 – 5
number of random actions	8 – 15
number of precondition features in an action	2 – 4
number of postcondition features in an action	1 – 5
number of *tt*s	5 – 15
number of precondition features in a *tt*	2 – 6
number of postcondition features in a *tt*	1 – 5
number of *ttf*s	5 – 10
number of precondition features in a *ttf*	2 – 6

Comparing to the KB values in Sections 4.13 and 6.8, the KB values in Table 7-1 are so chosen that there are more actions in the samples and that there are more alternative actions in a state for an agent to choose from. The intentions are, first, to make

the resource allocation problem more difficult. Second, we want the agents to have more action candidates to replace the suboptimal actions because our technique is useful only when the agents can identify suboptimal actions *and* can actually correct them. Also, the actions in this set of samples have bigger worst case execution times (*wcet*) so that the agents are more resource constrained – they can schedule fewer actions. It is set up this way so that we can show the improvement in the global resource allocation after running the Propose-Evaluate Protocol.

7.5.1 Evaluating the Propose-Evaluate Protocol

In our samples, the global utility increases, on average, by 8.46% (standard deviation 10.53%). To interpret these statistics, we have to realize that some samples have no agents that have suboptimal actions (43.50% of the samples), or that there are no alternative actions available to replace them. These agents cannot possibly apply the suboptimal action strategy to use with the Propose-Evaluate Protocol. The performance of the protocol thus appears to be not as compelling as it actually is. For the samples that improve the global resource allocation by using the Propose-Evaluate Protocol, the average increase in global utility is 15.78% (standard deviation 9.68%).

We argue that this performance improvement justifies the Propose-Evaluate Protocol. The reader is reminded that we are comparing plans that are generated using the protocol to the plans that are not. Yet, the agents apply to all these plans the reusing action strategy (Section 5.4), the Convergence Protocol (Section 6.3), and finally the unlikely state strategy (Section 4.10). Even if the agents do not use the Propose-Evaluate Protocol, they can already generate very good plans using the combination of the other three techniques. Thus, our evaluation of the protocol is done by comparing plans to the

already improved and provably good plans. As our overall approach is essentially a hill-climbing algorithm, there are more plateaus and, expectedly, smaller improvements as the agents refine their plans to get closer to the optimal.

Moreover, there are upper bounds on about how much a plan can be improved. The ideal utility (safety probability) of a plan without resource constraints is 1.0. So, if an already good plan (produced by using the techniques in the previous chapters) has a safety probability of 0.8, the maximum improvement is 25%. The average utility (safety probability) without applying the Propose-Evaluate Protocol in these samples is 0.76.[42] It increases to 0.88 if the protocol is applied. In light of this, 15.78% improvement for the agents having suboptimal actions is reasonably good. Likewise, one reason why some agents do not have suboptimal actions is that their utilities may already be close to the resource-permitting optimal. Thus, as a corollary to our conclusion, making a big improvement in a poor plan is easy; making a small improvement in a good plan is challenging.

In terms of cost, 33% of the agents that propose find a successful proposal (Section 7.4) in the first round of communication (the first time they propose). 32% of the agents need more than 5 rounds (or never find one). Note that some agents do not propose because either they are already schedulable, or they cannot correct any suboptimal actions, or no changes can lead to a higher local utility.

These agents are usually the ones who are the latest to propose, i.e., the agents having the highest utilities in their domains. By the time they can make proposals, the worse-off agents have already improved the global utilities to the point that little further

[42] This is slightly lower than that of the samples in the previous chapters because we intentionally increase the *wcet*s of actions.

improvement can be made. There are two reasons for this. First, as shown in Section 7.3.1, a worse-off agent makes more improvement per unit change in its local utility. Second, as discussed above, it is more difficult to improve a better plan.

Despite the fact that these agents are less likely to yield improvement to the global utilities, the Propose-Evaluate Protocol still requires them to search through all unlikely but possible proposals. This adds extra communication overhead to the cost of running the protocol. The overall cost of running the Propose-Evaluate Protocol measured by the number of rounds of communication is thus approximately proportional to the number of agents (multiplied by a small constant).

In summary, our experiments substantiate our claim that agents can achieve a good improvement in the global resource allocation at a relatively low cost of communication overhead using the Propose-Evaluate Protocol. The rationale behind this is that the agents can identify the potentially suboptimal actions using additional knowledge acquired from communication, e.g., a context change after the Convergence Protocol. Using the knowledge, they can refine their plans by proposing only the changes that are most likely to improve the resource allocation. The Propose-Evaluate Protocol is thus an iterative, hill-climbing approach that performs an *efficient*, *informed*, and *focused* search of plan improvements.

7.5.2 Evaluating the Suboptimal Action Strategy

Because the agents use the suboptimal action strategy with the Propose-Evaluate Protocol when they search for global resource allocation improvement, the effectiveness of the protocol is thus limited by how applicable the reusing action information is. Using the reusing action information, the agents identify suboptimal actions. If the information

cannot help the agents find any suboptimal actions, then it is useless. The information does not help them improve their plans. The agents would need other pieces of information to identify suboptimal actions so that they can use the protocol.

In general, whether the reusing information is useful depends largely on the planning algorithms that the agents use. As CIRCA applies the action replacement algorithm to replace actions by reusing other planned actions, intuitively it can gainfully examine whether some of these replacement decisions are still valid after a context change, especially when a significant number of actions are proven to be planned for unreachable states. In our experiments, 56.50% of the samples have agents that have suboptimal actions. For each of these agents that can find a proposal, 17.56% of their planned actions are suboptimal (stdev 8.21%). Our empirical results confirm the intuition.

7.6 Summary

In general, an agent selects actions for states based in part on the action choices in other states that it has already made (the partially-constructed plan) and/or the choices it might make in the future (lookahead). That is, the agent makes a planning decision based on a plan context. The plan context, however, may change when the agent communicates with other agents because it becomes more aware of the global activities, such as after performing the Convergence Protocol. In this case, the agent should in principle reconsider the entire combination of actions in its plan. Furthermore, as one agent changes its planned actions, it could trigger a chain reaction of changes to ripple through other agents' plans.

In this chapter, we have developed the Propose-Evaluate Protocol, which the agents can use to search for local plan improvements in a distributed, controlled manner

with a modest communication overhead. The rationale behind this is: using the additional knowledge acquired from context change, the agents can identify the potential suboptimal states. They can revise their plans by proposing to change the action choices that are most likely to be suboptimal to improve resource allocation. The protocol thus allows the agents to perform an efficient, informed, and focused search for plan improvements toward satisfying local resource constraints.

Chapter 8

Conclusions and Future Work

In this dissertation, we have studied the resource allocation problem for a resource-limited real-time agent in a dynamic and probabilistic world whether it acts alone or operates in a multiagent environment. In this chapter, we will summarize our conclusions and discuss possible future directions.

8.1 Summary of Results

In this dissertation, we have developed the unlikely state strategy (Chapter 4), the reusing action strategy (Chapter 5), the Convergence Protocol (Chapter 6), and the Propose-Evaluate Protocol using the suboptimal action strategy (Chapter 7). When a real-time agent in a multiagent environment is unschedulable, it will first use its initial knowledge to plan appropriate actions to preempt failures in the world. It will locally attempt to schedule these actions, including applying the reusing action strategy. If the agent succeeds, then it does not need to gather further information. Otherwise, it exchanges partial plans with other agents using the Convergence Protocol to discover and prune away unreachable states (and the actions it had planned for them).

The agent now becomes more aware of the global activities and can construct a more accurate worldview. It may realize that some action decisions it has made are now suboptimal in the new plan context. If the agent is still unschedulable, it can search for local changes to its plan to correct these suboptimal decisions using the suboptimal action strategy. The agent discusses the changes with other agents using the Propose-Evaluate

Protocol. If the agent still remains overloaded, it can finally use the unlikely state strategy to drop the least likely needed actions repeatedly until its plan becomes schedulable.

8.1.1 The Unlikely State Strategy

The idea behind the unlikely state strategy is to drop the least likely used actions from an unschedulable plan to make it schedulable. This strategy always succeeds. It leaves some failures unhandled so it should only be used as the last resort. Despite this, our experiments show that most of the time an agent can generate a better schedulable plan using the unlikely state strategy than simply dropping random actions. The utility is, on average, 26.48% higher.

To determine which actions are the least likely used, we have designed a probabilistic representation that captures the temporal dynamics of exogenous events and actions. This framework allows us to analyze the complex temporal coupling relations among concurrent transitions. Without enumerating all possible combinations of events, we can compute the transition probabilities of an action in different contexts from the temporal dynamic descriptions of the action and events.

In general, specifying the temporal dynamics (probability functions) of a transition and using it to compute transition probabilities in a continuous domain is extremely computationally expensive. We have introduced an efficient discretization method that approximates the continuous probability functions by piecewise constant probability rate functions. Using probability rate functions, not only can a user simplify the specification of transitions but he can also *estimate* transition probabilities in a simple step-wise manner. Moreover, we have proven that the accuracy of this approximation increases as the user uses a finer discretization of the timeline.

When the execution trajectory of a real-time agent is Markovian, we can compute the state probabilities analytically. Otherwise, we draw on tools from operations research and statistics to simulate the stochastic process to estimate the probabilities. Using the state probabilities, the agent can make informed resource allocation decisions if it is impossible to construct a plan that includes all failure-preempting actions.

Therefore, our probabilistic action and event representation enables researchers, who develop artificial agents in time-critical applications, to project the temporal trajectories of the agents. Using the discretization method, the researchers can simplify the descriptions of stochastic events and can make a tradeoff between computation cost and accuracy. Moreover, our results indicate to the researchers that the unlikely state strategy is effective in producing reasonably good schedulable plans. They should consider, among other factors, the state probabilities of actions – the probabilities of the actions being needed during execution – when their agents need to remove actions from their plans.

8.1.2 The Reusing Action Strategy

We have generalized the unlikely state strategy to the action replacement algorithm. Besides dropping an action, i.e., replacing the action by a NOOP, we allow an agent to replace the action by another cheaper action. Specifically, this action replacement algorithm iteratively improves an unschedulable plan toward satisfying the resource constraints by exploiting post-planning information. It is an informed, hill-climbing search for plan improvements.

To make the algorithm computationally tractable, we limit the scope of replacing actions to those that are already in the plan. That is, we replace an action by only another

action that is already planned. So, the marginal cost of a replacement is (usually) zero. Essentially, the agent reuses a planned action to satisfy the needs in multiple states. The utilization thus decreases after each replacement at the cost of having a lower utility. This is the reusing action strategy.

To select an action to replace, we can either order the actions by utilizations or reference counts. To replace an action, we can reuse an action that has the smallest utilization, biggest reference count, or largest TAP probability. To limit the search space size, we may or may not allow new states to be introduced into the state diagram during the replacement process. We also may or may not allow increasing schedule utilization. There are in total 24 different implementations of the reusing action strategy, and there is no dominant implementation. We have identified the circumstances where which of the heuristics and constraints are most appropriate.

One important conclusion is that the agent should always select a replacing action that will stay in the final schedule (after using the unlikely state strategy). Otherwise, not only does the corresponding failure become unhandled, but the agent might become worse off by using the reusing action strategy. The agent might have been able to use another replacing action to preempt the failure.

As a result, while the reusing action strategy helps, it may occasionally produce a worse plan. Our experiments show that, 78% of the time, the agent can beneficially use the reusing action strategy. The utility of the agent increases, on average, by 21%. We have found that if we do not impose constraints on the search space size, i.e., having a bigger search space, 12% more of the time an agent can successfully find a replacing

action to make a replacement. The computation cost in terms of trying more actions for each replacement increases modestly by 20% – 35%.

Therefore, researchers building resource-limited agents in real-time domains can apply the reusing action strategy to resource allocation in real-time domains. Traditionally, the resource allocation algorithms make restricted assumptions that are inappropriate for real-time applications such as that the transition probabilities are stationary, explicit state enumeration is possible, the time horizon is finite, and/or the environments are static and non-deterministic. Using the reusing action strategy, a real-time agent can reduce resource consumptions and still be able to preempt the transitions to failures.

8.1.3 The Convergence Protocol

In a multiagent environment, the uncertainty of other agents' plans impairs an agent's ability to model its temporal trajectory. The agents may ignorantly spend resources on planning actions in states that they will never reach during run-time. Our solution to this is: the agents first construct their reachability graphs using worst-case scenario analysis and then iteratively refine their plans using the Convergence Protocol. The agents cooperatively determine the set of states to which they need to react by exchanging partial plans to generate more coherent views of their activities. The beauty of this protocol is that the agents exchange just enough information for them to schedule their actions. They do not necessarily have and do not need identical views of the world.

Our experiments suggest that it is often worthwhile for agents to exchange partial details of their plans if they have inadequate local execution resources. More than 50% of the resources may be wasted when they are ignorant of the plans of other agents. We

have also determined the significant factors that contribute to the effectiveness of the Convergence Protocol. A user can estimate the effectiveness in his domain using our regression results to see if the cost to perform the protocol is justified. The percentage of actions planned for unconditional events in an agent's reachability graph is the single most decisive factor.

We have provided three analytical formulae for estimation. The first formula is based only on KB parameters. It is quick and cheap but its error is 22%. The second is based on reachability graph parameters. The average error is 14%. The third formula combines the KB and reachability graph parameters. It is more accurate and has error of only 12%. We have evaluated this formula using an independent set of samples (cross-examination). The majority of the results have error less than 10%.

While the global utility of the final plan of an agent is independent of its inquiry order and is independent of the communication order among agents, choosing the right choice functions can reduce the cost, e.g., communication overhead, bandwidth, and risk of eavesdropping, to run the Convergence Protocol by reducing the number of inquiries. We have shown the tradeoffs between the complexity and efficiency of various choice functions. The distance choice function turns out to be the best *on average*. It supports the hypothesis that the states closer to the initial states should be inquired about with higher priority.

Therefore, our results indicates to researchers in multiagent systems that, in addition to the problems of resolving conflicts and coordination, we have also to confront the problem that, for agents with limited resources, many of the resources could be unnecessarily wasted. Many events the agents think that they may encounter may never

arise during execution. Researchers building agents in multiagent environments and the agents can interact in many possible ways should therefore incorporate the Convergence Protocol into the agents in addition to their coordination mechanisms to make more informed resource allocation decisions.

8.1.4 The Propose-Evaluate Protocol

As the agents exchange partial plans using the Convergence Protocol, they become more aware of the global activities. The plan context in which some of the actions they have planned may now change. Consequently, the agents may want to change their actions, which may have now become suboptimal, including those that they have already announced to other agents.

To conduct the search for plan changes in a distributed and well-controlled manner, we have developed the Propose-Evaluate Protocol. It is an iterative, hill-climbing algorithm that performs an efficient, informed, and focused search to increase the global utility. The protocol is very flexible to accommodate whatever methods an agent uses to locally search to improve its local plan. Using the protocol, the agent then discusses with other agents its proposal, which is a set of local changes it wants to make, and evaluates how the changes affect other agents' plans globally.

We have also introduced the concept of ordering proposals. An agent ideally would want to try all possible proposals, but it is impractical to do so because there can be an exponentially large number of them. First, an agent would not be computationally able to compute all of them. Second, another agent would not have the chance to propose if it had to wait for the agent to finish first. By ordering the proposals, the agent generates and thus proposes only the proposals that are the most likely to increase the global utility

the most. A proposal is generated only when the previous one is rejected by all other agents. Consequently, the agents can compute fewer proposals. They also exchange fewer messages to propose and evaluate the plan changes.

Our experiments support our claim that agents can achieve a good improvement in global resource allocation at a relatively low cost of communication overhead using the Propose-Evaluate Protocol. For the agents that are able to improve their resource allocation, the global utility increases, on average, by 15.38%. 33% of these agents make only one proposal.

For an agent to efficiently compute a proposal, we have extended the action replacement algorithm. Using the knowledge available after a context change, the agent can identify suboptimal actions using the reusing action information. Specifically, for actions whose preference counts have dropped to zero after running the Convergence Protocol, the reason for reusing those actions is now gone. Thus, the agent may replace them by cheaper actions in the new context. Our experiments show that 56.50% of the samples have agents that have suboptimal actions. For each of these agents that can find a proposal, 17.56% of their planned actions become suboptimal after running the Convergence Protocol.

Therefore, we have shown that agents can take advantage of the context change information to identify what actions have become suboptimal in the new context. Using the action replacement algorithm, the agents can perform an efficient, informed, and focused search for plan improvements toward satisfying local resource constraints. On the other hand, the Propose-Evaluate Protocol allows agents to collaboratively propose

and evaluate in a distributed, controlled manner *only* the local changes that are most likely to be globally beneficial.

Compared to other generic algorithms, the protocol applies to only the (still large) class of problems involving distributed planning. Yet, it can reason about domain knowledge that is inherent in planning problems so it is more efficient to solve distributed planning problems. When researchers build agents in multiagent environments and when they foresee there will be room for the agents to improve their plans later on due to changes in plan context, they can have the agents apply the protocol to improve their existing plans.

8.2 Future Work

There are two main directions for future research in terms of allocating limited resources for a real-time agent. They are extending the action replacement algorithm for the single-agent case, and extending the Propose-Evaluate Protocol for the multiagent case. We are going to discuss them in this section.

8.2.1 Generalizing Action Replacement

The action replacement algorithm is a simple hill-climbing algorithm. It considers each replacement decision only once and does not consider the combinatorial effects of choosing different replacing actions. Essentially, we look at only one steepest ascent path in the search space of plans. We can enhance the performance of action replacement by adopting the idea of Graphplan [8, 9].

We can construct a planning graph that relates each possible improved plan made by using a different combination of action replacements. Useful information for

constraining search can be propagated through the graph as it is being built. For example, we can prune away the branches that are not promising to yield any further improvement in plan quality. In other words, we use a search control mechanism to explore more of the plan space to compare different combinations of replacing actions. Other search paradigms such as breadth-first-search and branch-and-bound are possible.

So far, we have limited our replacing actions to those that are already in the schedule so that the marginal cost of reusing them in other states is zero. When there are multiple actions to replace, the sum of the marginal costs of all replacements is still zero. As a result, the schedule utilization is guaranteed to decrease. The downside is that the agent may not be able to replace an action in a state because no planned action can preempt the *ttf* in that state.

If we allow the agent to replace an action by another cheaper action that is not necessarily in the plan, the agent will be more successful in finding a replacing action because it has more choices. We would therefore like to enhance the action replacement algorithm to allow for actions that do not have a zero marginal cost. Unfortunately, we will run into a difficult credit assignment problem. Although each replacing action costs less than the replaced action in a state, the total cost may exceed the reduction in schedule utilization when all replacements are considered together. It is in general very difficult to determine which of the (isolated) replacement decisions to blame as being responsible for making the new plan worse.

To account for using *any* cheaper actions, we would have to extend the action replacement algorithm with some search control mechanism such as a planning graph as in Graphplan to keep track of the consequences of making a replacement. When a new

plan that is worse off than the current plan is found, some sort of backtracking is needed. Therefore, not only does a search control mechanism allow an agent to explore more ways to improve its plan using the already planned actions, but it also lets the agent add new actions to the plan.

8.2.2 Multiagent Negotiation

Similarly, the improvements that the Propose-Evaluate Protocol can make are limited by the fact that some actions that can be replaced to improve the plan quality are not recognized. Our experiments show that 43.5% of the samples do not have agents that have suboptimal actions. This does not mean that their plans cannot be improved. Rather, it means that, using only the reusing action information, the agents do not find any actions that can be replaced. When there are no suboptimal actions found, the agents cannot take advantage of the Propose-Evaluate Protocol. So, one possible future direction is to research more non-domain-specific information that agents can exploit to identify more suboptimal actions.

Our experiments show that the last few agents to propose often do not yield any increase in the global utility. The plan might have already been improved to the point that making further improvements is very difficult. To reduce communication overhead, we can develop a mechanism to evaluate the cost and benefit of proposing further changes to the agents' plans. When it is estimated that an agent's local changes are unlikely to improve the global resource allocation, we would like to skip past it to let the next agent propose.

Moreover, the Propose-Evaluate Protocol so far allows agents to make only local changes. Future work will investigate the possibility of letting agents suggest remote

changes in the plans of other agents as well. In general, an agent, in a loosely-coupled multiagent environment, cannot expect other agents to be benign. Before running the Convergence Protocol, other agents might coincidentally select actions that are optimal for themselves but unfortunately impose more resource demands than the agent could possibly handle. Without a sufficiently detailed model of the agent, they do not know what in their plans might cause difficulty to that agent. So, they cannot include these changes in their proposals even if they are willing to.

To illustrate how an unschedulable agent might request other agents to change their plans to reduce its own resource consumption, we use a slightly modified version of Figure 7-1. Instead of having a cost of 0.2, the action QUICK-TEST now has a cost of 0.5. So, after agent 1 improves its local plan as described in Section 7.1, the utilization instead becomes $0.4 + 0.3 + 0.5 = 1.2 > 1$. Figure 8-1 shows the modified state diagram.

In this example, agent 1 is still unschedulable after replacing SHUT-DOWN with the lower cost action SCAN4. This is due to the fact that agent 2 has selected to do a quick test. If agent 2 had selected to do a full test, agent 1 does not need to perform another quick test. Agent 1 may identify this cause for over-utilization and try to persuade agent 2 to change to do a full test. Assuming that agent 2 has the capacity to do a full test, agent 1 can remove QUICK-TEST from its schedule. The new plan consists of SCAN1 (0.4) and SCAN4 (0.3). The utilization is now $0.4 + 0.3 = 0.7 < 1$. Agent 1 becomes schedulable. This is the optimal combination of agent plans for this example.

As this example demonstrates, exploiting the new information available after a context change usually involves modifying both the local plan (e.g., replacing action SHUT-DOWN with action SCAN4) and remote plans (e.g., requesting another to change

ttac A2:QUICK-TEST to *ttac* A2:FULL-TEST). We would therefore like to extend the Propose-Evaluate Protocol, which is now a distributed search algorithm, to a negotiation algorithm, so that an agent can negotiate remote changes with other agents.

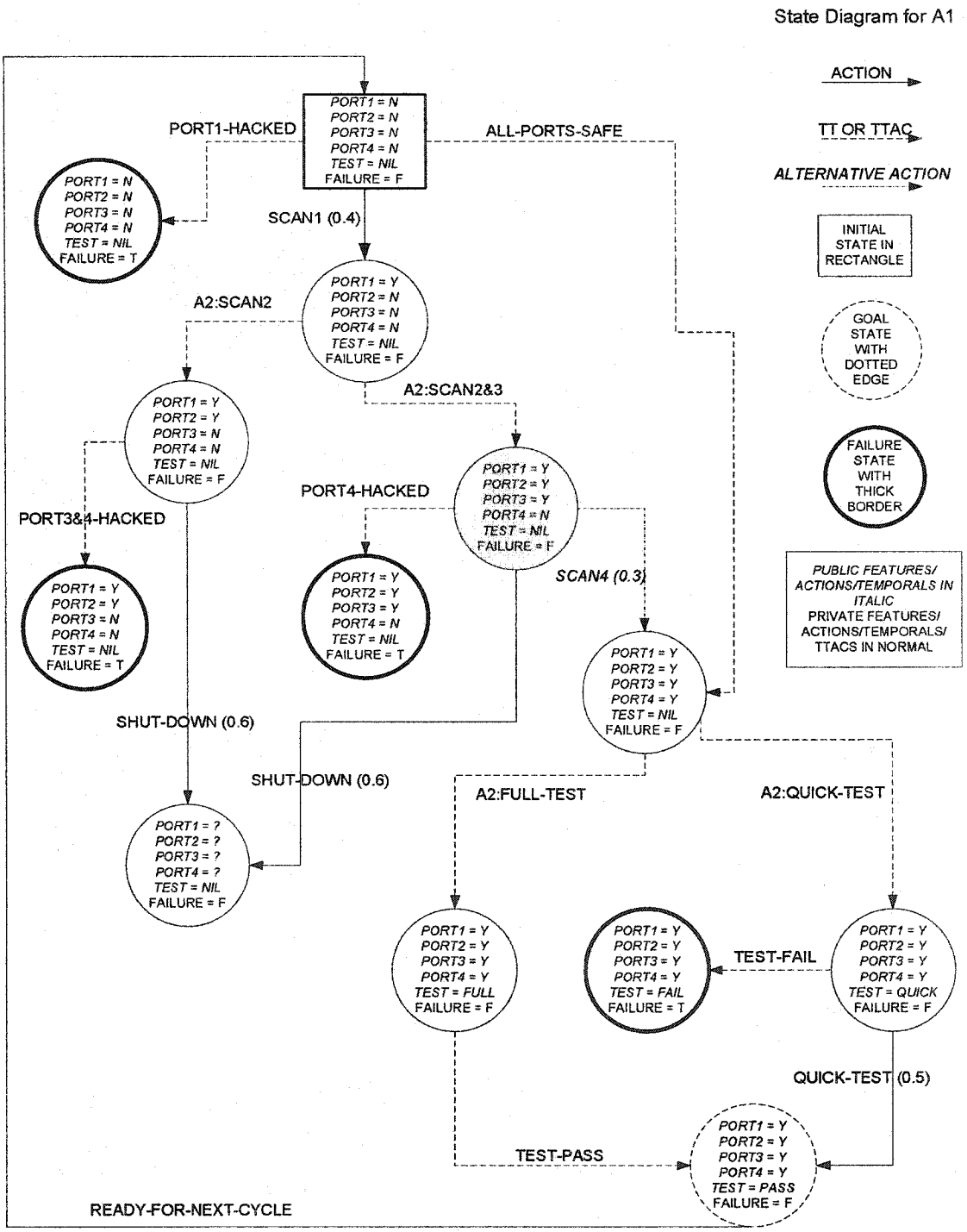


Figure 8-1: Multiagent Negotiation

APPENDICES

APPENDIX A

Proof of Convergence of the Logarithm Heuristic

In this appendix, we prove that the transition probabilities computed using the logarithm heuristic converges to the true values as the discretization gets finer. Before we proceed to prove this claim, we need to establish a few preliminaries. Recall that in eq. 4-3, for a transition, the function $\overline{F}(t) = (1 - F(t)) = P(T > t)$, where $F(t) = P(T \leq t)$ is the cumulative probability function. $\overline{F}(t)$ is called a survival function [57], and is equal to the probability that the transition fires *after* time t. It is a monotonic non-increasing function. The probability rate function, or hazard function, in a continuous domain is defined in eq. A-1. The continuous probability rate is the probability of the transition firing *per time unit*. It describes the instantaneous firing *rate* of the transition at time t if it has not fired before t.[43]

$$r(t) = \frac{f(t)}{1 - F(t)} = \frac{f(t)}{\overline{F}(t)} \qquad \text{eq. A-1}$$

We define the continuous probability rate in the time interval $[a,b)$ as in eq. A-2. ξ exists by the mean value theorem.

$$r_{a,b} = \frac{1}{(b-a)} \frac{\int_a^b f(t)dt}{1 - F(a)} = \frac{f(\xi)}{\overline{F}(a)}, \xi \in [a,b) \qquad \text{eq. A-2}$$

The discrete probability rate is the probability of the transition firing *per time interval*. The discrete probability rate $\tilde{r}_{a,b}$ in the time interval $[a,b)$ is given by eq. A-3.

[43] Note it is not the instantaneous probability because the instantaneous probability of firing at any time point is 0.

These values are the inputs in a CIRCA knowledge base, by which a user specifies the (discrete) probability rate functions.

$$\tilde{r}_{a,b} = \frac{\int_a^b f(t)dt}{1 - F(a)} \qquad \textbf{eq. A-3}$$

The relation between the continuous probability rate $r_{a,b}$ and the discrete probability rate $\tilde{r}_{a,b}$ in a time interval is shown in eq. A-4.

$$r_{a,b} = \frac{\tilde{r}_{a,b}}{(b-a)} \qquad \textbf{eq. A-4}$$

As the length of the interval goes to 0, i.e., $b \rightarrow a$, $\tilde{r}_{a,b}$ goes to 0 because $\xi \rightarrow a$. On the other hand, $r_{a,b}$ goes to $r(a)$, i.e.,

$$r_{a,a} = \frac{f(a)}{\overline{F}(a)} = r(a) \qquad \textbf{eq. A-5}$$

Since $F(t)$ is a monotonic non-decreasing function, $r_{a,b}$ in general does not equal the integral of $r(t)$ from a to b. $r_{a,b}$ is bounded below and above.

$$
\begin{aligned}
r_{a,b} &= \frac{1}{(b-a)} \frac{\int_a^b f(t)dt}{1-F(a)} = \frac{1}{(b-a)} \frac{\int_a^b \frac{f(t)}{1-F(t)}[1-F(t)]dt}{1-F(a)} \\
&= \frac{1}{(b-a)} \frac{\int_a^b \frac{f(t)}{\overline{F}(t)} \overline{F}(t)dt}{\overline{F}(a)} \\
&\geq \frac{\overline{F}(b)}{(b-a)} \frac{\int_a^b \frac{f(t)}{\overline{F}(t)} dt}{\overline{F}(a)} \\
&= \frac{\overline{F}(b)}{(b-a)\overline{F}(a)} \int_a^b r(t)dt \\
&= \frac{\overline{F}(b)}{\overline{F}(a)} r(\xi'), \xi' \in [a,b)
\end{aligned}
$$

eq. A-6

$$
\begin{aligned}
r_{a,b} &= \frac{1}{(b-a)} \frac{\int_a^b f(t)dt}{1-F(a)} \\
&\leq \frac{1}{(b-a)} \int_a^b \frac{f(t)}{1-F(t)} dt = \frac{1}{(b-a)} \int_a^b r(t)dt \\
&= r(\xi), \xi \in [a,b)
\end{aligned}
$$

eq. A-7

ξ and ξ' exist by the mean value theorem. Combining eq. A-6 and eq. A-7, we have:

$$
\frac{\overline{F}(b)}{\overline{F}(a)} r(\xi') \leq r_{a,b} \leq r(\xi), \xi', \xi \in [a,b)
$$

eq. A-8

Given the probability rate $\tilde{r}(h)=\tilde{r}_{t_h,t_{h+1}}$ in $[t_h,t_{h+1})$, we can construct the cumulative probability function.

$$F(0)=0$$
$$F(t)=1-\prod_{h}^{t/l}(1-\tilde{r}(h)) \qquad \textbf{eq. A-9}$$

$t>0$ is the beginning of a time interval. Similarly, we have:

$$\overline{F}(0)=1$$
$$\overline{F}(t)=\prod_{h}^{t/l}(1-\tilde{r}(h)) \qquad \textbf{eq. A-10}$$

Now, we are ready to prove that transition probabilities computed using the logarithm heuristic in Section 4.4 agree with eq. 4-3 in the limit. Essentially, the discrete probability rate function $\tilde{r}_{a,b}$ approximates the continuous rate function $r(t), t\in[a,b)$ by piecewise constant rates $r_{a,b}=\frac{\tilde{r}_{a,b}}{(b-a)}$. The strategy is to show that when the continuous probability rate functions are piecewise constant, the transition probabilities can be accurately computed by the heuristic.

Moreover, the values computed assuming piecewise constant functions are bounded below and above by the lower and upper sums of the integral in eq. 4-3. As the upper and lower sums converge when the length of the time intervals goes to 0, the heuristic value converges to the integral value.

We could rewrite the transition probability for tt_i, $p(tt_i)$, in eq. 4-3 using probability rate function instead of probability density function.

$$
\begin{aligned}
& p(tt_i) \\
&= \int_0^\infty \overline{F}_1(t)...\overline{F}_{i-1}(t)\overline{F}_{i+1}(t)...\overline{F}_n(t) f_i(t) dt \\
&= \int_0^\infty \overline{F}_1(t)...\overline{F}_{i-1}(t)\overline{F}_{i+1}(t)...\overline{F}_n(t)\overline{F}_i(t) r_i(t) dt \\
&= \int_0^\infty \left(\prod_{j=1}^{n} \overline{F}_j(t) \right) r_i(t) dt
\end{aligned}
$$

eq. A-11

Now, let's divide the time line into a lot of tiny intervals of length *dt*. The integrand is simply the probability of tt_i firing in a small time interval $[t, t+dt)$ given that no transitions (including tt_i) have fired before *t*. It equals the dependent probability rate of tt_i in $[t, t+dt)$.

At the beginning of the interval at time *t*, the first term in the integrand is:

$$
\begin{aligned}
& \prod_{j=1}^{n} \overline{F}_j(t) = \prod_{j=1}^{n} \left(\prod_{h=1}^{t/dt} (1 - \tilde{r}_j(h)) \right) \\
&= \prod_{h=1}^{t/dt} \left(\prod_{j=1}^{n} (1 - \tilde{r}_j(h)) \right) \\
&= \prod_{h=1}^{t/dt} (P_{NONE}(h)) \\
&= \overline{F}(t)
\end{aligned}
$$

eq. A-12

$\overline{F}(t)$ is the probability that no transition has fired before t. $\tilde{r}_j(h)$ is the "discrete" rate in the *h*-th time interval $[t_h, t_h + dt)$, $t_1 = 0$, as in eq. A-3. It is the input in a CIRCA KB. The first step in eq. A-12 is by eq. A-10, adding in the indices *j* indicating the transitions. The third step is by eq. 4-8, stripping off the second parameter for state

because it is irrelevant here.[44] The last step is by eq. 4-10, again stripping off the second parameter for state.

We denote the maximal dependent probability rate of tt_i firing in h-th time interval $[t,t+dt)$ by $M_i(h)$.[45] As the dependent probability rate of tt_i firing in $[t,t+dt)$ increases with its independent probability rate, $M_i(h)$ is the dependent probability rate of tt_i when:[46]

$$\begin{aligned} \tilde{r}_i(h) &= \operatorname*{lub}_{x\in[t,t+dt)} r_i(x) \\ r_{j\neq i}(t) &= \operatorname*{glb}_{x\in[t,t+dt)} r_j(x) \end{aligned} \qquad \textbf{eq. A-13}$$

Similarly, we define $m_i(h)$, the minimal dependent probability rate of tt_i in $[t,t+dt)$. The integrand in $[t,t+dt)$ is bounded below and above by:

$$m_i\overline{F}(t) \le \left(\prod_{j=1}^{n}\overline{F}_j(t)\right) r_i(t)dt \le M_i\overline{F}(t) \qquad \textbf{eq. A-14}$$

Consequently, the integral in eq. A-11 is bounded below and above by the lower and upper sums in eq. A-15. h is the index for the intervals.

$$\sum_{h=1}^{\infty} m(h)\overline{F}(h) \le \int_0^{\infty}\left(\prod_{j=1}^{n}\overline{F}_j(t)\right) r_i(t)dt \le \sum_{h=1}^{\infty} M(h)\overline{F}(h) \qquad \textbf{eq. A-15}$$

---

[44] Some notations in this section stipulate the second parameter because the state is implicit, hence irrelevant.

[45] We simplify $[t_h,t_h+dt)$ to $[t,t+dt)$.

[46] lub = least upper bound; glb = greatest lower bound

Denote the "true" dependent probability rate of tt_i in $[t, t+dt)$ by $r_i'(h)$. Then, M and m both converge to $r_i'(h)$, as dt goes to 0. That is,

$$\sum m(h)\overline{F}(h) = \sum r_i'(h)\overline{F}(h) = \sum M(h)\overline{F}(h) \qquad \textbf{eq. A-16}$$

So, the transition probability of tt_i, as given by eq. 4-3 and eq. A-11, is:

$$p(tt_i) = \int_0^{\infty} \left(\prod_{j=1}^{n} \overline{F}_j(t) \right) r_i(t)dt = \sum r_i'(h)\overline{F}(h) \qquad \textbf{eq. A-17}$$

Notice that eq. A-17 is the same as eq. 4-11, if and only if $r_i'(h) = \tilde{r}_i(h,s)$. In other words, to prove the logarithm heuristic satisfies the convergence requirement, we only need to show that as dt goes to 0, $\tilde{r}_i(h,s)$ computed by eq. 4-7 converges to $r_i'(h)$. Eq. A-18 reproduces eq. 4-7. Notice that $\tilde{r}_i(h,s)$ is a product of two terms.

$$\tilde{r}_i(h,s) = \left[\frac{\ln(1-\tilde{r}_i(h))}{\sum \ln(1-\tilde{r}_j(h))} \right] (1 - P_{NONE}(h)) \qquad \textbf{eq. A-18}$$

We can therefore complete the proof by proving eq. A-19 and that eq. A-19 takes the equal sign when $dt \to 0$.

$$\frac{r_i'(h)}{(1-P_{NONE}(h))} \approx \frac{\ln(1-\tilde{r}_i(h))}{\sum \ln(1-\tilde{r}_j(h))} \qquad \textbf{eq. A-19}$$

$\frac{r_i'(h)}{(1-P_{NONE}(h))}$ is simply the probability that tt_i fires during the h-th interval given that some transition will fire in the interval. It is:

$$\frac{r_i'(h)}{(1-P_{NONE}(h))} = \frac{\int_t^{t+dt}\left(\prod_{j=1}^{n}\overline{F}_j(x)\right)r_i(x)dx}{\prod_{j=1}^{n}\int_t^{t+dt}f_j(x)dx}$$

$$= \frac{\int_t^{t+dt}\left(\prod_{j=1}^{n}\frac{f_j(x)}{r_j(x)}\right)r_i(x)dx}{\prod_{j=1}^{n}\int_t^{t+dt}f_j(x)dx} \qquad \textbf{eq. A-20}$$

The last term is a complicated function of all $r_j(x)$, $x \in [t, t+dt)$ for all j. In general, the bigger $r_i(x)$ is relative to the other $r_j(x)$, the bigger $\frac{r_i'(h)}{(1-P_{NONE}(h))}$ is. Within a very tiny interval of length dt, we could approximate eq. A-20 by taking the $r_j(x)$'s as constants. Basically, we are approximating the integral in eq. A-20, hence eq. A-11 and eq. 4-3, by piecewise constant probability rate functions. Specifically, we can take $r_j(x) = r_{j,t,t+dt} = \frac{\tilde{r}_j(h)}{dt}$. According to eq. A-8, $r_{j,t,t+dt} = \frac{\tilde{r}_j(h)}{dt}$ is bounded below and above by $\frac{\overline{F}(t+dt)}{\overline{F}(t)} \operatorname{glb}_{x\in[t,t+dt)} r_j(x)$ and $\operatorname{lub}_{x\in[t,t+dt)} r_j(x)$. As $dt \to 0$, $r_{j,t,t+dt} \to r_j(t)$.

When the rates $r_j(x)$ are constant, they are stochastic processes that have exponential probability functions. We are now going to discuss some important properties of exponential probability functions before computing eq. A-20. We summarize various descriptions of an exponential distribution below:

- The probability density function is $f = \lambda e^{-\lambda t}$.
- The cumulative probability function is $F = 1 - e^{-\lambda t}$.
- The survival function is $\overline{F} = e^{-\lambda t}$.
- The probability rate function is $r = \lambda$!

λ is called the rate of the exponential distribution. In fact, exponential distributions are the only distributions that have constant rates. Moreover, an exponential probability function is said to be memory-less because the probability of a transition firing in any time interval $[t, t+dt)$, if it has not fired before t, is independent of t.

We now resume computing eq. A-20. We first compute the numerator.

$$\begin{aligned}
&\int_t^{t+dt} \left(\prod_{j=1}^{n} \overline{F}_j(x) \right) r_i(x) dx \\
&= \int_t^{t+dt} e^{-\lambda_1 x} \ldots e^{-\lambda_i x} \ldots e^{-\lambda_n x} \lambda_i dx \\
&= \int_t^{t+dt} e^{-(\lambda_1 + \ldots + \lambda_i + \ldots \lambda_n)x} \lambda_i dx \qquad \text{eq. A-21} \\
&= \int_t^{t+dt} e^{-\Lambda x} \lambda_i dx, \qquad \Lambda = \lambda_1 + \ldots + \lambda_n \\
&= \frac{\lambda_i}{\Lambda} \left[\frac{1}{e^{\Lambda t}} \left(1 - \frac{1}{e^{\Lambda dt}} \right) \right]
\end{aligned}$$

The second term $\left[\frac{1}{e^{\Lambda t}} \left(1 - \frac{1}{e^{\Lambda dt}} \right) \right]$ is simply the probability that some transition fires in the interval $[t, t+dt)$, and thus equals the denominator in eq. A-20.

$$\begin{aligned}
&\prod_{j=1}^{n} \int_t^{t+dt} f_j(x) dx = (1 - P_{NONE}(h)) \qquad \text{eq. A-22} \\
&= \left[\frac{1}{e^{\Lambda t}} \left(1 - \frac{1}{e^{\Lambda dt}} \right) \right]
\end{aligned}$$

Consequently, eq. A-20, when approximated by piecewise constant probability rates, simply becomes:

$$\frac{r_i'(h)}{(1-P_{NONE}(h))} \approx \frac{\lambda_i}{\Lambda} \qquad \textbf{eq. A-23}$$

In other words, the probability of tt_i firing in a time interval given that no transitions have fired before the beginning of the interval and that some transition will fire in the interval is proportional to its rate λ_i. Rearranging the terms in eq. A-23, we have eq. A-24. The term on the right hand side in eq. A-24 is bounded above and below by the M and m in eq. A-14 because of the way M and m are constructed. As M, the maximal, converges to m, the minimal, this term converges to $r_i'(h)$.

$$r_i'(h) \approx \frac{\lambda_i}{\Lambda}(1-P_{NONE}(h)) \qquad \textbf{eq. A-24}$$

$\frac{\lambda_i}{\Lambda}(1-P_{NONE}(h))$ looks very similar to eq. A-18. In fact, it is eq. A-18 with different symbols. We can write the exponential rate (λ) in the continuous domain in terms of the probability rate ($\tilde{r}$) in the discrete domain.

Recall that a rate $\tilde{r}$ in the discrete domain is the probability that a transition fires in the interval given that no transition has fired before the beginning of the interval. By eq. A-3, we have:

$$\tilde{r} = \frac{\int_t^{t+dt} \lambda e^{-\lambda x} dx}{1-F(t)} \qquad \textbf{eq. A-25}$$
$$= 1 - e^{-\lambda dt}$$

Therefore, it is possible to convert from the discrete rate, $\tilde{r}$, to the continuous rate, λ, and vice versa. Their relationships are:

$$\tilde{r} = 1 - e^{-\lambda dt}$$
$$\lambda = -\ln(1-\tilde{r})/dt \qquad \textbf{eq. A-26}$$

Substituting eq. A-26 into eq. A-24, we get eq. A-18.

$$\begin{aligned} & r_i'(h) \\ & \approx \frac{\lambda_i}{\Lambda}(1 - P_{NONE}(h)) \\ & = \frac{\ln(1-\tilde{r}_i)}{\sum_{j=1}^{n}\ln(1-\tilde{r}_j)}(1 - P_{NONE}(h)) \\ & = \tilde{r}_i(h,s) \end{aligned} \qquad \textbf{eq. A-27}$$

When $dt \to 0$, $r_i'(h) = \tilde{r}_i(h,s)$. In other words, the logarithm heuristic in eq. 4-7 to compute the dependent probability rate of a transition firing in a time interval agrees with eq. 4-7 in the limit, as we have claimed.

In fact, as shown by the mathematics in this section, if the continuous probability rate functions for the underlying processes are indeed piecewise constant, then our heuristic always computes transition probabilities accurately. Otherwise, its precision increases as we get finer discretization.

APPENDIX B

Notations

$P(E)$	the probability of an event E
$\tilde{r}(h)$	the discrete probability rate of a transition in the h-th time interval $[t_h, t_{h+1})$
$\tilde{r}_i(h,s)$	the discrete dependent probability rate function of transition i in state s
$P_{NONE}(h,s)$	the probability that no transitions fire in the h-th time interval in state s
$F_i(h,s)$	the cumulative probability function for a state-specific transition $trans_i$ in state s
$\overline{F}(h,s)$	the probability that the stochastic process is still in state s after t_h, or the h-1 time intervals
$p(trans_i, s)$	the transition probability of a state-specific transition i in state s
$f(t)$	the probability density function
$F(t)$	the cumulative probability function
$\overline{F}(t)$	the survival function
$r(t)$	the probability rate function
$\tilde{r}_{a,b}$	the discrete probability rate in the time interval $[a,b)$
$r_{a,b}$	the continuous probability rate in the time interval $[a,b)$
$\overline{F}(t)$	the probability that no transitions in a state fire before t
$r_i'(h)$	the "true" dependent probability rate of tt_i in $[t_h, t_h + dt)$
P	the period of an action
$r_s(h)$	the probability rate function for a dependent temporal transition in state s

BIBLIOGRAPHY

BIBLIOGRAPHY

1. Altman, E. 1999. *Constrained Markov Decision Processes.* Chapman and HALL/CRC.
2. Atkins, E. M. 1999. Plans Generation and Hard Real-time Execution with Application to Safe, Autonomous Flight. The Dept., of Electrical Engineering and Computer Science, the University of Michigan, Ann Arbor.
3. Atkins, E. M., Abdelzaher, T. F., Shin, K. G., and Durfee, E. H. 2001. Planning and Resource Allocation for Hard Real-time, Fault-Tolerant Plan Execution. *Journal of Autonomous Agents and Multi-Agent Systems* (Best of Agents '99 Special Issue).
4. Atkins, E. M., Durfee, E. H., and Shin, K. G. 1996. Plan Development Using Local Probabilistic Models. In *the Twelfth Conference on Uncertainty in Artificial Intelligence.*
5. Atkins, E. M., Durfee, E. H., and Shin, K. G. 1997. Detecting and Reacting to Unplanned-for World States. In *AAAI-97.*
6. Barlow, R. E., Marshall, A. W. and Proschan, F. 1963. Properties of probability distributions with monotone hazard rate. *Ann. Math. Stats.* 34: p. 375-389.
7. Besanko, D., Braeutigam, Ronald R. 2001. *Microeconomics: An Integrated Approach.* Wiley Text Books.
8. Blum, A., and Furst, M. 1997. Fast Planning Through Planning Graph Analysis. *Artificial Intelligence* (90): p. 281-300.
9. Blum, A., and Langford, J. 1999. Probabilistic Planning in the Graphplan Framework. In *the Proceedings of ECP'99.* Springer-Verlag.
10. Bockmayr, A., and Dimopoulos, Y. 1998. Mixed Integer Programming Models for Planning Problems. In *the Working Notes of the CP98 Workshop on Constraint Problem Reformulation.*
11. Boutilier, C. 1999. Sequential Optimality and Coordination in Multiagent Systems. In *IJCAI-99.*
12. Boutilier, C., and Goldszmidt, M. 1996. The frame problem and Bayesian network action representations. In *CSCSI-96.*
13. Boutilier, C., Dean, T. and Hanks, S. 1995. Planning under uncertainty: Structural assumptions and computational leverage. In *the 3rd European Workshop on Planning (EWSP'95).*
14. Boutilier, C., Dean, T. and Hanks, S. 1999. Decision Theoretic Planning: Structural Assumptions and Computational Leverage. *Journal of Artificial Intelligence Research* 11: p. 1-94.
15. Chen, Y., Finin, T., Labrou, Y., and Cost, S. 1999. Negotiating agents for supply chain management. In *the AAAI Workshop on Artificial Intelligence for Electronic Commerce.* Orlando, Florida.
16. Cheng, J., and Wellman, M. 1998. The WALRAS algorithm: A convergent distributed implementation of general equilibrium outcomes. *Computational Economics* 12: p. 1-24.
17. Chia, M., Neiman, D., and Lesser, V. 1998. Poaching and distraction in asynchronous agent activities. In *the International Conference on Multiagent Systems.*

18. Chien, S., Rabideau, G., Knight, R., Sherwood, R., Engelhardt, B., Mutz, D., Estlin, T., Smith, B., Fisher, F., Barrett, T., Stebbins, G. and Tran, D. 2000. ASPEN - Automating Space Mission Operations using Automated Planning and Scheduling. In *the Sixth International Symposium on Technical Interchange for Space Mission Operations and Ground Data Systems (SpaceOps 2000).*
19. Conry, S. E., Kuwabara, K., Lesser, V. R. and Meyer, R. A. 1992. Multistage Negotiation in Distributed Constraint Satisfaction. *IEEE Transactions on Systems, Man and Cybernetics* 21 (6): p. 1462-1477.
20. Conry, S. E., MacIntosh, D. J., and Meyer, R.A. 1990. DARES: A Distributed Automated Reasoning System. In *AAAI-90.*
21. Cooper, G. F. 1990. The computational complexity of probabilistic inference using bayesian belief networks. *Artificial Intelligence* 42 (2-3): p. 393-405.
22. Danzig, G. B. 1963. *Linear Programming and Extensions.* Reading, Princeton University Press.
23. Dean, T., and Kanazawa, T. 1988. Probabilistic temporal reasoning. In *the Seventh National Conference on Artificial Intelligence.*
24. Dean, T., and Kanazawa, T. 1989. A model for reasoning about persistence and causation. *Computational Intelligence* 5: p. 142-150.
25. Dean, T., and McDermott, D. V. 1987. Temporal data base management. *Artificial Intelligence* 32: p. 1-55.
26. Dean, T., Kaelbling, L. P., Dirman, J. and Nicholson, A. 1993. Planning With Deadlines in Stochastic Domains. In *AAAI.*
27. Dechter, R., Meiri, I., and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49 (61-95).
28. Decker, K., and Lesser, V. 1993. Quantitative modeling of complex environments. *International Journal on Intelligence Systems in Accounting, Finance and Management* 2(4).
29. Decker, K., and Lesser, V. 1995. Designing a Family of Coordination Algorithms. In *ICMAS-95.*
30. Decker, K., and Li, J. 1998. Coordinated hospital patient scheduling. In *the International Conference on Multi-Agent Systems.*
31. Dodson, B. 1994. *Weibull Analysis.* ASQC Quality Press.
32. Dolgov, D., and Durfee, E. H. 2002. Satisficing Strategies for Resource-Limited Policy Search in Dynamic Environments. In *the First International Joint Conference on Autonomous Agents and Multiagent Systems.*
33. Drabble, B., and Tate, A. 1994. The Use of Optimistic and Pessimistic Resource Profiles to Inform Search in an Activity Based Planner. In *AIPS-94.*
34. Durfee, E. H., and Lesser, V. R. 1991. Partial Global Planning: A Coordination Framework for Distributed Hypothesis Formation. *IEEE Transactions on Systems, Man, and Cybernetics, Special Issue on Distributed Sensor Networks, SMC* 21 (5): p. 1167-1183.
35. Eisenstat, S. C., Gursky, M. C., Schultz, M. H. and Sherman, A. H. 1977. Yale Sparse Matrix Package. Department of Computer Science, Yale University.
36. El-Kholy, A., and Richards, B. 1996. Temporal and Resource Reasoning in Planning: the parcPLAN approach. In *the Twelfth European Conference on Artificial Intelligence (ECAI-96).*

37. Ephrati, E., Pollack, M. E., and Rosenchein, J. S. 1995. A Tractable Heuristic that Maximizes Global Utility through Local Plan Combination. In *the First International Conference on Multi-Agent Systems*.
38. Ephrati, E., Pollack, M., and Milshtein, M. 1996. A cost-directed planner: Preliminary report. In *the Thirteenth National Conference on Artificial Intelligence*. Cambridge, MA. AAAI Press/MIT Press.
39. Fikes, R., Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* (2): p. 189-203.
40. Fishman, G. S. 2001. *Discrete-event simulation: modeling, programming, and analysis. Springer series in operations research.* New York, NY. Springer-Verlag, New York, Inc.
41. Flannery, B. P., Teukolsky, S. A., Vetterling, W. T. 1993. *Numerical Recipes in C: The Art of Scientific Computing*. 2nd ed. William H. Press. Cambridge University Press.
42. Frei, C., and Faltings, B. 1999. Resource allocation in networks using abstraction and constraint satisfaction techniques. In *Constraint Programming 1999*.
43. Freuder, E. C. 1978. Synthesizing constraint expressions. *Communications of ACM* 21(11): p. 958-966.
44. Freuder, E. C., and Hubbe, P. D. 1993. Using inferred disjunctive constraints to decompose constraint satisfaction problems. In *IJCAI-93*.
45. Georgeff, M. 1983. Communication and Interaction in multi-agent planning. In *The Third National Conference on Artificial Intelligence (AAAI-83)*.
46. Ghallab, M., and Laruelle, H. 1994. Representation and control in IxTeT, a temporal planner. In *the Second International Conference on AI Planning Systems.* Morgan Kaufmann.
47. Goldberger, A. 1991. *A course in econometrics.* Harvard University Press.
48. Hanks, S. 1988. Representing and computing temporally scoped belief. In *the Seventh National Conference on Artificial Intelligence.*
49. Hanks, S. 1990. Practical temporal projection. In *the Eighth National Conference on Artificial Intelligence.*
50. Hanks, S., and McDermott, D. V. 1994. Modeling a Dynamic and Uncertain World I: Symbolic and Probabilistic Reasoning about Change. *Artificial Intelligence* 66 (1): p. 1-55.
51. Haslum, P., and Geffner, H. 2001. Heuristic Planning with Time and Resources. In *European Conference on Planning.*
52. Heyman, D. P. 1982. *Stochastic Models in Operations Research.* Vol. I. McGraw-Hill, Inc.
53. Hirayama, K., and Yokoo, M. 1997. Distributed partial constraint satisfaction problem, in *Principles and Practice of Constraint Programming - CP97.* Smolka, G., Editor. Springer-Verlag. p. 222-236.
54. Hirayama, K., and Yokoo, M. 2000. An Approach to Over-constrained Distributed Constraint Satisfaction Problems: Distributed Hierarchical Constraint Satisfaction. In *International Conference on Multiagent Systems.*
55. Horling, B., Vincent, R., Mailler, R., Shen, J., Becker, R., Rawlins, K., and Lesser, V. 2001. Distributed sensor network for real time tracking. In *the fifth International Conference on Autonomous Agents, 2001.*

56. Huberman, B., and Clearwater, Scott H. 1995. A multi-agent system for controlling building environments. In *the First International Conference on Multi-Agent Systems (ICMAS)*. San Francisco, CA.
57. Johnson, K., and Balakrishnan. 1994. *Continuous Univariate Distributions*. 2nd ed. Vol. I and II. John Wiley and Sons.
58. Jonsson, A. K., Morris, P. H., Muscettola, N., Rajan, K. and Smith, B. 2000. Planning in Interplanetary Space: Theory and Practice. In *the Fifth International Conference on Artificial Intelligence Planning and Scheduling (AIPS-2000)*.
59. Kanazawa, K. 1992. Reasoning about Time and Probability. The Dept., of Computer Science, Brown University.
60. Karmarkar, N. 1984. A New Polynomial-Time Algorithm for Linear Programming. *Combinatorica* (4): p. 373-395.
61. Kautz, H., and Selman, B. 1992. Planning as satisfiability. In *ECAI-92*.
62. Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. In *AAAI-96*.
63. Kautz, H., and Walser, J. P. 1999. State-space Planning by Integer Optimization. In *the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*.
64. Kemeny, J. G., and Snell, J. L. 1960. *Finite Markov Chains*. Princeton, N.J. Van Nostrand.
65. Kitano, H., Todokoro, S., Noda, I., Matsubara, H., and Takahashi, T. 1999. Robocup rescue: Search and rescue in large-scale disaster as a domain for autonomous agents research. In *the IEEE International Conference on System, Man, and Cybernetics*.
66. Krishna, C. M., and Shin, K. G. 1997. *Real-time Systems*. McGraw-Hill.
67. Kulkarni, V. G. 1999. *Modeling, Analysis, Design, and Control of Stochastic Systems*. New York. Springer-Verlag New York, Inc.
68. Kushmerick, N., Hanks, S. and Daniel, W. 1994. An Algorithm for Probabilistic Planning. *Artificial Intelligence*.
69. Laborie, P., and Ghallab, M. 1995. Planning with sharable resource constraints. In *IJCAI-95*. Montreal.
70. Lawless, J. F. 1982. *Statistical Models & Methods for Lifetime Data*. John Wiley & Sons.
71. Lee, E. 1992. *Statistical Methods for Survival Data Analysis*. Second ed. Wiley-Interscience.
72. Lemaitre, M., and Verfaillie, G. 1997. An incomplete method for solving distributed valued constraint satisfaction problems. In *AAAI Workshop on Constraints and Agents*.
73. Littman, M. 1997. Probabilistic propositional planning: representations and complexity. In *AAAI-97*.
74. Liu, C., and Layland, J. W. 1973. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of ACM* 20 (1): p. 44-61.
75. Liu, J., and Sycara, K. 1996. Multiagent coordination in tightly coupled task scheduling. In *International Conference on Multi-Agent Systems*.
76. Liu, J. S., and Sycara, K. P. 1994. Distributed Meeting Scheduling. In *the Sixteenth Annual Conference of the Cognitive Science Society*. Atlanta, Georgia, U.S.A.

77. Malone, T. W., Fikes, R. E., Grant, K. R. and Howard, M. T. 1988. *Enterprise: A market-like task scheduler for distributed computing environments. The Ecology of Computer*. ed. Huberman, B. A. North-Holland.
78. Martin, N. G. 1993. Using Statistical Inference to Plan Under Uncertainty. The Dept., of Computer Science, University of Rochester, Rochester, NY.
79. Martin, N. G., and Allen, J. F. 1991. A language for planning with statistics. In *the Seventh Conference on Uncertainty in Artificial Intelligence*.
80. Martin, N. G., and Allen, J. F. 1993. Statistical Probabilities for Planning. The Department of Computer Science, University of Rochester. Rochester, NY.
81. McAfee, R., and McMillan, J. 1987. Auctions and bidding. *Journal of Economic Literature* (25): p. 699-738.
82. McVey, C., Durfee, Edmund H., Atkins, Ella M., and Shin, Kang G. 1997. Development of Iterative Real-time Scheduler to Planner Feedback. In *the Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI97)*.
83. Minton, S., Johnston, M., Philips, A., and Laird, P. 1992. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence* 58: p. 161-205.
84. Modi, P. J., Shen, W. M., Tambe, M., and Yokoo, M. 2003. An Asynchronous Complete Method for Distributed Constraint Optimization. In *Autonomous Agents and Multi-Agent Systems*.
85. Morris, P. 1993. The Breakout Method for Escaping From Local Minima. In *the Eleventh National Conference on Artificial Intelligence*.
86. Murphy, A. 1996. A Piecewise-Constant Hazard Rate Model for the Duration of Unemployment in Single Interview Samples of the Stock of Unemployed. *Economic Letters* 51: p. 177-183.
87. Musliner, D. J. 1993. CIRCA: The Cooperative Intelligent Real-Time Control Architecture. The Dept., of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor.
88. Musliner, D. J., Durfee, E. H. and Shin, K. G. 1994. Predictive Sufficiency and the Use of Stored Internal State. In *AIAA/NASA Conf. on Intelligent Robots in Field, Factory, Service, and Space*.
89. Nareyek, A. 2001. Beyond the Plan-Length Criterion, in *Local Search for Planning and Scheduling*. Nareyek, A., Editor. p. 55-78.
90. Nareyek, A. 2001. Local-Search Heuristics for Generative Planning. In *the Fifteenth Workshop on AI in Planning, Scheduling, Configuration and Design (PuK 2001)*.
91. Nareyek, A. 2004. EXCALIBUR online documentation. http://www.ai-center.com/projects/excalibur/documentation/application/orc/evaluation.html.
92. Nelson, W. 1990. *Accelerated Testing: Statistical Models, Test Plans and Data Analysis*. John Wiley & Sons.
93. Nemhauser, G. L., and Wolsey, L. A. 1988. *Integer and Combinatorial Optimization*. Reading, John Wiley & Sons, Inc.
94. Nishibe, Y., Kuwabara, K., Ishida, T., and Yokoo, M. 1994. Speed-up of distributed constraint satisfaction and its application to communication network path assignments. *Systems and Computers in Japan* 25 (12): p. 54-67.

95. Penberthy, J., and Weld, D. 1992. UCPOP: A sound, complete, partial order planner for ADL. In *the Third International Conference on Knowledge Representation and Reasoning, (KR'92)*. Boston, MA.
96. Powers, D. A., and Xie, Y. 2000. *Statistical Methods for Categorical Data Analysis*. Academic Press.
97. Reed, W. J. 1997. Estimating historical forest-fire frequencies for time-since-last-fire-sample data. *IMA Journal of Mathematics Applied in Medicine and Biology* 14 (1).
98. Rintanen, J., and Jungholt, H. 1999. Numeric State Variables in Constraint-based Planning. In *ECP-99*.
99. Ross, M. S. 1990. *A course in simulation*. Macmillan Publishing Company.
100. Sandhom, T. W. 1996. Limitations of the Vickrey auction in computational multiagent systems. In *the Second International Conference on Multi-Agent Systems (ICMAS)*. Keihanna Plaza, Kyoto, Japan.
101. Sandhom, T. W., and Suri, Subhash. 2001. Side constraints and non-price attributes in markets. In *the International Joint Conference on Artificial Intelligence Workshop on Distributed Constraint Reasoning*.
102. Schiex, T., Fargier, H., Verfaillie, G. 1995. Valued constraint satisfaction problems: Hard and easy problems. In *the International Joint Conference on Artificial Intelligence*.
103. Shedler, G. S. 1993. *Regenerative Stochastic Simulation*. San Diego, CA. Academic Press, Inc.
104. Shintani, T., Ito, T., and Sycara, K. 2000. Multiple Negotiations among Agents for a Distributed Meeting Scheduler. In *The Fourth International Conference on Multi-Agent Systems (ICMAS'2000)*.
105. Shoham, Y., and Tennenholtz, M. 1995. On Social Laws for Artificial Agent Societies: Off-Line Design. *Artificial Intelligence* 73 (1-2): p. 231-252.
106. Smith, R. G. 1980. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers* C-29 (12): p. 1104-1113.
107. Soh, L. K., and Tsatsoulis, C. 2001. Reflective negotiating agents for real-time multisensor target tracking. In *the International Joint Conference on Artificial Intelligence*.
108. Sorensen, J. B. 1999. STPIECE: Stata module to estimate piecewise-constant hazard rate models. Boston College Department of Economics. Boston.
109. Strassen, V. 1969. Gaussian elimination is not optimal. *Numerische Mathematik* 14 (3): p. 354-356.
110. Tate, A. 1996. *O-Plan: a Knowledge-Based Planner and its Application to Logistics. Advanced Planning Technology*. ed. Tate, A. AAAI Press.
111. Tsang, E. 1993. *Foundations of Constraint Satisfaction*. London, UK. Academic Press.
112. Van Beek, P., and Chen, X. 1999. CPlan: A Constraint Programming Approach to Planning. In *AAAI-99*.
113. Van Dyke Parunak, H. 1987. Manufacturing experience with the contract net, in *Distributed Artificial intelligence, Research Note in Artificial Intelligence*. Huhns, M. N., Editor. p. 285-310.

114. Vickrey, W. 1961. Counterspeculation, Auctions and Competitive Sealed Tenders. *Journal of Finance* 16: p. 8-37.
115. Vossen, T., Ball, M., Lotem, A., and Nau, D. 1999. On the Use of Integer Programming Models in AI Planning. In *the Sixteenth International Joint Conference on Artificial Intelligence(IJCAI-99).*
116. Wagner, T., Garvey, A. and Lesser, V. 1998. Criteria-Directed Task Scheduling. *International Journal of Approximate Reasoning* 19 (1/2): p. 91-118.
117. Waldspurger, C. A., Hogg, Tad, Huberman, Bernardo, Kephart, Jeffrey O, and Stornetta, Scott W. 1992. Spawn: A distributed computational economy. *IEEE Transactions on Software Engineering* 18(2): p. 103-117.
118. Walsh, W. E., and Wellman, M. P. 1998. A market protocol for decentralized task allocation. In *the Third International Conference on Multi-Agent Systems.*
119. Waltz, D. 1975. Understanding line drawing of scenes with shadows, in *The Psychology of Computer Vision.* Winston, P., Editor. McGraw-Hill. p. 19-91.
120. Wellman, M. 1993. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research*: p. 1-23.
121. Williamson, M., and Hanks, S. 1994. Optimal planning with a goal-directed utility model. In *the Second International Conference on Artificial Intelligence Planning Systems.*
122. Xuan, P., and Lesser, V. 2002. Multi-agent Policies: From centralized ones to decentralized ones. In *The First International Joint Conference on Autonomous Agents and Multi-Agent Systems.* Bologna, Italy.
123. Yampratoom, E. 1994. Using simulation-based projection to plan in an uncertain and temporally complex world. The Dept., of Computer Science, the University of Rochester.
124. Yokoo, M. 1995. Asynchronous weak-commitment search for solving distributed constraint satisfaction problems. In *the First International Conference on Principles and Practice of Constraint Programming.* Springer-Verlag.
125. Yokoo, M., Durfee, E. H., Ishida, T., and Kuwabara, K. 1992. Distributed constraint satisfaction for formalizing distributed problem solving. In *The Twelfth IEEE International Conference on Distributed Computing Systems.*