UNIVERSITY OF CALIFORNIA, SAN DIEGO

# Automated Planning in Very Large, Uncertain, Partially Observable Environments

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

in

Electrical Engineering
(Intelligence, Robotics and Controls)

by

James M. Vaccaro

Committee in charge:

Professor Clark Guest, Chair
Professor Charles Elkan
Professor William Hodgkiss
Professor Kenneth Kreutz-Delgado
Professor William McEneaney

2010

This dissertation of James M. Vaccaro is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____

_____

Chair

University of California, San Diego

2010

# Table of Contents

# List of Figures

# List of Tables

# Acknowledgements

Deepest thanks to my advisor, loving family, loyal friends, and all those people that evaluated my papers and pushed my limits. I would especially like to thank my advisor Clark Guest, and my coworkers Frank Oldham and Ross Shumway, for their countless technical discussion and insightful feedback. Thanks also to those at UCSD, ORINCON, and Lockheed Martin for their advice, patience and understanding in my pursuits.

The text in Chapter 7 contains material previously presented in three publications: Lecture Notes in Computer Science: Applications of Evolutionary Computation, IEEE Transactions on Evolutionary Computation, and IEEE Congress on Evolutionary Computation. I was the primary researcher and author, and the coauthor directed, and/or supervised in the research, which forms the basis of the chapter.

The text in Chapter 8 contains material previously presented in three publications: Lecture Notes in Computer Science: IEEE World Congress on Computational Intelligence, IASTED Applied Modeling and Simulation Conference, and Modelling, Simulation and Optimization (In Tech Publishing, Chapter 25). I was the primary researcher and author, and the coauthor directed, supervised and/or participated in the research, which forms the basis of the chapter.

# Vita

| | |
|---|---|
| 1987 | B.S. in Electrical Engineering, Clarkson University, Potsdam, NY |
| 1991 | M.S. in Electrical Engineering, Syracuse University, Syracuse, NY |
| 1987-2001 | Electronics Engineer, Air Force Research Laboratory, Rome, NY |
| 2001-2004 | Research Scientist, ORINCON Corp., San Diego, CA |
| 2004-2010 | Research Scientist, Lockheed Martin Corp., San Diego, CA |
| 2010 | Ph.D. in Electrical and Computer Engineering, University of California, San Diego |

## Publications


1. J. Vaccaro, C. Guest, "Modelling, Simulating and Autonomous Planning for Urban Search and Rescue," *Modelling, Simulation and Optimization*, Book: Chapter 25, In-Tech Publishing, March 2010.

2. J. Vaccaro, C. Guest, "Modeling Social Influence via Combined Centralized and Distributed Planning Control," MODSIM World Conference and Expo, Virginia Beach, VA, October 2009.

3. J. Vaccaro, C. Guest, "Modeling Urban Terrain for Simulating Search and Rescue Operations to Train Planners," *IASTED Applied Modeling and Simulation Conference, in press*, Corfu, Greece, June 2008.

4. J. Vaccaro, C. Guest, "Automated Dynamic Planning and Execution for a Partially Observable Game Model: Tsunami City Search and Rescue," *IEEE World Congress on Computational Intelligence (WCCI'08), in press*, Hong Kong, China, June 2008.

5. J. Vaccaro, C. Guest, "Learning Multiple Search, Utility and Goal Parameters for the Game RISK," *IEEE Congress on Evolutionary Computation*, Vancouver, Canada, July 2006, pp. 4351-4358.

6. J. Vaccaro, C. Guest, "Planning an Endgame Move Set for the Game RISK: A Comparison of Search Algorithms," *IEEE Transactions on Evolutionary Computation, Vol. 9, No. 6*, December 2005, pp. 641-652.

7. J. Vaccaro, C. Guest, "Evolutionary Bayesian Network Dynamic Planner for Game RISK," *Lecture Notes in Computer Science: Applications of Evolutionary Computation, EvoSTOC Proceedings*, Coimbra, Portugal, April, 2004, pp. 549-560.

8. J. Vaccaro, C. Guest, D. Ross. "Evolutionary Programming for Goal-Driven Dynamic Planning," *SPIE's 16$^{th}$ Aerosense Symposium*, Orlando, FL, April, 2002.

# Abstract of the Dissertation

Automated Planning in Very Large, Uncertain, Partially Observable Environments

by

James M. Vaccaro

Doctor of Philosphy in Electrical Engineering
(Intelligence, Robotics and Controls)

University of California, San Diego, 2010

Professor Clark Guest, Chair

Automated planning is investigated for very large, uncertain, partially observable environments. Experimental analysis is conducted on two widely different applications: the classic RISK game and a simulation of an Urban Search and Rescue Operation (US&RO). Many aspects of automated planning are explored simultaneously to limit assumptions, problem domain requirements, and subject matter expertise. Challenges addressed are in search capability, implementation strategy (scalability, modularity and extensibility), and machine learning for continual improvement.

A general architecture is designed for future applications, and includes a hierarchical framework, a well-defined set of functions, and a nine-step implementation strategy. Metrics are designed and implemented at seven levels of the hierarchy to address scalability, modularity and extensibility. Experimental results are analyzed for efficiency and effectiveness at three levels of simultaneous machine learning: plan generation (plan search and selection), planning cycle operation (incorporates plan-execution and -assessment as an iterative process), and planners' utility (measure of planners ability to accomplish a mission or win a game).

# Chapter 1

# Introduction

## 1.0 Abstract

In this dissertation, several new extensions to machine intelligence are presented that enable autonomous dynamic planning and execution (ADP&E) in very large, partially observable, and stochastic (VLPO&S) environments. In considering only discrete event space VLPO&S environments, a Partially Observable Markov Decision Process (POMDP) [1] has the ability to frame many ADP&E solutions. However, in real-world planning problems, as in some game problems, where the state space and the number of options are both very large, additional features beyond a POMDP must be considered: (1) problem domain-dependent plan lengths, (2) observability-dependent planning strategies, (3) the temporal cost of planning, (4) risk aversion to avoid plan failure, (5) expectation threshold for triggering re-planning, (5) the probability of successful plan completion, and (6) traceability for justifying a selected plan of action. In this dissertation, a hierarchical framework is constructed that encompasses these modular extensions. This approach is formulated, implemented, and demonstrated on two dissimilar VLPO&S problem domains. A planner improvement methodology is demonstrated on both problems with experimental results, and a general formulation is documented to allow future researchers to apply or extend the methodology.

## 1.1 Motivation

Our motivation for designing and implementing an ADP&E solution for VLPO&S planning problems is threefold: (1) there is a large gap between existing top-down and bottom-up approaches in planning, especially in continuously improving systems; (2) many real-world problems are not solvable using current methods; and (3) existing methods have left many unanswered questions concerning the design and implementation of a large scale ADP&E solution.

There is a large gap between traditional approaches to seeking emergent behavior in planning from a sensor-rich environment (i.e., bottom-up) and to controlling multiple agents in a modeled environment (i.e., top-down). More specifically, bottom-up approaches are limited in their ability to learn from many distributed actions to better pursue long-term goals, while top-down approaches are limited in their ability to apply long-term goals to better coordinate a large number of interactions. The class of planning problems investigated here is of such size and scope that one cannot use a bottom-up or top-down approach alone, because both approaches get too “bogged down” in their own pursuits and do not scale well. In other words, bottom-up approaches seldom learn from high-level goals, while top-down approaches seldom learn from a large number of individual actions.

This large gap is addressed in several ways. First, to ensure scalability, the real world or game environment is modeled as discrete event choices that separate the important static and dynamic features, and further subdivide these features into a modular framework to easily propagate information top-down and bottom-up. Secondly, to ensure

very fast simulation speed, each action response is pre-computed priori to implementation, and populates a lookup table for fast retrieval. Third, to avoid a biased search, action choice models are implemented to initially choose actions independent of goals. Fourth, to ensure a planner improvement capability, learning methods are implemented to adapt a planner's abilities through experience. Finally, to demonstrate this solution is amenable to many planning applications, this approach is implemented on two dissimilar VLPO&S planning problems.

Many VLPO&S planning problems cannot be solved using existing technology. One particular area of interest is incident management. For example, in natural disasters, such as the Indian Ocean tsunami (2004), the Katrina hurricane (2005), the Myanmar cyclone (2008), the China's Sichuan province earthquake (2008), or the Haiti earthquake (2010) history has shown that many victims did not get help in a timely manner. Often the resources were available, but mobilizing them in an efficient and effective manner presents a complex and partially observable planning problem not previously implemented in a tractable or automated form.

Another area of interest are game problems, where there may be stochastic outcomes due to rolling dice (e.g., RISK, backgammon), partially observable or hidden states, such as cards (e.g., RISK, poker), or board pieces (e.g., Battleship, Stratego). Many of these game problems may also have a very large state space (e.g., RISK, Go, Chess), or a very large number of action choices (e.g., RISK, Go, Monopoly, Axis & Allies). Building on models from a POMDP baseline, this methodology can be extended from generating only rudimentary actions to efficiently handling more complex

coordinated action plans for both real world planning exercises and in adapting game players.

## 1.2 Challenges

Designing and implementing an ADP&E system for a wide range of VLPO&S planning problems, three areas are considered: (1) architecture; (2) process flow; and (3) continual improvement. The architecture has broad application by minimizing the functional components to a minimal set, and including a modular hierarchy that can handle large problems without restricting or biasing action choices. The process flow has a variety of modes that depend on the current planning phase (e.g., planning, execution, assessment) and available information. The continual improvement uses a general approach across different applications, independent of individual plans, and interdependent on long-term results of completed games or real-world missions simulations.

More specifically, there are several architecture features. First, a minimal set of essential functional building blocks is issued in the ADP&E architecture. Second, dependencies and functional interactions are explored in this architecture to uncover the required behaviors and interactions. Third, a modular scalable representation is found that subdivides a large variety of VLPO&S planning problems. Fourth, the static and dynamic portions of an application are both encoded into the architecture, considering autonomy and very fast simulation speed. Finally, we design and implement virtual and real models, one for planning and reassessment and the other for execution and observations, respectively.

There are six major process flow concerns. First, an end of a plan criterion is investigated and developed that is dependent on application type. For example, should the system be task completion or resource consumption driven? More specifically, how many projected tasks are completed or how much consumption of the available resources constitutes a plan? Second, a metric for choosing individual outcomes is investigated. For stochastic problems, plans often turn out better or worse than expected. For these cases, a risk aversion parameter can play a role in choosing outcomes that mitigate risk. Choosing an acceptable level of risk can save on resources, computational search time, or allow a plan to reach goals conventional methods may find unattainable. Third, given a set of chosen actions, an algorithm for calculating the probability of successful completion of a plan is developed. In other words, how does one mathematically combine outcome probabilities with action sequences to propagate the probability of success? Fourth, a good way of knowing when to keep a plan or to re-plan is investigated. Fifth, metrics for having a cost of planning, and its impacts on re-planning are investigated. Finally, risk metrics for balancing between the cost of planning, and the risk of continuing the current plans are investigated.

There are four important continuous improvement issues. First, while initially allowing any possible action options, algorithms for coordinating activity of many heterogeneous agents are investigated. In other words, how does one balance the breadth of choosing from many actions, while slowly improving ability to plan? Second, the pros and cons of planning by search and comparison, verses stringing together single action choices are investigated. Many times one must incur a short-term cost to achieve a long-term benefit, because getting into a larger reward region may require suboptimal initial

actions. In other cases, stringing together single action choices may be more advantageous in an unknown environment, because observations may change plans too often for search to be cost effective. Third, the circumstances where individual action trials are useful are investigated. Action and state spaces are very large for the problems studied here, thus a very small percentage of action-state combinations can be processed over the course of many simulations. In other words, given their limited visibility, can state-action pairs provide a useful basis for planning improvement? Finally, how a planner can improve on its ability to plan, including goals, search, risk, and other important variables are investigated. There are many variables in planning that may impact the effectiveness and efficiency of a plan. Can these variables be learned from experience, and if so, what are the mechanisms that lead to improvement, and what are plausible stopping criteria?

## 1.3 Accomplishments

As described above, the motivation and challenges involved in using an ADP&E approach for solving VLPO&S planning problems are extensive. There are many design and implementation tradeoffs that are inherently necessary. Examination of these tradeoffs has led to eight primary accomplishments in this dissertation that address the concerns described above: (1) leverage and extend existing POMDP building blocks to include adaptable, tractable and traceable planning strategies; (2) design and implement an ADP&E scalable framework that handles both partially observable and stochastic environments; (3) include risk aversion and probability of success control for planning within stochastic environments [2]; (4) develop a more efficient planning search

algorithm for large stochastic environments [3]; (5) include the temporal cost of planning to autonomously increase planning efficiency [4]; (6) design and implement a methodology for adapting internal parameters within the ADP&E framework to improve planning effectiveness [5], [6]; (7) demonstrate the utility and versatility of the developed ADP&E approach using two dissimilar applications; and (8) formulate a general implementation strategy for designing, building and testing new applications.

## 1.4 Dissertation Organization

The dissertation organization is sequential, in that it builds on itself. It is best to read the document from start to finish, because many of the later chapters assume definitions and descriptions from previous chapters. Chapter 2 describes the problem domain in detail, and provides definitions for many planning terms to distinguish this work from other planning approaches. Chapter 3 describes related concepts and technologies that are already well established and form a baseline from which to build. Related research will be discussed in detail in Chapter 4. Given the definition of the planning problem in Chapter 2, and the related work in Chapter 4, new challenges are described in Chapter 5. These challenges make the automation of planning unique from those problems already addressed in the literature. Chapter 6 describes the ADP&E approach for general planning problem domains. Chapter 7 describes that approach applied to the RISK game, which includes a description of the implementation details and application results. Chapter 8 describes that approach applied to an urban search and rescue operation (US&RO) mission, which includes a description of the implementation details and application results. Chapter 9 describes generalization of the approach based

on the implementation strategies and results described in Chapters 7 and 8. Chapter 10 provides a summary discussion of the dissertation and describes some potential ideas for future extensions.

## References:


1. R. S. Sutton & A. Barto, *Introduction to Reinforcement Learning*, MIT Press / Bradford Books, Cambridge, MA, 1998.

2. J. Vaccaro, C. Guest, "Evolutionary Bayesian Network Dynamic Planner for Game RISK," *Lecture Notes in Computer Science: Applications of Evolutionary Computation, EvoSTOC Proceedings*, Coimbra, Portugal, April, 2004, pp. 549-560.

3. J. Vaccaro, C. Guest, "Planning an Endgame Move Set for the Game RISK: A Comparison of Search Algorithms," *IEEE Transactions on Evolutionary Computation, Vol. 9, No. 6,* December 2005, pp. 641-652.

4. J. Vaccaro, C. Guest, "Learning Multiple Search, Utility and Goal Parameters for the Game RISK," *IEEE World Congress on Computational Intelligence (WCCI'06),* Vancouver, Canada, July 2006.

5. J. Vaccaro, C. Guest, "Automated Dynamic Planning and Execution for a Partially Observable Game Model: Tsunami City Search and Rescue," *IEEE World Congress on Computational Intelligence (WCCI'06),* Hong Kong, China, June 2008.

6. J. Vaccaro, C. Guest, "Modeling Urban Terrain for Simulating Search and Rescue Operations to Train Artificial Planners," *The 17$^{th}$ IASTED International Conference on Applied Simulation and Modelling (ASM'08),* Corfu, Greece, June, 2008.

# Chapter 2

# Problem Domain

## 2.0 Abstract

The purpose of this dissertation is to provide a novel ADP&E approach to address discrete event space VLPO&S planning problems that are beyond the capabilities of existing approaches. The purpose of this chapter is to define each of the terms in the preceding sentence and their component parts, so the scope of the problem and its solution may be soundly understood.

First, the general terms of ADP&E solution spaces and VLPO&S planning problems are defined. Second, their essential functional objects are defined, followed by a detailed description of their component parts. Third, metrics and behavioral analysis terms are defined. Finally, the domain requirements are described.

## 2.1 General Description

In an *autonomous dynamic planning* and *execution* (ADP&E) solution, *planning* is the generation of a sequence of actions to achieve one or more stated goals, such as winning a game or completing a mission [1]. *Execution* is a process of taking actions within an environment. The combination of planning and execution is enhanced by the inclusion of *assessment*. *Assessment* is the monitoring of action results as they are executed. The purpose of assessment is to determine the degree of success of a plan,

possibly to trigger re-planning if actual results diverge too far from intended goals. The term *dynamic* refers to both a changing environment, and the process of adaptively changing plans while cycling through the planning, execution, and assessment phases. The term *autonomous* refers to a planning system that is self-governing, and continually improves based on previous planning experiences [2].

The problem domains are *very large, partially observable* and *stochastic* (VLPO&S) planning environments. We are not considering continuous real-time systems, but will focus on discrete event space problems that have a well-defined set of actions, and a finite state model that represents the environment. *Very large* refers to both the size of the state space representation and the number of action choices per planning iteration. *Partially observable* environments have unknown values of internal state, and in turn, uncertain action choices and outcomes. Thus, planned actions may need adjustment based on new observations. *Stochastic* environments have more than one probable outcome for a given action; thus planned actions may need to be adapted based on unanticipated outcomes.

## 2.2 Essential Functional Objects

In its most primitive form, an ADP&E solution for VLPO&S planning problems has two essential functional objects: an *environment* and a *planner*. The interaction between the planner and environment are fundamental to the study of planning systems. Basic terms of environments will now be described, followed by basic planner terminology. The following sections will describe more detailed characteristics of environments and planners, respectively.

The problems of interest here are *discrete event space* models of *environments*. *Discrete event* is a measurable change in the state of a system based on actions taken within the state space, and includes the time displacement in making that change [3]. *Discrete space* is a representation in which each spatial dimension is divided into discrete samples, such as a finite set of regions. *States* are defined here as a "condition" or "status" of the environment, thus the current value of event and space variables is considered a state. *Actions* are defined here as elements of a set of all interactions between planners and environments; each action can either change or sense the environment, or both. For discrete time and space problems, *discrete choice models* can represent the *environment* in a simulation. Here, *discrete choice models* [4] are defined as finite state machines that can process actions, track *state transitions* and update state variables with *observations*. When considering each action, a *state transition* is a model's action response or the natural time evolution of some model's state variables, such as health of survivors without food supplies. *Observations* are the receiving of knowledge from an environment through sensed information. In the context of planning within a simulation, the environment is an external model, and the actions include sensors that acquire knowledge as current values of state variables.

Environments can either be *cooperative* or *non-cooperative*. These terms come from game theory [5], and within a game theory context, games are considered equivalent to environments. In game theory, *cooperative* games study coalitions and coordinative aspects of players' interaction, while *non-cooperative* games study single player options based on incentives and the reactions of other players [6]. In terms of planning, players

are equivalent to planners within a game theory context. Thus, single or multiple planners may be considered, depending on the problem.

The *planner* uses a discrete choice model of the environment to generate *plans* and corresponding *expectations* in an effort to direct the model to a more desirable state or *goal*. *Plans* are a selected sequence of actions that project state transitions, observations, expectations, and *schedule*. The *schedule* is the time order for taking each action. In a game, the schedule may simply be the order, and for a real world simulation, the schedule may be the predicted start time and time length of execution. *Expectations* are measures of state deemed important to represent progress made by the planner. Planners develop plans that direct *agents* to interact within environments. *Agents* are defined here as entities that have actuators and sensors and are controlled by the planner. Each agent has *agent-resources*. In the context of planning, *agent-resources* are any attribute of an agent that is in limited quantity and constrains an agent's capability to carry out a plan, such as the fuel capacity of a vehicle agent. Agents with different actuators, sensors, or resource constraints are considered different *agent types*. Planners develop plans that direct the agents of various types and eventually lead to an *endgame* or *end-state*. *Goals* are a measure of the endgame or end-state that determines how well the planner performed over the course of an entire game or simulation. We refer an *endgame* as the completion of a game simulation and an *end-state* as the completion of a real world simulation. In this dissertation, they will be used interchangeably.

## 2.3 Environment

In VLPO&S planning problems, a planner interacts with two *models*. One *model* represents the real environment, where action choices make a permanent change in the model during execution. This model is called the *real world* or *execution-model*. Another *model* is within the planner, where action choices are both tested and evaluated. This model is called the *planning-model*. Both *real world* and *planning-models* are built on a *static infrastructure* and *rules*. The *static infrastructure* is the part of the model that does not change, such as the board layout for a game or a finite set of discrete locations in a real world simulation model. *Rules* define both the available action options based on the current model condition, and the state transitions and observations based on the chosen actions.

Both the real world model and planning-model have the same “very large” finite state set. In this context, “very large” often means that the finite state set is beyond $10^{100}$ elements in discrete event space. An important difference in the two models is that the real world model holds the “true” state values, since it represents the real environment. Many of these state values are partially observable. Thus, the planning-model does not know many “true” state values and often must guess these values in lieu of observation. Therefore, a *planning-model update* is where planning model characteristics are updated to reflect observations of the real world model so that future plans can make use of a more accurate model.

## 2.4 Planners

The environments considered here are not limited by the number of planners in a game or simulation, or the number of agents each planner controls. Thus, one application could have a single planner coordinating resources to complete a real world simulation, while another application could have multiple planners competing to win a game. These applications are referred to as *single-planner* and *multi-planner* games or simulations, respectively. Likewise, single agent applications are referred to as *single-agent* games or simulations, and multiple agent applications are referred to as *multi-agent* games or simulations.

In order for planners to be autonomous, they must include all three *plan-phases*: *plan-generation*, *plan-execution*, and *plan-assessment*. The *plan-phases* are discrete portions of the *planning cycle*. The *planning cycle* is the iteration of the plan-phases as a repeatable ordered sequence. Continual use of the planning cycle until the simulation or game is complete ensures that the planning system is autonomous (note that multiple planning cycles can be working simultaneously for other planners or agent sets). *Plan-generation* is the process of selecting a set of actions using the planning-model with *predicted* outcomes and expectations. *Plan-execution* is the process of sequentially executing each action of a plan in the real world model and observing outcomes. *Plan-assessment* is the process of re-enacting the *plan-remainders* using the planning-model with updated observations and then, comparing new expectations with what was previously predicted in plan-generation for the same set of agents. *Plan-remainders* are the remaining portion of plans after they are partially executed. In the context of

planning, the term *predicted* refers to indicating results in advance of plan-execution. Predictions are only relevant to applications where the model has *uncertainties*. *Uncertainties* refer to both a model's aspects of being partially observable and/or having stochastic outcomes. The following subsections identify and define all terms related to the three-phase planning process.

### 2.4.1 Plan-Generation

In this dissertation, plan-generation comprises many aspects of planning: (1) handling different types of agents; (2) choosing the number of each agent type used; (3) planning action timing and coordination across agents; (4) choosing a plan length; (5) selecting actions; (6) selecting among plans; (7) predicting outcomes under uncertainty; (8) determining a plan's success probability under uncertainty; (9) choosing an observation update function for the planning-model; (10) choosing state features for measuring progress toward goals and expectations; (11) incorporating a cost of planning; and (12) deriving metrics for combining and coordinating these aspects of planning. These twelve aspects do not form an exhaustive list, but each aspect has a set of terms, which when defined, aids in understanding the many facets of plan-generation. The following paragraphs define terms related to these aspects of planning.

The first three aspects described above deal with agents. The handling of agents, choosing which agent types to use, and planning action timing and coordination across agents are just a few of the *variable choices* in plan-generation. *Variables choices* are considered as any choices available to the planner in any of the three phases of planning. In the handling of agents, variable choices include selecting a portion of the agents (i.e.,

number and type) to consider in choosing actions. Not all available agents need to be considered at once when generating plans. The chosen agents are directly connected to the chosen plans of actions described next, because agents are considered as planner *resource objects*. These *resource objects* are tangible entities that may also have internal resources that indicate the health status of an agent, such as fuel level.

Many of these variable choices in the plan-generation phase fall under the term *selection criteria*. *Selection criteria* refer to the methods used when selecting specific choices based on a list of options, such as actions and/or plans with their corresponding agents. Aspects four through seven above are related to the selection criteria. Selection criteria in the domain of projecting future actions are defined in three areas: *action-selection, plan-selection,* and *plan-termination*. Other selection criteria related to an uncertain environment will be discussed later in this section. *Action-selection* entails the *decision-making* methods used to choose an action, given a state. *Decision-making* refers to machine learning technologies used for assigning these actions to states [7]. *Plan-selection* refers to methods for choosing a plan among *comparable plans*. *Comparable plans* are plans that consider the same agent or same group of agents over a similar time period.

*Plan-termination* refers to methods that determine when a plan is complete. *Plan length* is the number of actions required to complete a plan. Plan length is considered a variable choice, so as to not restrict the planner's ability to plan at variable depths. Depending on the type of application problem, plan-termination can either be *turn-based* or *task-based*. *Turn-based* plan-termination is used in some multi-player game problems,

where a player or planner must pass control to other planners over the course of a game, such as in chess or RISK. *Task-based* plan-termination is used in applications where there are no turns, and planners and their agents can operate concurrently in the environment. Furthermore, both task- and turn-based planning can restrict plan length through these three types of plan-terminations that stop after: (1) a number of *task completions*, (2) a *time horizon*, or (3) an amount of *resource expenditures*. *Task completions* are a *task-driven* approach that restricts plan length in terms of meeting expectations, such as reaching a tangible milestone. *Time horizons* are a *time-driven* approach that restricts plan length in terms of projected future time limits, such as an hour or week in real world time. *Resource expenditures* are a *resource-driven* approach that restricts plan length in terms of a limit in portion of resources consumed, such as fuel or agent losses.

Aspect five above, selecting actions, addresses decision-making methods. Decision-making methods for action-selections consider all available options called a *finite action set* [8]. A *finite action set* is all allowable actions and is dependent on state, agent type under consideration, and the number of agents available. This dissertation considers only “very large” finite actions sets. In this context, “very large” means that the finite action set can exceed$10^3$ available options at each action selection step.

Aspect six, selecting among plans, deals with search. Developing comparable plans in plan-selection requires *search*. In plan-generation, *search* refers to a three-stage operation: generating plans, comparing plans, and selecting a plan based on plan-selection described earlier. There are many types of search. This dissertation explores some search approaches called *heuristic search*. These heuristic search methods are

described in detail in Chapter 3: Background. The plan resulting from search is called the *selected plan*. As in the real world, a player is timed in a game, such as in chess, where search for more optimal moves (i.e., plan of actions) is restricted by real time limits. To account for these time limits, here the search process is constrained in the number of *action contemplations* per mission or game. *Action contemplations* are defined here as the combination of an action selection, the integration of that action as a step in a plan, the evaluation of that action step with reward and probability of success, and a comparison of that action's outcome predictions with previously predicted states.

Aspects seven through nine deal with an uncertain environment. When planning under uncertainty, the selected plan has to consider *selecting outcomes*, *probability of successful plan completion*, and an *observation update function* for the planner. *Selecting outcomes* is another selection criterion and refers to choosing a single outcome for each action choice when there is more than one *probable outcome*. For stochastic models, each action can have a set of probable outcomes, and based on a selection criterion, an outcome can be selected to represent the *predicted outcome* or local changed state values (i.e., *selected outcome*). The predicted outcome and selected outcome are considered as equivalent here, and are used interchangeably throughout this dissertation. The selection criterion refers to the methods used in selecting an outcome based on rank-ordering the outcomes in terms of reward (a metric described later). Methods can range from taking the expected value, the maximum likelihood, or a *risk-averse* selection of predicted outcomes. Here, *risk-aversion* is defined as a factor (i.e., from 0 to 1), which allows the planner to choose outcomes based on perceived risk. A full description of how risk aversion is applied is described in Chapter 7.

In planning, predicting outcomes is especially important, because taking sequential actions requires knowledge of previously predicted outcomes. For stochastic models, the *probability of successful plan completion* refers to combining the probabilities of all outcomes to determine the overall probability of success (described in detail in Chapter 7). Successfully predicted outcomes are considered as completed tasks (both described earlier). For both partially observable and stochastic models, the *observation update function* determines when and how often the planning model should be updated by the observations of the real world model. For instance, planning deeply without planning model updates in an uncertain environment can be an inefficient use of planning resources.

Whether an approach uses a decision-making method to choose individual actions to construct a plan or uses search to construct many plans and then chooses one, there must be *value metrics* to determine progress made by the planner. In economics, *value* is the market worth of goods and services [9], and *metrics* are systems of measure of distance [10]. In the context of planning, *value metrics* are the desirability placed on the outcomes of a generated action or plan. In planning, these value metrics contribute to deriving a *fitness function,* which is used here to select a plan from among searched plans. A *fitness function* for plan selection considers factors such as achievable state, probability of successful plan completion, and other factors described more fully in Chapter 6.

Aspects ten and eleven deal with value metrics. For planning systems, the measurable commodities are predicted *state-features. State-features* are some

combination of state variables with their corresponding values. These values are used in selecting actions and/or plans. State-feature values can reflect the *costs* and/or *rewards* of carrying out an action or plan. *Costs* are a function of the state-features that determine a negative impact on choosing actions/plans (e.g., time delays, expended resources, etc.), while *rewards* are a function of the state-features that determine a positive impact on choosing actions/plans (e.g., completed tasks, high probability of success, etc.). The next few paragraphs will describe terms related to state-features, cost, and rewards, respectively.

As described earlier, the goal is to reach an end-state or endgame. However, reaching the end-state or endgame on a single plan in a VLPO&S planning environment is considered infeasible due to the number of action choices available, the size of the state-space, and uncertainties in observations and outcomes. Thus, the many plans that are generated over the course of a game or real world problem are given a measure of desirability for their selection. Using a set of selected state-features (by SME), **costs** and **rewards** can be identified that determine this desirability.

In this dissertation, **costs** are state-features that measure constraints that thwart progress toward winning a game or succeeding in a mission of a real world simulation. In the context of planning, costs are the resources used to do something, such as generating, selecting, or executing actions and/or plans. Thus, the idea here is to minimize these costs. Two state-features considered are the *cost of planning*, and *cost of executing*. The *cost of planning* is the required resources it takes to generate and select a plan for execution. For example, in a multiplayer turn-based game, the cost of planning could be

the amount of search time required to plan a turn in a game. For a real world problem, the cost of planning could be a time penalty representing the time to instruct a human to delegate a new plan to an agent. There are many feasible ways to penalize a planner based on the resources required to generate a plan, depending on the problem under consideration. The *cost of execution* is the predicted resources it takes to execute a plan. For example, each action may have a predicted time to execute in the real world model, which is based on real world time delays. Also, the cost of execution for a plan could be the amount of fuel consumption of a vehicle agent associated with executing all actions in that plan.

Here, **rewards** are state-features that measure progress toward winning a game or optimizing the execution of a real world simulation. Rewards are used to choose actions and plans that lead to these circumstances. In choosing actions, rewards are measured as *weighted action-choice conditions* that aid in action-selection. *Action-choice conditions* are functions of state-features that link individual actions to chosen state-features with the desire to achieve the goal of a game or mission. In choosing plans, rewards are measured as *weighted sub-goals* to aid in plan-selection. *Sub-goals* are functions of state-features that measure some reward value of a plan that may aid in achieving the overall goal of a game or mission. For example, one sub-goal for the game chess is the measure of board pieces a player has compared to his opponent. The term *weighted* refers to the value an action-choice condition or a sub-goal has in choosing actions or plans, respectively. For example, in chess, the value of a queen is more than a rook, and therefore has a greater weight placed on capturing or saving the queen [11].

Some state-features can be used as value metrics that measure progress based on the uncertainties associated with the differences in *predicted-outcomes* versus *realized-outcomes* of a plan. *Realized-outcomes* are plan-execution outcomes resulting from the real world model. Plan-execution is described in the next subsection. *Predicted-outcomes* are plan-generation or plan-assessment outcomes resulting from using the planning-model, where there are outcome uncertainties based on partially observable state variables and stochastic state transitions. Partially observable state variables need to be guessed or predicted in the planning-model, and stochastic outcomes need to be selected based on outcome selection in the planning-model.

Aspect twelve, choosing *utility metrics* for combining and coordinating the first eleven aspects of planning, is central to planning (i.e., plan-generation, plan-execution, and plan-assessment). In economics, *utility metrics* is a measure of desirability of goods and services based on their level of consumption [12], [13]. In plan-generation, utility metrics refer to the values of *planner parameters* based on their usefulness in fulfilling the ultimate goal of winning a game or optimizing performance of a real world mission simulation. The metric function used to determine the success or failure of a mission or loss of a game is called the *mission value function*. *Planner parameters* refer to all control parameters for plan-generation, plan-execution, and plan-assessment. The metric function that values the usefulness of these parameters is called the *planner utility function*. The metric function used to evaluate the effectiveness of the overall planning cycle is called the *planning-cycle assessment function*. Plan-execution and plan-assessment are described in the next two subsections. All metrics used in the ADP&E system are described in Section 2.5: Behavioral Analysis. The variable planner

parameters chosen are approach dependent, thus they are described in Chapter 6: Approach.

### 2.4.2 Plan-Execution

After plans are generated and selected in the plan-generation phase, the plan-execution phase begins. In this dissertation, each planner is restricted to occupy only one phase at a time. However, in the multi-planner case, planners can be in different phases. Plan-execution attempts each action of a plan in the scheduled order as identified in the selected plans. Each agent has a plan, thus the execution of actions are allowed to proceed concurrently among agents. Once each action is executed in the real world model, the realized outcomes in the model are irreversible. Reversing a plan can only be done in the planning-model, which is used in both the plan-generation and plan-assessment phases. It is assumed that the agent executing the actions retains all observations made over the course of plan-execution. The observation update function defined earlier determines when the plan-execution phase ends and the plan-assessment phase begin.

### 2.4.3 Plan-Assessment

Over the course of executing a plan, due to uncertainty, expectations may not be achievable. For instance, in stochastic planning problems, where there is more than one probable outcome for a given action, predicted outcomes are often not realized, and the model could be steered into an under-achieving state. In partially observable planning problems, new information may be acquired through plan-execution that values the plan-remainder below expectations. In both cases, a planner may consider *re-planning* if the

cost of taking the existing plan outweighs the costs of re-planning. Evaluating this tradeoff is the *assessment function*. The *assessment function* considers whether expectations are acceptable or not for a given plan. If the assessment function decides to retain the existing plan-remainder, it is called *plan-continuation*. If the assessment function decides to discard the existing plan-remainder and generate a new plan, it is called *re-planning*. Once all agent plans have been assessed in the plan-assessment phase, the plan-generation phase begins again. In the plan-continuation case, that agent's plan is not changed in the next plan-generation phase. In the re-planning case, a new plan is generated for that agent from some point in the existing plan.

## 2.5 Behavioral Analysis

There are seven metrics mentioned above that determine: (1) outcome selection using a risk sensitive (i.e., risk averse) prediction, (2) action selection using preselected feature conditions, (3) whether to re-plan or continue with current plan based on achieving an acceptable level of expectations, (4) plan selection using a reward based fitness function, (5) monitoring task completion efficiency using planning-cycle assessment, (6) comparing planners' usefulness within a single mission or game using planner utility, and (7) comparing planner results across multiple missions or games to determine mission value. Likewise, there are seven corresponding behavioral justifications for: (1) selecting particular outcomes to be risk-averse (outcome-selection justification), (2) selecting particular actions to advance toward desirable states (action-selection justification), (3) re-selecting or keeping particular plans to meet expectations and avoid traps or dead ends (plan-assessment justification), (4) selecting a particular

plan to achieve the probable best forecast (plan-fitness justification), (5) balancing computational resources throughout the planning cycle to increase cycle efficiency (planning-cycle justification), (6) comparing planners' abilities to meet mission or game goals to select the best planners (planner-utility justification), and (7) comparing mission or game results to improve future planners (mission-value justification).

## 2.6 Domain Requirements

The environments of concern here have been described above as VLPO&S planning problems. The planners have been described as ADP&E solution spaces. The interaction of these two objects leads to the following domain requirements:

- State-space representation (very large > $10^{100}$)
- Finite set of available actions (preferably large > $10^{3}$)
- Known state transitions (possibly stochastic)
- Finite set of measurable state-features (possibly partially observable)

The four requirements are aspects of a POMDP. However, these attributes go beyond a POMDP due to the size and complexity of the problems studied. POMDPs will be described further in Chapter 3: Background, and the challenges associated with extending current approaches to handle VLPO&S planning problems and ADP&E solution spaces is described in Chapter 5: New Challenges.

**References:**


1. J. Vaccaro, C. Guest, “Evolutionary Bayesian Network Dynamic Planner for Game RISK,” *Lecture Notes in Computer Science: Applications of Evolutionary Computation, EvoSTOC Proceedings*, Coimbra, Portugal, April, 2004, pp. 549-560.

2. J. Vaccaro, C. Guest, “Learning Multiple Search, Utility and Goal Parameters for the Game RISK,” *IEEE Congress on Evolutionary Computation*, Vancouver, Canada, July 2006, pp. 4351-4358.

3. Jeff S. Steinman, Discrete-event simulation and the event horizon, ACM SIGSIM Simulation Digest, v.24 n.1, p.39-49, July 1994.

4. Train, K(2003). "Discrete Choice Methods with Simulation", Massachusetts: Cambridge University Press.

5. D. Fudenberg and D. K. Levine, *The Theory of Learning in Games*, MIT Press, 1998.

6. D.K. Levine, “Game theory,” *Encyclopedia of Cognitive Science: Psychology Articles*, V1, due Nov. 2002.

7. Charles H. Kepner, Benjamin B. Tregoe (1965). The Rational Manager: A Systematic Approach to Problem Solving and Decision-Making. McGraw-Hill, June 1965.

8. Rath, Kali P. Representation of Finite Action Large Games, *International Journal of Game Theory*, 1995, vol. 24, issue 1, pages 23-35.

9. Steve Keen *Debunking Economics*, New York, Zed Books (2001) p. 271.

10. Rolewicz, Stefan (1987). *Functional Analysis and Control Theory: Linear Systems*. Springer.

11. Tarrasch, Siegbert (1987). *The Game of Chess*. Courier Dover Publications.

12. Neumann, John von and Morgenstern, Oskar *Theory of Games and Economic Behavior*. Princeton, NJ. Princeton University Press. 1944 sec.ed. 1947.

13. Nash Jr., John F. The Bargaining Problem. *Econometrica* 18:155 1950.

# Chapter 3

# Background

## 3.0 Abstract

In this background chapter we introduce **concepts** and **technologies** used or referenced that relate to our research. **Related concepts** are *discrete choice models* and *Markov models* used in *decision-making* within the context of *machine learning*. Machine learning is concerned with the design and development of algorithm-based learning [1]. Machine learning algorithms are methods that improve a system's skill or knowledge in a particular domain through experience. Machine learning can include interaction with the real world [2], [3], [4] or interaction within a computer-generated world [5], [6]. We chose the latter for our study, and our computer-generated worlds are simulations that represent real world or game applications. *Decision-making* is an algorithmic process that leads to selecting a course of action (COA) or a plan from among many alternatives [7]. In economics, "*discrete choice problems"* involve choices between two or more discrete alternatives [8]. In machine learning, the term "*discrete choice models"* refers to the use of a finite set of states or symbols that represent all alternatives and outcomes [9]. *Markov models* are a subset of discrete choice models, where the future state is only dependent on the current state and is independent of past states [10].

**Related technologies** within the planning domain are numerous. Depending on the application, some algorithms may be more advantageous than others. To maintain a manageable scope, only algorithms that are employed in our model, and direct competitors to them, are reviewed. New algorithms will be discussed in Chapters 6, 7 and 8, while well-established algorithms are discussed here. For instance, dynamic Bayesian networks (DBN) are used for output processing [11], and a variety of search algorithms were examined for deep planning. These algorithms include three broad areas — evolutionary, stochastic, and tree search — and, along with the model concepts mentioned above, will be described in detail in this chapter.

## 3.1 Related Concepts

The related concepts described here are discrete choice and Markov models. Discrete choice models attempt to model the decision process of an individual or agent in a particular context [7]. Choice modeling may also be used to estimate non-market environmental costs and benefits [9]. A Markov model is a type of quantitative modeling that involves a specified set of mutually exclusive and exhaustive states, and for which there are transition probabilities of moving from one state to another (including remaining in the same state) [10].

### 3.1.1 Discrete Choice Models

Given that machine learning approaches are studied within a computer simulation, a model representation that is discrete in event space and choice is required. As described in Chapter 2, discrete choice models for decision-making require a framework with three

fundamental functions: (1) state representation of the environment, (2) action options available based on state, and (3) an evaluation function to measure progress.

This simple discrete choice model can be improved through: (1) *learning from the environment*, (2) *incorporating human expertise*, and (3) *maintaining progress toward goals*. Learning from the environment is often referred to as "data mining". Data mining is the process of sorting through large amounts of state or environmental data and extracting relevant and useful information [12]. Human expertise, often encoded as "inference rules," is represented in a symbolic format fit for machine processing [13]. Inference rules are logical operators that manipulate a set of premises in order to reach a set of conclusions [14]. Maintaining progress toward goals refers to designing metrics that measure progress toward achieving goals based on a changing environment [15]. These three improvement strategies form a broad and complementary set of approaches in machine learning. We will refer to these approaches as **data-driven**, **expert-defined** and **goal-directed**, respectively.

**Data-driven** approaches fall into two broad classes: *supervised* and *unsupervised learning*. *Unsupervised learning* is an autonomous process that requires no oversight and is often referred to as self-organization [16]. This approach strictly processes the data with a mathematical formulation to achieve some steady state representation, such as clustering [17]. These formulations often stem from the principles of self-organization, or from statistical [16] or economical analysis [18]. Some well-known unsupervised approaches are self-organizing maps [16], statistical classifiers, such as Bayesian inference [19], principle component analysis [20], and independent component analysis [21]. On the other hand, *supervised learning* approaches require either a human-in-the-

loop, or data that is pre-labeled, so as data is analyzed, some generalized associative mapping is formed between the data and prescribed labels. Some well-known supervised approaches are back-propagation networks (BPN) [22], support vector machines (SVM) [23], and recurrent neural networks (RNN) [24].

Unsupervised learning approaches are used to a limited extent in our ADP&E system. Given the size and complexity of the environments used, geographical clustering is done on the terrain environment used in our US&RO application (described in Chapter 8) to allow for offline processing of all possible planning paths. Self-organization techniques are also used in assigning diversity metrics for planning search in the RISK game application described in Chapter 7.

Supervised learning is not used in our approach. Approaches such as BPN, SVM, and RNN do not translate to a straightforward methodology for justifying plans of actions. In other words, these networks include a complex connectionist model that has many eloquent properties [25], but their internal representation requires a complex interpreter to extract logical meaning of taking a particular action [26]. In addition, it is well known that these models degrade in performance with the length of decision sequences [27]. Planning in complex environments often requires long sequences of actions.

**Expert-defined** approaches can be applied in one of two ways: *a priori knowledge*, where human expertise is encoded as symbolic computational models [28], or *a posteriori knowledge*, where humans sit in judgment over a system to monitor, analyze, and correct a plan generated by the planning system. *A priori knowledge* systems are

often classified as *inductive* or *deductive* reasoning [29]. *Inductive* reasoning makes generalizations from a set of individual instances. In other words, induction reasons from a large number of particular examples to generalized rules. *Deductive* reasoning is drawing a logical conclusion from a set of premises or predetermined conditions. Both of these types of reasoning can be translated into a rule- or case-based reasoning (CBR) format. CBR is an analogy-making approach where past experiences are encoded to solve new similar problems [30]. For example, CBR could form a diagnosis template for an auto-mechanic to determine what is wrong with a car based on past experiences of similar diagnostic results on similar cars. Rule-based approaches are logical statements of judgment or truths that are represented as linked arrangement of premises or antecedents to prescribed conclusions or consequences. CBR and rule-based reasoning approaches often require a tremendous amount of human knowledge to be useful. Their implementation of human knowledge is typically ad hoc in nature and suffers from the "curse of dimensionality" [31]. The curse of dimensionality refers to the exponential increase in dimensionality of the space when one adds another dimension to the problem. For example, adding an action to a plan typically increases the number of possible choices of that action exponentially.

A very large computer system that relies on this approach is called Cyc (pronounced like psych). Cyc has had an incredible amount of effort expended in an attempt to build a core artificial intelligence engine by infusing common sense human expertise. In spite of this effort, the computer system cannot perform the reasoning capability that was initially proposed [32]. Cyc was intended to communicate with

humans using natural language and was intended to establish a common vocabulary for automatic reasoning [33].

We have not relied heavily on human expertise in our approach. We only use human knowledge to select initial sub-goals and state-features that may be important in action and/or plan selection. Based on the experience of simulating results, the system can choose to use sub-goals or state features, or not. How the use of sub-goals and state features are modified and adapted is described in Chapter 6: Approach.

*A posteriori knowledge* systems rely on an individual or team of individuals to reliably monitor and correct plans. Humans make mistakes due to fatigue, and often lack attention for monotonous tasks. Humans are inherently biased based on their experiences, take time to plan, cost money for their efforts, and often have to be trained to make practical plans in the particular domain of interest. Including a human-in-the-loop would preclude running thousands of autonomous example simulations for training planners. Due to these challenges, posterior influence on planning is avoided in our approach.

**Goal-directed** approaches are referred to here as those that have a simple long-term objective, such as winning a game or completing a mission. For the VLPO&S planning problems studied, the challenge is finding the near optimal sequences of short-term action sets that best lead to that objective. Goal-directed approaches for planning are numerous, but they can be grouped into two distinct classes: *goal-state* and *sub-goal-state* feedback systems. *Goal-states* are only those states that represent a completed task or endgame situation. *Sub-goal-states* are a set of features that lead or provide guidance to reaching the goal-state.

*Goal-state* feedback systems can rely on direct feedback from the model environment, and can be applied to propagate values back from the goal-states to other visited states to eventually "value" all states to encode optimal action paths. These approaches, such as dynamic programming [34] or reinforcement learning, assume that the "values" will eventually converge within a reasonable number of random trials. These approaches also assume that the number of state-action pairs in the environment is small enough so that each pair can be visited multiple times to ensure convergence on a solution [15].

*Sub-goal-state* feedback systems rely on selected information about states to direct plans to an endgame or mission-accomplished state. Sub-goal-states are a limited set of states deemed as important contributors in achieving the goal-state, but are available throughout the simulation and not just an end-state. This approach avoids the problem of finding the endgame or goal-state, as many games and real world problems are not guaranteed to reach the endgame through random actions. For example, in many games, such as Monopoly, RISK, or even Chess, random moves may never reach an endgame situation. For real world models and other games, these approaches may be sub-optimal, because the selected sub-goal-states and their relative importance are initially based on biased human expertise. To approach a more optimal plan, these approaches must not only be able to find the best action sets, but also adapt sub-goal-state parameters to improve play.

In summary, goal-state feedback systems have better convergence properties [35], such as reinforcement learning, but are not feasible for large-scale problems. However,

subgoal-state feedback systems can limit the number of states searched to ensure an endgame solution, but are more difficult to train. Since we are dealing with large-scale problems, our emphasis is placed on the multiple stage learning systems of sub-goal-state feedback systems. Chapter 4: Related Work, will give a more complete comparison of data-driven, expert-defined and goal-directed approaches, and Chapter 6: Approach, will provide a detailed description of our sub-goal-state feedback approach. Specific application examples are described in Chapters 7 and 8.

### 3.1.2 Markov Models

A Markov model is a probabilistic process, which has the distinct property that, given the present state, future states are independent of past states. In other words, the present state fully captures the information for evolving future states. This is an important property in planning, because one can restrict search to starting from the current state. We make this assumption in our approach.

Markov models can be represented as graphs of states connected by transition probabilities. Nodes and edges of a graph can represent a variety of information, depending on the type of Markov process in use. Markov models can either be explorative, such as Markov chains or hidden Markov models (HMMs), or exploitative, such as Markov decision processes (MDPs) or POMDPs. The difference between explorative and exploitative Markov models is the use of control. Exploitative models allow the use of control, such as choosing courses of action, while the model parameters are fixed. Explorative models do not allow complete control but allow for the learning of the model parameters from external sources, such as transition probabilities. The

difference between Markov Chains, HMMs, MDPs, and POMDPs is observability [15], [36], [37]. Markov chains and MDPs are fully observable, while HMMs and POMDPs are only partially observable. In HMMs, one can find the hidden parameters of a model from observed parameters, and a HMM is the simplest example of a dynamic Bayesian network (DBN). DBNs are used extensively in our approach for *outcome processing* and are more thoroughly described below in the Section 3.2 Related Technologies. *Outcome processing* in the context of planning are the functions used in calculating probabilities of future outcomes. Our approach considers both fully and partially observable models.

A MDP is a 4-tuple process ($S, A, P_a, R_a$), where $S$ is the state space, $A$ is the action space, $P_a$ is the transition probability space for action $a$, and $R_a$ is the reward space for action $a$. In dynamic programming or reinforcement learning, the goal is to maximize some cumulative function of the rewards. For a POMDP, the process is the same as an MDP, except that many states may be unknown. For this purpose, it is useful to define the function that corresponds to taking the action $a$ and then continuing to optimize:

$$Q(s,a) = R(s) + \gamma \sum_{s'} P_a(s,s')V(s') \qquad \text{(eq. 3.1)}$$

where $Q(s,a)$ is the cumulative reward of state action pair $s$ and $a$, $s'$ is the projected state, $\gamma$ is the discount rate ($0 < \gamma \leq 1$), $R(s)$ is the reward at state s, and $V(s')$ is the value of state $s'$. If the discount rate is zero, only immediate rewards are averaged, or if the discount rate is one, all previous rewards to the beginning are averaged equally. If the discount rate is between zero and one, then reward is weighted by $\gamma^n$ where n is the number of time steps prior to the current state. In our approach, we use POMDPs to represent the unchanging aspects of the environment with variable state values. POMDPs

are however incomplete for handling planning problems because of hierarchical and dynamical concerns that will be further described in Chapter 6: Approach.

## 3.2 Related Technologies

The related technologies described in this section are outcome processing and search algorithms. Outcome processing refers to the algorithms used to handle actions with multiple probable outcomes in chains of selected actions. Search algorithms are concerned with techniques explained in this dissertation to search for and select plans.

### 3.2.1 Outcome Processing

Outcome processing is performed by combining three technologies: DBN, rank-based ordering, and data clustering. A Bayesian Network (BN) is a directed acyclical graph, where the nodes represent a set of variables, while their connections (edges) represent the conditional probabilities among the variables. A DBN is a BN that can model sequences of variables, such as a sequence of outcomes from a plan. Sequences of variables are processed through a DBN uses Bayes' Theorem [38]. Bayes' Theorem is represented as $P(A/B) = P(B/A)P(A)/P(B)$ and relates the conditional and marginal probabilities of outcomes A and B, where B has probability greater than zero. P(A) is the marginal probability of A, since it does not take into account any information about B. P(A/B) is the conditional probability of A given B, because A depends on the value of B. Likewise, P(B/A) is the conditional probability of B given A. P(B) is the marginal probability of B, and acts as a normalizing constant. In our approach, DBNs are performed on all action outcomes to calculate each subsequent outcome.

Given that subsequent outcomes can grow exponentially in number, and many are identical in cost or reward, a *rank-based scheme* with *data clustering* is used to consolidate the outcomes after each DBN propagation step. *Rank-based schemes* refer to items that can be compared to see which items come first, second, etc., until all items are labeled [39]. In planning outcomes, there are various rank-based schemes, but a useful way is to use the value given to each outcome. *Data clustering* is a technique of partitioning a data set into groups that have common traits [40]. In the rank-based scheme, data is ranked according to its numerical values (i.e., outcome states) or as processed reward values (a metric described in Chapter 2). Thus, the values can be grouped together via data clustering, which avoids having the number of outcomes grow too quickly after each action step. Many elements of planning can be clustered to reduce the complexity of the model, such as outcomes, actions, state values, and state nodes. In Chapter 7: RISK Game Application, use of clustering on outcome processing will be explained in detail. In Chapter 8: Urban Search and Rescue Operation Application, state node clustering will be described to reduce the complexity of the state representation.

### 3.2.2 Search Algorithms

The role of the search algorithm within a planning solution is to search an internal game (i.e., planning-model) or real world model (i.e., execution-model) to achieve a goal. Given a set of possible actions, a search algorithm can sequence these actions together to form a plan that explores a path through the internal model. The plan is graded based on a prescribed reward system that measures the effectiveness of the plan by predicted outcomes. The reward system depends on the application, and the reward systems for

RISK and US&RO applications are described in detail in Chapters 7 and 8, respectively. Each action within the plan may have multiple possible outcomes. As mentioned earlier, outcomes for each action can be ranked-based on value or reward and assigned probabilities by the DBN.

All search strategies used are in the family of heuristic search algorithms. Seven search strategies were used to compare and contrast approaches within the stochastic RISK Game application. Results of these search algorithm comparisons are given in Chapter 7: RISK Game Application. These approaches include classical methods (heuristic depth-first, breadth-first and best-first search), random search, gradient search, simulated annealing, and evolutionary search.

#### 3.2.2.1 Classical Tree Search Methods

Search algorithms for decision-trees use nodes to signify each decision point in the tree. Each node represents an action and there is a set of possible outcomes from applying that action. For our purposes, a branch in the tree is not an outcome, but another action taken from the predicted outcome. Thus, each node represents an action and all the probable outcomes with an attached reward based on predicted state and an occurrence probability. For the purposes of heuristic search, nodes are either on a closed or open list. Nodes on the closed list are either termination nodes or nodes where all subsequent actions have been opened. Nodes on the open list are subject to further exploration. Expanding nodes in the tree can be done in a variety of ways; for our purposes, we use the classical depth-first, breadth-first, and best-first algorithms [41] as described below. We include heuristics in the algorithms to strengthen the search toward the goal.

All three algorithms begin the same way. One initially expands all possible initial actions from the root node. All initial actions are put on an open list while the root node is put on a closed list. From this point, the most rewarded action is chosen. Next, the selected action node is expanded for all possible next actions again. The selected node is put on the closed list, while all new action nodes are placed on the open list.

For heuristic depth-first search, the new set of nodes is now the focus of the search, and the fittest member (one of greatest reward value) of that group is selected and expanded. Then the whole process repeats itself until either a termination node or goal node is reached. At a termination node, the search expands the next highest fitness at the greatest depth of open nodes and the process repeats itself. At a goal node the search is complete and stops. In case of a tie in both depth and fitness level, an action from that tied group is selected randomly.

For heuristic breadth-first search, all the nodes nearest to the root node and still open are the focus of the search and the fittest member of that group is selected. The whole process repeats itself until either a termination node or goal node is reached. At a termination node, the search expands the next-highest fitness at the least depth area of open nodes and the process repeats itself. At a goal node the search is complete and stops. In case of a tie in both depth and fitness level, an action from that tied group is selected randomly.

For heuristic best-first search all the open nodes are the focus of the search and the fittest member of that group is selected and expanded. The whole process repeats itself until either a termination node or goal node is reached. At a termination node, the

search expands the next highest fitness of open nodes and the process repeats itself. At a goal node the search is complete and stops. In case of a tie in fitness, an action from that tied group is selected randomly.

#### 3.2.2.2 Random Search Method

A good baseline for comparison of search algorithms is the random search method, where each action is chosen randomly at every step of the plan until the plan reaches an end. In our case, the end is where a task is completed, resources are exhausted, or the plan reaches an accomplished mission or endgame. Random search is a form of depth first search that does not expand the nodes at each step in the plan. The random search of interest here is a random walk [42], where each plan is generated from beginning to end, choosing random actions.

#### 3.2.2.3 Gradient Ascent Search Method

Gradient search [43] is concerned with ascending up the fitness slope over multiple dimensions to reach a goal. The dimension of the planning space can be seen as each step in the plan. Since the fitness landscape is not a smooth surface in the planning space, gradient search will seldom reach the goal by ascending at each step in the plan. Thus, a heuristic form of gradient search was used to get more competitive results. Chapter 7: RISK Game Application will describe in detail a gradient ascent search method that fits the problem.

#### 3.2.2.4 Simulated Annealing Search Method

Simulated annealing [44] uses a declining temperature factor to initially allow wide variation in choices and then over time the declining temperature restricts the search

to only a few choices. This approach overcomes the rough fitness surfaces of many planning spaces, such as observed in the RISK game application. Much like random search described above, simulated annealing is only concerned with modifying one selection, in this case, one plan. Chapter 7: RISK Game Application will describe in detail a simulated annealing search method that fits the problem.

### 3.2.2.5 Evolutionary Search Method

Evolutionary computation has four iterative components: *reproduction*, *modification*, *competition*, and *selection* [45]. *Reproduction* ensures there are multiple plans for comparison. *Modification* includes both mutation and recombination of plans. Mutation is a stochastic variance of a plan, while a recombination or crossover is a combination of two or more plans. *Competition* allows a mechanism for comparison and *selection* and is based on the results of the competition. Random variation is employed in all four components to enhance the exploratory aspects of evolution and to keep the approach robust in avoiding local minimums. This approach is also a heuristic approach and convergence on a solution typically takes many iterations. Chapter 7: RISK Game Application will describe in detail an evolutionary search method that fits the problem.

**References:**

1. Mitchell, T.M., 1997, *Machine Learning*, McGraw-Hill.

2. K.Z. Haigh and M.M. Veloso, “Planning with dynamic goals for robot execution,” *AAAI Symposium*, Nov. 1996, pp. 61-71.

3. J.S. Zelek, “A framework for mobile robot concurrent path planning & execution in incomplete & uncertain environments,” *Integrating Planning, Scheduling and Execution in Dynamic and Uncertain Environments*, AAAI Technical Report WS-98-02, AAAI Press, 1998.

4. K. Deb, D.K. Pratihar and A. Ghosh, "Learning to avoid moving obstacles optimally for mobile robots using a genetic-fuzzy approach," *PPSN 1998*, pp. 583-592.

5. S. Torma, "A model for the dynamic planning of industrial projects," *Acta Polytechnica Scandinavica, Mathematics, Computing and Managament in Engineering*, Series No. 85, Espoo 1997.

6. G. Armano, G. Cherchi, and E. Vargiu, "An agent architecture for planning in a dynamic environment," *Lecture Notes in Computer Science, V. 2175*, 2001, pp. 388-393.

7. Phil Faye & Emily Andrew, "Joint Synthetic Battlespace 'Applying Simulation to Acquisition, Mission Effectiveness, and Course of Action Analyses'," *MSIAC's M&S Journal*, V. 3 N. 1, Fall 2001.

8. McFadden, Daniel L. (1984). *Econometric analysis of qualitative response models*. Handbook of Econometrics, Volume II. Chapter 24. Elsevier Science Publishers BV.

9. Ben-Akiva, M. and Bierlaire, M. (2003). Discrete choice models with applications to departure time and route choice, *in* R. Hall (ed.), *Handbook of Transportation Science, 2nd edition*, Kluwer.

10. An Introduction to Hidden Markov Models, L. R. Rabiner and B. H. Juang, *IEEE ASSP Magazine*, Vol. 3, No. 1, pp. 4-16, January 1986.

11. Z. Ghahramani, Learning Dynamic Bayesian Networks (1997), Lecture Notes In Computer Science, Vol. 1387, 168-197.

12. Kantardzic, Mehmed (2003). *Data Mining: Concepts, Models, Methods, and Algorithms*. John Wiley & Sons.

13. A. Wolf, "A rule-based approach to dynamic constraint satisfaction problems," *Proceedings of the 12th International Conference on Applications of Prolog*, 1999.

14. Muggleton, S., and De Raedt, L., 1994, Inductive logic programming: theory and methods, *Journal of Logic Programming*, 19-20:629-679.

15. R. S. Sutton & A. Barto, *Introduction to Reinforcement Learning*, MIT Press / Bradford Books, Cambridge, MA, 1998.

16. T. Kohonen, *Self-Organizing Maps*, Springer-Verlag Berlin Heidelberg, 1995, pp. 1-50.

17. Jain, Murty and Flynn: *Data Clustering: A Review*, ACM Comp. Surv., 1999.

18. Kahn, Matthew E. and Vaughn, Ryan K. (2009) "Green Market Geography: The Spatial Clustering of Hybrid Vehicles and LEED Registered Buildings," *The B.E. Journal of Economic Analysis & Policy*: Vol. 9 : Iss. 2 (Contributions), Article 2.

19. Bernardo, José M. and Smith, Adrian F. M. (1994). *Bayesian Theory*. Wiley.

20. Lay, David. (2000). Linear Algebra and It's Applications. Addison-Wesley, New York.

21. Bell, Anthony and Sejnowski, Terry. (1997) "The Independent Components of Natural Scenes are Edge Filters." Vision Research 37(23), 3327-3338.

22. Stuart Russell and Peter Norvig. *Artificial Intelligence A Modern Approach*. p. 578.

23. Corinna Cortes and V. Vapnik, "Support-Vector Networks", Machine Learning, 20, 1995.

24. Mandic, D. & Chambers, J. (2001). *Recurrent Neural Networks for Prediction: Learning Algorithms, Architectures and Stability*. Wiley.

25. Rumelhart, D.E., Hinton, G.E. and Williams, R.J. (1986). "Learning internal representation by error propagation", in Parallel distributed processing: Exploration in microstructure of cognition, Vol. (1) (D.E. Rumelhart, J.L. McClelland and the PDP research groups, edn.) Cambridge, MA: MIT Press, 318-362.

26. Warner, B. and Misra, M. (1996). Understanding neural networks as statistical tools. *American Statistician*, 50, 284-93.

27. Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, 5(2), 1994, pp. 157-166.

28. Greenberg, Robert. "Kant's Theory of A Priori Knowledge," Penn State Press, 2001.

29. Hurely, P. J. (2000) A Concise Introduction to Logic - 7th Edition.

30. An, A. and N. Cerone. 1998. Multimodal reasoning with rule induction and case-based reasoning. In Multi-modal Reasoning: Papers from the 1998 AAAI Spring Symposium. AAAI Press, Menlo Park, CA pp. 118-123.

31. H.J. van den Herik, J. W.H.M. Uiterwijk & J. van Rijswijck, “Games solved: now and in the future,” *Artificial Intelligence V. 134*, 2002, pp. 277-311.

32. D.B. Lenat, Cyc: A Large-Scale Investment in Knowledge Infrastructure. Communications of the ACM 38, no. 11, November 1995.

33. M. Witbrock, E. Coppock, and R. C. Kahlert, Uniting A Priori and A Posteriori Knowledge: A Research Framework in *Knowledge Representation (IR-KR): Proceedings of the IJCAI-09 workshop*, pp. 22-28, Pasadena, CA. 2009.

34. Bellman, R. E. (1957). Dynamic Programming. Princeton University Press, Princeton, NJ.

35. Singh, S., Kearns, M. (2002). Near-Optimal Reinforcement Learning in Polynomial Time. Machine Learning journal, Volume 49, Issue 2, pages 209-232, 2002.

36. N. L. Zhang & W. Zhang, “Speeding up of convergences of value iteration in partially observable Markov decision processes,” *Journal of Intelligence Research*, V. 14, 2001, pp. 29-51.

37. B. Bakker, “Reinforcement learning with long short-term memory,” *Advances in Neural Information Processing Systems, V. 14*, 2001.

38. T. Bayes. (1763). “An Essay towards solving a Problem in the Doctrine of Chances.” *Philosophical Transactions of the Royal Society of London* **53**: 370–418.

39. Blickle, T., Thiele, L.: A comparison of selection schemes used in genetic algorithms. Technical Report 11, Swiss Federal Institute of Technology (ETH), Computer Engineering and Communication Networks Lab, Zurich, Switzerland, December (1995).

40. Jain, Murty and Flynn: *Data Clustering: A Review*, ACM Comp. Surv., 1999.

41. N. Nilsson, *Artificial Intelligence: A New Synthesis*, Morgan Kaufmann Publishers, Inc., San Francisco, CA 1998.

42. Barry D. Hughes (1996), *Random walks and random environments*, Oxford University Press.

43. Jan A. Snyman (2005). Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms. Springer Publishing.

44. E. Weinberger, Correlated and Uncorrelated Fitness Landscapes and How to Tell the Difference, Biological Cybernetics, 63, No. 5, 325-336 (1990).

45. D. B. Fogel. Evolutionary Computation. Toward a New Philosophy of Machine Intelligence. IEEE Press, Piscataway, NJ, 1995.

# Chapter 4

# Related Work

## 4.0 Abstract

Solving VLPO&S planning problems within an ADP&E solution space can be attempted by a broad range of technologies. VLPO&S planning problems include many games, such as RISK, Poker, or Stratego, and real world navigation, wargaming, or logistical operations problems, respectively. Our focus is on finite event space domain problems. Related work within this domain is judged based on whether the technology can handle the complexity of VLPO&S planning problems, and whether it fits within the ADP&E solution space.

Related work includes all technologies that can: be captured within a state, option, value construct; have autonomous interaction with a simulated environment; address some of the POMDP limitations; and address some of the current machine learning challenges. Many of these characteristics are discussed in detail in Chapters 2 and 3. Our goal here is to systematically present planning solutions that fit within this domain. In other words, we examine tools or techniques that could autonomously generate, execute and assess plans, and improve on their behavior.

As discussed in Chapter 3: Background, there are three broad design, development, and improvement approaches in implementing a planning solution. The

approaches of data-driven, expert-defined, and goal-directed cover a broad range of solutions. Here, we will discuss these three solution domains in more detail and present their general characteristics, example methods, and their strengths and weaknesses. Secondly, we will present hybrid solutions that attempt to extend these approaches to improve on their strengths for solving a set of game problems. Finally, we will present a summary of important characteristics for the presented approaches.

## 4.1 General Approaches

ADP&E solutions represent a large area of study, and thus we first examined general approaches with broad appeal, such as data-driven, expert-defined, and goal-directed introduced in Chapter 3. These three approaches are fundamentally different and complimentary. The following paragraphs describe the pros and cons of using these methods within an ADP&E solution. This includes: how each approach fits within the state, options, value construct; a general description of each methodology; example techniques; and their advantages and limitations.

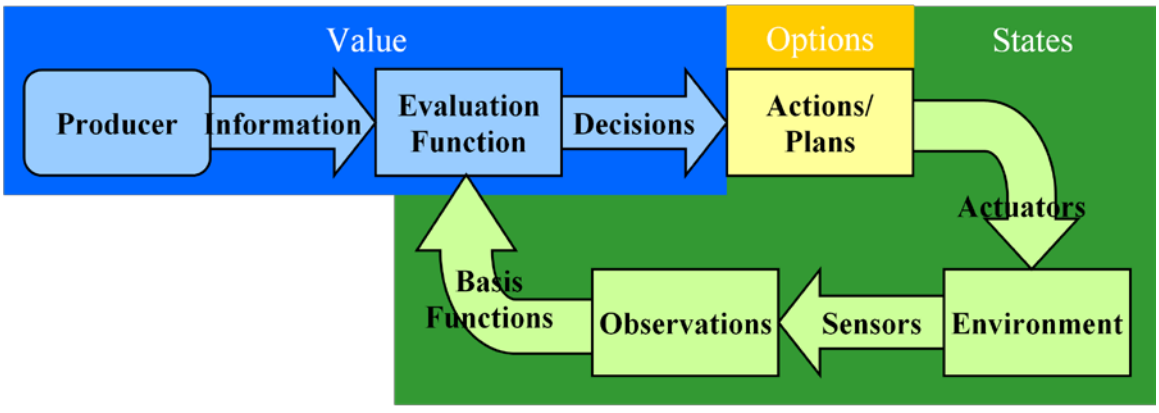


Figure 4.1. States, Options, Value Construct

VLPO&S planning problems by their nature cannot be solved deterministically, or without interactive feedback from the environment. Thus, attempted approaches require a methodology that includes a complete concept and cyclical framework, such as a state, options, value construct. The **state**, **options**, **value** construct is defined here, and can be represented within the block diagram shown in Figure 4.1. The first four parts of the model colored blue comprise the evaluation function. The second part is the actions/plans block in yellow, and the final five-part section represents the state in green. **Value** is calculated within the "Evaluation Function" block, **options** are selected within the "Actions/Plans" block, and **state** is represented within the "Observations" block. Note that value and options primarily represents the ADP&E solution space, while the state primarily represents the VLPO&S planning problem domain. This diagram is not universal, but is a simplified representation that captures the nature of the problem domain, and solution possibilities. This diagram shows the iterative nature of applying ADP&E solutions to VLPO&S planning problems, and is general-purpose enough to fit all three broad approaches.

The data-driven, expert-defined and goal-directed approaches are different in the way they make decisions. The significant difference among these approaches is in the value construct described next. As shown in Figure 4.2, depending on the approach chosen, the value is made up of different components. Expert-defined systems require experts' knowledge to define rules (discrete or fuzzy) and through the use of these rules, inferences are made about what actions to take [1]. Data-driven systems require data in the form of labeled quantities from both environmental observations and selected actions [2] [3]. Goal-directed systems have a goal that is measurable, either by reward or fitness

[4] [5] [6] [7] [8]. This value function is a policy for choosing future actions based on the desirability of future states or proximity to achieving goals.

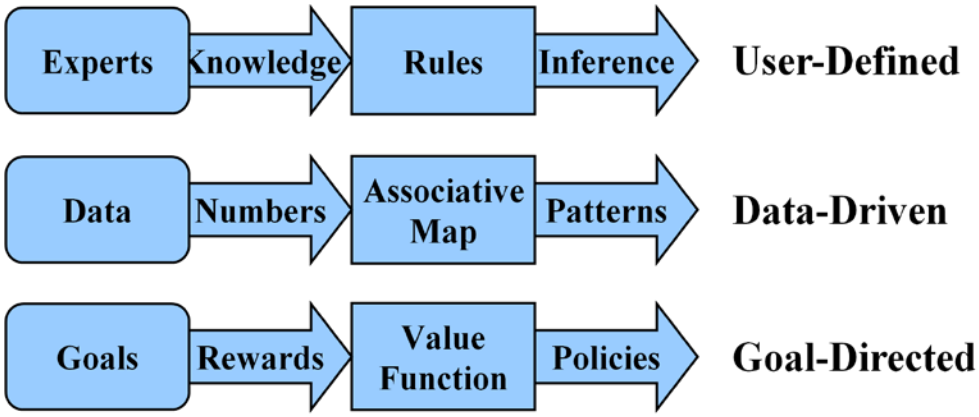


Figure 4.2. Evaluation Functions

Another alternative is a hybrid approach, where techniques are combined among the three approaches [2] [9] [10] [11] [12] [13] [14]. Next, we will describe these three approaches in detail, followed by a comparison of hybrid approaches.

### 4.1.1 Expert-Defined Approach

As introduced in Chapter 3, expert-defined approaches are techniques used to represent human knowledge, where the knowledge is entered by hand, organized in a tractable form, and accessed upon request. Representations for these systems are typically discrete rules [1], fuzzy-logic [9], or case-based [15], and are usually represented as decision-trees of if-then-else statements or finite state logic [16]. These systems work well on domain specific problems of limited size, however, they fail to handle the 'curse of dimensionality' and grow exponentially when many actions are considered at each time step and in succession [17]. Thus, expert methods cannot handle VLPO&S planning

problems of sufficient breadth necessary to associate a large number of simultaneous observations with action choices.

Another drawback of rule, fuzzy, and case-based approaches is their inability to adjust or adapt to circumstances unfamiliar to the expert embedding the knowledge. In other words, when the environment changes outside the prescribed domain, rules become useless and modifying these rules to fit the new situations is difficult [18]. There are many approaches that attempt to alleviate this deficiency, but they all still fail to handle the ‘curse of dimensionality’ [17].

### 4.1.2 Data-Driven Approach

As described in Chapter 3, data-driven approaches fall into one of two categories: supervised or unsupervised learning. Both of these categories are within the realm of *associative networks*, where data is linked based on its underlying characteristics. *Associative networks* represent a highly connected network or graph of symbols or qualitative nodes, which are either connected to the environment (i.e., inputs), to actions (i.e., outputs), or with each other (i.e., internal state representation). Supervised learning approaches clamp the output nodes (i.e., best known action) to learn the links between states and actions by back-propagating error into the network (i.e., back propagation networks (BPNs)) [12]. Unsupervised learning approaches use statistical metrics to learn connection strengths among the nodes to form classification clusters [19] or to do pattern recognition [20]. These associative or connectionist networks, such as statistical clustering, Bayesian networks (BNs), support vector machines (SVMs), BPNs, or

recurrent neural networks (RNNs), are built to mine large amounts of data, and to generalize (reason via induction) over these datasets.

In planning problems, these data-driven approaches can be used to handle many action choices (i.e., breadth) well by generalizing observations to actions. However, for very large problems, these approaches degrade as plans increase in sequential depth [21]. In other words, if a plan requires many sequential actions to complete a task or turn, a data-driven approach will fail to recognize each action's significance and optimize over a few actions without sufficiently considering the overall plan. For example, an RNN could be a viable approach for functionally approximating the next best action, and learn the connections to subsequent and preceding actions via back-propagation through time [5]. However, it has been shown that RNNs degrade in their results when generating multiple action choices without direct or full-state feedback from the environment [21]. Since we are dealing with VLPO&S planning problems, where the environment may not always be observable or have direct feedback after each action is executed, data-driven approaches cannot be considered in developing long-term plans.

Another drawback of a data-driven approach is its inability to improve over a supervisor. In other words, supervised approaches are trained via a priori knowledge, which may be biased, or the appropriate action may be unknown given current observations. For example, a BPN would be a viable approach for functionally approximating the next best action provided a supervisor is available to train the network [22]. However, the network would not improve over the supervisor's ability, nor explore alternative action choices that may lead to a better strategy in generating plans.

### 4.1.3 Goal-Directed Approach

By definition, goal-directed approaches assume a goal is known to exist, and that goals can be sought through a 'trial and error' process to achieve them. For some applications, many states have a perceived value (i.e., reward value) that can guide a planner to the end-goal, such as winning a game. For example, in chess, each board-piece has a perceived value if captured. In other applications, state values can be approximated by their relevance to other nearby states that do have value. For instance, when achieving a single goal state such as an endgame situation, backtracking through preceding states can lead to learning a path to that goal by distributing reward along the way. Note this is only the case if the goal state can be reached by reasonable chance in action selections [23]. For example, applying random actions in a backgammon game reaches the endgame with almost certainty [12]. In other games, such as Monopoly or RISK, there is no guarantee of a winner with random action choices. The five goal-directed approaches listed in Figure 4.3 are described below.

Depending on the size and scope of a problem, some goal-directed approaches are more useful than others. For example, when a planning problem is deterministic and of limited size, a dynamic programming (DP) approach has been shown to converge to an optimal strategy [5]. DP requires a perfect model of the environment, and an extensive search through the action space to converge on an optimal solution [5]. DP is heavily reliant on having a small set of action choices at each interval in time. For problems of great breadth (i.e., action choices are numerous at each instance), DP fails due to the

imposed computational burden. Problems this method solves well are deterministic games such as freecell, or Sudoku.

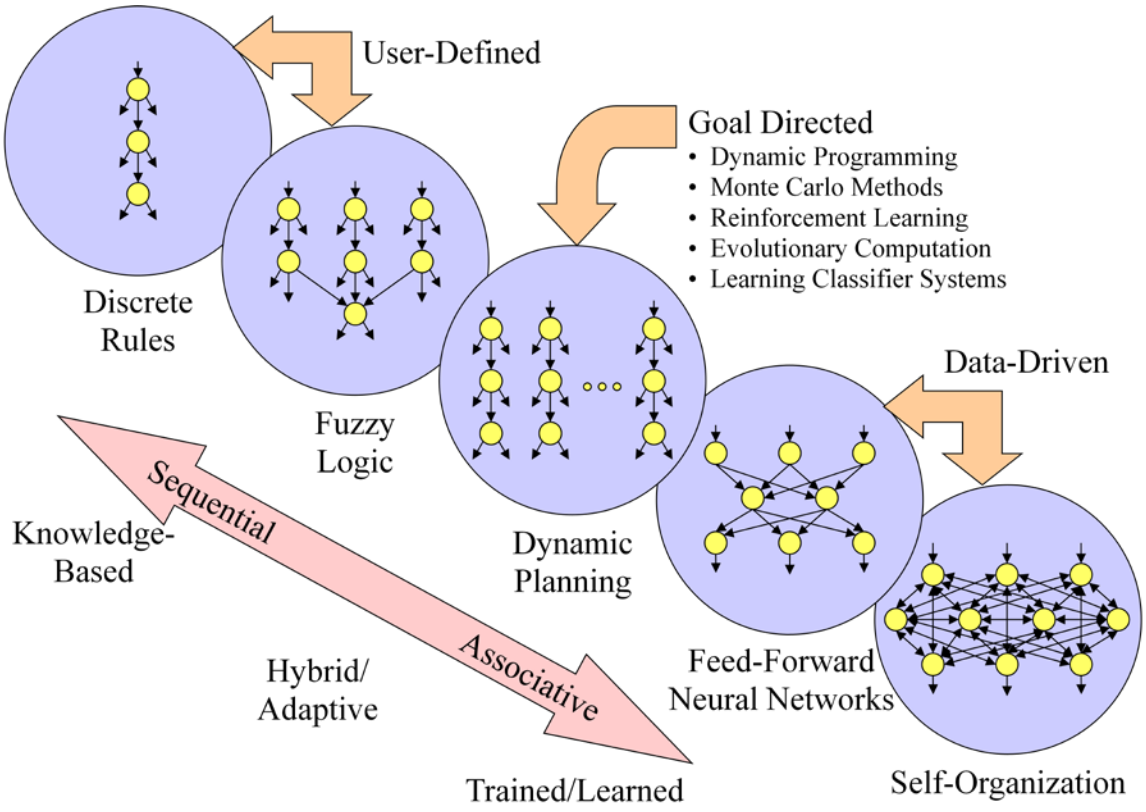


Figure 4.3. Summary of Three General Approaches

For planning problems that are larger in size and possibly stochastic, the DP approach has been modified into a reinforcement learning (RL) or Q-learning approach [5]. This approach does not require perfect knowledge of every state as in the DP approach, but it can still guarantee an optimal strategy if sufficient exploration is done on state-action pairs that lead to a goal [5]. RL algorithms, such as Q-learning, try to effectively balance the depth and breadth search of a problem by rewarding each action based on not only its immediate reward (i.e., current state-action pair), but also on the rewards near in temporal sequence to that action [13] [24]. This approach works well for

problems of great depth and breadth where the state space representation or set of action-choices is not too large. RL is often used in simpler planning problems, such as path planning for robots [25], and card games such as Blackjack [5]. If the state space is too large, where every state cannot be represented, a function approximator can be included to confine the search, as in the backgammon game [12]. This game example will be discussed in more detail later in this chapter. These RL approaches fail for very large problems, because they assume that a goal can be reached by random chance, and that enough state-action pairs can be encountered to learn the optimal strategy in a reasonable time.

Another goal-directed approach that does not depend on learning the value of state-action pairs is a Monte Carlo (MC) method. MC methods are very general statistical techniques [26]. MC methods have three basic processes: (1) define a set of options, (2) generate a random draw from this set, and (3) aggregate the results of the chosen options into the final result [27]. These three processes can be applied in principle to the planning problem as a goal-directed approach. A MC method can randomly select actions from a set of possible options and continue selecting until a plan is formed. For instance, if each action is selected based on equal probabilities, then it can be referred to as a ‘random walk’ [28]. Only after the entire plan is formed does the algorithm evaluate the resulting plan’s value, if a state can be evaluated. Several plans are usually generated using this method, because many will not reach a high valued state. These techniques are used in handling high dimensional problems with great breadth, such as the ‘Go’ game problem [29], [30] discussed later in this chapter. However, they often fail to converge on an optimal solution when the planning depth is great, because these methods do not

associate relationships well among near time action choices in the pursuit of a goal [5] as do RL methods.

Another goal-directed approach is the application of evolutionary computation (EC) or genetic algorithms (GA) to planning. EC and GA work well for problems of great depth and breadth, but require an explicit reward space for every possible state encountered [31], because unlike RL, EC and GA do not associate one state with the next. Instead, EC and GA concentrate on exploring the space more broadly by comparing sets of competing solutions (e.g., plans vs. single actions). The reward system used is called an evaluated state space and for many problems a fitness function can be prescribed for this purpose. However, these fitness functions are seldom optimal for they are usually expert-defined. In addition, since EC and GA methods require many offspring to compare and contrast, the methods scale linearly in their use of computational resources in proportion to the number of offspring, in comparison to RL methods. In other words, instead of learning from single plans, as RL does, EC and GA compare and contrast many plans. When the problem is of significant size, however, these methods can be made more efficient [31], because they do not need to converge on a value-function, and do not require a function approximator for high dimensional spaces when there are a great many possible states or action choices.

The balancing of exploration and exploitation to achieve a goal is fundamental and essential in designing an ADP&E solution. Learning Classifier Systems (LCS) methods attempt to balance exploration and exploitation by combining the RL methodology in exploiting existing rules, while using EC or GA methodology to explore

new rules that map states to actions [32], [33]. Both LCS and the Extended Classifier Systems (XCS) approaches divide each *prescribed condition* of a state-space into three categories: each condition has a positive, negative, or none impact within a rule. A *prescribed condition* is a binary feature of the state-space considered important by the 'trial and error' of attempting rule-action pairs. The actions that produce the greatest value over many trials become most strongly linked to the corresponding condition rule. Since this approach is similar in scope to RL problems, in the sense they map rule-action pairs as opposed to state-action pairs, LCS or XCS methods may suffer when the state-space and action choice sets are very large. XCS has been used in Poker [34], [35] and other applications [36] with some success. The poker example will be discussed in more detail later.

In summary, the five goal-directed approaches have particular advantages and limitations. (1) The advantages of DP methods are that they guarantee convergence, handle stochastic problems, are deterministic and tractable. The limitations of DP methods are that they require a perfect model of the environment, grow exponentially with action choices, and can handle only domain specific problems well. (2) The advantages of RL methods are that they handle both large depth and breadth search, include incremental learning, handle stochastic problems, are deterministic and tractable, and can incorporate a temporal difference factor that links rewards over time. The limitations of RL methods are that they do not scale well for very large problems due to the size of state-action pairs. (3) The advantages of Monte Carlo methods are that they require limited a priori knowledge, require only sample transitions and are non-deterministic yet tractable. The limitations of Monte Carlo methods are that they require

much exploration, their convergence properties are not well-defined, and they cannot handle associative tasks well. (4) The advantages of EC methods are that they can search very large combinatorial spaces, can handle stochastic problems, and are tractable [4]. The limitations of EC methods are that they require expert knowledge in the form of prioritized goals (a fitness function), require exploitation and exploration parameters that are domain specific [5], and do not take into account state transition probabilities, thus results can be overly optimistic and consequently not risk averse (defined in Chapter 2). (5) The advantages of LCS methods are that they can handle large depth and breadth problems and learn a small set of rules that link states to actions. The limitations are that they are not well understood mathematically and have not been proven contributors to very large problems due to the size of rule-action pairs.

Goal-directed approaches have been used to bridge the gap between having either too much information from environmental sensors [37] (data-driven) or too biased information from domain experts (expert-defined). Figure 4.3 provides a summary of the architectures, technologies and relationships among the three general approaches. Expert-defined approaches handle sequential tasks well, but cannot handle large breadth spaces or their associative properties well. They rely on human expertise that is often incomplete and biased, and fail to address scalability or the curse of dimensionality. Data-driven solutions handle associative tasks well, but degrade when the sequential depth of the task becomes too long to integrate without providing prescribed guidance, such as a supervisor. Also, data-driven approaches cannot learn beyond the supervisor. Goal-directed efforts aim at finding a compromise among these approaches in an attempt to better provide more flexible ADP&E solutions to VLPO&S planning problems.

## 4.2 Hybrid Approaches

By definition, hybrid approaches are methodologies that use two or more technologies to achieve an objective. In attempting to solve VLPO&S planning problems, applying two or more techniques can have advantages. One way to understand the advantages of using multiple techniques is through the study of game problems that reflect some of the VLPO&S planning problem characteristics. As opposed to real world problems that have many different possible interpretations of the problem, most game problems are well defined and thus, can establish a better baseline in measuring results than real world problem solutions. In other words, in real world problems, the implementation is often highly tied to the planning approach taken [38] [39], and in game problems, the rules-of-the-game and problem interpretations are pre-defined. Thus, game problems are often independent of the planning solution, and thus, their solutions are more easily compared.

There are many games that can be considered as planning problems. Five game problems studied heavily over the past few decades are Chess, Checkers, Backgammon, Go, and Poker. These game problems vary in their number of players, state space size, action choices, observability, and uncertainty of outcomes. All these factors play a role in choosing an appropriate planning strategy to win a game. We will examine these five games by describing their characteristics; the hybrid approaches used in attempting to solve them, and the limitations of these approaches in solving VLPO&S planning problems.

Chess is a two-player game where the number of possible states exceeds $10^{40}$, thus no computer could ever plan all the moves in a game until the endgame is near. The action-choices are at most dozens at each instance, each action outcome is deterministic, and the game is full-state observable. Chess has been solved to a great extent through combining expert-defined value functions with decision-tree minimax search [40], such as used in the chess-playing computer, Deep Blue [11], [41]. These techniques can now beat Chess Masters, and are getting better as computers increase in processing power (i.e., search speed) and memory (i.e., search capacity) [42]. The techniques used in chess, however, cannot be directly applied to VLPO&S planning problems, because chess relies on a minimax search tailored for two-player games, is full state observable, and has no element of chance. Also, we are concerned with planning that improves through play and the best chess approaches rely on search using hard and fast rules [42].

Checkers is another two-player game with even fewer action choices, deterministic outcomes, and is full-state observable. Note that the state-space of checkers is much smaller than chess. Checkers has been solved to a certain extent by applying an artificial neural network (ANN). With EC, a more effective player evolves over many trials of playing the game [43], [44]. Even though evolving an ANN over many trials does provide a good exploration of the space, it has not yet been proven effective over problems where the input (states) and output (actions) combinations are far greater than checkers. Also, the checkers game can now be solved perfectly, given enough computer resources [45]. Thus, given its limited size and scope, checkers' solutions cannot be considered for VLPO&S planning problems.

Backgammon is a two-player game where the state-space exceeds $10^{20}$. As in chess and checkers, the game has few action choices, and is full-state observable [46]. However, the game is not fully deterministic due to a required two dice roll at each turn. Games such as backgammon have been solved to a great extent through a goal-directed temporal difference RL approach. Through competing against itself, a BPN can be trained over the course of many games [47]. However, backgammon does not have a very large set of action choices at every instance, and does not require planning actions beyond a single turn as in chess. Thus, this backgammon approach cannot be applied to a VLPO&S planning problem in its current form.

Go is another two-player, full-state observable, and deterministic game. However, the number of possible states far exceeds chess [48], and Go has both great breadth and depth of action-choices [49]. In addition to posing this difficult search problem, Go has been resistant to learning an evaluation function that can map progress of winning a game, as in chess [50]. In other words, chess has a well-defined value for each board piece, while in Go, the value of one position over another is not as well established as in chess. Also, given the number of possible action choices at each turn, one cannot search as deep as chess using classical tree search methods. Thus, probabilistic tree-search approaches have been used such as MC. MC methods have had some success in playing Go [51]. MC's ability to explore large action sets has been shown to be beneficial and is incorporated into our approach as described in Chapters 7 and 8. The hybrid MC methods used for Go do consider a very large state-space, but do not address partially observable or uncertain outcomes.

Poker is a multiplayer game that has both a partially observable state and uncertain outcomes. The number of action choices is few at each instance (call, raise, or fold), but the total number of possible states is very large [52]. Given the size and scope of this problem and the variability in opponent play, learning an opponent can be beneficial in game play [53] [54]. For instance, someone playing against a conservative poker player tends to fold when the conservative player raises, knowing the player probably has a good hand. One approach attempting to solve computer poker uses a combination of SVMs with LCS [32]. This approach attempts to optimize each action choice, but does not consider an overall planned strategy of winning a tournament. As in most computer poker programs [34] [35], optimizing play over a hand limits the search to a small subset of play, rather than learning an overall strategy of winning a tournament. This approach is a good first step toward achieving a good player, however, it has not yet been scaled to planning beyond a single hand.

There are many other games and other real world problems that could be considered. For this dissertation, we consider a game problem, RISK and a real world mission problem, urban search and rescue operation (US&RO). RISK is a multiplayer game problem that has many actions per turn, and many actions have stochastic outcomes [55]. Unlike backgammon, rolling the dice occurs throughout the execution of many actions, rather than just at the beginning of a turn. Our US&RO mission problem is a single player game with a partially observable state-space. Both RISK and US&RO have very large number of action choices and state-spaces. Chapters 7 and 8 will describe them in detail, respectively.

For VLPO&S planning problems, such as RISK or US&RO, one cannot focus solely on any single technique to solve these problems well. For instance, (1) having some endgame motivation such as is used in solving chess; (2) an exploration capability as used in the checkers' solution; (3) a reward feedback mechanism such as used in backgammon; (4) an action choosing mechanism as used in Go; and (5) an adaptive planning ability as shown in Poker provides a better overall approach than applying any of these techniques individually. Applying all of these techniques to all planning problems is not the answer we seek, but determining when to apply which method based on the planning problem characteristics is one of the goals of this dissertation.

**References:**


1. Wolf, "A rule-based approach to dynamic constraint satisfaction problems," *Proceedings of the 12th International Conference on Applications of Prolog*, 1999.

2. Kenneth N. Ross & Ronald D. Chaney, "Neuro-dynamic programming for adaptive fusion complexity control," *SPIE Conference on Sensor Fusion: Architecture, Algorithms, and Applications III*, April 1999, *V. 3719*, pp. 398-409.

3. G. Tesauro, "Programming backgammon using self-teaching neural nets," *Artificial Intelligence, V. 134*, 2002, pp. 181-199.

4. L. P. Kaebling, M. L. Littman & A. W. Moore, "Reinforcement learning: a survey," *Journal of Artificial Intelligence Research*, V. 4, 1996, pp. 237-285.

5. R. S. Sutton & A. Barto, *Introduction to Reinforcement Learning*, MIT Press / Bradford Books, Cambridge, MA, 1998.

6. Aluizo F. R. Araujo & Arthur P.S. Braga, "Reward-penalty reinforcement learning scheme for planning and reactive behavior," *IEEE International Conference on Systems, Man and Cybernetics, V. 2*, 1998, pp. 1485-1490.

7. W.M. Spears, "The role of mutation and recombination in evolutionary algorithms," *Ph.D. Dissertation: George Mason University*, 1998.

8. D.B. Fogel "The advantages of evolutionary computation," *Bio-Computing and Emergent Computation* 1997, D. Lundh, B. Olsson, and A. Narayanan (eds.), Sköve, Sweden, World Scientific Press, Singapore, pp. 1-11.

9. O. Cordon, F. Herrera & P. Villar, "Generating the knowledge base of a fuzzy rule-based system by the genetic learning of the data base," *IEEE Transactions on Fuzzy Systems, V. 9*, N. 4, August 2001, pp. 667-674.

10. X. Yao, "Evolving artificial neural networks by evolutionary algorithms," *IEEE Proceedings on Computational Intelligence*, September 1999, pp. 1-45.

11. R. Levinson, F. H. Hsu, J. Schaeffe, T. A. Marsland, & D. E. Wilkins, "The role of chess in artificial-intelligence research," *ICCA Journal*, V. 14, N. 3, 1991, pp. 153-161.

12. G. Tesauro, "Temporal difference learning and TD-Gammon," *Communications of the ACM*, V. 38, N. 3, 1995, pp. 58-68.

13. D.F. Beal, M.C. Smith, "Temporal difference learning for heuristic search and game playing," *Information Sciences V. 122*, 2000, pp. 3-21.

14. J.L. Shipiro and J. Wearden, "Reinforcement learning and time perception – a model of animal experiments," *Advances in Neural Information Processing Systems, V. 14*, 2002.

15. An, A. and N. Cerone. 1998. Multimodal reasoning with rule induction and case-based reasoning. In Multi-modal Reasoning: Papers from the 1998 AAAI Spring Symposium. AAAI Press, Menlo Park, CA pp. 118-123.

16. M. L. Littman M. Kearns and S. Singh, "An efficient, exact algorithm for solving tree-structured graphical games", *Advances in Neural Information Processing Systems, V. 14*, 2002.

17. H.J. van den Herik, J. W.H.M. Uiterwijk & J. van Rijswijck, "Games solved: now and in the future," *Artificial Intelligence V. 134*, 2002, pp. 277-311.

18. Gil, Y., and Tallis, M. 1997. A Script-based Approach to Modifying Knowledge-based Systems. In Proceedings of the Fourteenth National Conference on Artificial Intelligence.

19. Richard O. Duda, Peter E. Hart, David G. Stork: *Unsupervised Learning and Clustering*, Ch. 10 in *Pattern classification* (2nd edition), p. 571, Wiley, New York, ISBN 0-471-05669-3, 2001.

20. S. Kotsiantis, P. Pintelas: *Recent Advances in Clustering: A Brief Survey*, WSEAS Transactions on Information Science and Applications, Vol 1, No 1 (73-81), 2004.

21. Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, 5(2), 1994, pp. 157-166.

22. S. Kotsiantis, Supervised Machine Learning: A Review of Classification Techniques, Informatica Journal 31 (2007) 249-268.

23. Heylighen F. (1992): "Principles of Systems and Cybernetics: an evolutionary perspective", in: Cybernetics and Systems '92, R. Trappl (ed.), (World Science, Singapore), p. 3-10.

24. M.L. Littman, "Value-function reinforcement learning in Markov games," *Journal of Cognitive Systems Research, V. 2*, 2001, pp. 55-66.

25. K. Deb, D.K. Pratihar and A. Ghosh, "Learning to avoid moving obstacles optimally for mobile robots using a genetic-fuzzy approach," *PPSN 1998*, pp. 583-592.

26. Metropolis, N.; Ulam, S. (1949). "The Monte Carlo Method". *Journal of the American Statistical Association* **44** (247): 335–341.

27. Robert, C. P.; Casella, G. (2004). *Monte Carlo Statistical Methods* (2nd ed.). New York: Springer.

28. William Feller (1968), *An Introduction to Probability Theory and its Applications* (Volume 1), Chapter 3.

29. Bruegmann, B. (1993). Monte Carlo Go.

30. Bouzy, B. and Cazenave, T. (2001). Computer Go: an AI oriented survey. *Artificial Intelligence*, Vol. 132, pp. 39–103.

31. C. R. Stephens, "'Effective' fitness landscapes for evolutionary systems," *Proceedings of the Congress on Evolutionary Computation, V. 1*, 1999, pp. 703-714.

32. John H. Holland. Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In Mitchell, Michalski, and Carbonell, editors, *Machine learning, an artificial intelligence approach. Volume II*, chapter 20, pages 593-623. Morgan Kaufmann, 1986.

33. Kamran Shafi, Tim Kovacs, Hussein A. Abbass, Weiping Zhu, Intrusion Detection with Evolutionary Learning Classifier Systems. *Natural Computing*, 8(1), pp. 3–27. March 2009.

34. Smith, Stephen, F., Flexible learning of problem solving heuristics through adaptive search, IJCAI Conference on Artificial Intelligence, Proceedings of the Eighth International Conference on Artificial Intelligence Volume 1. Karlsruhe, Germany, 1983.

35. Billings, D., Davidson, A., Schaeffer, J., and Szafron, D. (2002). The challenge of poker. *Artificial Intelligence*, Vol. 134, pp. 201–240.

36. Kamran Shafi, Tim Kovacs, Hussein A. Abbass, Weiping Zhu, Intrusion Detection with Evolutionary Learning Classifier Systems. *Natural Computing*, 8(1), pp. 3–27. March 2009.

37. A.R. Mccallum, "Learning to use selective attention and short-term memory in sequential tasks," *From Animals to Animats, Fourth International Conference on Simulation of Adaptive Behavior*, (SAB), 1996.

38. S. Chien, R. Hill Jr., X. Wang, T. Estin, K. Fayyad, and H. Mortenson. "Why Real-Worls Planning is Difficult: A Tale of Two Applications," *Jet Propulsion Laboratory Technical Report*, Oct 1995.

39. R.S. Aylett, G. J. Petley, P.W.H Chung, B. Chen, and D.W. Edwards. "AI Planning: Solutions for Real World Problems," Knowledge-Based Systems, Volume 13, Issues 2-3, April 2000, Pages 61-69.

40. T. Taipale, S. Pieska, & J. Riekki, "Dynamic planning based control of autonomous machine," *Proceedings on Real-Time Systems*, 1991, pp. 120-126.

41. H.J. van den Herik, J. W.H.M. Uiterwijk & J. van Rijswijck, "Games solved: now and in the future," *Artificial Intelligence V. 134*, 2002, pp. 277-311.

42. D. Rasskin-Gutman. "Chess Metaphors: Artificial Intelligence and the Human Mind," MIT Press. December 2009.

43. K. Chellapilla and D.B. Fogel, "Anaconda defeats Hoyle 6-0: a case study competing an evolved checkers program against commercially available software," *Proceedings of the 2000 Congress on Evolutionary Computation*, IEEE Press, Piscataway, NJ, 2000, pp. 857-863.

44. D. Fogel. "Blondie24: Playing to the Edge of AI," Morgan Kaufmann Publishers, Inc. San Francisco, CA.

45. J. Schaeffer, N. Burch, Y. Bjoernsson, A. Kishimoto, M. Mueller, R. Lake, P. Lu, S. Sutphen. “Checkers Is Solved,” *Science*, September 2007. Vol. 317 no. 5844, pp. 1518-1522.

46. B. Robertie. “Backgammon for Winners (Third Ed.),” *Cardoza*, 2002.

47. G. Tesauro. “Programming Backgammon Using Self-Teaching Neural Nets,” *Artificial Intelligence 134 (1)* 2002. pp. 181-199.

48. S. Tzu translated by Ralph D. Sawyer. “The Art of War,” *Barnes & Nobles*. 1994.

49. N. Yoshiaki. “Basic Techniques of Go,” Kiseido Publishing Company.

50. X. Chen, D. Zhang, X. Zhang, Z. Li, X. Meng, S. He, and X. Hu. “A Functional MRI Study of High-Level Cognition: II. The Game of GO,” *Cognitive Brain Research*, Volume 16. Issue 1, March 2003, pp. 32-37.

51. B. Bouzy and B. Helmstetter. “Developments on Monte Carlo Go,” *Advances in Computer Games 10*, 2003.

52. R. Brenner and G. Brenner. “Gambling and Speculation: A Theory, a History, and a Future of some Human Decisions,” *Cambridge University Press*, 1990.

53. A. Davidson, D. Billings, J. Shaeffer, D. Szafron. “Improved Opponent Modeling in Poker,” *Proceedings of the 2000 International Conference on Artificial Intelligence*, 2000. pp. 1467-1473.

54. A. Davidson, D. Szafron, R. Holte, and W. Pedrycz. “Opponent Modeling in Poker: Learning and Acting in a Hostile and Uncertain Environment,” 2002.

55. J. Vaccaro, C. Guest, “Evolutionary Bayesian Network Dynamic Planner for Game RISK,” *Lecture Notes in Computer Science: Applications of Evolutionary Computation, EvoSTOC Proceedings*, Coimbra, Portugal, April, 2004, pp. 549-560.

# Chapter 5

# New Challenges

## 5.0 Abstract

In an effort to solve VLPO&S planning problems with ADP&E solutions, many new challenges arise beyond the methods discussed in Chapter 4: Related Work. Research in planning architectures has not addressed extensibility across a broad range of VLPO&S planning problems or do not consider an ADP&E solution. Current research does not formulate VLPO&S planning problems as an extension to well-defined POMDP models. Currently, top-down ADP&E planning approaches use extensive knowledge engineering [1] [2] [3], and the inclusion of bottom-up planning approaches does not consider the size and scope of automating VLPO&S planning problems [4] [5]. In addition, while using machine learning for training ADP&E planners, current approaches do not address the size and uncertainties associated with VLPO&S planning problems [6] [7] [8].

The purpose of this chapter is to uncover and define the new challenges imposed by investigating VLPO&S planning problems with ADP&E solutions. Each identified challenge represents an aspect of planning that previous research has not addressed adequately, if at all. The challenges can be divided into four main areas: general characterization and extensibility challenges in handling a variety of VLPO&S planning

problems with an ADP&E solution framework, specific VLPO&S environmental challenges, specific ADP&E planner challenges, and general learning challenges. The next four sections describe extensibility, environment, planner, and learning challenges, respectively.

## 5.1 Characterization and Extensibility

As described in Chapter 2: Problem Domain, the ADP&E approach employs two essential functional objects: an environment and a planner. These two objects have a wide variety of attributes, depending on the application at hand. To study the flexibility of a proposed approach, both the environment and the planner are divided into different aspects to help define the scope of the problem domain. These aspects help define a planning problem in terms of its characteristics and extensibility. In this dissertation, two applications are studied that span across these aspects to ensure the developed approach has a broad applicability to many problems. Next, many characterization terms are introduced to establish a more extensible approach for both the environment and the planner. Chapters 6 through 9 describe the use of these characteristics to aid in designing approaches tailored to specific and new applications.

For the environment, each application has different: (1) *planners required*, (2) *breadth*, (3) *depth*, (4) *model observability*, (5) *action response*, and (6) *goal evaluation*. The *planners required* are the number of competing and/or cooperating planners in an application environment. The *breadth* is the number of considered action choices available for a given state, player, and agent. The *depth* is the length of a plan and refers to the number of actions or tasks required within a plan. The *model observability* refers to

whether the real model is partially or full-state observable. The *action response* refers to whether actions have stochastic or deterministic outcomes. The *goal evaluation* refers to whether the application is planned within a cooperative or non-cooperative environment. The challenge here is to examine how these various dimensions impact the design of planning approaches.

For the planner, each application has different: (1) *planning complexity*, (2) *planning coordination*, (3) *action processing*, (4) *required agents*, (5) *model update frequency*, (6) *outcome selection*, and (7) *control parameters*. *Planning complexity* refers to whether the application has multiple tasks or turns to complete an entire game or real world simulation. Single task games, such as solitaire, freecell or sudoku, are considered trivial, because the search spaces are small and deterministic, and are not studied here. *Planning coordination* refers to whether the application's control is centralized by the planner, or decentralized, where the agents have their own decision-making abilities and often plan their own actions. The approach taken in this dissertation only considers the centralized planning problem. Having decentralized or distributed decision-making is an interesting addition to the problem domain studied, and is a topic of future work described in Chapter 10. *Action processing* refers to whether the outcomes of actions are processed off-line (e.g., stored as a look-up table), or calculated on the fly. This dissertation concentrates on plans that have many action sets, so to save processing time, all trivial single action choices are preprocessed offline so outcomes can be looked-up quickly during simulation. *Required agents* refer to whether the application has single or multiple agents. *Model update frequency* refers to how often the internal planning model is updated with observations from executed actions. The update frequency could be

continual (i.e., after each executed action), but considering the cost of planning, a more intermittent update frequency often proves more valuable, depending on the application. *Outcome selection* refers to whether an outcome is selected via classical methods, such as expected value or maximum likelihood, or through a risk-averse strategy. The risk-averse strategy is described in the Chapter 6: Approach, and application of the strategy is described in Chapter 7: RISK Game Application. *Control parameters* for planning systems are often static, such as fixed search strategy or value function. This is a trivial case. The challenge is to study the situation where the control parameters are dynamic and adaptive.

## 5.2 VLPO&S Domain Problems

Planning for large complex problems cannot be handled by POMDPs as currently defined [9]. In this dissertation, the focus is on six features currently limiting POMDPs: (1) *planning strategies*, (2) *plan lengths*, (3) *planning costs*, (4) *outcome selection*, (5) *probability of successful plan completion*, and (6) *expectations and re-planning*. These limitations are addressed by describing challenges for each feature. In later chapters, these challenges will be further addressed with well-defined terms, algorithms, and results. Terms and algorithms are detailed in Chapter 6: Approach. Results are shown for two applications in Chapters 7 and 8. In handling large complex problems, challenges are surmounted to extend the current POMDP model to include these new features for future planning applications.

*Planning strategies* refers to choices that allow the planner to vary its ability in selecting actions and/or plans dependent on the planning problem. Planning strategies are

not currently defined as a variable choice in POMDP models. This new variable choice raises many challenges. One challenge is deciding whether to plan ahead single or multiple actions. Another challenge is determining how dependent plans are on environmental changes. A third challenge is determining the dependency of one agent's plan on another agent's plan. A fourth challenge is determining how to adapt the planning strategy based on environmental observations.

*Plan lengths* refers to the number of turns or tasks used. Plan lengths are dependent on the type of application, and extend a POMDP model to the planning domain by addressing additional challenges. One challenge is determining what constitutes a plan's progress, such as a completed task or turn in a game. Another challenge is determining the number of actions required to complete a task or turn. A third challenge is determining what is a feasible number of tasks or turns to plan ahead, given the cost of planning, and uncertainty of the environment.

*Planning costs* are used in plan-generation and plan-execution, and refer to the search time required in planning and/or the logistics required to support planning, such as communication delays. Planning costs also extend a POMDP model to the planning domain by addressing additional challenges. The first challenge is determining whether there is a logical place to include planning costs, such as in the generation and/or execution of a plan. Another challenge is determining if there are real world logistical costs that need to be modeled. A third challenge is determining the amount of cost to include in searching for a new plan, or the cost of communicating information among agents on new plans or new observations. A fourth challenge is accessing the impact of

the costs of planning on choosing other variable choices, such as planning strategy or plan lengths.

*Outcome selections* refer to the functions used in predicting an outcome when there are multiple possibilities. In a POMDP framework, outcome selections are a function of the state transition probabilities. However, a POMDP can be extended to include a selection criteria based on risk instead of maximum likelihood [10] or expected value [11]. If there is more than one possible outcome, then there are several challenges to consider when selecting a specific outcome. One challenge is how to rank order the outcomes. Another challenge is how to select among the ranked outcomes. A third challenge is formulating a way to adapt the selection based on perceived risk or overall success of the planner.

The *probability of success* is the likelihood that a plan will succeed as expectations are defined. The probability of success for plan completion is not currently handled in POMDPs. A POMDP can include probability of success by addressing a few challenges. One challenge is determining what constitutes a successful action or plan. Another challenge is determining how to calculate the probability of successful plan completion. A third challenge addresses how to use the probability of success to limit the depth of search. A fourth challenge is determining how probability of success weighs in plan selection and/or assessment, and how to make this weight adjustable.

*Expectations and re-planning* refer to the variable choice of keeping an existing plan or re-planning. Expectations and re-planning are not functions in a POMDP framework. Several challenges must be addressed to allow for variable choices in using

expectations and re-planning. One challenge is determining how to balance the cost of re-planning with the risk of continuing execution of a preexisting plan. Another challenge is determining a good measure of expectations and how it relates to achieving goals. A third challenge is determining an expectation acceptance threshold metric for variable triggering of re-planning. A fourth challenge is determining a method for adapting this acceptance metric.

## 5.3 ADP&E Solution Space

Central to an ADP&E solution is the planner, and all the planner parameters used to orchestrate the planning process. A planner might be defined specifically for each application, but this would lead to special purpose algorithms, not useful to other applications. The challenge is to design an ADP&E approach that can be used for many VLPO&S planning problems and can improve its ability to plan. Thus, the planner parameters must be identified to vary planner control to provide enough flexibility to handle a variety of planning problems. There are modifiable controls in nine major categories: (1) *choosing individual actions*; (2) *searching and comparing plans*; (3) *application dependent planning depth*; (4) *risk aversion*; (5) *success probability*; (6) *temporal cost of planning*; (7) *evaluating plans*; (8) *diversity in plan selection;* and (9) *justifying system behavior*.

First, in *choosing individual actions*, consider the balance between exploring new options versus exploiting proven ones. This has long been a difficult problem in machine learning. For planning, this challenge remains, and there are several considerations. They include whether the application is stochastic or deterministic, whether the application is

partially observable or has full-state feedback, whether there are few or many actions at each time step, whether plans are for many agents or a single agent, and whether the state space is traversable or too large to consider every state. Planning problems that are small, deterministic and have full-state feedback are not considered here, because they have trivial solutions. When considering individual action choices, one challenge is to look at a variety of different types of planning problems, and to develop a control scheme that is useful and adjustable to solve a broad set of problems.

Second, in *searching and comparing plans*, consider the cost of search versus the risk and benefits of executing the resulting plans. Whether an environment is deterministic or not, or has full-state feedback or not, could determine whether search is an important part of planning. There may be circumstances where the cost of search is too large to be included, or circumstances where search is cost effective and performed not only before a plan is executed, but while it is executed. Also, if search is used, there are a large variety of search algorithms to choose from, and some may prove more valuable than others, depending on the application. When searching and comparing plans, one challenge is to examine different search approaches to determine their value in planning, based on efficiency (speed of finding a solution) and effectiveness (solution quality).

Third, in *application dependent planning depth*, consider whether to approach the problem from a resource consumption perspective, or in terms of completing tasks. Depending on the application, one approach may be more successful than the other. For instance, in competitive games, such as turn games (Chess, Backgammon, Go, Poker, RISK, etc.), resource monitoring may be a better measure of planning progress. In more

cooperative planning problems, where a shared responsibility among agents is important, a task-oriented scheme may be more feasible, such as real world humanitarian missions (China 2008, Haiti 2010, etc.). When considering application dependent planning depth, one challenge is to examine planning depth from both perspectives, and determine which type of planning depth metric proves more valuable for different application types.

In considering stochastic problems, *success probability* plays an important role in determining a plan's value and chance of completion. One challenge is to examine how to combine the probability of success for all actions and their impact on the overall value of a plan. Also, probability of success may play an important role in reassessing a plan as it is executed. A plan's success is an important component in whether a planner re-plans or not. When considering a plan's success, one challenge is to determine how to calculate probability of successful plan completion and how to assess its impact on a plan's value.

*Risk aversion* can play an important role in two areas: outcome selection and expectation assessment. When a planning problem is stochastic, selecting outcomes based on risk must be considered. Choosing the best value of risk aversion for outcome selection may depend on the application, and on the state of the model. In assessing expectations, risk (i.e., an acceptability threshold) is important in determining whether to keep a plan or re-plan. One challenge is to design and implement mechanisms to monitor and adapt risk for both outcome selection and expectation assessment.

In determining how one plans, when one plans, and how often one plans, a planner needs to consider the *temporal cost of planning*. Temporal costs are considered in two ways. There are the computational costs in searching through plans, and there are

logistical costs in planning for real world problems. For search, there are computational costs that refer to time or resource allocations. For instance, in some board games (e.g., Chess, RISK, Go), a planner could have limited time to search for the best plan [12] [13] [14]. In addition, time could be split among: initial planning, planning during execution (continual planning case), and re-planning (when plans do not proceed as expected). In real world applications, a human-in-the-loop is often used, and may take time to commit computer generated plans, such as contacting agents in the field as a dispatcher coordinates drivers. These communication lags as well as other logistical costs are involved in many real-world planning problems. One challenge is to examine temporal costs as an integral part of planning, and to learn improved control strategies to counter these costs.

In *evaluating plans* for selecting the best ones over the course of an entire simulation or game, the final goal of the planner (e.g., winning the game) is not always a good metric in choosing the best plans. For instance, in nontrivial games, winning a game takes many turns, and in some of those games, an endgame may not be reached through random actions (e.g., RISK, Axis and Allies, Diplomacy) [13] [15] [16]. In these games, as in real world planning applications, it is necessary to guarantee reaching the endgame or end-state. Sub-goals or action-choice conditions can be a way of avoiding this “no-progress” problem. Sub-goals and action-choice conditions, as defined in Chapter 2, refer to features of state that measure progress of projected plans of action that lead the planner toward the endgame. Initially, sub-goals or action-choice conditions are not necessarily good qualifiers for choosing the best actions, but subject matter expertise (SME) can be used to choose sub-goals or action-choice conditions that may or may not be important in

planning. Also, in choosing the best actions, one should avoid restricting action choices purely based on SME. Thus, one of the challenges is choosing and adapting sub-goals or action-choice conditions to avoid this “no-progress” problem, and to improve the planning strategy through experience, while ensuring that a large portion of planning options is explored independent of SME.

In the case when plan search is deemed important, a planner may consider whether only to exploit the best projected plans, or to explore other plans that are more diverse than these best plans. For example, when one considers more than one plan, exploiting only the best plans could lead to a local minimum based on restricting the search area. In allowing for *diversity in plan selection*, the planner may be more likely to pursue other options while retaining the best plans. Another challenge is to include plan diversity as a control mechanism in applications that require search, and to allow for variability in plan selection based on long-term plan success.

Finally, *justifying systems behavior* for an ADP&E system can be determined in seven ways as described in Chapter 2: outcome-selection justification, action-selection justification, plan-assessment justification, plan-fitness justification, planning-cycle justification, planner-utility justification, and mission-value justification. Due to the complexity of VLPO&S planning problems, these justifications are interdependent and span across many aspects of planning. In justifying system behavior, the challenge is to adequately identify, define, and converge the parameters associated with each of these seven justifications with regard to the planning problem. In other words, justifying system behavior requires traceability of choosing value function metrics for all associated

parameters, because ADP&E systems require justification before deployment. The parameters associated with the seven justifications are application dependent and are described in more thoroughly in Chapters 7 and 8.

## 5.4 Learning Challenges

Planning problems for very large, partially observable, stochastic environments are difficult to solve for four main reasons: (1) balancing exploration for new information vs. exploitation of pre-existing information; (2) providing a scalable framework, where the speed of the simulation is minimally affected by the number of agents or actions available; (3) determining where to assign credit in the action-response domain, especially for long action sequences; and (4) providing traceable results that justify each planned and executed action.

There are many concerns in balancing exploration versus exploitation. First, exploring action options should initially be done in an unbiased manner. Second, choosing between searching entire plans versus making decisions for each action should be planning problem dependent. If one chooses search, how much is necessary, and how often should it be done? Also, how many searched plans are necessary for comparison? Next, when considering problem observability, one needs to consider the impact on plan length and model update frequency. Finally, when considering strategies for adapting planners' parameters, one needs to balance between exploiting and exploring control parameters. For example, how are parameters modified, how often do they change, and how many games or simulations is enough to determine adequate planner fitness? Challenges concerning exploration versus exploitation will be addressed in the context of

one stochastic and another partially observable planning problem described in Chapters 7 and 8, respectively.

The curse of dimensionality is an important problem when designing and implementing a scalable framework. There are several characteristics to consider when designing and implementing a scalable architecture: (1) limiting assumptions by creating a simplified architecture with only necessary and sufficient parts increases usefulness across multiple applications; (2) limiting fixed parameters to a small, required set allows for an adaptable and trainable implementation; (3) limiting required SME minimizes humans' biased constraints; (4) preprocessing of individual action responses speeds up the ability to simulate planning; (5) requiring the architecture's hierarchy to be modular and extensible allows one to include or preclude functionality deemed important for each particular application; and (6) ensuring the hierarchy has an object-oriented design allows for easy modification of dependent planning processes, such as multithreading agents. This dissertation describes our efforts to include these six characteristics to meet the challenge of having a scalable framework.

Attempting to solve the credit assignment problem for planning tasks presents many difficulties. First, given the variation in planning problems, such as cooperative versus non-cooperative environments, deterministic versus stochastic outcomes, full-state versus partially observable state feedback, and single agent versus multiple agent tasking, one needs a credit assignment strategy that considers all these domains. Second, one needs to include the features of the environment deemed as important factors in developing a good plan. Third, one needs to investigate where, when, and how often to

assign credit to these factors. One must avoid decoupling planners' parameters, which compete and adapt together, at different levels within the hierarchy. For instance, when a planner contemplates and executes a plan, there must be feedback from both the environment and the game or simulation results, which can propagate to one common level in the hierarchy (i.e., planner, described in Chapter 6). Finally, the architecture should restrict hierarchical dependency to only one layer above and below to allow information to pass easily from one layer to the next. These challenges can be addressed in an attempt to assign credit properly and improve on the planner's ability to plan.

In developing ADP&E solutions to VLPO&S planning problems, providing traceable and justifiable plans is very important in applications that puts peoples' lives at risk, such as an urban search and rescue operation described in Chapter 8. To ensure traceability and justifiability in a planning process, an important challenge is investigating modularization of the approach to allow tunable parameters for many different aspects of the problem: (1) outcome selection, (2) action selection, (3) whether to re-plan or not, (4) plan selection, (5) monitoring task completions, (6) comparing planners' usefulness, and (7) comparing missions or games results. All these aspects are investigated with their own adaptable parameters. The goal here is to abstract a justification for taking a plan of action through examining the adapted parameters described in this chapter.

## 5.5 Summary

In summary, applying ADP&E solutions to VLPO&S planning problem provides many interesting challenges. In the ADP&E solution space, there are challenges in

designing and implementing the essential functional objects, control, general framework and process flow. In the VLPO&S planning problem domain, there are challenges in meeting the requirements. As mentioned previously, the overall goal is to design and implement a planner that can change its ability to plan based on an application's needs, and improves its ability through planning experience.

**References:**


1. E. Sacerdoti. "The Nonlinear Nature of Plans," *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI),* (1975) pp. 206 – 214.

2. B. Clement, E. Durfee, and A. Barrett. "Abstract Reasoning for Planning and Coordination," *Journal of Artificial Intelligence Research*, 28 (2007) pp. 453-515.

3. D. Leake. "Case-Based Reasoning: Experiences, Lessons, and Future Directions," *AAAI Press/MIT Press*. 1996.

4. J. Kurian,and R. Nagendran. "Top Down And Bottom Up Approach For Sustainability Of Waste Management In Developing Countries," *Proceedings Sardinia 2007, Eleventh International Waste Management And Landfill Symposium, S. Margherita di Pula, Cagliari, Italy;* 1 - 5 October 2007.

5. H. K. Jacobsen. "Integrating the bottom-up and top-down approach to energy–economy modelling: the case of Denmark," *Energy Economics, Volume 20, Issue 4*, 1 September 1998, pp. 443-461.

6. D. Aberdeen, S. Thiebaux, and L. Zhang. "Decision-theoretic military operations planning." *International Conference on Automated Planning & Scheduling 2004 (ICAPS'04)*. British Columbia, Canada.

7. D. Aberdeen, and O. Buffet. "Concurrent probabilistic temporal planning with policy-gradients," *International Conference on Automated Planning & Scheduling 2007 (ICAPS'07)*. Providence RI, USA.

8. A. Kolobov, Mausam, and D. Weld. "Regressing Deterministic Plans for MDP Function Approximation," *Workshop on A Reality Check for Planning and Scheduling Under Uncertainty at ICAPS.* Sydney, Australia, 2008.

9. L.P. Kaelbling, M.L. Littman, and A.R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence Journal, Vol. 101*, pages 99-134, 1998.

10. A. Hald. “On the history of maximum likelihood in relation to inverse probability and least squares,” *Statistical Science* **14** (2): 1999, pp. 214–222.

11. P. Billingsley. “Probability and Measure,” *New York, Toronto, London: John Wiley and Sons.* 1979.

12. R. Levinson, F. H. Hsu, J. Schaeffe, T. A. Marsland, & D. E. Wilkins, “The role of chess in artificial-intelligence research,” *ICCA Journal*, V. 14, N. 3, 1991, pp. 153-161.

13. S. Blatt. “RISKy business: An in-depth look at the game RISK" *Undergraduate Math Journal* **3** (2), 2002.

14. Bouzy, B. and Cazenave, T. (2001). Computer Go: an AI oriented survey. *Artificial Intelligence*, Vol. 132, pp. 39–103.

15. Reid, Thomas M. (2007), "Axis & Allies", in Lowder, James, *Hobby Games: The 100 Best*, Green Ronin Publishing, pp. 17–20.

16. Calhamer, Allan. "The Invention of *Diplomacy*", in *Games & Puzzles*, No. 21, January 1974.

# Chapter 6

# Approach

## 6.0 Abstract

Solutions for many real-world planning problems require a multi-pronged approach. In other words, most VLPO&S planning problems require a mixture of goal-directed, expert-defined, and data-driven approaches, taking advantage of their strengths, minimizing their weaknesses, while bringing together a practicality and realism to the modeling and simulation of games or missions. In this dissertation, the adopted approach attempts to build planners that can autonomously and dynamically plan and execute actions through constructing a model with: (1) simplified and adaptable SME (i.e., expert-defined), (2) dynamically tunable environmental data structures (i.e., data-driven), and (3) an underlying goal-directed architecture. Both the expertise and data structures are modifiable to meet the ultimate goal of learning better strategies for winning a game or completing a mission. However, the expertise and data structures are application dependent, and described in detail in Chapters 7 and 8. This chapter will describe the more general aspects of this ADP&E approach, the goal-directed architecture.

The goal-directed architecture will be described from various perspectives to provide a clear understanding of its functionality and composition. This chapter begins with a general description section that presents a conceptual understanding of the planner

and the environment. The goal-directed architecture description will include: (1) basic building blocks, (2) hierarchical decomposition, (3) functional flow and interactions, and (4) three phase core planning cycle. Second, a detailed description section follows with more in-depth material, which includes all functions and variables. This section will revisit the three phase planning cycle, and includes a general description of an approach to learning better planners. The final section describes the incremental order in which to design and implement the goal-directed architecture for the core planning cycle. Chapter 9: Design and Implementation of Future Applications will provide a full hierarchical description on how to build future applications, which includes lessons learned from the two applications described in Chapters 7 and 8.

## 6.1 General Description

The ultimate goal is to design and develop an ADP&E approach to train planners for winning a game or completing a mission. The application representation is restricted to the discrete event (in time and space) domain. The environment can be cooperative or non-cooperative, and the number of planners (or players) in the environment is only restricted to be a finite number, instead of one or two as defined in many applications [1] [2] [3] [4] [5]. The descriptions that follow in Sections 6.1 and 6.2 are structural rather than functional. That is, the building blocks of the entire planning process are named in sequential and nested hierarchical fashion along with the inputs and outputs of these blocks, and how they connect. What those blocks do is suggested by their names, but their operational details are application dependent. So, this chapter focuses on structure, while Chapters 7 and 8 that follow fill the details of operation for their respective

applications. In summary, this chapter is an attempt to formalize the planning process, a process that has not been formalized to this extent before.

The environment ($\Xi$) is defined as having two functional blocks: model (M) and state representation (X) (i.e., $\Xi = \{M_\Xi, X_\Xi\}$). Each planner ($\Psi$) is defined as having four functional blocks: model ($M_\Psi$), state ($X_\Psi$), action (A) options, and value (V) function (i.e., $\Psi = \{M_\Psi, X_\Psi, A, V\}$). Figure 6.1 shows a diagram of the environment and a single planner.

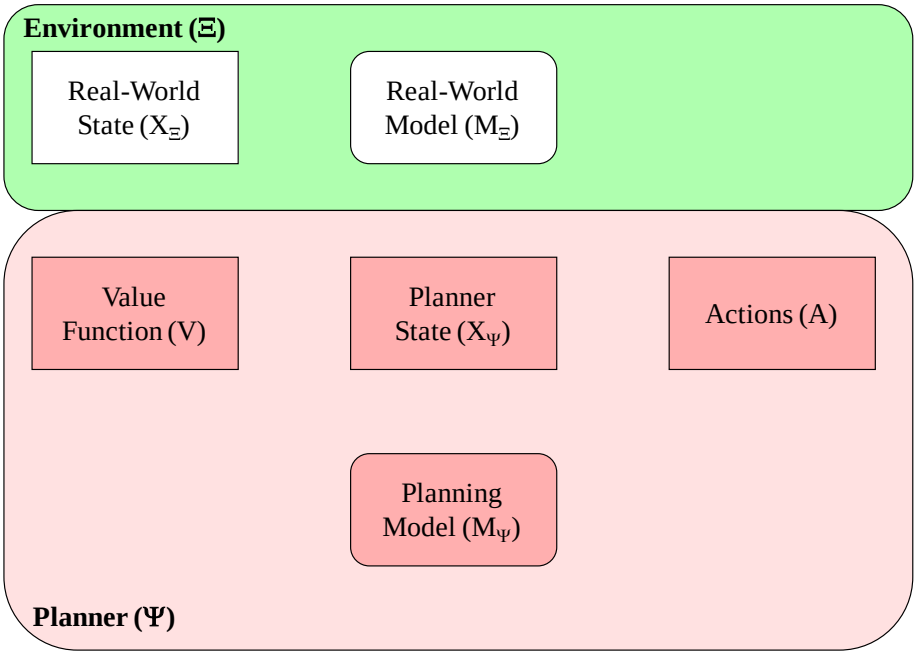


Figure 6.1. Planner and Environment with Functional Building Blocks

### 6.1.1 Basic Building Blocks

Models are used and represented as both objects and functions (i.e., $M = \{M_{obj}, M_{fun}\}$). The model objects include the environment and all planner models (i.e., $M_{obj} = \{M_\Xi, M_{\Psi 1}, M_{\Psi 2}, \ldots, M_{\Psi k}\}$, where k is the number of planners. The environmental model

is called the execution- or real-world model ($M_{\Xi}$), and each planner's model is called a planning-model ($M_{\Psi k}$). The functional portion of the models is the static infrastructure ($\mathcal{I}$) and state transition rules ($\mathcal{R}$) (i.e., $M_{fun} = f(\mathcal{I}, \mathcal{R})$). The static infrastructure includes the parts of the model that represent all non-varying features, such as a game board for a game, or a modeled terrain for a mission simulation. State transition rules are a functional representation of the game or mission rules, and are a function of the state and chosen actions (i.e., $\mathcal{R} = f(X, A)$). Static infrastructure and state transition rules are application dependent and are described in detail for specific cases in Chapters 7 and 8.

State is a combination of the model-state and simulation-state (i.e., $X = \{X_{mod}, X_{sim}\}$). There are several model-states, and they include the state of the environment and state of each planner (i.e., $X_{mod} = \{X_{\Xi}, X_{\psi 1}, X_{\psi 2}, \ldots, X_{\psi n}\}$). Each model-state ($X_{\Psi}$) is the planner's perceived state of the environment, and can also be referred to as a local state. The environmental or real-world-model state ($X_{\Xi}$) is defined as ground truth and can only be modified through a plan's execution. There is only one simulation-state and that is the current value of all the objects in the goal-directed hierarchy. The environment ($X_{\Xi}$) and simulation-state ($X_{sim}$) are both considered global state variables. Simulation state will be revisited when the hierarchy is described later in this section.

Generally speaking, action options are a function of the model and the state (i.e., $A = f(M, X)$). The use of the model and state to provide action options for search and select in forming plans, is highly dependent on the application under consideration. Also, search and select is an iterative process where the model and state are also dependent on the chosen actions in each plan. The choices available in searching and selecting actions

will be described in more detail in Section 6.2.1 Plan-Generation. Particular implementation details in performing search and selection are application specific and described in Chapters 7 and 8.

Value is composed of seven measurement systems: games or missions values, planners' utility, planning cycle assessment, plan fitness, expectation assessment, action selection, and outcome selection (i.e., $V = \{V_G, U_\Psi, C_\Gamma, F_P, E_Q, Z_A, Z_Y\}$). Each of the seven value components measures different aspects in the goal-directed process. Mission value ($V_G$) measures the overall performance of planners across a set of missions or games. Performance differences are shown by the direct competition of the planners on similar games or missions. Planner utility ($U_\Psi$) measures the usefulness of planner parameters within a single game or mission. Planner utility is thus a measure of how well parameters compare across other planners in their ability in achieving similar goals. Planning-cycle assessment ($C_\Gamma$) measures the ability to adjust to unexpected environmental changes. Planning-cycle assessment determines when a plan requires changes or is kept in continual execution. Plan-fitness ($F_P$) measures the ability to select a more optimal plan given a set of multiple plans acquired via search using computational and/or time constraints, probabilities and risks, rewards on reaching desirable states, and having diversity in the selection process. Fitness is the competitive function that balances the cost, risks, and reward of searching and selecting plans. Fitness is thus a measure of how well a plan performs. Expectation assessment ($E_Q$) is a function that determines whether a particular plan is accepted for continued execution or will be modified via re-planning. Once a plan is selected based on fitness, the expectations are compared during

plan-assessment to see if these expectations are met. If these expectations are met to some quantitative degree of acceptance, the plan can continue to be executed, otherwise re-planning occurs. Action selection ($Z_A$) is an important part of plan-generation. Specifically, the action selection function determines value for all action choices, and is used to make the choice of every action. Outcome selection ($Z_Y$) is an important part of every action choice, provided the number of possible outcomes exceeds one. If the number of outcomes is greater than one, the outcome selection function selects an outcome based on the risk the planner is willing to take. Section 6.2 identifies where value functions $F_P$, $E_Q$, $Z_A$, and $Z_Y$ work within the core planning cycle, and all the inputs and outputs of these functions are defined. Value functions $V_G$, $U_\Psi$, and $C_\Gamma$ are used in the analysis and learning portion of the ADP&E architecture. Where these functions are used in the ADP&E architecture is described in Section 6.2.4 Design. More specifics on these value functions are described in applied form in Chapters 7 and 8.

As shown in Table 6.1, mission value is a function of the top four tiers of the hierarchy (i.e., $V_G = \{\Omega, \Theta, M, \Psi\}$). Planner utility is a function of four of the top five tiers in the hierarchy (i.e., $U_\Psi = \{\Theta, M, \Psi, \Gamma\}$). Planning cycle assessment is a function of the middle five tiers of the hierarchy (i.e., $C_\Gamma = \{\Psi, \Gamma, P, T, A\}$). Plan fitness is a function of the bottom five tiers of the hierarchy (i.e., $F_P = \{P, T, A, Y, O\}$). Expectation assessment is a function of two of the middle tiers of the hierarchy (i.e., $E_Q = \{P, T\}$). Action selection is a function of two of bottom three tiers of the hierarchy (i.e., $Z_A = \{A, Y\}$). Outcome selection is only a function of the possible outcomes (i.e., $Z_Y = \{Y\}$).

These hierarchical dependencies are better understood using specific application details, thus these dependencies are explained by example in Chapters 7 and 8.

### 6.1.2 Hierarchical Decomposition

The goal-directed hierarchy is composed of ten levels from general to specific: (1) tournaments ($\Omega$), (2) rounds or competitions ($\Theta$), (3) games or missions ($G$), (4) players or planners ($\Psi$), (5) agents ($\Gamma$), (6) plans ($P$), (7) tasks ($T$), (8) actions ($A$), (9) outcomes ($Y$), and (10) observations ($O$). A tournament is defined as a process that controls a set of rounds or competitions that goal-directs the ADP&E architecture to find more qualified planners. A tournament is considered complete when the planner is not improving substantially from one tournament round to the next. A round is defined as a function that controls sets of similar games or missions that test the performance of different planners to determine their utility (described above). The planners with higher utility are admitted to the next round, where their modified copies recompete on similar games and missions. A game is a function of a single or multiple player(s) or planner(s). A game or mission is a simulation environment with a clear and concise goal, such as winning a game or completing a mission scenario. The model environment can be stochastic and/or partially observable, but there are well-defined rules for generating action choice possibilities that lead to the completion of each game or mission. A planner is a function that controls a variety of agents in order to win a game or complete a mission. Each agent is assigned one of several possible plans generated by the planner. Each plan can perform a variety of tasks. The number of tasks completed or the resources used in a plan determines the length of a plan. A completed task is a discrete change in state that contributes to the

overall goal of winning a game or completing a mission. The expectation assessment function described above measures the completed tasks to determine if a plan should proceed or be re-planned. An action is considered as an atomic movement in a game or mission. An action can have many resulting outcomes or observations when enacted within a model. An outcome is defined as a state change produced when an action is performed in a model. Outcomes can be nondeterministic in stochastic models, or unobservable before execution in partially observable models. Each outcome can have a set of observations of state variables within the model environment. An observation is defined as a state value that can be sensed by performing an action in the model.

Table 6.1. Ten-Tier Hierarchical Framework

| Hierarchical Level | Composition | One to Many Relationship | Tuple Relation | Metric Operators |
|---|---|---|---|---|
| 1. Tournaments | $\Omega = \{\omega_1, \omega_2, ..., \omega_i\}$ | $\omega = \{\Theta\}$ | $\overline{\omega} = (\omega, \Theta)$ | $V_G = \{\Omega, \Theta, G, \Psi\}$ |
| 2. Rounds | $\Theta = \{\theta_1, \theta_2, \ldots, \theta_i\})$ | $\theta = \{M\}$ | $\overline{\theta} = (\omega, \theta, G)$ | |
| 3. Games | $G = \{g_1, g_2, ..., g_i\}$ | $g = \{\Psi\}$ | $\overline{g} = (\theta, g, \Psi)$ | $U_Y = \{\Theta, M, \Psi, \Gamma\}$ |
| 4. Planners | $\Psi = \{\psi_1, \psi_2, \ldots, \psi_i\}$ | $\psi = \{\Gamma\}$ | $\overline{\psi} = (g, \psi, \Gamma)$ | |
| 5. Agents | $\Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_i\}$ | $\gamma = \{P\}$ | $\overline{\gamma} = (\psi, \gamma, P)$ | $C_\Gamma = \{\Psi, \Gamma, P, T, A\}$ |
| 6. Plans | $P = \{p_1, p_2, ..., p_i\}$ | $p = \{T\}$ | $\overline{p} = (\gamma, p, T)$ | $F_P = \{P, T, A, Y, O\}$ |
| 7. Tasks | $T = \{\tau_1, \tau_2, \ldots, \tau_i\}$ | $\tau = \{A\}$ | $\overline{\tau} = (p, \tau, A)$ | $E_Q = \{P, T\}$ |
| 8. Actions | $A = \{a_1, a_2, ..., a_i\}$ | $a = \{Y\}$ | $\overline{a} = (t, a, Y)$ | $Z_A = \{A, Y\}$ |
| 9. Outcomes | $Y = \{y_1, y_2, ..., y_i\}$ | $y = \{O\}$ | $\overline{y} = (a, y, O)$ | |
| 10. Observations | $O = \{o_1, o_2, ..., o_i\}$ | | $\overline{o} = (y, o)$ | $Z_Y = \{Y\}$ |

More formally stated, $\Omega$ is a collection of tournaments and $\omega$ is a single tournament (i.e., $\Omega = \{\omega_1, \omega_2, \ldots, \omega_n\}$). $\Theta$ is a collection of rounds or competitions and $\theta$ is a single round or competition (i.e., $\Theta = \{\theta_1, \theta_2, \ldots, \theta_n\}$). $G$ is a collection of games or

missions and $g$ is a single game or mission (i.e., $G = \{g_1, g_2, \ldots, g_n\}$). $\Psi$ is a collection of players or planners and $\psi$ is a single player or planner (i.e., $\Psi = \{\psi_1, \psi_2, \ldots, \psi_n\}$). $\Gamma$ is a collection of agents and $\gamma$ is a single agent (i.e., $\Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_n\}$). $P$ is a collection of plans and $p$ is a single plan (i.e., $P = \{p_1, p_2, \ldots, p_n\}$). $T$ is a collection of tasks and $\tau$ is a single task (i.e., $T = \{\tau_1, \tau_2, \ldots, \tau_n\}$). $A$ is a collection of actions and $a$ is a single action (i.e., $A = \{a_1, a_2, \ldots, a_n\}$). $Y$ is a collection of outcomes and $y$ is a single outcome (i.e., $Y = \{y_1, y_2, \ldots, y_n\}$). $O$ is a collection of observations and $o$ is a single observation (i.e., $O = \{o_1, o_2, \ldots, o_n\}$). As mentioned earlier, a simulation state is the current value of all the objects in the goal-directed hierarchy (i.e., $X_{sim} = \{\Omega, \Theta, G, \Psi, \Gamma, P, T, A, Y, O\}$).

Each collection or single hierarchical class has an associated object and tuple relationship. In terms of objects, observations are elements of an outcome, outcomes are elements of an action, actions are elements of a task, tasks are elements of a plan, plans are elements of an agent, agents are elements of a game, games are elements of a round, and rounds are elements of a tournament (i.e., $o \in y \in a \in \tau \in p \in \gamma \in \psi \in g \in \theta \in \omega$). In other words, each tournament is a set of rounds (i.e., $\omega = \{\Theta\}$). Each round is a set of games (i.e., $\theta = \{G\}$). Each game is a set of planners (i.e., $g = \{\Psi\}$). Each planner is a set of agents (i.e., $\psi = \{\Gamma\}$). Each agent is a set of plans (i.e., $\gamma = \{P\}$). Each plan is a set of tasks (i.e., $p = \{T\}$). Each task is a set of actions (i.e., $\tau = \{A\}$). Each action is a set of outcomes (i.e., $a = \{Y\}$). Each outcome is a set of observations (i.e., $y = \{O\}$).

The tuple relationships of each hierarchical class are divided such that the each level only refers to the objects one level above and below. More specifically, a

tournament tuple is a 2-tuple of a specific tournament and set of rounds (i.e., $\bar{\omega} = (\omega, \Theta)$). A round tuple is a 3-tuple of a specific tournament, a specific round and set of games (i.e., $\bar{\theta} = (\omega, \theta, G)$). A game tuple is a 3-tuple of a specific round, a specific game and set of planners (i.e., $\bar{g} = (\theta, g, \Psi)$). A planner tuple is a 3-tuple of a specific game, a specific planner and set of agents (i.e., $\overline{\psi} = (g, \psi, \Gamma)$). An agent tuple is a 3-tuple of a specific planner, a specific agent and set of plans (i.e., $\overline{\gamma} = (\psi, \gamma, P)$). A plan tuple is a 3-tuple of a specific agent, a specific plan and set of tasks (i.e., $\bar{p} = (\gamma, p, T)$). A task tuple is a 3-tuple of a specific plan, a specific task and set of actions (i.e., $\bar{\tau} = (p, \tau, A)$). An actions tuple is a 3-tuple of a specific task, a specific action and set of outcomes (i.e., $\bar{a} = (\tau, a, Y)$). An outcomes tuple is a 3-tuple of a specific action, a specific outcome and set of observations (i.e., $\bar{y} = (a, y, O)$). An observation tuple is a 2-tuple of a specific outcome and a set of observations (i.e., $\bar{o} = (y, O)$). Table 6.1 summarizes the general relationships within the hierarchy. More detailed relationships will be described later in this chapter.

### 6.1.3 Functional Flow and Interactions

Functional flow and controlled interactions happen at two distinct locations in the goal-directed architecture. The upper portion of the hierarchy is the training portion and includes two major processes: a hierarchical control chain (HCC) that keeps track of tournaments, rounds, missions and planners, and learning and analysis (L&A) that evaluates the results of the core planning cycle. The lower portion of the hierarchy has only one major process and that is the core planning cycle (CPC) that controls all interactions from planners to observations. The HCC controls the start and stop of

tournaments ($\Omega$), rounds ($\Theta$), missions (G), and planners ($\Psi$). The number of rounds per tournament, the number of missions per round, and the number planners per mission are application dependent and described in detail in Chapters 7 and 8. The L&A analyzes the results of the CPC to train planners and statistically analyze the progress of planner parameters. The training parameters used are determined by an evolutionary learning function ($\Lambda_\theta$) and the statistical analysis function ($\Sigma_\omega$) that tracks the means and standard deviations of the planner parameters. The type and number of evolutionary learning function parameters and CPC planner parameters are application dependent and described in detail in Chapters 7 and 8. The CPC is at the heart of the ADP&E architecture and will be described in general in subsequent paragraphs. Section 6.2 will provide a more detailed look at HCC, L&A and CPC.

In the core planning cycle, the environment ($\Xi$) and each planner ($\Psi$) have six basic building blocks as described above (i.e., $\Xi = \{M_\Xi, X_\Xi\}$ and $\Psi = \{M_\Psi, X_\Psi, A, V\}$) and shown in Figure 6.1. These building blocks have interactions that make up the complete operational structure of the planning process for each planner. These interactions are: (1) planner-state to actions; (2) actions to real-world-model; (3) real-world-model to real-world-state; (4) real-world-model to planner-state; (5) planner-state to planning-model; (6) actions to planning-model; (7) planner-state to value-function; and (8) value-function to actions. These relationships are shown in Figure 6.2.

The core planning cycle controls the use of the lower portion of the simulation state (i.e., $X_{sim} = \{\Gamma, P, T, A, Y, O\}$) to produce these eight interactions. The CTC operates in three iterative phases ($\Phi$): plan-generation ($\Phi_1$), plan-execution ($\Phi_2$) and plan-

assessment ($\Phi_3$). Each agent ($\gamma$) of a planner can be in only one phase at a time. Each phase controls which of the eight interactions are used and how they perform. Interactions (1), (5), (6), (7), and (8) are used in the plan-generation phase, interactions (1), (2), (3), (4) and (5) are used in the plan-execution phase, and interactions (1), (5), (6) and (7) are used in the assessment phase. A more detailed description of all the interactions is provided in the planning cycle subsection 6.1.4.

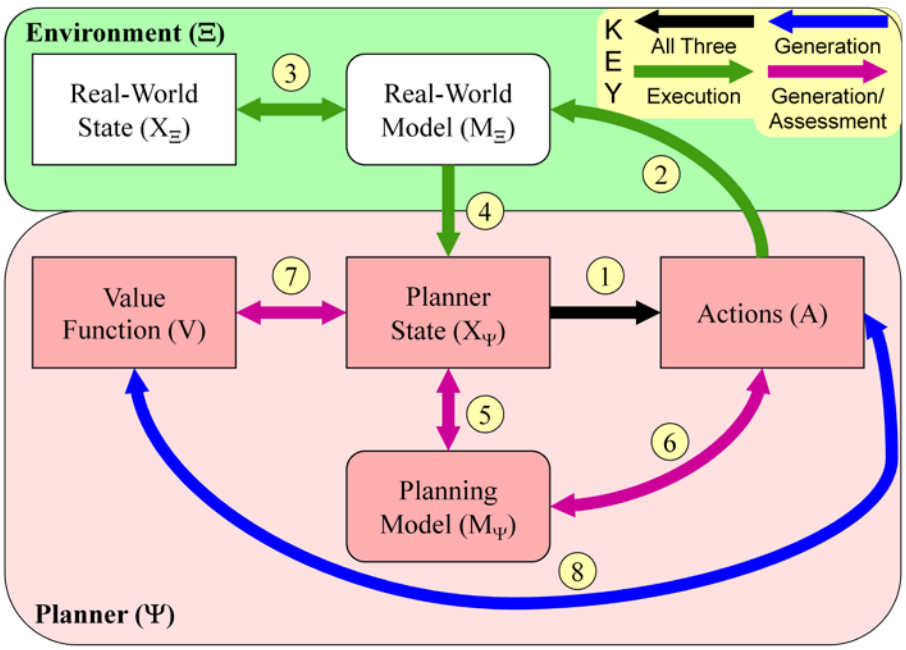


Figure 6.2. Functional Building Blocks with Eight Interactive Processes

As shown in Figure 6.3, there are five functional interactions in the plan-generation phase: (1) *search for and select plans*; (2) *determine action choices*; (3) *select actions*; (4) *model/state interactions*; and (5) *evaluate fitness*. *Search for and select plans* comprise the process of comparing plans to choose the most desirable for execution. This process controls all four subsequent processes in this phase and iterates them until a plan for each agent is considered complete. When an agent receives a selected plan, it can go

to plan-execution phase. First, *determine action choices* process provides action options given the current state. As actions are attempted in planning-model and planner-states change, new actions become available. Next, *select actions* process is a bidirectional interaction of selecting actions using the planning-model and state response. Selecting actions could be done randomly or could take into account planning costs, resources available, and SME selected state conditions that are considered important in choosing actions. Next, *model/state interactions* process is the bidirectional interactions that handle outcomes and observations based on a chosen action. Outcomes and observations are represented as state-transitions using the current plan and planning-model. Finally, *evaluate fitness* process combines measures of the performance of a plan, such as the reward value placed on state changes, the probability of successful completion of a plan, and the diversity in plan selection.

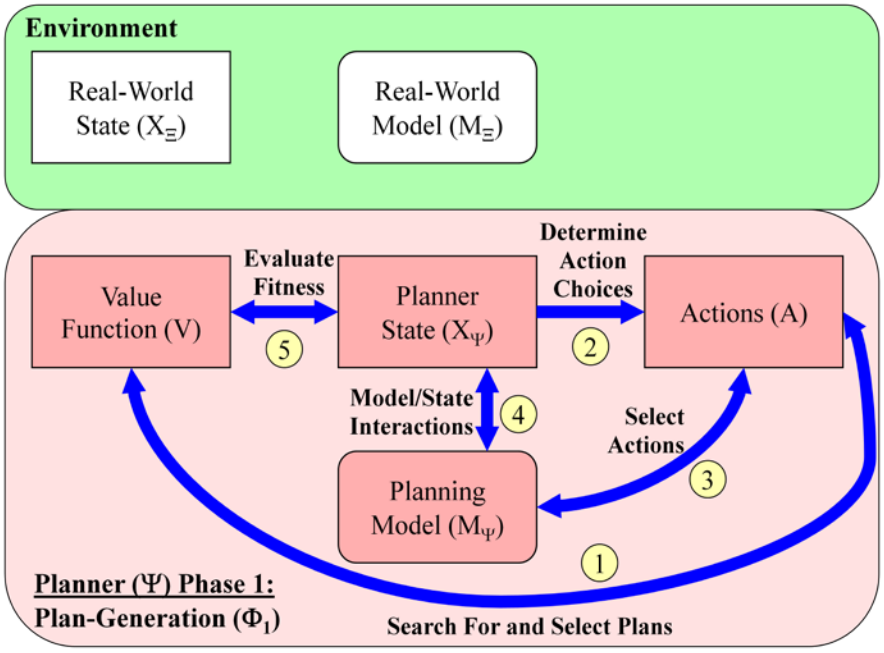


Figure 6.3. Planner's Plan-Generation Phase ($\Phi_1$) with Five Interactive Processes

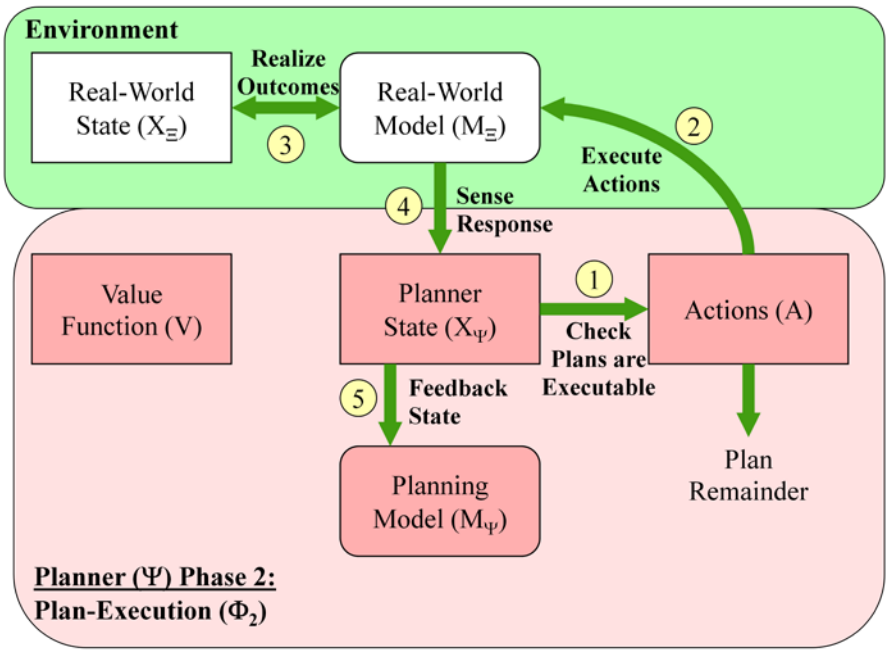


Figure 6.4. Planner's Plan-Execution Phase ($\Phi_2$) with Five Interactive Processes

As shown in Figure 6.4, there are five functional interactions in the plan-execution phase: (1) *check that plans are executable*; (2) *execute actions*; (3) *realize outcomes*; (4) *sense response*; and (5) *feedback state*. First, *check that plans are executable* is the process of making sure the next actions in the plan are still viable given the perceived state of the environment (i.e., planning-model). If still viable, the next action can be executed, otherwise the plan is cancelled and the agent awaits a new plan. Second, *execute actions* is the process of executing each action in time order concurrently for all agents. Next, *realize outcomes* is the process of determining outcomes represented as state-transitions using the current plan and real-world model. Fourth, *sense response* is the process of monitoring observable outcomes passed on from the real-world model. Finally, *state feedback* is the process of updating the planning-model with the updated state using new observations from the environment (i.e., real-world-model). The *plan-*

*remainder* shown as output from the actions block represents the remaining portion of the plan not yet executed.

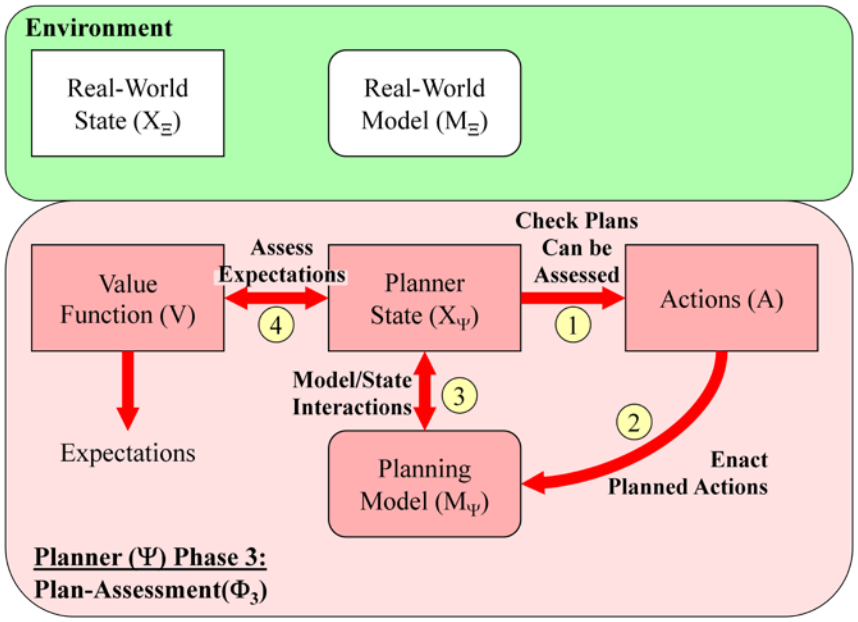


Figure 6.5. Planner's Plan-Assessment Phase ($\Phi_3$) with Four Interactive Processes

As shown in Figure 6.5, there are five functional interactions in the plan-assessment phase: (1) *check that plans can be assessed*; (2) *reenact planned actions*; (3) *model/state interactions*; and (4) *assessment*. First, *check that plans can be assessed*, is similar to the process used in plan-execution phase, and determines if a plan is viable for assessment. If still viable, the next action can be assessed, otherwise the plan is truncated and the agent awaits a new plan in the next planning phase, plan-generation. Next, *enact planned actions* is the process of reenacting the plan-remainder (identified above) in the planning-model with the new updated state of the planning-model based on recent observations. Next, *model/state interactions* are the process of generating the new perceived outcomes and observations using the updated planning-model. Finally, the *assessment* process generates new expectations and compares these newly perceived

expectations with the original planned expectations produced in the previous plan-generation phase.

### 6.1.4 Three Phase Planning Cycle

As mentioned above, the planner operates in three iterative phases ($\Phi$): plan-generation ($\Phi_1$), plan-execution ($\Phi_2$) and plan-assessment ($\Phi_3$). In this three phase cycle, there are three objects exchanged and/or updated: (1) plans; (2) model updates; and (3) expectations. First, plan-generation creates plans with corresponding expectations for each agent by sequencing actions in a planning-model via search. The resulting plans are sent to plan-execution and the corresponding expectations are sent to plan-assessment. Next, plan-execution begins to execute the scheduled order of actions using the real-world model. After a predefined number of actions are executed or a number of tasks are complete, the planning-model is updated with any new observations acquired since the last model update. The planning-model is updated in both plan-generation and plan-assessment simultaneously. Note plan-execution uses the real-world model, while plan-generation and -assessment use a planning-model. Also, the remainder of the plan not executed is passed forward to plan-assessment to reassess this plan-remainder by enacting it using the updated planning-model to forecast new expectations. New and old expectations are compared for each agent to determine whether re-planning is necessary. If expectations are within desired constraints, no re-planning is necessary and the plans and expectations are passed through plan-generation directly to plan-execution again. If re-planning is necessary, plan-assessment passes empty plans and expectations to plan-generation for re-planning and the cycle continues. Note that the changes in the planning-

model are revertible. However, changes in the real-world model cannot be reverted due to the nature of action execution. Thus, careful planning in the plan-generation phase can minimize any unforeseeable problems in the real-world model. This description is summarized in Figure 6.6.

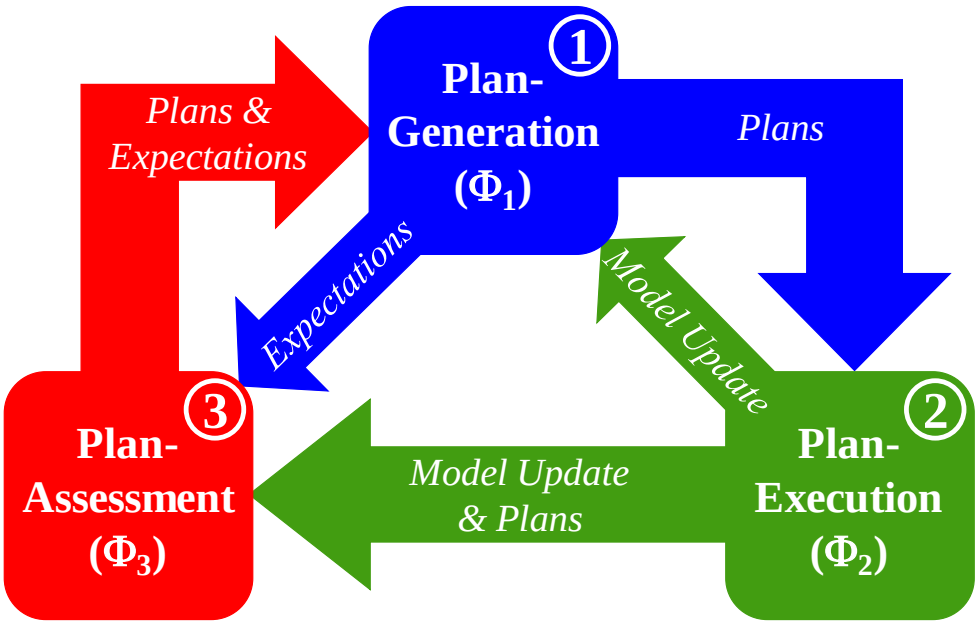


Figure 6.6. Planner's Full Planning Cycle

The three phase planning cycle is described more formally here. Each of the three phases (i.e., plan-generation ($\Phi_1$), plan-execution ($\Phi_2$), plan-assessment ($\Phi_3$)) contains a model (M), plan(s) (P), and expectation(s) (Q) (i.e., $\Phi$ = {M, P, Q}). All three phases operate independently for each planner. In other words, each planner is independent in where they are in the three-phase process from other planners. However, each planner's phase is dependent on the model(s) (M) they use, which is changeable when agents generate or execute plans. A plan-generation phase for a particular planner ($\phi_{1\psi}$) is a function of: a planning-model of a particular planner ($M_\psi$), a set of plans for all agents of

the planner ($P_\Gamma$), and expectations for each plan of all agents ($Q_P$) (i.e., $\phi_{1\gamma} = f(M_\psi, P_\Gamma, Q_P)$). Each agent has a selected-plan (i.e., $P_\Gamma = \{p_{\gamma 1}, p_{\gamma 2}, \ldots, p_{\gamma m}\}$), and each agent's plan has corresponding expectations (i.e., $Q_P = \{q_{p1}, q_{p2}, \ldots, q_{pm}\}$), where m is the number of agents or selected-plans. A plan's expectations are directly related to the set of completed tasks for that plan (i.e., $q_p = \{\tau_1, \tau_2, ...,\tau_n\}$), where n is the number of completed tasks in that plan. The plan-execution phase is a function of the execution-model ($M_\Xi$), selected plans for all corresponding agents ($P_\Gamma$), and expectations for all corresponding plans ($Q_P$) (i.e., $\phi_{2\psi} = f(M_\Xi, P_\Gamma, Q_P)$). The plan-assessment phase uses the planning model of a particular planner ($M_\psi$), the remainder of the selected-plans after execution for all corresponding agents ($P_\Gamma$), and expectations of all corresponding selected-plans ($Q_P$) (i.e., $\phi_{3\psi} = f(M_\psi, P_\Gamma, Q_P)$).

Each phase modifies portions of model(s), plan(s), and expectation(s) for all agents of each particular planner. The plan-generation function modifies: (1) a planning-model for a particular planner ($M_\psi$) and passes it to the plan-assessment phase for that planner ($\phi_{3\psi}$), (2) a plan for each agent and passes a selected-plan ($p_\gamma$) to the plan-execution phase for that planner ($\phi_{2\psi}$), and (3) expectations for each selected plan ($q_p$) and passes it to the plan-assessment phase ($\phi_{3\psi}$) for later comparison. The plan-execution function modifies: (1) the execution-model ($M_\Xi$) and passes portions of the model based on each planner's observations to both the plan-generation ($\phi_{1\psi}$) and plan-assessment ($\phi_{3\psi}$) phases for that planner, and (2) a plan for each agent is truncated to reflect the actions not yet executed ($p_\gamma$) and passes corresponding plan-remainders to the plan-

assessment phase for that same planner ($\phi_{3\psi}$). If an action cannot be executed for an agent due to unforeseen environmental (i.e., real-world-model) changes, that plan is set to null and passed to plan-assessment (i.e., $p_\gamma$ = null). The plan-assessment function modifies: (1) a planning-model for a particular planner ($M_\psi$) based on execution-model observations received from the plan-execution phase, and (2) the plan-remainders depending on whether the plans still meet expectations ($Q_P$) and passes them to the next plan-generation phase ($\phi_{1\psi}$). Plan-assessment compares expectations received from the previous plan-generation phase with the new forecasted expectations taking into account the outcomes realized from partial plan-execution.

## 6.2 Detailed Description

The detailed description of the goal-directed architecture for ADP&E will be presented in four subsections. The first three subsections provide an in-depth description of the three phases of the planning process: plan-generation, plan-execution and plan-assessment. These subsections include all corresponding sub-processes, their associated terminology, and a functional description. Implementation details and specific differences related to application domain are described in Chapters 7 and 8. These first three subsections describe the middle and lower level functionality of the architecture. The fourth and final subsection describes top levels of the architecture, which measure utility and define the approach to learning better planners.

### 6.2.1 Plan-Generation

As described above, the plan-generation phase has five interactive processes: (1.1) *search for and select plans,* (1.2) *determine action choices,* (1.3) *select actions,* (1.4)

*interactions of model and state*, and (1.5) *evaluate plan fitness*. Two of these general processes can be broken-down into several sub-processes, while the other three processes consist of only a single process. *Search for and select plans* can be broken into three sub-processes: (1.1.1) *search for plans*, (1.1.2) *select plans*, and (1.1.3) *forecast expectations*. *Interactions of model and state* can be broken into four sub-processes: (1.4.1) *predict outcomes*, (1.4.2) *select outcomes*, (1.4.3) *observe state changes*, and (1.4.4) *update plan's model state*. All of these ten sub-processes are summarized in Table 6.2. Throughout these next three subsections, sub-processes will be defined as functions with a set of variables. The nomenclature used is such that functions are non-italicized and capitalized, and variables are italicized and capitalized, except for the variable time (*t*). Coefficients are italicized and lowercase, and subscripts are used to help qualify a function, variable or coefficient.

For plan-generation, the first sub-process is the *search for plans* function (1.1.1: $S_P$), and this function orchestrates the generation of multiple plans for each agent. $S_P$ is a search algorithm for exploring the planning-model to develop action sequences that are planned before real-world interaction. The role of the search algorithm within a game or mission simulation is to search an internal model (planning-model) to achieve some goals or tasks and to receive rewards that measure this progress. The *search for plans* function uses five input variables: (1) the planner's search parameters ($S_\psi$); (2) completed tasks per plans ($T_P$); (3) proportion of completed tasks constraint coefficient ($\tau_P$); (4) resources available ($R$); and (5) portion of used resources constraint coefficient ($u_R$). The function outputs a set of possible plans ($P$) (i.e., $P = S_P(S_\psi, T_P, \tau_P, R, u_R)$). The planner's search

parameters determine when to search and how much to search. Tasks completed per plans and/or resources available are used with their corresponding coefficients to determine how deeply a plan should be searched. The proportion of completed tasks constraint coefficient, and the portion of resources used coefficient are both fractions between zero and one. What constitutes a completed task or usable resource is application dependent and will be described in detail in Chapters 7 and 8.

Table 6.2. Phase 1: Plan-Generation Processes

| *Planning Phase* | *General Processes* | *Sub-Processes* |
|---|---|---|
| *Phase 1: Plan-Generation* | *1.1. Search For and Select Plans* | *1.1.1. Search For Plans* |
| | | *1.1.2. Select Plans* |
| | | *1.1.3. Forecast Expectations* |
| | *1.2. Determine Action Choices* | |
| | *1.3. Select Actions* | |
| | *1.4. Interaction of Model and State* | *1.4.1. Predict Outcomes* |
| | | *1.4.2. Select Outcomes* |
| | | *1.4.3. Observe State Changes* |
| | | *1.4.4. Update Plan's Model State* |
| | *1.5. Evaluate Plan Fitness* | |

The planner's search parameters make the *search for plans* function different from all other functions introduced here, because search requires all other sub-processes used in plan-generation. Search requires a set of actions (1.2: *determine action choices*), an action selection function (1.3: *select actions*), outcomes and observations (1.4: *interactions of state and model*), and a reward system to aid in choosing the best plans (1.5: *evaluate plan fitness*). In other words, given a set of possible actions, a search algorithm strings these actions together to form plans that explore paths through the internal (i.e., planning) model. The plans are graded based on a reward system that measures the effectiveness of the plans by their predicted outcomes and observations.

Plans are generated to complete tasks and use resources. If a plan is restricted by task completions, then the completed tasks-per-plan variable ($T_P$) determines when a plan is complete. If a plan is resource constrained, then that plan ends when the resources ($R$) used reaches the resources constraint fraction ($u_R$). Resource constraints are application dependent, thus a detailed example is shown in Chapter 7.

The other following four processes (1.2-1.5) are performed in each iterative search cycle of plan-generation until the final plans are selected. The iterations are closed out using the sub-processes *select plans* function ($Z_P$) and *forecast expectations* function ($K_Q$). The *select plans* function uses the fitness ($F$) and corresponding searched plans ($P$) to determine which plans to select ($P_Z$) for the next iteration of search (i.e., $P_Z = Z_P(F, P)$). The next sub-process is the *forecast expectations* function ($K_Q$). This function takes the selected plans ($P_Z$) and their associated tasks-per-plan ($T_P$) to determine the planned expectations ($Q$) of the selected plans (i.e., $Q = K_Q(P_Z, T_P)$). Each plan's expectations are a quantitative measure of all completed tasks for each plan. Specific expectations metrics are application based and described in Chapters 7 and 8.

Within a *search for plans* function, the search parameters drive the other four processes (1.2-1.5). Before any plan can be searched, action choices must be made available using the *determine action choices* function ($I_A$). This function is dependent on: (1) the agent under consideration ($\gamma$ described above), (2) the planner's planning-model ($M_\Psi$ described above) and (3) the current state of the plans ($X_P$) (i.e., $A = I_A(\gamma, M_\Psi, X_P)$). Note that determine action choices function is used once after each action in a plan. Thus,

there is a separate state representation for every action within a plan, and the next action choices depend on the most recent state values, identified as $X_P$.

The third process of plan-generation is the *select actions* function (1.3: $Z_A$). This function uses six variable inputs: (1) set of agents under consideration for planning ($\Gamma$ described above); (2) action options ($A$ described above); (3) selected features ($H$); (4) available resources ($R$ described above); (5) action costs ($C_A$); and (6) time ($t$) (i.e., $A_Z = Z_A(\Gamma, A, H, R, C_A, t)$). The selected features are a subset of the overall plans' state ($X_P$) determined by SME. The action costs are associated with selecting an action, such as fuel expended or time to complete an action. The time variable identifies which agent to consider when selecting an action based on time-ordering the action choices. For instance, the next agent to plan is the one with the shortest plan in terms of time length. If time is not a factor, then choosing an agent to take an action is an arbitrary choice. The selected features and action costs will be described more fully in the application chapters, 7 and 8.

The fourth process of plan-generation is the *interaction of model and state* function (1.4). This process has four sub-processes identified in Table 6.2 (1.4.1-1.4.4). The first of these four processes is the *predict outcomes* function (1.4.1: $K_Y$). This function uses four variable inputs: (1) the selected actions ($A_Z$ described above); (2) the corresponding agent under consideration ($\gamma$ described above); (3) the planning-model of current planner ($M_\Psi$ described above); and (4) the current state of the corresponding plan ($X_P$ described above) (i.e., $Y = K_Y(A_Z, \gamma, M_\Psi, X_P)$). This function outputs a set of outcomes ($Y$) that represent a set of probable state transitions.

The second of the four sub-processes is *select outcomes* function (1.4.2: $Z_Y$). This function uses two inputs: (1) the set of outcomes output by the previous process ($Y$ described above); and (2) outcome risk aversion coefficient ($r_Y$). This coefficient is a real number between zero and one and determines the amount of acceptable risk to take when selecting an action. This parameter is best described with an example, and will be described fully in Chapter 7. The output of the select outcomes function is a selected outcome for each action chosen ($Y_Z$) (i.e., $Y_Z = Z_Y(Y, r_Y)$).

The third of the four sub-processes is *observe state changes* function (1.4.3: $O_X$). This function uses four inputs: (1) selected outcomes ($Y_Z$ described above); (2) agent under consideration ($\gamma$ described above); (3) the planning-model of current planner ($M_\Psi$ described above); and (4) the current state of the corresponding plan ($X_P$ described above) (i.e., $O = O_X(Y_Z, \gamma, M_\Psi, X_P)$). This function outputs a set of observations ($O$), which represent a set of observable state variables.

The fourth of the four sub-processes is *update plan's model state* function (1.4.4: $X_U$). This function uses two variable inputs: (1) the set of agents under consideration for planning ($\Gamma$ described above); (2) the observations ($O$ described above); and (3) the task update coefficient ($\tau_U$). This coefficient is an integer number of tasks that need to be completed by any agent before the planner updates its' internal planning-model. The output of this function is a new state representation for the internal planning-model for each plan and at every action step (i.e., $X_P = X_U(\Gamma, O, \tau_U)$).

The fifth and final process of plan-generation is the *evaluate plan fitness* function (1.5: $F_P$). This function uses four variable inputs: (1) the current state of the

corresponding plan ($X_P$ described above); (2) the planner's sub-goals ($J_\Psi$); (3) the probability of successful completion of planned tasks ($Pr(T_P)$); and (4) diversity ($D$). The planner's sub-goals are functions that measure the current state. These functions are defined by SME and dependent on the application. The probability of successful completion of planned tasks is the total probability of completing all tasks in a plan. This is best explained with an example, which will be shown in Chapter 7. Diversity is a metric that provides reward for choosing plans different from those already selected, and based on having different outcomes. The fitness function outputs all fitness for each plan, including fitness at each action step in a plan, which may determine a possible point to prune a plan (i.e., $F = \mathrm{F_P}(X_P, J_\Psi, Pr(T_P), D)$).

### 6.2.2 Plan-Execution

As described above, the plan-execution phase has five interactive processes: (2.1) *check actions are executable*, (2.2) *execute actions*, (2.3) *realize outcomes*, (2.4) *sense response* and (2.5) *feedback state*. Unlike plan-generation, plan-execution only has a single sub-process per general process. All of these five processes are summarized in Table 6.3.

Table 6.3. Phase 2: Plan-Execution Processes

| *Planning Phase* | *General Processes* |
|---|---|
| *Phase 2: Plan-Execution* | *2.1. Check Plans are Executable* |
| | *2.2. Execute Actions* |
| | *2.3. Realize Outcomes* |
| | *2.4. Sense Response* |
| | *2.5. Feedback State* |

The first process of plan-execution is the *check actions are executable* function (2.1: $P_{\Xi}$). This function uses four variable inputs: (1) the selected plans ($P_Z$ described above); (2) the corresponding agent under consideration ($\gamma$ described above); (3) the planning-model of current planner ($M_{\Psi}$ described above); and (4) the current state of the corresponding plan ($X_P$ described above). The output of this function is the plan-remainder ($P'$) (i.e., $P' = P_{\Xi}(P_Z, \gamma, M_{\psi}, X_P)$). The plan-remainder is the portion of the plan that is still considered executable.

The second process of plan-execution is the *execute actions* function (2.2: $A_{\Xi}$). This function uses two variables: (1) plan-remainder ($P'$ described above); and (2) time ($t$ described above). The *execute actions* function outputs two variables: (1) the first time-ordered actions that can be executed; and (2) a new plan-remainder that eliminates the first time-ordered action from it's list (i.e., $\{A_1, P'\} = A_{\Xi}(P', t)$). As actions are executed, eliminating the executed action from the plan-remainder shortens it and prepares the plan-remainder either for further execution or for the plan-assessment phase.

The third process of plan-execution is the *realize outcomes* function (2.3: $Y_R$). This function uses four variables: (1) first time-ordered actions ($A_1$ described above); (2) corresponding agent under consideration ($\gamma$ described above); (3) the real-world or environmental model ($M_{\Xi}$ described above); and (4) the current state of the environment ($X_{\Xi}$ described above). This function outputs realized outcomes ($Y'$) (i.e., $Y' = Y_R(A_1, \gamma, M_{\Xi}, X_{\Xi})$), which are considered as permanent changes in the environmental model.

The fourth process of plan-execution is the *sense response* function (2.4: $O_{\Xi}$). This function uses four input variables: (1) realized outcomes ($Y'$ described above); (2) the corresponding agent under consideration ($\gamma$ described above); (3) the real-world or environmental model ($M_{\Xi}$ described above); and (4) the current state of the environment ($X_{\Xi}$ described above). This function outputs realized observations ($O'$) (i.e., $O' = O_{\Xi}(Y', \gamma, M_{\Xi}, X_{\Xi})$), which are considered local observations of each agent. These observations are not shared with other agents of a particular planner until the planning-model is updated with feedback.

The fifth and final process of plan-execution is the *feedback state* function (2.5: $X_U$). This function uses three input variables: (1) a set of agents under consideration for planning ($\Gamma$ described above); (2) the observations ($O$ described above); and (3) the task update coefficient ($\tau_U$ described above). The output of this function is a new state representation for the internal planning model (i.e., $X_{\psi} = X_U(\Gamma, O', t_U)$). Unlike *update plan's model state* function (1.4.4: $X_U$), the *feedback state* function makes permanent changes in the planning-model for all agents controlled by a particular planner.

### 6.2.3 Plan-Assessment

As described above, the plan-assessment phase has four interactive processes: (3.1) *check plans can be assessed*, (3.2) *reenact planned actions*, (3.3) *interactions of model and state*, and (3.4) *assess expectations*. The last two of these general processes can be broken-down into several sub-processes, while the first two processes have only a single process. *Interact model and state* process can be broken into four sub-processes: (3.3.1) *predict outcomes*, (3.3.2) *select outcomes*, (3.3.3) *observe state changes*, and

(3.3.4) *update plan's model state*. Second, *assess expectations* process can be broken into three sub-processes: (3.4.1) *forecast new expectations*, (3.4.2) *compare expectations*, and (3.4.3) *initiate re-planning*. All of these nine sub-processes are summarized in Table 6.4.

The first process of plan-assessment is the *check plans can be assessed* function (3.1: $P_{\psi}$). This function uses four input variables: (1) plan-remainder ($P'$ described above); (2) the corresponding agent under consideration ($\gamma$ described above); (3) the planning-model of current planner ($M_{\Psi}$ described above); and (4) the current state of the corresponding plan ($X_P$ described above). The output of this function is a new plan-remainder ($P''$) (i.e., $P'' = P_{\psi}(P', \gamma, M_{\psi}, X_P)$). The plan-remainder is the portion of the plan that is still considered as a plausible plan and is executable. This function operates similarly to *check plans for execution* (2.1: $P_{\Xi}$).

Table 6.4. Phase 3: Plan-Assessment Processes

| *Planning Phase* | *General Processes* | *Sub-Processes* |
|---|---|---|
| *Phase 3: Plan-Assessment* | *3.1. Check Plans can be Assessed* | |
| | *3.2. Enact Planned Actions* | |
| | *3.3. Interaction of Model and State* | *3.3.1. Predict Outcomes* |
| | | *3.3.2. Select Outcomes* |
| | | *3.3.3. Observe State Changes* |
| | | *3.3.4. Update Plan's Model State* |
| | *3.4. Assess Expectations* | *3.4.1. Forecast New Expectations* |
| | | *3.4.2. Compare Expectations* |
| | | *3.4.3. Initiate Replanning* |

The second process of plan-assessment is the *reenact planned actions* function (3.2: $A_{\psi}$). This function uses two input variables: (1) the new plan-remainder ($P''$ described above); and (2) time ($t$ described above). The *reenact planned actions* function outputs two variables: (1) the first time-ordered actions that can be assessed; and (2) a

new plan-remainder that eliminates the first time-ordered action from it's list (i.e., $\{A_1, P''\} = A_\psi(P'', t)$). As actions are assessed, eliminating the assessed action from the plan-remainder shortens it and prepares the plan-remainder either for further assessment or for the plan-generation phase for possible re-planning. This function operates exactly as *execute actions* (2.2: $A_\Xi$) except the model used is different.

The third process of plan-assessment is the *interacting model and state* function (3.3). This process has four sub-processes identified above: (3.3.1) *forecast outcomes* function ($K_Y$); (3.3.2) *select outcomes* function ($Z_Y$); (3.3.3) *observe state changes* function ($O_X$); and (3.3.4) *update plan's model state* function ($X_U$). These four sub-processes behave exactly the same as the *interaction of model and state* function (1.4.1-1.4.4) described in plan-generation. The equations vary slightly because the outcomes, observations, and plans' model state have different values than those used in plan-generation: $Y''$, $O''$, and $X_P''$, respectively. The four functional processes are (3.3.1) $Y'' = K_Y(A_1, \gamma, M_\Psi, X_P)$; (3.3.2) $Y_Z'' = Z_Y(Y'', r_Y)$; (3.3.3) $O'' = O_X(Y_Z'', \gamma, M_\Psi, X_P)$; and (3.3.4) $X_P'' = X_U(\Gamma, O'', t_U)$.

The fourth and final process of plan-assessment is the *assess expectation* function (3.4). This process has three sub-processes identified in Table 6.4 (3.4.1-3.4.3). The first of these three processes is *forecast new expectations* function (3.4.1: $K_Q$). This function uses two input variables: (1) the new plan-remainders ($P''$ described above); and (2) completed tasks-per-plans ($T_P$ described above). The output of the *assess expectations* function is a new set of expectations ($Q''$), one for each plan-remainder ($Q'' = K_Q(P'',$

$T_P$)). This function operates the same as the *forecast expectations* function of plan-generation (1.3: $K_Q$).

The second of the three sub-processes is the *compare expectations* function (3.4.2: $E_Q$). This function uses three input variables: (1) the original expectations computed in plan-generation (*Q* described above); (2) the new expectations computed in the previous process (*Q″* described above); and (3) the expectation acceptance threshold coefficient ($r_Q$). The expectation acceptance threshold coefficient is a real number between zero and one and determines the portion of acceptable task completions necessary to keep existing plans; otherwise re-planning modifies the plans in the next plan-generation phase. Expectations are either a positive integer of tasks completed (e.g., in Chapter 7) or a scoring system that measures the degree of tasks completed (e.g., in Chapter 8). If original expectations multiplied by expectation acceptance threshold coefficient is greater than or equal to the new expectations (i.e., $Q \bullet r_Q \geq Q''$,), then the compare expectations function outputs the new expectations; otherwise the output expectations are reset to zero (i.e., $Q = E_Q(Q, Q'', r_Q)$), and re-planning is triggered.

The third and final sub-process of *assess expectations* (3.4) is the *initiate re-planning* function (3.4.3: $I_R$). This function uses two input variables: (1) the new plan-remainder (*P″* described above); and (2) the reset expectations output from the previous process (*Q* described above). The outputs of the *initiate re-planning* function are new plans, which are passed to the next plan-generation phase (i.e., $P = I_R(P'', Q)$). The new plans are equal to the plan-remainder if the expectations are greater than zero ($Q > 0$);

otherwise, the plans are reset to null and re-planned in the following plan-generation phase.

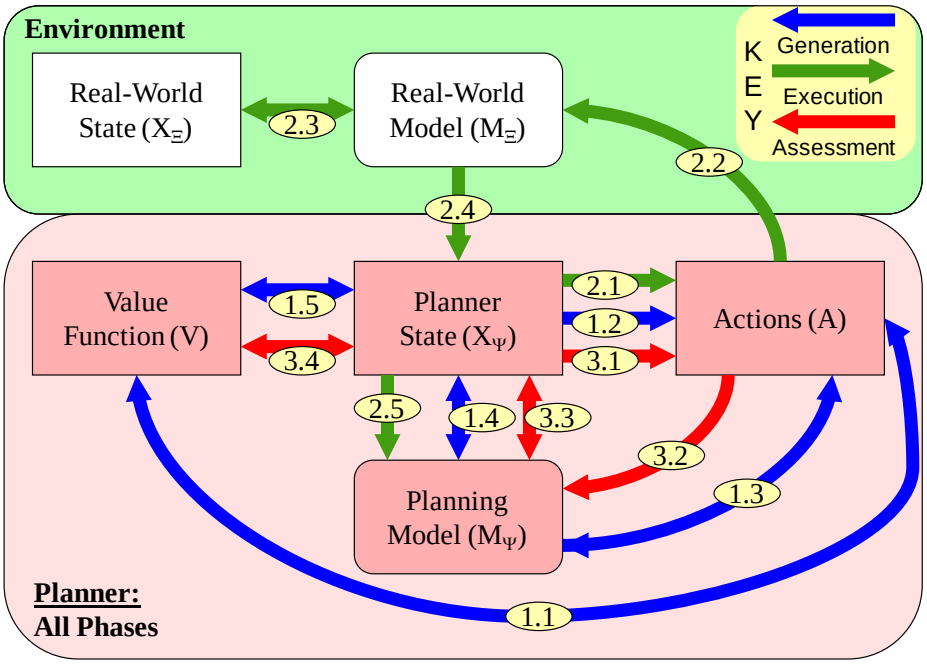


Figure 6.7. Functional Representation of 14 General Processes

As a summary of Tables 6.2 to 6.4, the 14 general processes are shown in Figure 6.7 in a functional representation. Each general process is labeled on an arrow connecting two functional blocks. This figure is a composite of Figures 6.3 - 6.5, and combines all three phases of the planning process.

A summary of all 24 processes for all three phases of planning and their related functions and variables is shown in Table 6.5. This table includes all the sub-processes and corresponding functions and variables. Note that since the search parameters of process 1.1.1 controls 1.2-1.5, these processes occur before 1.1.2 and 1.1.3. However, all of these processes for each phase are iterated until complete, because they reoccur for each action of every agent considered. Phase 1 (1.1-1.5) iterates until all considered

agents have a plan, phase 2 (2.1-2.5) iterates until feedback state function initiates a model update, and phase 3 (3.1-3.4) iterates until all plans are assessed. More detailed description of these functions and variables is provided in Chapters 7 and 8. Section 6.3 provides an ordered development schematic for designing, building and testing these processes for a future application domain.

Table 6.5. Processes for All Phases of Planning

| *Planning Phase* | *General Processes* | *Sub-Processes* | *Functions and Variables* |
|---|---|---|---|
| *Phase 1: Plan-Generation* | *1.1. Search For and Select Plans* | *1.1.1. Search For Plans* | $P = S_P(S_{IP}, T_P, \tau_P, R, u_R)$ |
| | | *1.1.2. Select Plans* | $P_Z = Z_P(F, P)$ |
| | | *1.1.3. Forecast Expectations* | $Q = K_Q(P_Z, T_P)$ |
| | *1.2. Determine Action Choices* | | $A = I_A(\gamma, M_\Psi, X_P)$ |
| | *1.3. Select Actions* | | $A_Z = Z_A(\Gamma, A, H, R, C_A, t)$ |
| | *1.4. Interaction of Model and State* | *1.4.1. Predict Outcomes* | $Y = K_Y(A_Z, \gamma, M_\Psi, X_P)$ |
| | | *1.4.2. Select Outcomes* | $Y_Z = Z_Y(Y, r_Y)$ |
| | | *1.4.3. Observe State Changes* | $O = O_X(Y_Z, \gamma, M_\Psi, X_P)$ |
| | | *1.4.4. Update Plan's Model State* | $X_P = X_U(\Gamma, O, \tau_U)$ |
| | *1.5. Evaluate Plan Fitness* | | $F = F_P(X_P, J_\Psi, Pr(T_P), D)$ |
| *Phase 2: Plan-Execution* | *2.1. Check Plans are Executable* | | $P' = P_\Xi(P_Z, \gamma, M_{IP}, X_P)$ |
| | *2.2. Execute Actions* | | $\{A_I, P'\} = A_\Xi(P', t)$ |
| | *2.3. Realize Outcomes* | | $Y' = Y_R(A_I, \gamma, M_\Xi, X_\Xi)$ |
| | *2.4. Sense Response* | | $O' = O_X(Y', \gamma, M_\Xi, X_\Xi)$ |
| | *2.5. Feedback State* | | $X_{IP} = X_U(\Gamma, O', \tau_U)$ |
| *Phase 3: Plan-Assessment* | *3.1. Check Plans can be Assessed* | | $P'' = P_{II}(P', \gamma, M_{IP}, X_P)$ |
| | *3.2. Enact Planned Actions* | | $\{A_I, P''\} = A_{II}(P'', t)$ |
| | *3.3. Interaction of Model and State* | *3.3.1. Predict Outcomes* | $Y'' = K_Y(A_I, \gamma, M_\Psi, X_P)$ |
| | | *3.3.2. Select Outcomes* | $Y_Z'' = Z_Y(Y'', r_Y)$ |
| | | *3.3.3. Observe State Changes* | $O'' = O_X(Y_Z'', \gamma, M_\Psi, X_P)$ |
| | | *3.3.4. Update Plan's Model State* | $X_P'' = X_U(\Gamma, O'', \tau_U)$ |
| | *3.4. Assess Expectations* | *3.4.1. Forecast New Expectations* | $Q'' = K_Q(P'', T_P)$ |
| | | *3.4.2. Compare Expectations* | $Q = E_Q(Q, Q'', r_Q)$ |
| | | *3.4.3. Initiate Replanning* | $P = I_R(P'', Q)$ |

An alternative description of Figure 6.7 is shown in Figure 6.8, where functional processes are drawn as boxes rather than arrows. Figure 6.8 also includes all 24 sub-processes of the core planning cycle (instead of only the 14 main processes), and the hierarchy is referenced as well. Each row of functional boxes corresponds to a hierarchical level from agents to observations (levels 5 to 10). Note that for the core

planning cycle tournaments to planners (levels 1 to 4) is not included. Also note that the 24 functions in Figure 6.8 map directly to the functions shown in Table 6.5. However, the functions are labeled from 5 to 28 instead of 1.1.1 to 3.4.3. The names of the functional processes are labeled in a more simplified form than in Table 6.5. In Figure 6.8, planning model, and real world model are shown with the functions that use them. The value functions are separated from these planning functions, and action functions are now represented as a level in the hierarchy. This alternative representation of the core planning cycle is used here to help introduce the next subsection, Training, which describes the higher-level portions of the ADP&E architecture. Chapter 9 will use this representation exclusively to map functional requirements directly to future planning applications.

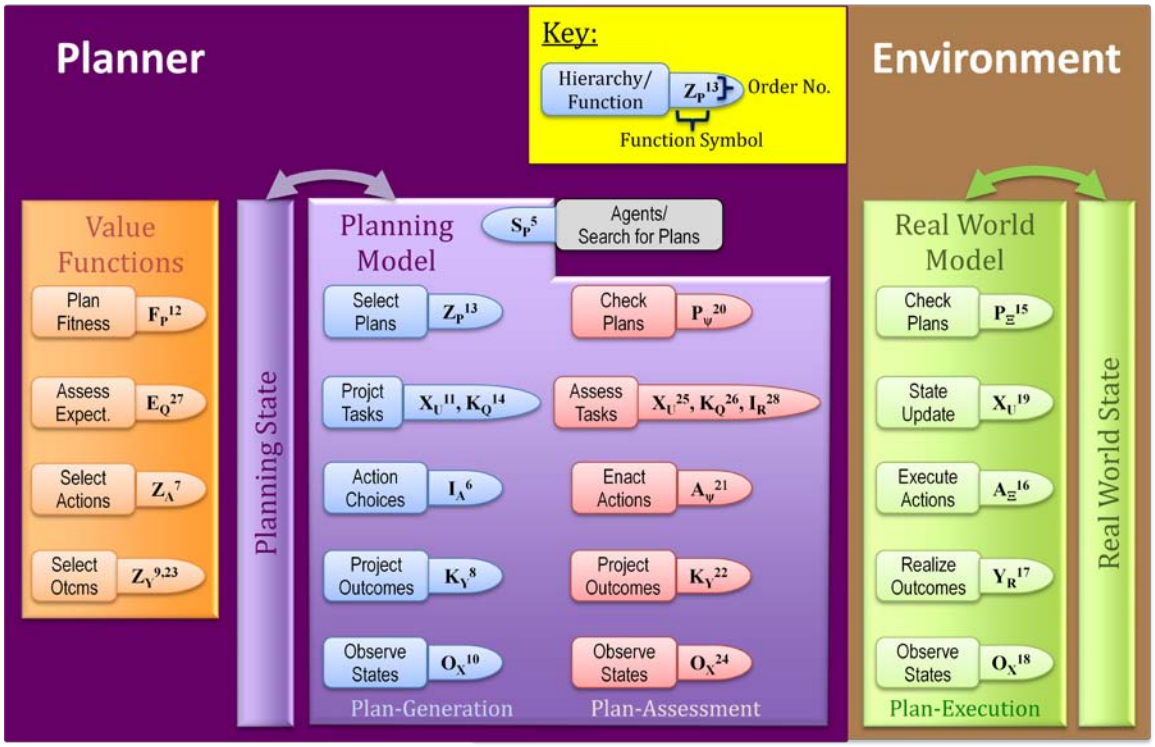


Figure 6.8. ADP&E Architecture for Core Planning Cycle

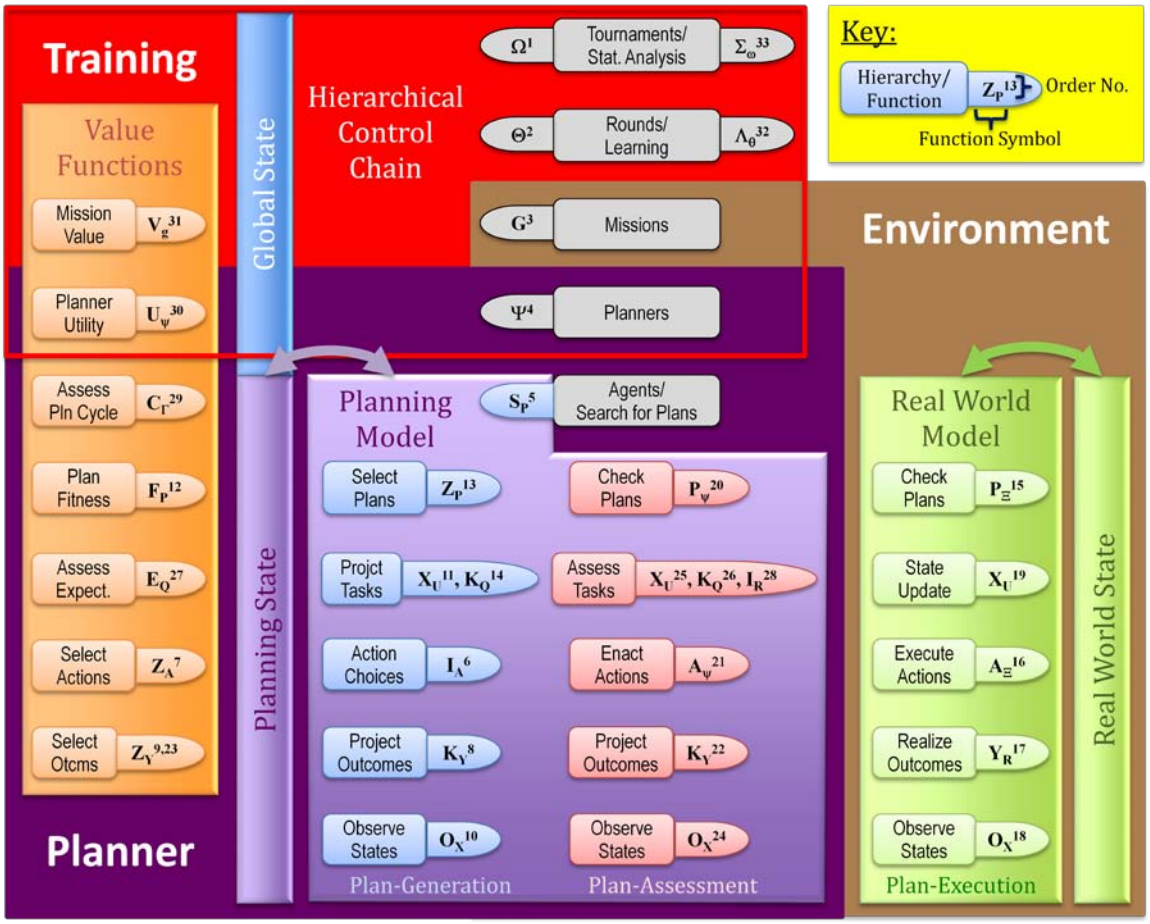


Figure 6.9. Overall ADP&E Architecture

### 6.2.4 Training

The full architecture is shown in Figure 6.9. The three main areas of the architecture are the planner, the environment, and training. The planner and the environment compose the core planning cycle described in the previous sections. As introduced in Section 6.1, training is composed of a goal-directed HCC that coordinates multiple rounds, missions, and planners, and a data-driven L&A that aggregate results and improve planners' capabilities. The global state, which overlaps training and the planner, keeps track of the HCC variables, such as the particular tournament, round, mission or planner under consideration. Also, one portion of the value function in

training is the mission value function that computes the value of each planner in every corresponding mission or game. The planner is composed of the value function, planning model, and planner state. The environment includes the real world model and real world state.

There are 24 functions in the CPC (described in previous subsection), 4 functions in the HCC, and 5 functions in the L&A totaling 33 functions for the ADP&E system. These functions are labeled in order of flow in both Figures 6.9 and 6.10. In Figure 6.9, the function symbols are labeled in the bullets and connected to boxes giving the functional names. In Figure 6.10, functions are labeled along the left side of the page with their corresponding symbols.

First, Figure 6.9 will be examined more closely. The first four functions begin with the HCC: tournaments ($\mathbf{\Omega}$), rounds ($\mathbf{\Theta}$), missions ($\mathbf{G}$), and planners ($\mathbf{\Psi}$). The last five functions end with L&A: planning cycle assessment ($\mathbf{C_{\Gamma}}^{29}$), planner utility ($\mathbf{U_{\psi}}^{30}$), planner value ($\mathbf{V_g}^{31}$), evolutionary learning ($\mathbf{\Lambda_{\theta}}^{32}$), and statistical analysis ($\mathbf{\Sigma_{\omega}}^{33}$). The makeup of the planner is defined by the planner parameters, which control all aspects of the CPC: plan-generation, plan-execution, and plan-assessment. Plan-generation has seven different functions (blue background) in six different levels of the hierarchy ($\mathbf{S_P}^{5}$, $\mathbf{I_A}^{6}$, $\mathbf{K_Y}^{8}$, $\mathbf{O_X}^{10}$, $\mathbf{X_U}^{11}$, $\mathbf{Z_P}^{13}$, $\mathbf{K_Q}^{14}$). Plan-execution has five different functions in five different levels of the hierarchy ($\mathbf{P_{\Xi}}^{15}$, $\mathbf{A_{\Xi}}^{16}$, $\mathbf{Y_R}^{17}$, $\mathbf{O_X}^{18}$, $\mathbf{X_U}^{19}$). Plan-assessment has seven different functions used in five different levels of the hierarchy ($\mathbf{P_{\Psi}}^{20}$, $\mathbf{A_{\Psi}}^{21}$, $\mathbf{K_Y}^{22}$, $\mathbf{O_X}^{24}$, $\mathbf{X_U}^{25}$, $\mathbf{K_Q}^{26}$, $\mathbf{I_R}^{28}$). The value function has seven different value functions used in seven different levels of the hierarchy. As described earlier, the value function and plan-

generation and -assessment share functionality. More specifically, outcome selection ($\mathbf{Z_Y}^{9}$), action selection ($\mathbf{Z_A}^{7}$), and plan fitness ($\mathbf{F_P}^{12}$) functions are used in plan-generation, and outcome selection ($\mathbf{Z_Y}^{23}$), and assess expectations ($\mathbf{E_Q}^{27}$) functions are used in plan-assessment. The functional details of all CPC components, and the order of these operations have already been described in Section 6.2.

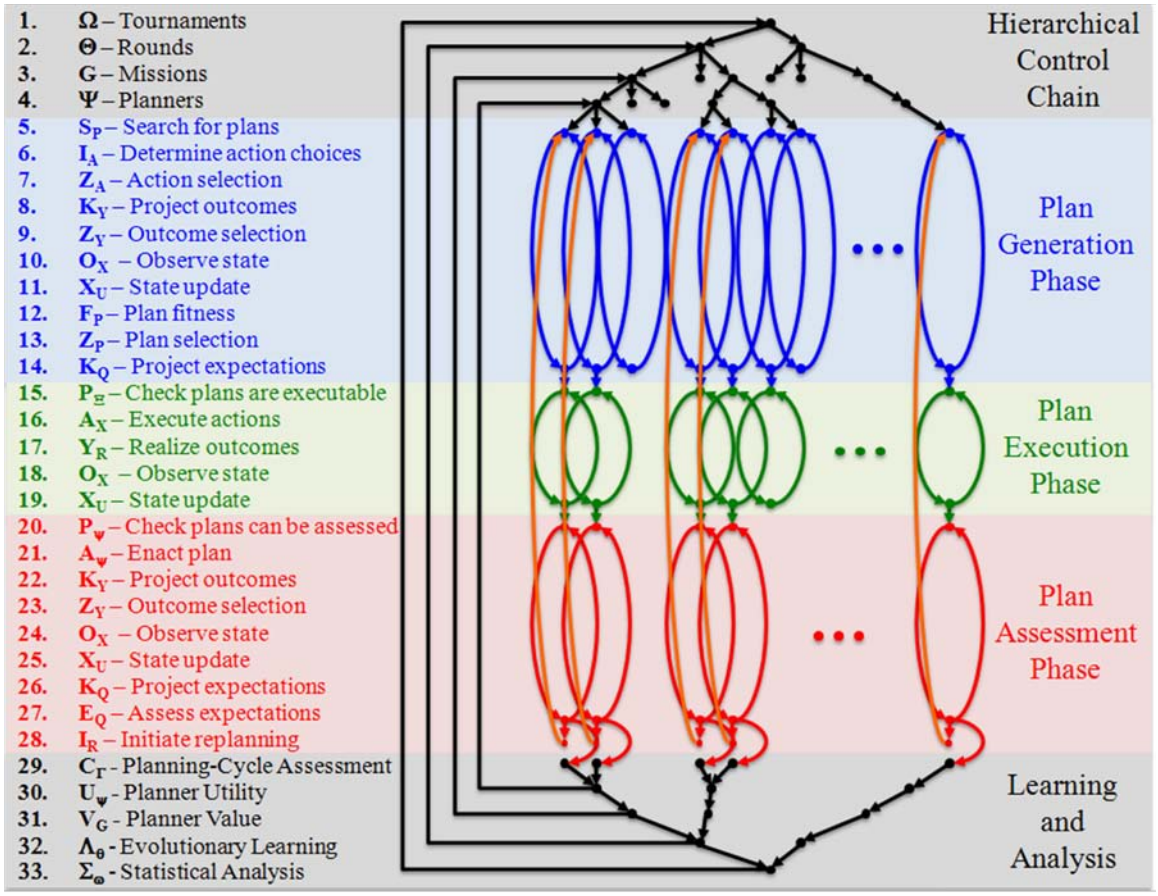

Figure 6.10. Overall ADP&E Process Flow Diagram

Figure 6.10 shows the flow of information throughout the system as it relates to the five phases of the system (hierarchical control chain, plan-generation, plan-execution, plan-assessment, and learning and analysis) and 33 functions. Note that the HCC expands from a single tournament to multiple rounds, to a greater number of missions, etc. This

expansion leads to generation of multiple plans for multiple agents, where some plans are considered and selected while others are pruned or thrown away. Once plans are selected, they are iteratively executed and assessed in the plan-execution and plan-assessment phases. Once plans are complete or re-planning is triggered, plan-generation begins again. The CPC process then iterates until a mission is complete or a game produces a winner. Note that multiple planners can be operated at once or on a turn-basis, based on rules and constraints of the mission or game.

## 6.3 Design, Build and Test

The detailed description above of the CPC (i.e., plan-generation, -execution, and -assessment) can be daunting when building this goal-directed hierarchy for a new application. However, an incremental design, build and test (DB&T) approach can be used to reduce this challenge into manageable parts. The DB&T approach can be divided into nine incremental steps: (1) **execute arbitrary actions**; (2) **update planning-model**; (3) **choose and execute single actions**; (4) **generate arbitrary multi-action plans**; (5) **generate and choose plans**; (6) **assess plans**; (7) coordinate using the full **core planning cycle** (i.e., CTC: generate, execute, and assess) for a single planner; (8) coordinate a multiple planner ADP&E system for **a single game or mission**; and (9) implement **a complete tournament** meta-learning strategy for training planners to autonomously improve performance.

The first DB&T procedure is to construct an **execute arbitrary actions** cycle. This requires the development of the first four processes of the plan-execution phase (2.1-2.4) as shown in Figure 6.11. The first step is to implement the real-world model for the

application ($M_{\Xi}$) as a finite state machine. All variables in the finite state model are defined as state variables and define the real-world state. For this step, it is assumed that the real-world model is fully observable, so that the planning-state is an exact copy of the real-world state. All variable choice blocks in the finite state machine are defined as action choices in the model. For simplicity, processes 2.1-2.4 only consider single action plans, where the actions are chosen randomly from the set of all possible actions. This cyclical **execute arbitrary actions** procedure tests two important features of the overall architecture: (1) does the game or real-world simulation play out according to the rules of the finite state machine; and (2) does the game or simulation run autonomously and indefinitely.

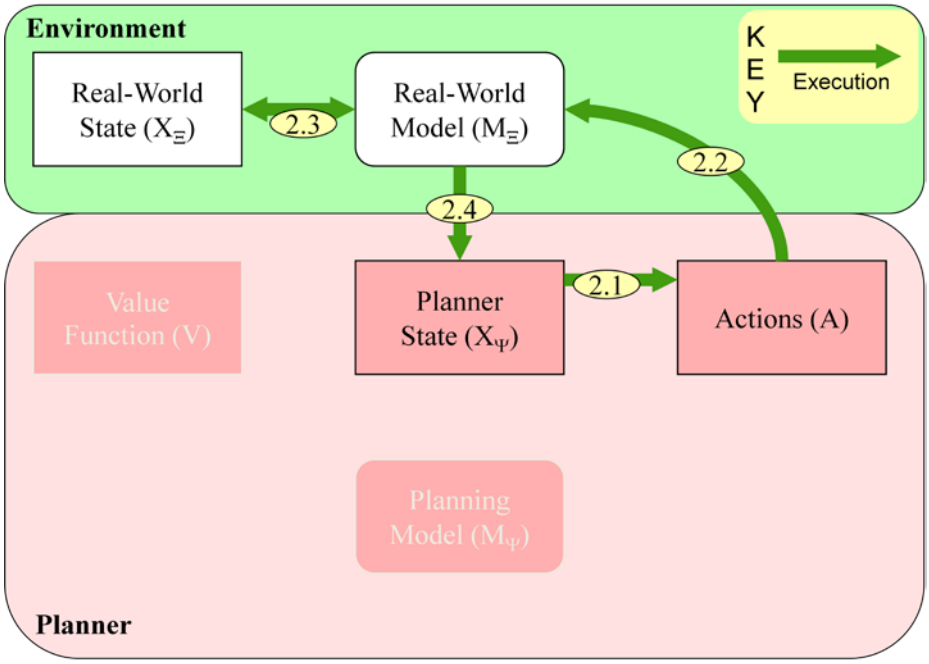


Figure 6.11. Execute Arbitrary Actions

The second procedure is to finish the entire plan-execution cycle, which includes **update planning-model**. This requires inclusion of the final process of the plan-

execution phase (2.5) as shown in Figure 6.12. In this build, the environment or real-world model is not considered observable. Thus, a planning-model is constructed with the same finite state machine structure as the real-world model, but some of the planning-model states are chosen randomly within the limits imposed by the application. Thus, the planning-state is not an exact copy of the real-world state. Running this build and **execute arbitrary actions** tests whether observations can be updated to the planning-model.

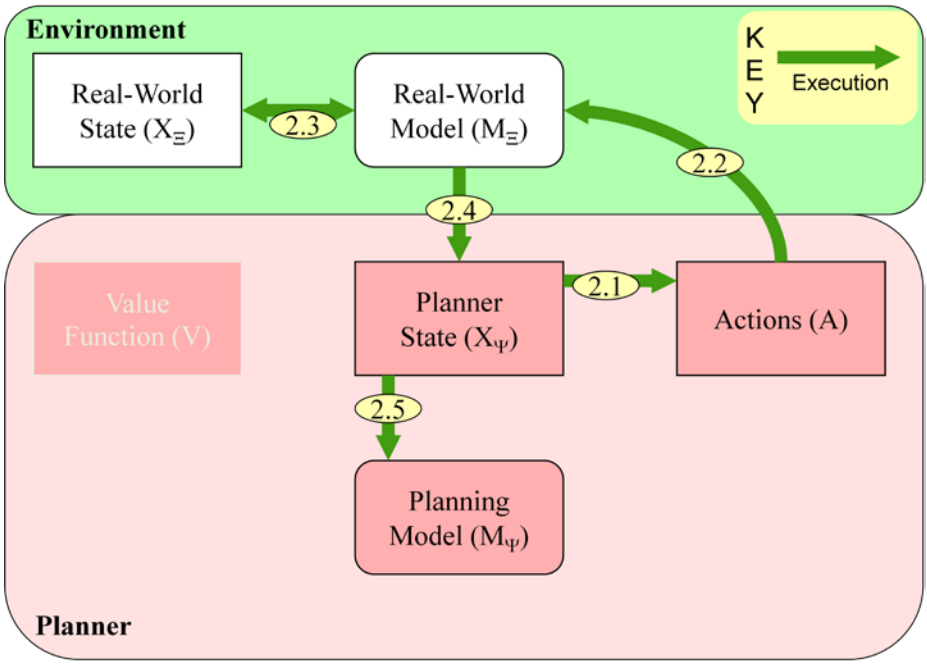


Figure 6.12. Update Planning Model

The third procedure is to construct a **choose and execute single actions** cycle. This cycle still only considers plans as single actions. The procedure requires part of the existing plan-execution phase with the addition of part of the plan-generation phase as shown in Figure 6.13. A *select actions* function (1.3) portion of the value function block is created for choosing single action plans, and the second two parts of *select plans* (1.1.2) and *forecast expectations* (1.1.3) are created for the *search for and select plans*

function (1.1). The *determine action choices* function (1.2) is created by modifying the existing *check plans are executable* function (2.1). This procedure tests the coordination of using plan-generation with plan-execution.

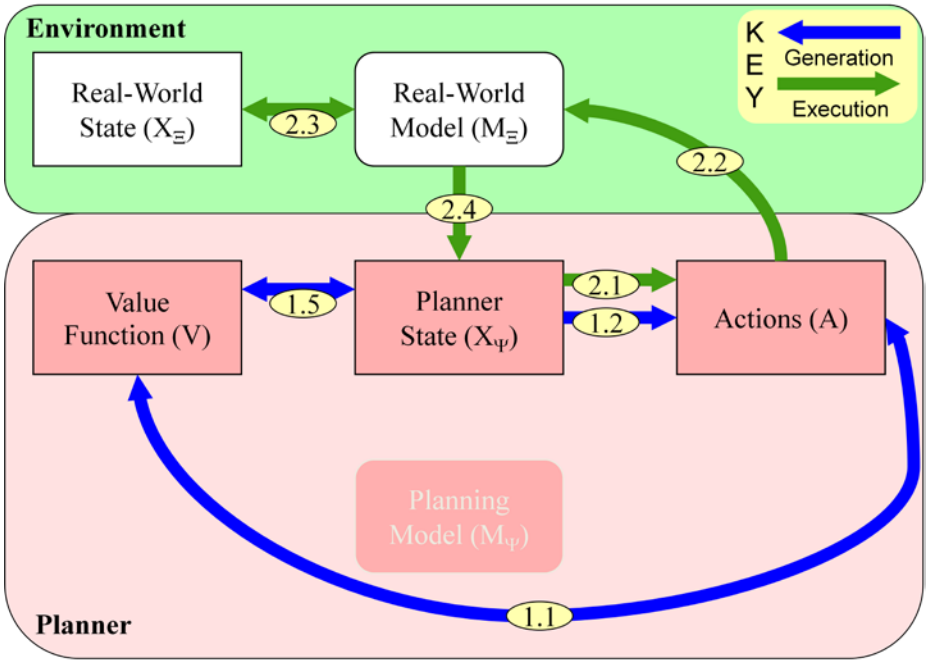


Figure 6.13. Choose and Execute Single Actions

The fourth procedure is to construct the ability to **generate arbitrary multi-action plans** cycle for plan-generation. This cycle now considers plans of variable length and includes the heart of the plan-generation process: (1) *determine action choices* (1.2), (2) *select actions* (1.3), and (3) *interactions of model and state* (1.4) as shown in Figure 6.14. When built, the *select actions* function tests use of resources, choosing SME feature conditions, action costs, and choosing actions in a time sequential order. The *interaction of model and state* function tests selecting single outcomes when there are multiple outcomes per action. Also, the overall **generate arbitrary multi-action plans** cycle tests handling of multiple observation states for each action in a plan.

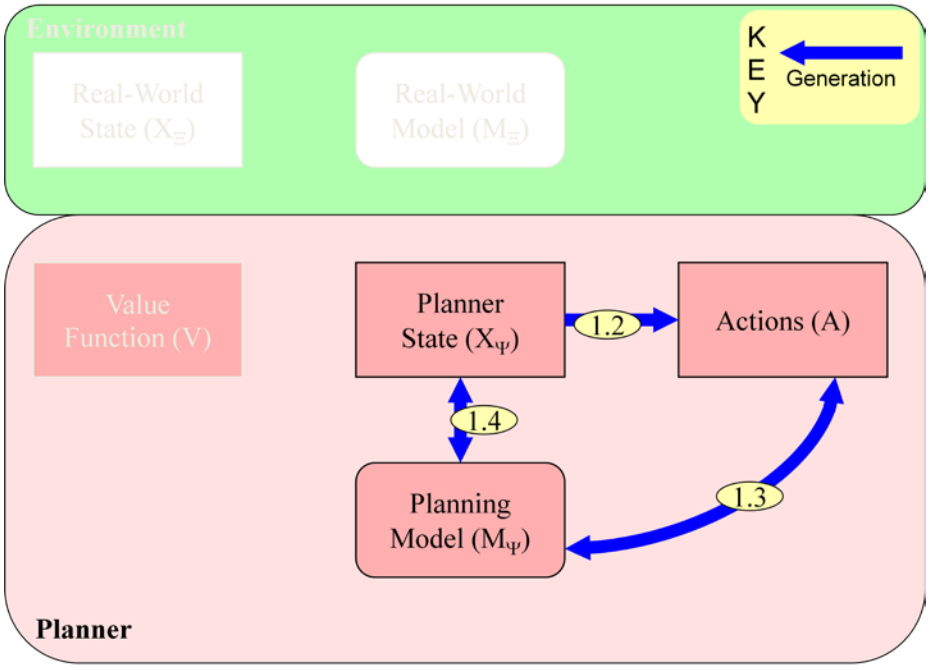


Figure 6.14. Generate Arbitrary Multi-Action Plans

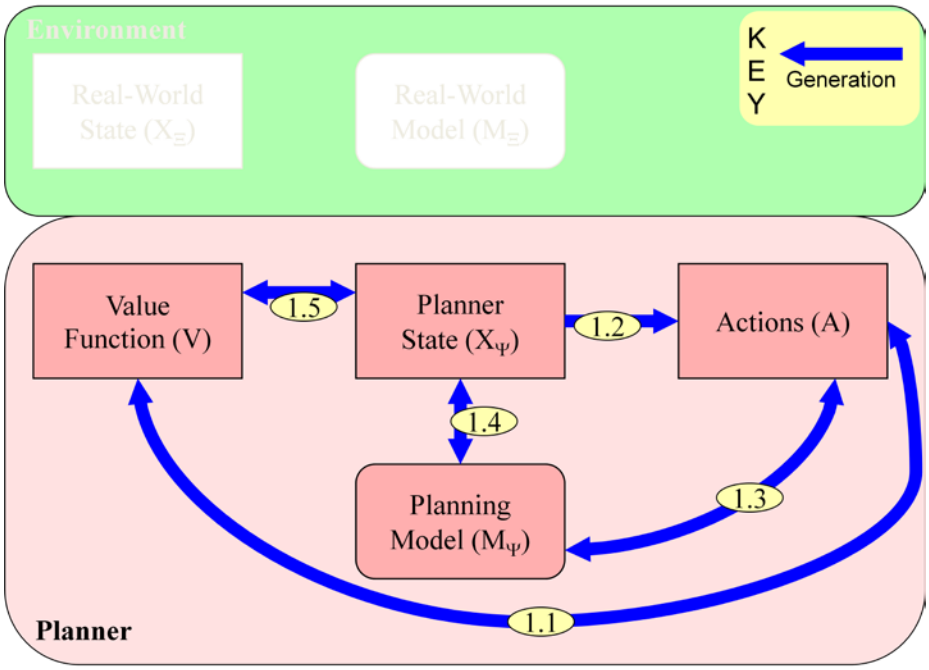


Figure 6.15. Generate and Choose Plans

The fifth procedure is to construct the entire plan-generation planning cycle, **generate and choose plans**. This cycle now considers all parts of plan-generation,

including *search for and select plans* function (1.1) as shown in Figure 6.15. The *search for plans* function (1.1.1), a sub-process of search for and select plans function, tests the use of tasks and resources to decide the lengths of plans, and the use of the search parameters to control processes 1.2 through 1.5. The *select plans* function (1.1.2) is tested to ensure it handles multiple plans per agent, compares the plans, and select them based on fitness. The *forecast expectations* function (1.1.3) is tested to see if expectations can be calculated for selected plans.

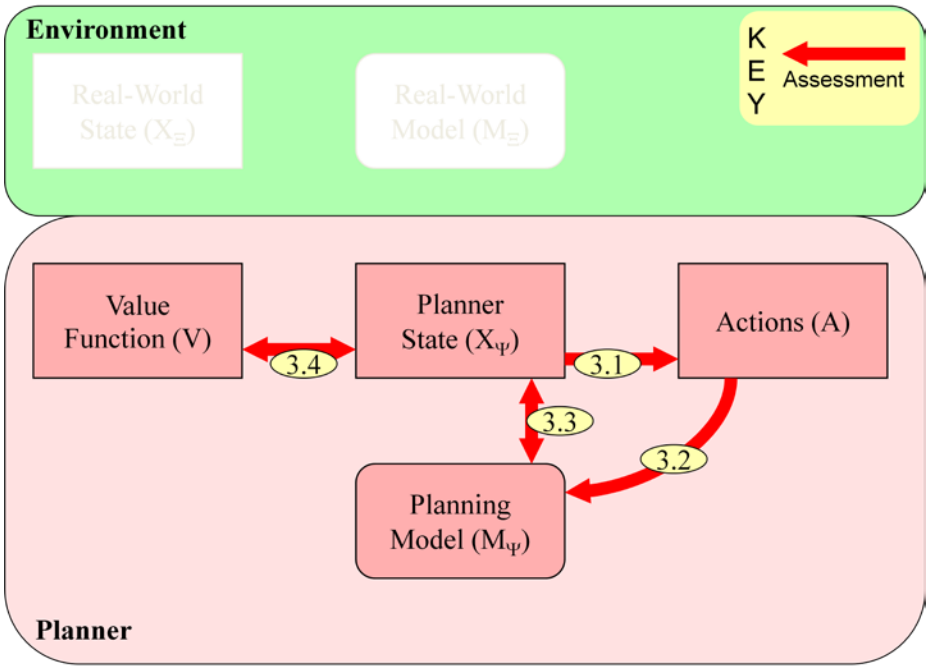


Figure 6.16. Assess Plans

The sixth procedure is to construct the entire plan-assessment planning cycle, **assess plans**, and is shown in Figure 6.16. This cycle has duplicate parts of already built designs on three of the four processes (3.1-3.3). Processes 3.1, 3.2, and 3.3 are duplicates of processes 2.1, 2.2, and 1.4 with minor modifications, respectively. The only new part

in this cycle is the *assess expectations* function (3.4). This function is tested to see if plans' expectations can be compared to determine if re-planning is necessary.

The seventh procedure is to combine the previous six builds into one to coordinate all three phases of the planning process, the **core planning cycle** (CPC), as shown previously in Figure 6.7. This system requires having all three planning phase cycles working in a coordinated way. This build tests whether there is a correct exchange of data and smooth transition among the three phases: (1) plan-generation to plan-execution; (2) plan-execution to plan-assessment; and (3) plan-assessment to plan-generation.

The eighth procedure would coordinate the complete operation of **a single game or mission**. This procedure is to include multiple planners if the application requires them. Another planner would operate the same as any other, except that a planner may have different agents at its disposal, a different value function for choosing actions and/or plans, and a different assessment function to assess progress. Figure 6.17 shows an illustration of a two planner functional representation. The application RISK in Chapter 7 will discuss a problem that uses eight players or planners.

The ninth procedure is to implement **a complete tournament** using a meta-learning strategy for training planners to autonomously improve performance. The strategy used in this dissertation is an evolutionary computation approach. Details of the procedures used are application dependent and described in Chapters 7 and 8. Note that using evolutionary computation methods is not a necessary technique to improve planners' behavior, but is shown to be sufficient to improve planners' abilities through

autonomous interaction. Our approach is not restricted to using only this technique. In other words, other techniques such as multi-objective optimization [6] [7] [8] can be applied.

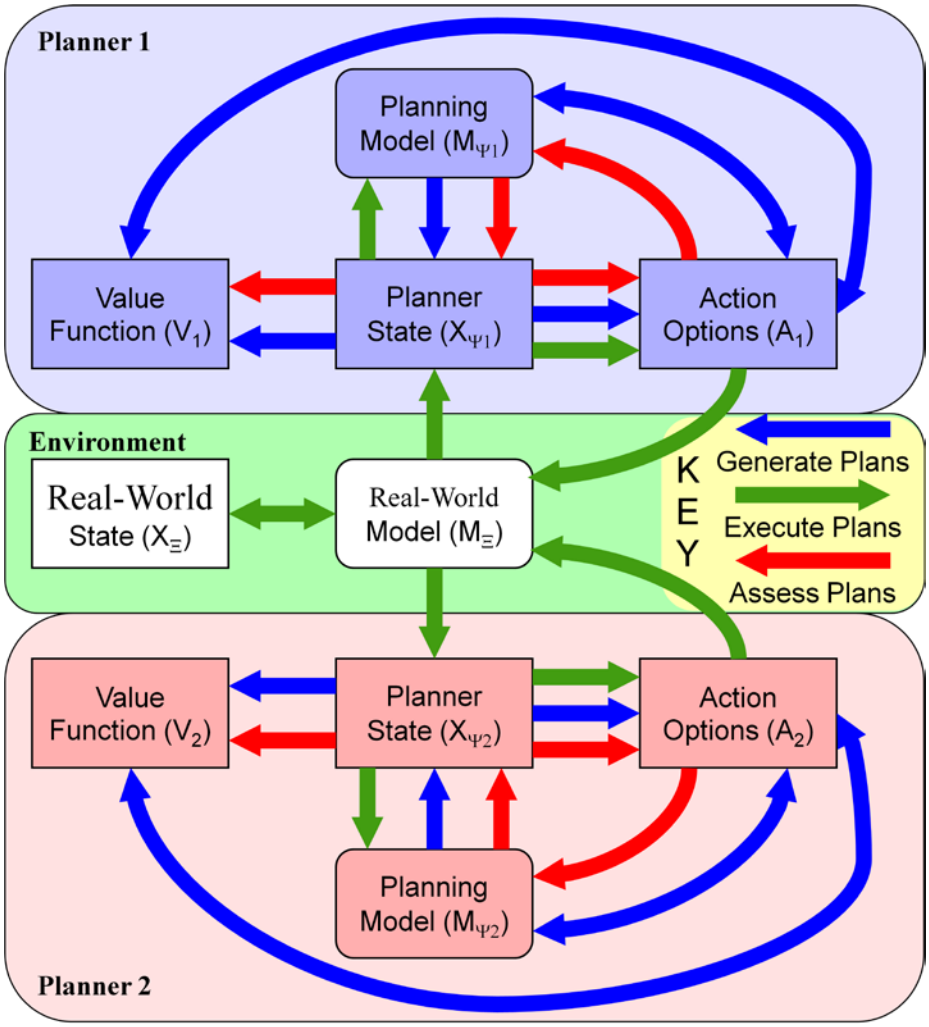


Figure 6.17. Two-Planner Functional Block Diagram

This DB&T will be reiterated in Chapters 7, 8, and 9. Chapters 7 and 8 will provide a detailed description of two specific applications. Chapter 9 will provide more insight into DB&T for future applications based on the greater understanding achieved in Chapters 7 and 8. Chapter 9 will describe the DB&T from a broader perspective as shown

in Figure 6.9. All the detailed CPC processes described in this chapter provide all the necessary and sufficient terms and definitions to understand the implementations in Chapters 7 and 8, and provide the underpinnings to build upon in Chapter 9.

**References:**


1. R. Levinson, F. H. Hsu, J. Schaeffe, T. A. Marsland, & D. E. Wilkins, “The role of chess in artificial-intelligence research,” *ICCA Journal*, V. 14, N. 3, 1991, pp. 153-161.

2. K. Chellapilla and D.B. Fogel, "Anaconda defeats Hoyle 6-0: a case study competing an evolved checkers program against commercially available software," *Proceedings of the 2000 Congress on Evolutionary Computation*, IEEE Press, Piscataway, NJ, 2000, pp. 857-863.

3. G. Tesauro. “Programming Backgammon Using Self-Teaching Neural Nets,” *Artificial Intelligence 134 (1)* 2002. pp. 181-199.

4. B. Bouzy and B. Helmstetter. “Developments on Monte Carlo Go,” *Advances in Computer Games 10*, 2003.

5. A. Davidson, D. Billings, J. Shaeffer, D. Szafron. “Improved Opponent Modeling in Poker,” *Proceedings of the 2000 International Conference on Artificial Intelligence*, 2000. pp. 1467-1473.

6. I. Das and J. E. Dennis. Normal-Boundary Intersection: A New Method for Generating the Pareto Surface in Nonlinear Multicriteria Optimization Problems. SIAM Journal on Optimization, 8:631–657, 1998.

7. A. Messac and A. Ismail-Yahaya and C.A. Mattson: The normalized normal constraint method for generating the Pareto frontier. Structural and multidisciplinary optimization, vol. 25, no2, pp. 86-98, 2003.

8. Daniel Mueller-Gritschneder, Helmut Graeb and Ulf Schlichtmann: A Successive Approach to Compute the Bounded Pareto Front of Practical Multiobjective Optimization Problems. SIAM Journal on Optimization, Volume 20, Issue 2, pp. 915-934, 2009.

# Chapter 7

# RISK Game Application

## 7.0 Abstract

The problem domain chosen here is the game RISK [1]. This chapter will describe the game, provide implementation details of an automated planning player, and present results. The game will be presented in terms of the ADP&E goal-directed hierarchy. Implementation details are described in the nine-step design, build and test (DB&T) strategy. The results are evaluated using three key areas of value: plan fitness, planning cycle assessment, and planner utility. The hierarchy, DB&T, and value function definitions and details are described in Chapter 6.

## 7.1 General Hierarchical Description

As described in Chapter 6, the goal-directed hierarchy is composed of ten levels (see Table 6.1). The order of the RISK description is levels 3 through 10 first, followed by level 2 (rounds), and finally level 1 (tournaments).

### 7.1.1 Games or Missions (*G*)

RISK is a non-cooperative stochastic game. Unlike chess, checkers, backgammon, or GO, RISK can have more than two players. Each player makes multiple moves per turn and many of the moves have stochastic outcomes based on multiple dice rolls [2]. The objective of a RISK game is to conquer the world [1] [2] [3] [4]. There are

forty-two territories on the board, a set of five dice for battling forces, and a deck of cards (44: one for each territory and two wildcards) for accumulating bonus forces when a three-card match is reached. Figure 7.1 provides a sample view of a game board with eight players after initial territories are picked and each player's remaining forces are placed on the board. The implemented game is the classic RISK game [1], including Hasbro II rules which handles up to eight players [1], In this game, each turn in of cards yields greater number of force allocations (i.e., 4, 6, 8, 10, 12, 15, 20, 25, etc.). Also, when cards are turned in, two forces are placed on each territory for the current player that matches the territory on a card.

At the start of the game, each player is allocated a number of forces depending on the number of players: for two players, 40 forces each; for three players, 35 forces each; for four players, 30 forces each; for five players, 25 forces each; and for six, seven, or eight players, 20 forces each. Next, each player picks one of forty-two territories in turn until all forty-two territories have a single force on it from a player. Depending on the number of players and the order of selection, each player will have a different number of territories. Specific territory selection is as follows for '2' to '8' players, respectively: for '2' players, every player has twenty-one territories; for '3', every player has fourteen territories; for '4', two players have eleven territories and two players have ten territories; for '5', two players have nine territories and three players have eight territories; for '6', every player has seven territories; for '7', every player has six territories; and for '8', two players have six territories, and six players have five territories. For implementation simplicity, territory selection here is done randomly for all players. Table 7.1 shows the

number of combinations considering the number of players and selection among forty-two territories.

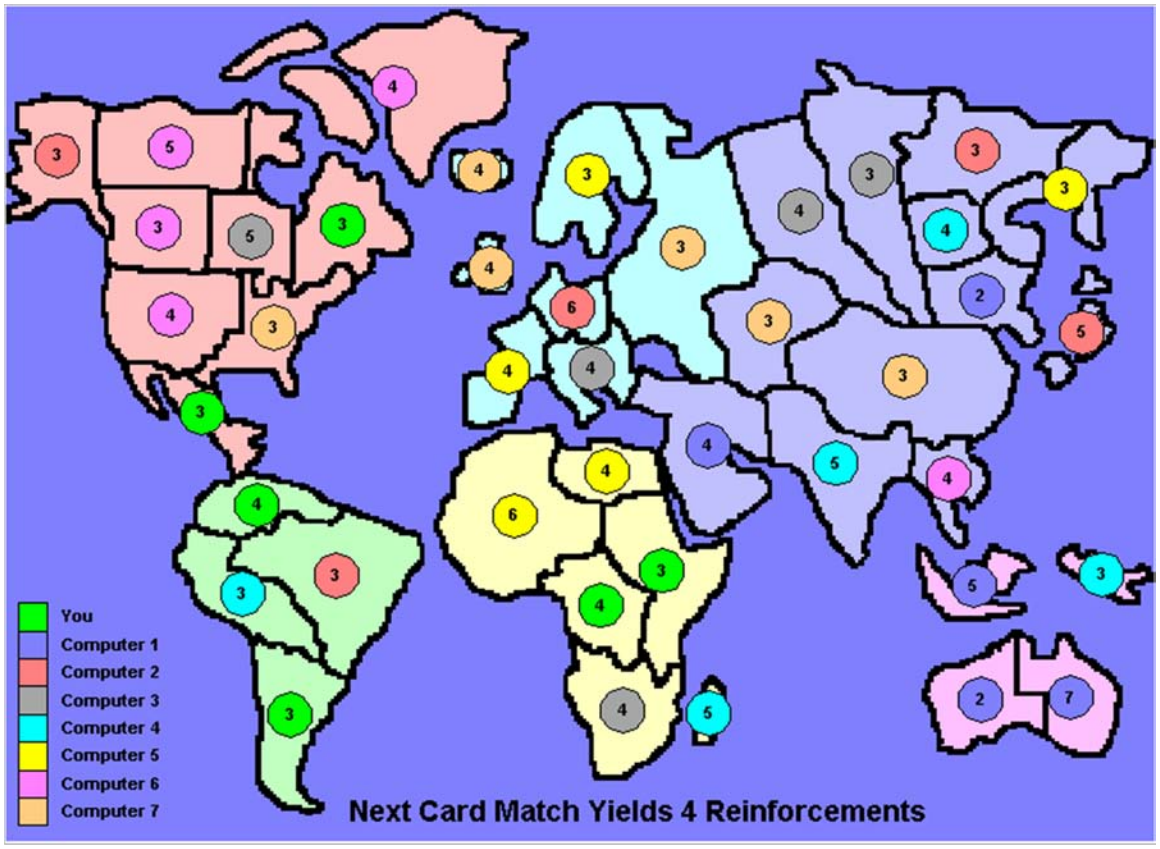


Figure 7.1. A Random Initial Board Setup for Eight Players

Each player allocates their remaining forces on any territory they occupy, such as placing all remaining forces on one territory or placing nearly equal forces on every occupied territory. The total combinations of force allocations depend on the number of forces left after territory selection. Specific force allocations are as follows for (2) to (8) players, respectively: (2) each player allocates nineteen remaining forces; (3) every player allocates twenty-one remaining forces; (4) two players allocate nineteen remaining forces and two players allocate twenty remaining forces; (5) two players allocate sixteen remaining forces and three players allocate seventeen remaining forces; (6) every player

allocates thirteen remaining forces; (7) every player allocates fourteen remaining forces; and (8) two players allocate fourteen remaining forces, and six players allocate fifteen remaining forces. In the implemented version, allocations are selected based on a fitness function described later in this chapter. Table 7.2 shows the number of combinations, considering the number of players and remaining forces not yet allocated.

Table 7.1. Game Starting Territory Selection Combinations

| players | territory selection combinations | # of combs. |
|---|---|---|
| 2 | $42!/(21!)^2$ | 5.38E+11 |
| 3 | $42!/(14!)^3$ | 2.12E+18 |
| 4 | $42!/((10!)^2 \bullet (11!)^2)$ | 6.70E+22 |
| 5 | $42!/((8!)^3 \bullet (9!)^2)$ | 1.63E+26 |
| 6 | $42!/(7!)^6$ | 8.57E+28 |
| 7 | $42!/(6!)^7$ | 1.40E+31 |
| 8 | $42!/((5!)^6 \bullet (6!)^2)$ | 9.08E+32 |

Table 7.2. Game Starting Territory Allocation Combinations

| players | # of forces | territory allocation combinations | # of combs. |
|---|---|---|---|
| 2 | 40 | $39!/(19! \bullet 20!)^2$ | 4.75E+21 |
| 3 | 35 | $34!/(21! \bullet 13!)^3$ | 7.99E+26 |
| 4 | 30 | $29!/(20! \bullet 9!)^2 \bullet 29!/(19! \bullet 10!)^2$ | 4.02E+28 |
| 5 | 25 | $24!/(17! \bullet 7!)^3 \bullet 24!/(16! \bullet 8!)^2$ | 2.24E+28 |
| 6 | 20 | $19!/(13! \bullet 6!)^6$ | 3.99E+26 |
| 7 | 20 | $19!/(14! \bullet 5!)^7$ | 2.87E+28 |
| 8 | 20 | $19!/(15! \bullet 4!) \bullet 19!/(14! \bullet 5!)^2$ | 4.58E+29 |

Territory selections and force allocations comprise the beginning setup of a game. The total combinations of opening game states are shown in Table 7.3. This dissertation concentrates on the eight-player game, where there are $4.16 \times 10^{62}$ possible starting states.

Table 7.3. Game Starting Territory Selection and Allocation Combinations

| players | territory selections and allocations | # of combs. |
|---|---|---|
| 2 | $42!/(21!)^2 \bullet 39!/(19! \bullet 20!)^2$ | 2.56E+33 |
| 3 | $42!/(14!)^3 \bullet 34!/(21! \bullet 13!)^3$ | 1.69E+45 |
| 4 | $42!/((10!)^2 \bullet (11!)^2) \bullet 29!/(20! \bullet 9!)^2 \bullet 29!/(19! \bullet 10!)^2$ | 2.69E+51 |
| 5 | $42!/((8!)^3 \bullet (9!)^2) \bullet 24!/(17! \bullet 7!)^3 \bullet 24!/(16! \bullet 8!)^2$ | 3.65E+54 |
| 6 | $42!/(7!)^6 \bullet 19!/(13! \bullet 6!)^6$ | 3.42E+55 |
| 7 | $42!/(6!)^7 \bullet 19!/(14! \bullet 5!)^7$ | 4.02E+59 |
| 8 | $42!/((5!)^6 \bullet (6!)^2) \bullet 19!/(15! \bullet 4!) \bullet 19!/(14! \bullet 5!)^2$ | 4.16E+62 |

### 7.1.2 Players or Planners ($\Psi$)

RISK is a multi-player game that can have eight players (i.e., $\Psi = \{\psi_1, \psi_2, \psi_3, \psi_4, \psi_5, \psi_6, \psi_7, \psi_8\}$). As in many games, the RISK game has turns, and each player takes their turn in a constant cyclic order until the game is complete. Each turn is composed of four phases: (I) turning in cards, (II) reinforcing troops, (III) battling opponents, and (IV) moving forces. The rules and procedures for all four phases of a turn are shown in Figure 7.2. Shaded shapes represent decisions imposed by the rules of the game, while white shapes represent decisions made by a player. The diamond-shaped boxes represent yes/no, or binary, decisions while the rectangular boxes represent decisions that have multiple choices.

### 7.1.3 Agents ($\Gamma$)

There are only two agent types in the game RISK. One agent type ($\Gamma_1$) is each card a player holds in their hand, and the other agent type ($\Gamma_2$) is each force located on the game board (Figure 7.1). The number of cards held can vary from zero to six to begin or end a turn and from zero to eleven during a turn (i.e., $\Gamma_1 = \{\gamma_{1.1}, \gamma_{1.2}, ..., \gamma_{1.11}\}$). The

number of forces a player has can vary from zero to many, typically between twenty and a hundred (e.g., $\Gamma_2 = \{\gamma_{2.1}, \gamma_{2.2}, ..., \gamma_{2.100}\}$). A player with zero forces surrenders their cards to the player that attacked them, and is eliminated from the game.

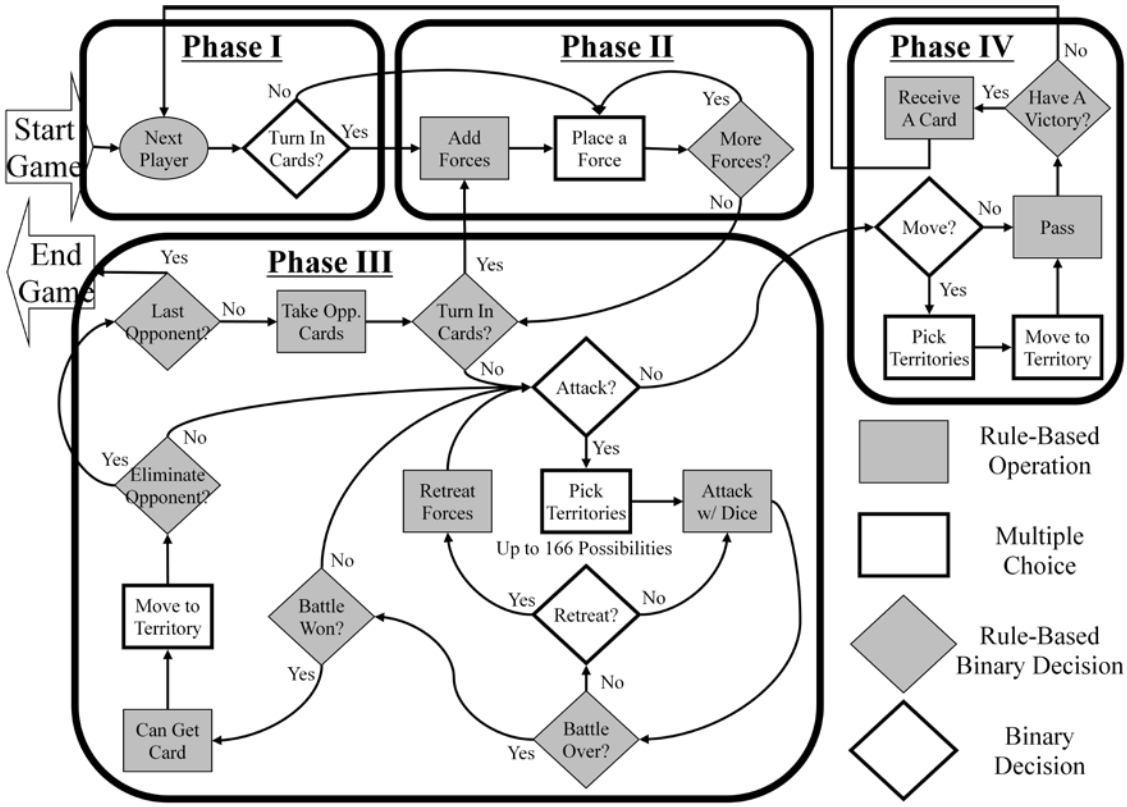


Figure 7.2. Functional Flow Diagram of RISK Game Rules

Agents are controlled as shown in Figure 7.2. For instance, the turn-in-cards phase has some options for when a player can turn in cards. Turning in cards is required at the beginning of a turn when a player has five or more cards, but a player with three or four cards can choose not to turn in cards even if they have a match. Postponing a turn-in of cards can lead to more forces being available for the next turn, but also makes a player more vulnerable to elimination. For instance, if the player is weak and has limited forces, they can be eliminated before their next turn. A player who takes out an opponent

receives all the cards of the eliminated player (phase III), thus a player with cards is more likely to be attacked. Note that, as opponents are eliminated, turning in cards can be done multiple times in a turn. If the number of cards received from another player combined with the current player's cards, exceeds five, all matches must be turned in until that player has four or fewer cards.

Forces can be controlled in many ways as shown in Figure 7.2. Forces are controlled by choosing: (1) where to place forces at the beginning of a turn or when cards are turned in (phase II); (2) which territories to attack and from where (i.e., phase III); (3) when to retreat a single attack (phase III); (4) when to stop attacking new territories (phase III) and pass to phase IV, and (5) how many forces to move after a victorious attack and at the end-of-turn final move (phase IV). Controlling forces represents the brunt of complexity for the game RISK. Thus, controlling forces represents a majority of the plans discussed next.

### 7.1.4 Plans (*P*)

For the game RISK, a plan (*p*) might be considered as anything from a single action to the complete orchestration of multiple turns of all players. Given the complexity and variability of a single turn, and a player's inability to control an opponent's turn, a plan is considered here as a set of ordered actions chosen to complete the current player's turn in a game. In other words, the depth of planning multiple players or turns ahead restricts the ability to explore the breadth of possible actions a player can contemplate in a single turn. Thus, a single player's turn is used as the extent of a plan, so that during

plan generation a player can concentrate on establishing a good situation before passing control to other players.

As shown in Figure 7.2, plan length can be considered in terms of individual actions (each choice taken in a multiple choice box) or in terms of the number of phases (each transfer across phases). This dissertation considers both, and does not restrict plan length based on the number of actions or phases. In terms of phases, the minimum length of a plan is one, because if a player is holding less than five cards, this player could pass on Phase I, place his forces in Phase II, and pass on Phases III, and IV. The maximum length of a plan in terms of phases is fifteen. More specifically, if eight players are currently in a game, and one player eliminates all seven other players in one turn it might take fifteen phases to complete the planned turn if each player has enough cards to trigger a new turn in of cards with every elimination (i.e., turn = {I, II, III, II, III, II, III, II, III, II, III, II, III, II, III}). In other words, after eliminating each opponent, a player could conceivably turn in cards. This complex endgame problem will be discussed further in the Section 7.3: Results.

### 7.1.5 Tasks (*T*)

The game RISK is a battle for territory and cards. A player must eliminate another player's territories to get their cards, and is also given a card at the end of their turn if they defeat at least one territory. Also, as will be detailed later in Subsection 7.1.7, taking territory entails an element of risk because there are dice rolls associated with battling for territories. However, taking cards is a discrete event, and does not require dice rolls. In other words, taking a territory is not guaranteed, while taking out an opponent guarantees

the attacker gets their cards. Thus, a common element of a task ($\tau$) considered here is a territory takeover. If a plan projects *n* territory takeovers, then the projected tasks are $T = \{\tau_1, \tau_2, \ldots, \tau_n\}$. The probability of successful plan completion is defined as the probability of completing all projected tasks (i.e., Pr($T$)).

In the process of taking over territories, each territory takeover can be (1) dependent, (2) intra-dependent, (3) independent, or (4) interdependent of other territory takeovers. This depends on where the attacks originate and where they end up. First, if battles in a sequence of attacks all originate from the same territory and only follow a single path to a single end point, then this is considered a dependent battle sequence. Second, if battles in a sequence of attacks all originate from the same territory, but follow multiple paths and do not re-attack a territory from more than a single territory, then these battles are considered as intra-dependent battle sequences. Third, if battles in a sequence of attacks originate from different territories and do not attack the same territories, then these battles are considered as independent battle sequences. Finally, if battles in a sequence of attacks cross over other battle sequences by attacking any territory from more than one territory, then these battle sequences are considered interdependent battle sequences. All independent battle sequences are defined as battle campaigns.

The probability of task completions is subject to combining battle success probabilities among dependent, intra-dependent, independent, and interdependent battle sequences. Figure 7.3 shows a variety of battle sequences. In the figure, there are eight players with one player planning fourteen battles against seven opponents. For battling territories, there is a connecting band between the territories. The wide end of the band is

where the attack is originating. The numbers in the band refer to a projected transfer of forces from the originating territory to the attacked territory. The number closer to the originating territory is the number of projected forces left behind, and the number nearer to the attacked territory is the number of projected forces moved forward. If the projected forward force is greater than zero, then a victory is predicted. Note that there are three numbers in the attacked territory. The top number indicates the original number of opponent forces on that territory before executing the plan, the bottom number indicates the projected opponent forces left after executing the plan, and the number to the right indicates the battle number or order of battle (e.g., battle 4 is Greenland attacking Northwest Territories). In the originating or attacking territory, there are one or two numbers. If there are two numbers, the top number indicates the number of allocated forces, and the bottom number indicates the total forces on that territory after the allocations (e.g., Greenland, Ontario, and Argentina each received one force). If there is only one number, no additional allocations were placed, and that number indicates the number of forces on that territory. Note there are fourteen battles shown here.

There are five independent battle campaigns shown in Figure 7.3 and fourteen individual battles, which refer to tasks $\{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5, \tau_6, \tau_7, \tau_8, \tau_9, \tau_{10}, \tau_{11}, \tau_{12}, \tau_{13}, \tau_{14}\}$. Note that the battle number and task number are equivalent here (i.e., the number to right in attacked territories). Two of these are dependent battle sequences: (1) a two-battle sequence $\{\tau_1, \tau_{10}\}$; and (2) a single-battle sequence $\{\tau_4\}$. The other three are intradependent battle sequences: (1) a four-battle sequence $\{\tau_2, \tau_8, \tau_{12}, \tau_{14}\}$; (2) a five-battle sequence $\{\tau_3, \tau_5, \tau_7, \tau_9, \tau_{11}\}$; and (3) another two-battle sequence $\{\tau_6, \tau_{13}\}$. Note

that battle number ‘7’ is projected to be a failure. Thus, it is considered as an incomplete task. The projected completed tasks are the set of all other territory takeovers (i.e., $T = \{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5, \tau_6, \tau_8, \tau_9, \tau_{10}, \tau_{11}, \tau_{12}, \tau_{13}, \tau_{14}\}$). Calculation of the probability of successful completion of tasks will be discussed in detail in the Section 7.3: RISK Game Results.

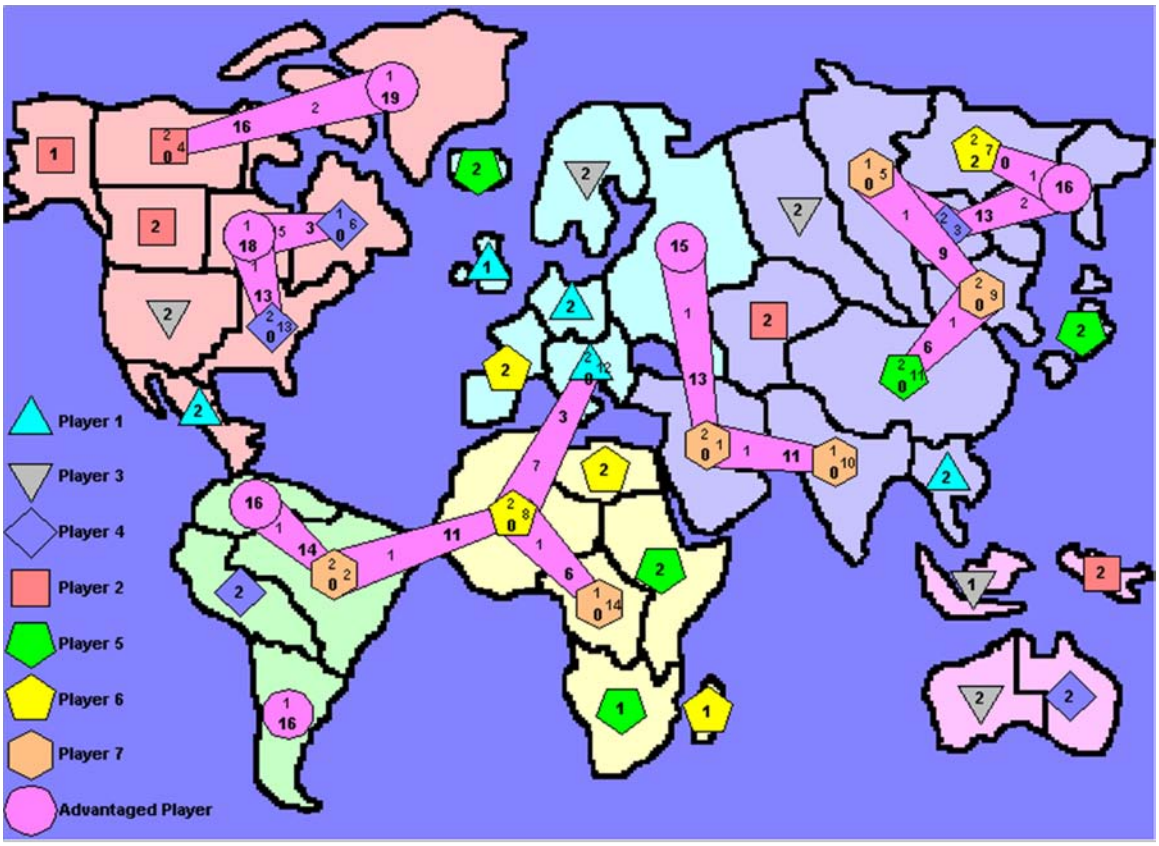


Figure 7.3. Example RISK Plan with 14 Battle Tasks

### 7.1.6 Actions (*A*)

In order to complete tasks, actions must be sequenced together to form plans. Available actions are defined in the functional diagram shown in Figure 7.2. The first action chosen for a plan is whether the player can or cannot turn in cards (Phase I). If a

player can turn in their cards, they may do so and allocate more forces. Once this decision is made, force allocations are the next chosen actions (Phase II). In this action, a player can place their new forces on any territory they occupy. The number of combinations is:

$$(m+n-1)!/((m-1)!\bullet n!), \quad \text{(eq. 7.1)}$$

where $m$ is the number of occupied territories (typically 5 for an eight player game, but can range from 1 to 41) and $n$ is the number of force allocations (typically 3 to 7, but can range from 3 to 100+). Note that if $m = 0$, then the player is out of the game and the combinations are zero.

To simplify the allocation process, a Monte Carlo method is used to select the force allocations based on random chance, but using some subject matter expertise (SME). Note that a player cannot attack another unless the attacking player has a territory with at least two forces on it, and that territory borders an opponent's territory. Thus, to simplify the allocation process, allocations are only given to territories that border opponent territories. This significantly reduces the search space for allocation in many cases, especially in mid-game and endgame situations. Note that if each force were applied randomly with equal probability to each territory, many situations would be unlikely to occur, such as allocating all forces on a single territory. For instance, if there were $n$ forces to be placed on $m$ border territories, the probability of allocating all on one territory with an even probability distribution function is m divided by the number of total combinations (shown above) or probability equal to:

$$(m!\bullet n!)/(m + n - 1)!. \quad \text{(eq. 7.2)}$$

To achieve a broader Monte Carlo search, a beta distribution [5] [6] is used instead of uniform selection, where *v* and *w* are equal to 0.3 and 0.5, respectively. The graph of the probability density function is shown in Figure 7.4. For each territory, a number from zero to one is chosen using this density function and afterward, all numbers are normalized to sum to one. These real numbers indicate the percentage of forces allocated to each territory. More specifically, the real numbers are multiplied by the number of total allocated forces. This product is rounded to the nearest whole number, and represents the number of forces allocated to each corresponding territory. Since many of these numbers will be zero, a greater number of combinations can be reached using this method than using an equally likely random selection.

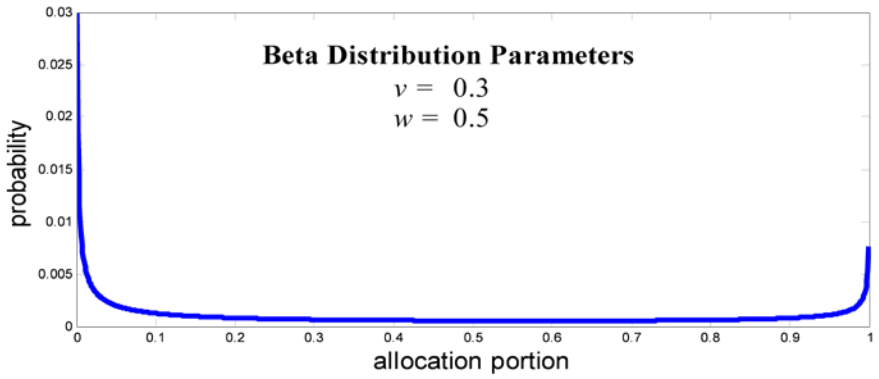


Figure 7.4. Probability Density Function for Allocating Forces

The next set of actions is determining: (1) where to attack, (2) how long to attack, and (3) how many territories to attack (Phase III). In RISK, an attack action is chosen from among all bordering territories (up to 166 possibilities), where the current player has at least two forces on their territory and it borders an opponent. Here, choosing where to attack is done by randomly choosing among the possible territories available. The length of an attack is dependent on the number of forces on both sides of the attack, and the dice

rolls by both attacker and defender. Finally, the number of territories a player can attack depends on a player's resources (number of forces) verses their opponents' resources. More detailed analysis of attacks is provided in the next subsection.

### 7.1.7 Outcomes (*Y*)

For the game RISK, turning in of cards, allocating forces, and moving forces have outcomes that are a one to one mapping to actions, while attacking territories have outcomes that are a many to one mapping to actions. In the first case, each action produces a single outcome, while in the second case, each action can produce one of many possible outcomes. The first case is trivial and not discussed here. The second case, attacking territories, is a stochastic element of the game RISK. First, the rules pertaining to attacking territories will be discussed in detail, followed by a specific example to clarify how outcomes are determined in multiple successive attacks.

Table 7.4. Probability Table of a Single Attack

| Attack Dice | Defend Dice | Defensive Losses | | | Offensive Losses | | |
|---|---|---|---|---|---|---|---|
| | | -2 | -1 | 0 | 0 | -1 | -2 |
| 3 | 2 | 0.3717 | 0.3358 | 0.2926 | 0.3717 | 0.3358 | 0.2926 |
| 3 | 1 | 0.0 | 0.6597 | 0.3403 | 0.6597 | 0.3403 | 0.0 |
| 2 | 2 | 0.2276 | 0.3241 | 0.4483 | 0.2276 | 0.3241 | 0.4483 |
| 2 | 1 | 0.0 | 0.5787 | 0.4213 | 0.5787 | 0.4213 | 0.0 |
| 1 | 2 | 0.0 | 0.2546 | 0.7454 | 0.2546 | 0.7454 | 0.0 |
| 1 | 1 | 0.0 | 0.4167 | 0.5833 | 0.4167 | 0.5833 | 0.0 |

An attack as defined in RISK is a two-player battle, where an attacker and a defender roll dice to determine losses on both sides. The number of dice that can be rolled depends on the number of forces on the two battling territories, and the forces each player is willing to risk in each dice roll. If the attacking player has *m* forces on its

attacking territory, then the attacker can roll up to the minimum of three and *m*-1 dice (i.e., **min**(3, *m*-1)). If a defender has *n* forces on its defending territory, the defender can roll up to the minimum between two and *n* dice (i.e., **min**(2, *n*)). An attacking player has the option to roll one or two dice instead of three, but the probability of being successful is less in this case. A defender also can roll one dice instead of two, but his odds of winning the roll are also less. There are very few circumstances where one rolls fewer dice than given, so this RISK game implementation assumes each player rolls the maximum number of dice. During a single attack, both players roll six-sided dice, and the dice are sorted from highest to lowest and matched up to determine who wins the attack. The attacker wins for each die greater than a defender's die, and a defender wins for each die greater than or equal to an attacker's die. The lowest die or dice are not used when the attacker and defender use a different number of dice. For each attack, zero, one or two forces are lost on either side. The probabilities for each dice roll and corresponding losses are shown in Table 7.4. As shown in the table, the probabilities favor an attacker if the player has more dice than the defender; otherwise the defender has favorable probabilities.

A single attack does not often take a territory, and an attacker can wage a battle until there is a winner (complete forces lost on one side or the other). Using the table above, the full battle probabilities for successive attack/defend rolls can be calculated using a Bayesian network. However, calculating every possible outcome with its probability for each battle would take valuable computational time. Thus, this calculation was done offline to speed the performance of calculating multiple battle outcomes. A state transition probability mass function (pmf) was calculated a priori for all likely single

battles. More specifically, all pmfs for battles from 2 to 300 offensive forces vs. 1 to 200 defensive forces were calculated a priori and stored for quick retrieval. This database forms the initial battle pmfs for all single battle sequences (see Figure 7.5 upper-right graph for example pmf of 9 vs. 2 forces). In Figure 7.5, we use number pairs to represent possible battle outcomes. These number pairs are under each bar on the three graphs shown to the right. The first number indicates the maximum number of offensive troops left to move onto a defeated territory and the second number indicates the number of defending troops left after a battle is completed.

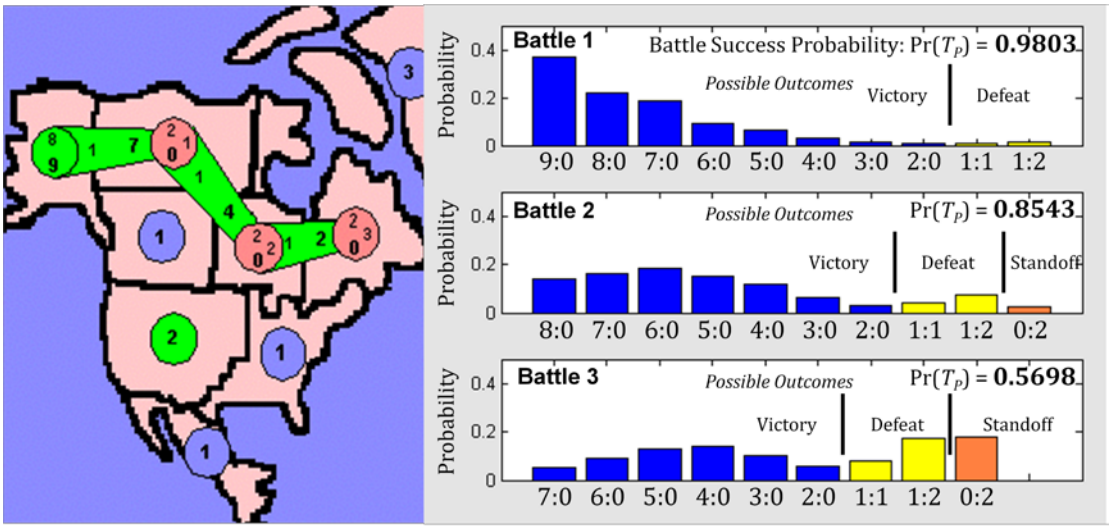


Figure 7.5. Example Dependent Battle Sequence

When planning, all possibilities should be considered, but often the number of possibilities grows exponentially as a function of plan length. To counter this, a dynamic Bayesian network (DBN) [7] [8] is used to chain individual state transition probabilities (see Figure 7.5, battle 2 and 3). This is done by multiplying the probability of all previous outcomes (A) by the conditional probability that an event occurs (i.e., Pr(A,B) = Pr(A)Pr(B|A)) [9]. Battle 1 is looked up in the probability table calculated offline, while

Battles 2 and 3 are calculated on the fly. For example, consider Battle 2. The Pr(A) is each of the ten possible pmf bars (i.e., Pr(A) = Pr({$a_1$, $a_2$, $a_3$, $a_4$, $a_5$, $a_6$, $a_7$, $a_8$, $a_9$, $a_{10}$}) in the Battle 1 graph. Now, each Battle 2 possibility is calculated with these ten conditional cases, and with the reuse of the lookup table. For $a_1$, the battle lookup is 8 versus 2; for $a_2$, the battle lookup is 7 versus 2 and so on. These resulting conditional probabilities are multiplied by Pr(A) to get the pmf shown in Battle 2. The same is done again to calculate Battle 3. Note that the same outcomes are summed together to avoid exponential expansion of the tree of probable outcomes. Further actions from this state can take into account this new pmf, thus reducing the computation required for multiple chained dependent actions. Through the use of this DBN, a network is generated on the fly that represents a set of outcomes based on a generated plan.

### 7.1.8 Observations (*O*)

The game RISK is almost completely observable. All players can observe: all force actions, the number of cards a player holds, and each set of cards turned in. The only element of the game not observable is what cards a player holds in their hands. This has little impact on the game, and thus is not modeled extensively. However, the probability of having a match with three and four cards is modeled, and is taken into consideration when planning.

### 7.1.9 Rounds or Competitions ($\Theta$)

A round is a set of games used to compare players for the RISK game. The round algorithm used here was for an eight-player game where each round acts as a generation in an overall tournament or evolutionary strategy. The population of players was kept at

a constant level for each generation and all players play each other at least once. The reward is winner-take-all, where the top performers are measured by the percentage of games won. More details will be provided in the Subsection 7.3.3 in this chapter.

### 7.1.10 Tournaments ($\Omega$)

A tournament for the game RISK is considered as a set of rounds or evolutionary generations. The constants in a tournament are the rules of the game and number of players starting the game. Most tournament experiments use eight players, and all use the classic RISK rules described earlier. A tournament is considered complete when the parameters have converged on stable values, and can also outperform previous generational players. More details will be provided in the Subsection 7.3.3 in this chapter.

## 7.2 Implementation Strategy

The implementation strategy follows the nine-step DBT approach described in Chapter 6. The approach for the game RISK is divided into these nine incremental steps: (1) **execute arbitrary** (random) **actions**; (2) **update planning-model**; (3) **choose and execute single actions**; (4) **generate arbitrary** (random) **multi-action plans**; (5) **generate and choose** (the fittest) **plans;** (6) **assess plans**; (7) coordinate the **core planning cycle** for a single player's turn; (8) coordinate multiple players to complete **a single game or mission**; and (9) orchestrate **a complete tournament** as a meta-learning strategy for training players to autonomously improve performance.

### 7.2.1 Execute Arbitrary Actions

**Execute arbitrary actions** implementation requires four building blocks: game model, game state, planning state, and action choices, as shown in Figure 6.7 in Chapter

6. For the RISK game, the game board, cards, players' forces, and the rules determine the game model. The game state is determined by the placement of forces on the board, the cards dealt to players, the number of card matches turned in, which player is up, and the current phase of the turn as shown in Figure 7.2. The planning state is equivalent to the game state at the beginning of a player's turn. In this implementation step, the planning state will be equal to the game state. For the player up, action choices are a function of the player's current state: cards in hand, placement of forces on the board, and phase of a turn.

As shown in Figure 6.7, the building blocks require four functional interactions for **execute arbitrary actions**: (2.1) *check that actions are executable* (i.e., $P' = P_{\Xi}(P_Z, \gamma, M_{\Psi}, X_P)$); (2.2) *execute actions* (i.e., $\{A_1, P'\} = A_{\Xi}(P', t)$); (2.3) *realize outcomes* (i.e., $Y' = Y_R(A_1, \gamma, M_{\Xi}, X_{\Xi})$); and (2.4) *observe response* (i.e., $O' = O_{\Xi}(Y', \gamma, M_{\Xi}, X_{\Xi})$). The four functions here were tailored for the game RISK and are described in detail below.

The *check that actions are executable* function requires four inputs: (1) the selected plans ($P_Z$); (2) the corresponding agent under consideration ($\gamma$); (3) the planning-model of current planner ($M_{\Psi}$); and (4) the current state of the corresponding plan ($X_P$). In **execute arbitrary actions**, the selected plan is a single action generated from the functional block diagram shown in Figure 7.2. The agents under consideration depend on the phase of the turn. For Phase I, the agents are the held cards, for Phases II, III, and IV, the agents are the randomly selected forces used to attack, and/or move. Here, the planning model is made equal to the execution model. The current state is the player up, the board layout, the held cards, number of matches turned in, and the phase of the turn.

This function outputs a one-action plan ($P'$), which demonstrates and tests that the rules of the game are properly implemented.

The *execute actions* function requires two inputs: (1) plan-remainder ($P'$ described above); and (2) time ($t$). Since only one player plays at each instant, time stamping each action is not required in the RISK application. The actions are already placed in the order of execution, and only one player plays at a time. For **execute arbitrary actions**, the plan-remainder is null. This function outputs two separate variables, the first action ($A_1$) and the new plan-remainder ($P'$). The *execute actions* function is the same as the process 3.1 shown in Table 6.5. After plans are sequenced together, the plan-remainder will be adapted as described for process 3.1 described later.

The *realize outcomes* function plays out a single action in the real-world simulation of the game board. This function uses four already described variables: (1) time-ordered actions ($A_1$); (2) corresponding agent under consideration ($\gamma$); (3) the real-world model ($M_\Xi$); and (4) the current state of the environment ($X_\Xi$). This function outputs realized outcomes ($Y'$), which can be a turn in of cards, the results of an attack (dice roll), or a force movement.

The *observe response* function observes each action's outcome and modifies the current state to match this observation. This function uses four already described variables: (1) realized outcomes ($Y'$); (2) the corresponding agent under consideration ($\gamma$); (3) the real-world or environmental model ($M_\Xi$ described above); and (4) the current state of the environment ($X_\Xi$ described above). This function outputs realized observations

($O'$), which are a one to one mapping with all realized outcomes ($Y'$), except for attack outcomes described earlier (Section 7.1.7).

### 7.2.2 Update Planning-Model

In implementing the **update planning-model**, the difficult task is coordinating between the planning-model changes and real-world model changes. The real-world model changes are permanent, while the planning-model changes are temporary. In updating the planning-model from the new real-world model after some action execution, these new planning-model changes are permanent and irreversible (see Figure 6.9). For the game RISK, the environment is almost fully observable and known (except for cards in other players' hands). Thus, these planning-model changes are considered only as a change in the current state, such as a player: turning in cards, placing forces on the game board, rolling dice in an attack, or moving forces.

**Update planning-model** uses the *feedback state* function (2.5: $X_U$). This function uses three input variables: (1) a set of agents under consideration for planning ($I'$); (2) the observations ($O$ described above); and (3) the task update coefficient ($\tau_U$). The set of agents are the cards in each player's hands and forces on the board. These are modified permanently, based on observations. For the game RISK, the task update coefficient is set to a single action such as the completion of a single battle. A battle is considered as a single action for the game RISK, which may require many dice rolls. This gives the current player the opportunity to change their plan after every territory battle, because dice rolls might make the difference in the projected success or failure of subsequent battles in a plan. The output of this function is a new state representation for the internal

planning-model (i.e., $X_{\psi} = X_U(\Gamma, O', t_U)$). All new plans will use this state as a starting point.

### 7.2.3 Choose and Execute Single Actions

The **choose and execute single actions** implementation of the RISK game requires the final building block, the *value* function (V), shown in Figure 6.10. The *value* function includes all control parameters of a player. In choosing and executing actions, only a small portion of these parameters are used, and are those parameters used in selecting individual actions. Choosing of actions for the game RISK is done randomly in selecting among all the possible choices available based on the finite state machine representation of a player's turn (Figure 7.2). This finite state machine is used as the *determine action choices* function based on the current state (i.e., 1.2: $A = I_A(\gamma, M_{\Psi}, X_P)$). *Select actions* function is done using a Monte Carlo method (i.e., 1.3: $A_Z = Z_A(\Gamma, A, H, R, C_A, t)$). For the game RISK, the method does not use any selected features ($H$) or time ($t$), because actions are randomly selected and time-stamping actions is not necessary in RISK. Only the agents ($\Gamma$), their available resources ($R$), action costs ($C_A$), and action choices ($A$) are considered when choosing individual actions. More specifically, choosing whether to turn in cards or not is decided randomly with equal probability. Choosing where to place allocations is described in Section 7.1.6. Choosing where to battle next is chosen randomly with equal probability among all available choices. This procedure tests the coordination of using plan-generation with plan-execution.

### 7.2.4 Generate Arbitrary Multi-Action Plans

The **Generate arbitrary multi-action plans** implementation for the game RISK is equivalent to generating multiple action sequences. Each action is chosen as described in Section 7.2.3. However, plans are not executed until a plan considers all resources. This requires interactions of the model and state process for each action considered in a plan. The interaction of model and state process has four sub-processes: (1.4.1) *predict outcomes*, (1.4.2) *select outcomes*, (1.4.3) *observe state changes*, and (1.4.4) *update plan's model state*, as shown in Table 6.5.

For the game RISK, the *predict outcomes* function (i.e., $Y = \mathrm{K}_{\mathrm{Y}}(A_Z, \gamma, M_\Psi, X_P)$) uses the selected action ($A_Z$) described above and calculates the outcomes ($Y$) based on using the previously described DBN (Section 7.1.7). The *select outcomes* function (i.e., $Y_Z = \mathrm{Z}_{\mathrm{Y}}(Y, r_Y)$) uses the set of outcomes ($Y$) with its associated pmf, and outcome risk aversion coefficient ($r_Y$) to select an individual outcome ($Y_Z$). The outcome risk aversion coefficient for the game RISK will be described in subsequent paragraphs. The *observe state changes* function (i.e., $O = \mathrm{O}_{\mathrm{X}}(Y_Z, \gamma, M_\Psi, X_P)$) uses the selected outcome ($Y_Z$) as one element of the observable state transitions for each selected action. Finally, the *update plan's model state* (i.e., $X_P = \mathrm{X}_{\mathrm{U}}(\Gamma, O, \tau_U)$) uses the observations ($O$) to update the planning-model state ($X_P$) for each action chosen within a plan. The task update coefficient ($\tau_U$) is not needed for RISK, because the game is nearly full state observable. Note that each step or selected action in a plan has a stored planning-model state for later evaluation.

When predicting outcomes of actions for a stochastic game, there are a variety of approximations one can choose that can have a substantial influence in determining future rewards. For instance, one can take the maximum likelihood [10], or the expected value [11] of projected outcomes to determine the predicted outcome. Instead, the outcome risk aversion coefficient is used to make these approximations. If one knows the underlying pmf based on each action-state pair, then one can calculate the pmfs forward as one projects actions (see Figure 7.6). These predicted outcomes are essential in prescribing the follow-on move sets.

More specifically, in examining Figure 7.6, the maximum likelihood prediction is '9:0', '6:0' and '4:0', because maximum likelihood predicts the outcome that is the most likely within a set of possible outcomes, or the tallest bar in this case. Another predictor is the expected value, which is the average outcome given all possible outcomes. This value is shown as the bar with the dot in it as '8:0', '5:0' and '3:0'. One can incorporate an outcome risk aversion parameter ($r_Y$), where '$r_Y = 0$' represents no risk aversion and thus expects the best outcome and '$r_Y = 1$' represents full risk aversion and thus expects the worst outcome. Thus, for $r_Y = 0$, we would predict '9:0', '8:0' and '7:0' and for $r_Y = 1$ we would predict '1:2', '0:2' and '0:2' for the plan shown in Figure 7.6. The far left and far right arrows over the figure illustrate $r_Y = 0$ and $r_Y = 1$, respectively, while the middle arrow represents expected value (i.e, $r_Y = 0.5$).

Specifically for the case in Figure 7.6, Battle 2 is a battle of eight vs. two, seven vs. two, six vs. two and so on down to zero vs. two. The probability for all of these outcomes from Battle 1 is propagated forward giving a pmf reflected in Battle 2 and

Battle 3. However, given that we propagate this expected value as our projected state; the reward metric of that state is biased based on that assumption. Therefore, we have modified the projected state algorithm here to include an outcome risk aversion parameter ($r_Y$) that ranges from zero to one. As described above, if $r_Y$ is set to '0', the parameter projects completely optimistic outcomes, where the DBN projects winning every battle in the plan. As applied to RISK, if $r_Y$ is set to '1', the DBN assumes the plan will lose every battle and thus will have a negative impact on reward when projecting battle plans. If $r_Y$ is set '0.5', the system projects the expected value outcome after each action. Thus, as the parameter ranges from '0' to '1' the outcomes become more conservative or risk averse in their estimates. Thus, the projected state ($X_p'$) is not only dependent on the actions taken and the current state, but also on the selected outcome risk aversion parameter ($r_Y$).

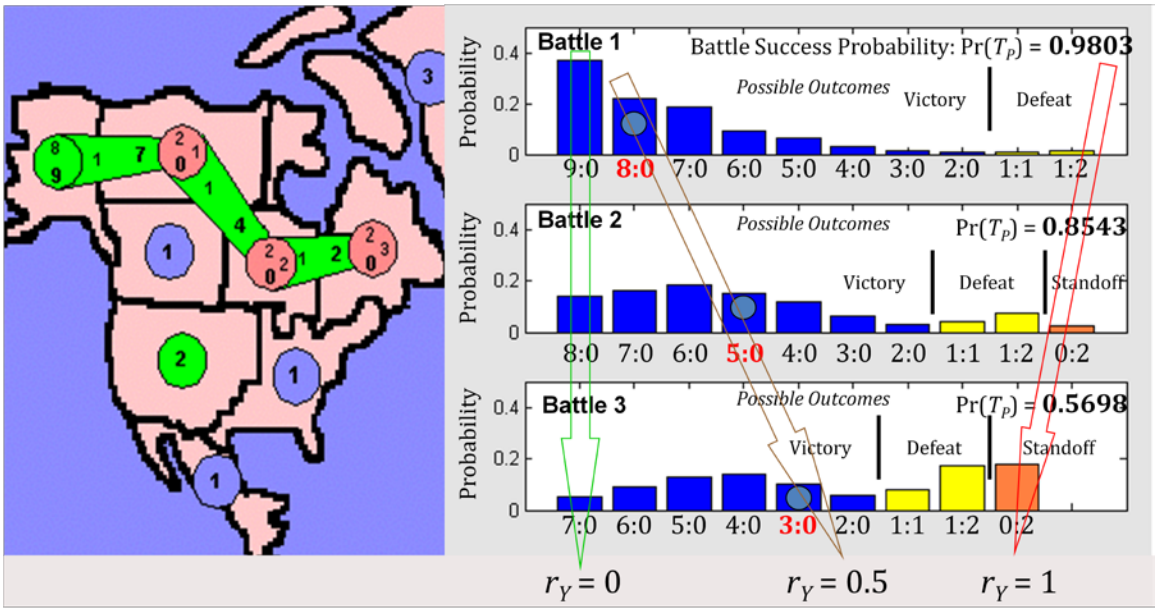


Figure 7.6. Using Risk Aversion Coefficient to Select Battle Outcomes

### 7.2.5 Generate and Choose Plans

**Generate and choose plan** implementation uses the full capacity of the plan-generation phase of the planning cycle for an individual player in a single turn. More specifically, **generate and choose plan** requires the rest of the plan generation processes: (1.1) a *search and select plans* function, and (1.5) an *evaluate plan fitness* function. The *search and select plans* function includes the use of fitness to choose plans, search parameters to search through many possible action choices to form plans, and a *forecast expectations* function to measure the completed tasks for each plan. The *evaluate plan fitness* function includes SME in choosing the sub-goals, probability of successful completion of plans, and a diversity metric to aid in selecting variable plans.

#### 7.2.5.1 Search and Select Plans

The first part of the *search and select plans* function (1.1.1) is the *search for plans* function (i.e., $P = S_P(S_\psi, T_P, \tau_P, R_\gamma, u_R)$). This function includes: (1) the planner's search parameters ($S_\psi$); (2) completed tasks per plan ($T_P$); (3) number of tasks constraint coefficient ($\tau_P$); (4) resources available ($R$); and (5) portion of used resources constraint coefficient ($u_R$). For the game RISK, an evolutionary or genetic algorithm search approach is used. Other approaches have been used and compared (described in Chapter 3: Background), but the evolutionary approach proves to be the most effective here. A comparison is shown in Subsection 7.3.1.

The search parameters used (i.e., $S_\psi = \{s_{es}, s_{rs}, s_{cs}, s_{se}, s_{of}, s_{re}\}$) will be described in detail next. For the game RISK, the number of tasks constraint is set to infinity, because a player is not restricted by tasks, but by the resources available (agents: cards and forces).

The length of a plan is agent resource ($R_\gamma$) constrained due to the number of forces on the board. Resources are considered as the forces on the game board, including the forces acquired via projected card turn-ins. Each held card is considered equivalent to one-third the value of the next turn-in, since a turn-in requires three cards. The resource constraint coefficient ($u_R$) is used to limit the proportional use of resources from none ($u_R = 0$) to full ($u_R = 1$). For some experiments this coefficient is set to one to indicate full use of all agent resources, otherwise it is learned via tournament play.

The planner's parameters for evolutionary search include: (1) how much to search at the beginning of a turn ($\{s_{es}\}$), (2) how much to search after every task within a turn given a plan is "predicted to succeed" ($\{s_{cs}\}$), (3) how much to search when a plan is "predicted to fail" in completing projected future tasks ($\{s_{rs}\}$), and (4) what constitutes a single generation of search ($\{s_{se}, s_{of}, s_{re}\}$). At the beginning of a turn, the evolutionary search parameter ($\{s_{es}\}$) indicates the number of generations to search (1 to $i$). The number of generations to search after each task (i.e., a territory takeover) is completed is given by the continuous search parameter ($s_{cs} = 0$ to $j$). The number of generations to search after a plan is predicted to fail is given by the revolutionary search parameter ($s_{rs} = 0$ to $k$). The other parameters determine the amount of search in each individual generation: (1) the selections parameter ($s_{se} = 1$ to $l$) indicates the number of selections per generation; (2) the offspring parameter ($s_{of} = 0$ to $m$) indicates the number of offspring per generation, and (3) the recombinations parameter ($s_{re} = 0$ to $n$) indicates the number of recombinations per generation. These parameters are either preset or learned, depending on the experiment.

More specifically, for each generation, the *selections* ($s_{se}$) are the number of plans selected before and after performing mutations and recombinations. The *offspring* ($s_{of}$) is the number of duplicate selections that are pruned randomly based on fitness (fitness is described further in Section 7.2.5.2). The *recombinations* ($s_{re}$) is the number of times each phase of a plan can be scrambled, while the remaining plan stays the same. For all these experiments, $s_{se}$, $s_{of}$, and $s_{re}$ were chosen by experimentation to yield a competent player. Table 7.5 below provides a more concrete description of the evolutionary search method used for plan-generation.

Table 7.5. Evolutionary Search for Plan-Generation

| Step | Algorithm |
|---|---|
| 1. | Generate each plan randomly until $u_R$ fraction of resources are used in each plan ($s_{se} \bullet s_{of}$ plans) and project a random final move at the end of each plan |
| 2. | Prune out projected defeats and re-project plans ($s_{se} \bullet s_{of}$ plans) |
| 3. | Optional: Phases ($\phi_n$ per plan) are chosen from each plan for recombination, includes pruning out as above<br>a. Copy each plan $\overline{\phi_n} \bullet s_{re}$ times, where $\overline{\phi_n}$ is average number of phases per plan<br>b. Use recombination on selected phases for each plan $((s_{se} \bullet s_{of}) \bullet (\overline{\phi_n} \bullet s_{re} + 1)$ plans) |
| 4. | All plans are pruned back to their fittest point $((s_{se} \bullet s_{of}) \bullet (\overline{\phi_n} \bullet s_{re} + 1)$ plans) |
| 5. | Fittest plans are selected from all pruned plans ($s_{se}$ plans) |
| 6. | Offspring are additionally pruned copies of selected plans ($s_{se} \bullet s_{of}$ plans) |
| 7. | If a plan meets all goals or time expires, terminate, otherwise go to step 1 (all the offspring are the beginning plans for the next generation) |

Many EC algorithms have some form of *mutation*, *recombination*, *reproduction*, and *selection*. In the algorithm above, the *mutation* portion of the algorithm is step 1. The *mutation* method used is synonymous to a random walk, where each action is chosen randomly at each step of a plan. The *recombination* portion of the algorithm is step 3.

The recombining method is unique and a more detailed description of recombination is described in the next paragraph. *Reproduction* occurs when the number of plans grows in number or length. The reproduction portion of the algorithm is seen in steps 1 through 3, and 6. Generating plans, adding parents to the selection pool, and copying plan portions all constitute additional flexibility in plan choices. *Selection* occurs when a large number of possible plans are drawn down to a chosen few. Selection is used in step 5 where the algorithm prunes back plans to their fittest point and selects from among the new plans.

The recombination method selects a single portion of a plan (i.e., a phase). All prior portions of a plan are left unchanged. The targeted portion is recombined, constraining actions of that portion to actions that complete similar tasks (step 3). For example, the selected recombining phase is the takeover of a particular player. For recombination, the only actions allowed in this phase are the ones that takeover the same territories (i.e., same tasks) as before. Thus, a shorter more probable takeover path may be found. The probability of selecting a particular portion of a plan for recombination is proportional to the allocations used in that phase multiplied by the change in probability of successful plan completion (as described in Section 7.1.5) for that phase. It can be argued that phases that expend a high number of allocations and have reduced success probabilities have the greatest potential of improving the overall plan. Once a phase is chosen, the actions in that phase are chosen randomly with the possibility of yielding similar outcomes having higher success probabilities and thus a higher fitness (step 4).

Due to the stochastic nature (dice rolls) and length of a player's turn (possibly tens of actions) in the game RISK, evolutionary search can be done at different strategic

points within a turn. At the beginning of a turn ($\{s_{es}\}$), after every task in a turn ($\{s_{cs}\}$), and when a plan is failing to complete projected tasks ($\{s_{rs}\}$) are these strategic points. These parameters control the number of generations of search used at these strategic points. These parameters are learned based on competing planners in tournaments, where there are a limited number of actions generated over the course of a game.

### 7.2.5.2 Evaluate Plan Fitness

*Evaluate plan fitness* for the game RISK is a function of state ($X_P$ described above), reward ($J_\Psi$), probability of success ($Pr(T_P)$ described above), and diversity ($D$) (i.e., $F = F_P(X_P, J_\Psi, Pr(T_P), D)$). The reward is measured by a set of ten sub-goals ($J_\Psi$) derived by SME and shown in Table 7.6. Diversity is a measure of similarity in plans based on accomplished tasks. Diversity is described in detail later in this section.

Table 7.6. Ten RISK Game Sub-Goal Objectives

| Sub-Goals to Maximize | Reward Values |
|---|---|
| 1. Overall Strength | $\frac{\text{Individual's Force – No. of Territories}}{\text{Total Forces – Total Territories}}$ |
| 2. Reoccurring Reinforcements | $\frac{\text{Territory Reinforcements}}{\text{Total Territory Reinforcements}}$ |
| 3. Onetime Reinforcements | $\frac{\text{Individual's Cards Reinforcements}}{\text{Total Cards Reinforcements}}$ |
| 4. No. of Protected Territories | $\frac{\text{Individual's Protected Territories}}{\text{Total Protected Territories}}$ |
| 5. No. of Protected per Protector Territories | $\frac{\text{Individual Protected Territories}}{\text{Individual Protector Territories}}$ |
| 6. Forces on Protector Territories | $\frac{\text{Individual Protector Forces– No. of Protector Territories}}{\text{Individual Total Forces – No. of Territories}}$ |
| 7. Smallest Force on Protector Territories | $\frac{\text{Individual's Smallest Protector Force - 1}}{\text{Total Smallest Protector Forces – No. of Current Players}}$ |
| 8. Forces on Border Territories | $\frac{\text{Individual Border Forces – No. of Border Territories}}{\text{Individual Total Forces – No. of Territories}}$ |
| 9. Smallest Force on Border Territories | $\frac{\text{Individual's Smallest Border Force - 1}}{\text{Total Smallest Border Forces – No. of Current Players}}$ |
| 10. Fewest Unprotected Border Territories | $\text{minimum}\left(2, \frac{1}{\text{No. of Unprotected Borders}}\right)$ |

*Evaluate plan fitness* for the game RISK will be described as a function of goals, risk aversion, probability of success, and diversity. The state representation ($X_P$) is measured via a set of reward functions ($J_\Psi$), which measures progress of a plan (i.e., set of actions $a$) based on the following change in reward ($\Delta R$) where the $X_0$ is the initial state and $X_P$ is the projected state:

$$\Delta R = J_Y(X_P) - J_Y(X_0) \text{ where the initial reward } R_0 = J_Y(X_0) \quad \text{(eq. 7.3)}$$

The sub-goals ($J_Y(X_P)$) are used to measure the progress of a player in a game. Each sub-goal objective is measured for all players and normalized to sum to one. Each of these sub-goals is not considered as equal value, so weights are assigned to each sub-goal. For each player, $W$ is a vector of ten real numbers ranging from zero to one where their sum is one. These weight values can be pre-assigned or learned via tournament play described in detail later. In essence $\Delta R$ is an eight by ten matrix representing the reward change separately for each player and each sub-goal objective. A dot product of $\Delta R$ and $W$, plus the addition of the current reward, can represent fitness as shown here:

$$F_p^{(1)} = (\Delta R \bullet W) + R_0 \quad \text{(eq. 7.4)}$$

If we also consider the overall probability of success of a plan by multiplying all the independent battle campaigns together, we can have an additional term added to the fitness to reflect this:

$$F_p^{(2)} = (\Delta R \bullet W) \prod_{i=1}^{N} Pr(T_P) + R_0 \quad \text{(eq. 7.5)}$$

Here, $Pr(T_P)$ is a set of scalar numbers ranging from zero to one that represent each independent battle success probability at each successful leaf node. There are $N$

successful leaf nodes for a given plan. Specifically, $Pr(T_P)$ equals the sum of all successful outcomes, as illustrated in the example in Figure 7.6 Battle 3.

If one does not assume that the overall probability of success (i.e., $\prod Pr(T_P)$) plays a substantial role in determining the best fitness function, one can include a probability of success factor $\pi$ (ranging from zero to one) that favors having probability of success included, and $1 - \pi$ for attenuating the probability of success feature. This results in a new fitness function, which now includes risk, reward and probability of success:

$$F_p^{(3)} = (\Delta R \bullet \mathrm{W})\ \pi \prod_{i=1}^{N} Pr(T_P) + (\Delta R \bullet \mathrm{W})\ (1-\pi) + R_0 \qquad \text{(eq. 7.6)}$$

Since we are using an evolutionary approach that considers more than one plan at a time in search of the best plan, one additional factor is considered. Specifically, evolution implies a population of possible plans and thus one can consider modifying the selection process based on the diversity of the population in contrast to selecting based solely on fitness score. The selection of the best plans can be diversified using a diversity factor ($d$) that increases the likelihood of selecting diverse plans over plans with higher rewards. Incorporating diversity into the fitness function changes it as such:

$$F_p^{(4)} = F_p^{(3)}(1-d) + D*d \qquad \text{(eq. 7.7)}$$

The measure of diversity of one plan compared to others is $D$, and $d$ is the proportion of importance placed on using diversity. To formulate $D$ one considers two vectors, such as y and z $(y \bullet z)/(||y||\ ||z||) = Cos(\phi)$ where $\phi$ represents the angle between these vectors, $(y \bullet z)$ represents the inner product of the two vectors and $Cos(\phi)$ represents

the similarity between these two vectors [12]. If we consider the value of a state as a vector then diversity can be evaluated as $D = 1 - Cos(\phi)$. In this case $Cos(\phi) = X_p * X_p^T$ divided by the magnitude of all states considered where $X_p'$ represents the projected state of a plan for the game RISK.

The plan state used is a vector with 540 values ranging from 0 to 1. There are 42 territories of ownership, 166 possible battles, 166 possible victories and 166 possible portions of force movements that make up the state $X_p'$. $D$ is a function of the inner product of these 540 plan projected state values. Factor $d$ can be learned from experience. Diversity is only applied to the difference between selected plans and plans left in the remaining pool. Thus, an iterative process is used to implement diversity. The first selection in this process is the fittest plan, which provides a basis for comparison that every successive selection uses. If $d$ is set to one, each subsequent selection is the most diverse member of the pool of selected plans, and if $d$ is set to a number less than one, the fittest plan and most diverse plan share in the selection process based on the value of $d$.

### 7.2.5.3 Forecast Expectations

*Forecast expectations* function for the game RISK requires the selected plans ($P_Z$) described above with their corresponding completed tasks ($T_P$) (i.e., $Q = K_Q(P_Z, T_P)$). The accomplished tasks are considered territory takeovers with a probability of success that exceeds the perceived risk (i.e., outcome risk aversion coefficient $r_Y$). The projected expectations ($Q$) is the sum total of the successful territory takeovers.

### 7.2.6 Assess Plans

**Assess plans** implementation is the DB&T of the plan-assessment phase. Three of the four processes in this phase are already complete as described in Section 6.2.3. Processes 3.1, 3.2, and 3.3 are duplicates of processes 2.1, 2.2, and 1.4 respectively with minor modifications. The only new algorithm to be implemented is process 3.4, the *assess expectations* function. This function has three sub-processes identified in Table 6.4 (3.4.1-3.4.3). The first of these three processes is *forecast new expectations* function (3.4.1: $K_Q$). This function operates exactly the same as the *forecast expectations* function of plan-generation (1.3: $K_Q$).

The second of the three sub-processes is the *compare expectations* function (3.4.2: $E_Q$). As described above, expectations are the number of tasks completed. If original expectations multiplied by the expectation acceptance threshold coefficient ($r_Q$) is greater than or equal to the new expectations (i.e., $Q \bullet r_Q \geq Q''$), then the *compare expectations* function outputs the new expectations, otherwise the output expectations are set to zero (i.e., $Q = E_Q(Q, Q'', r_Q)$). The third and final sub-process of *assess expectations* (3.4) is the *initiate re-planning* function (3.4.3: $P = I_R(P'', Q)$). The new plans are equal to the plan-remainder if the expectations are greater than zero ($Q > 0$); otherwise, the plans are reset to null. For simplicity in all experiments, the outcome and expectations acceptance threshold coefficients are considered equal ($r_Y = r_Q$).

### 7.2.7 Core Planning Cycle

The implementation of a single player's turn orchestrates the integration of all three phases of the **core planning cycle**: plan-generation, plan-execution and plan-

assessment. This includes the initial generation of a plan done via search parameters $s_{es}$, $s_{se}$, $s_{of}$, and $s_{re}$. After this initial planning is complete, each action is executed to the fidelity of an individual attack or battle completion, and plan and state changes are passed to plan-assessment. If a plan's success falls below the expectation acceptance threshold coefficient ($r_Q$), then plan-generation generates a new plan via these "predicted to fail" search parameters $s_{rs}$, $s_{se}$, $s_{of}$, and $s_{re}$ (described in Subsection 7.2.5.1). On subsequent planning cycles, after every completion of a task (territory takeover) in the plan-execution phase, the next plan-generation phase uses the "predicted to succeed" search parameters $s_{cs}$, $s_{se}$, $s_{of}$, and $s_{re}$ (described in Subsection 7.2.5.1). to generate the next plan.

### 7.2.8 A Single Game or Mission

The implementation of **a single game or mission** coordinates the global state parameters: whose turn it is, number of card turn-ins, which players are still alive, and reshuffling of the card deck if necessary. The initial player who starts the game is chosen randomly. For all experiments, no more than seventy turns are allowed in a single game and games are played with eight players. Note that more than seventy turns is considered extremely unlikely in human play. Also, using eight players is considered the most interesting example of planning because it imposes the greatest computational complexity.

### 7.2.9 A Complete Tournament

**A complete tournament** requires many games and rounds of games. The particular parameters for each player or planner in the RISK game used to complete a tournament must consider all aspects of plan-generation, planning cycle, and planner

capabilities. Section 7.3.3 will describe all the parameters used in a complete tournament and how these parameters are trained with corresponding results.

## 7.3 RISK Game Results

Results are presented in three areas: (1) plan-generation, (2) the integration of plan-execution and plan-assessment into a planning cycle, and (3) the complete orchestration of tournament play for the training of RISK players. The results are in chronological order of development, and built from simple action selections to orchestrating a large number of games. Plan-generation results concentrate on the searching of plans. Integration of the planning cycle will demonstrate the usefulness of execution and continual reassessment. Training of RISK players will demonstrate an approach for improving players through an autonomous offline tournament competition. Sections 7.3.1, 7.3.2 and 7.3.3 will describe the results for plan-generation, planning-cycle, and tournament play, respectively.

### 7.3.1 Plan-Generation

In plan-generation, there are three main results. The first result demonstrates the use of evolutionary search in planning moves ahead to play the game RISK against other computer players. The second result demonstrates the benefit of including the probability of success in the fitness function. The third result shows that evolutionary search is more valuable for the game RISK than other established search methods.

#### 7.3.1.1 Planning Using Evolutionary Search

The first attempts to use planning with evolutionary search are simplified versions of the current methods described in Section 7.2, Table 7.5. The first evolutionary search

used is described in Table 7.7. The main difference is that recombinations for allocations and battles are done separately in this case. Also, the search parameters are not variable and the generations, selections, offspring, allocation recombinations, and battle recombinations are set to 10, 8, 8, 1, and 2, respectively. The sub-goals used are also different from the current methods described in Table 7.6. The sub-goals used (shown in Table 7.8) are only five, and their weights are fixed, set to 0.4, 0.3, 0.1, 0.1, and 0.1 as shown in Table 7.8.

Table 7.7. First Evolutionary Search Algorithm

| Steps | Number of Individuals |
|---|---|
| Loop Generations (10) | N/A |
| 1. Mutate | $s_{se} \bullet s_{of}$ |
| 2. Prune | $s_{se} \bullet s_{of}$ |
| 3. Add Parents | $s_{se} \bullet s_{of} + s_{se}$ |
| 4. Recombinations | $[(s_{se} \bullet s_{of}) + s_{se}] \bullet (1 + s_{reA}) \bullet (1 + s_{reB})$ |
| 5. Prune | $[(s_{se} \bullet s_{of}) + s_{se}] \bullet (1 + s_{reA}) \bullet (1 + s_{reA})$ |
| 6. Add Parents | $[(s_{se} \bullet s_{of}) + s_{se}] \bullet (1 + s_{reA}) \bullet (1 + s_{reA}) + s_{se}$ |
| 7. Select Fittest | $s_{se}$ |
| 8. Truncate and Copy | $s_{se} \bullet s_{of}$ |

Table 7.8. First Fitness Function

| **Sub-Goals to Maximize** | **Reward Values** | **Weight** |
|---|---|---|
| Overall Strength | $\frac{\text{Individual Force}}{\text{Total Forces}}$ | 0.4 |
| Expected Reinforcements | $\frac{\text{Individual Reinforcements}}{\text{Total Reinforcements}}$ | 0.3 |
| No. of Defensive Fronts | $1 - \frac{\frac{\text{Individual Frontal Regions}}{\text{Individual Total Regions}}}{\frac{\text{Total Frontal Regions}}{\text{Total Regions}}}$ | 0.1 |
| Strength of Fronts | $\frac{\text{Individual Frontal Forces}}{\text{Total Individual Forces}}$ | 0.1 |
| Logistical Support | $\frac{\frac{\text{Individual Frontal Forces}}{\text{Individual Support Forces}}}{\frac{\text{Total Frontal Forces}}{\text{Total Support Forces}}}$ | 0.1 |

Figure 7.7a through i shows an example game to illustrate the use of the first dynamic planner for plan-generation. The game is played against two other players (three player game). The competing players are automated computer players from the RISK II game built by Hasbro. To play against these players, the developed dynamic planning player had to be run offline and then executed manually, because the algorithms used by Hasbro are proprietary and inaccessible.

The situational information is fed into the Matlab program, and then a human applies the results generated by the dynamic planner to play the turn. The convergence criteria that ends the evolutionary process is a simple run of 10 generations, and it is at this point where the best evolutionary strategy is selected and applied.

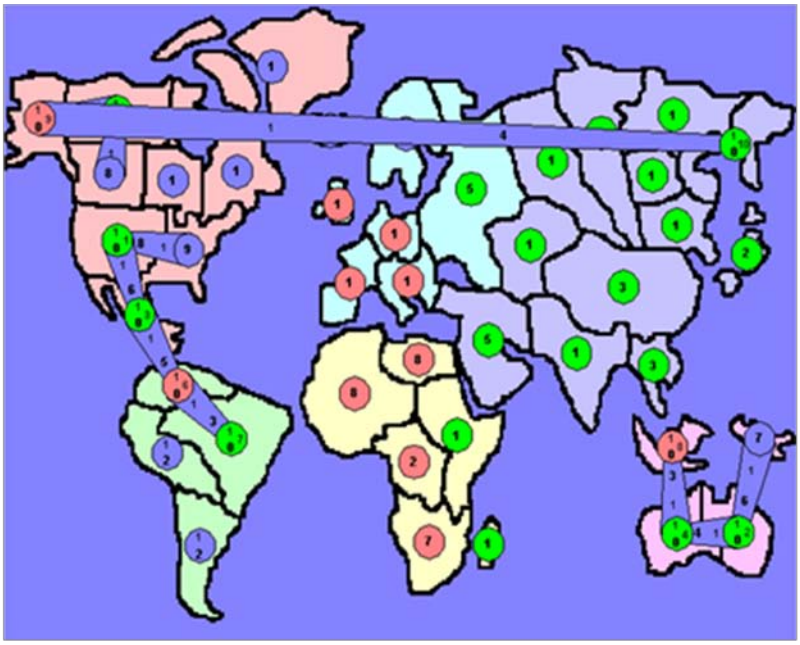

Figure 7.7a. Example Solution for Three Player Game: Turn 1

During this RISK game, the dynamic planning (plan-generation) player was player 2, thus the starting position of the game shown in Figure 7.7a illustrates the player 2 plan after player 1's turn where player 1 took over Asia. The player 2's plan generated

in the first turn (Figure 7.7a) can be summarized as follows: take over the Americas, and Australia and deny Asia to player one. The dice rolls were average and the plan proved successful for player 2.

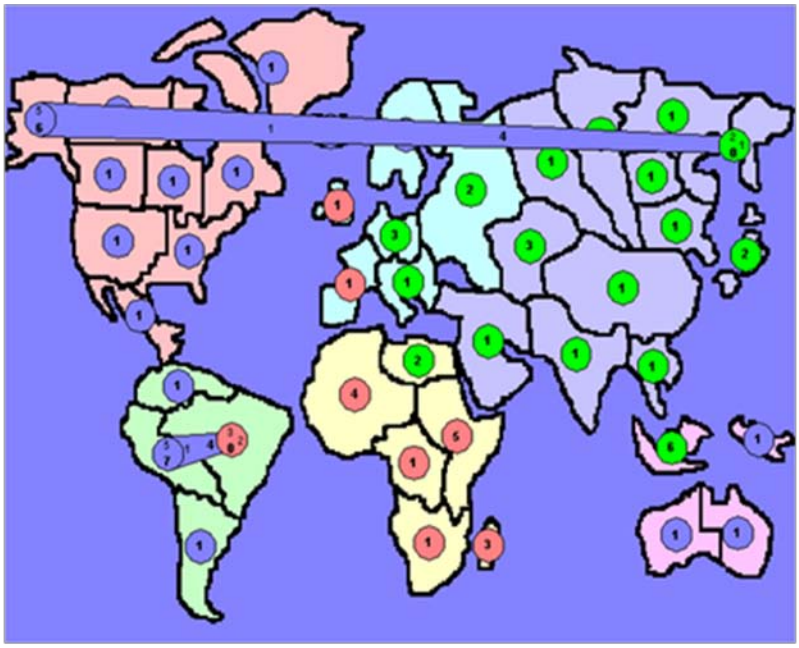

Figure 7.7b. Example Solution for Three Player Game: Turn 2

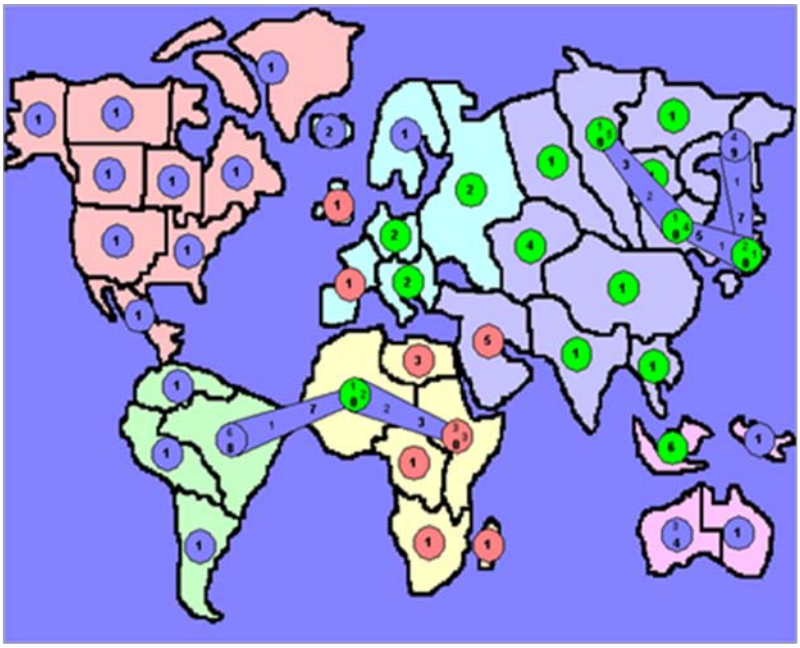

Figure 7.7c. Example Solution for Three Player Game: Turn 3

Prior to player 2's second turn, as shown in Figure 7.7b, the other opponents denied Australia and South America to player 2 and player 1 recaptured Asia. For turn two, the automated player 2 decided to retake South America and deny Asia again. The turn is successful and player 2 is able to hang onto South America and deny Asia until turn three.

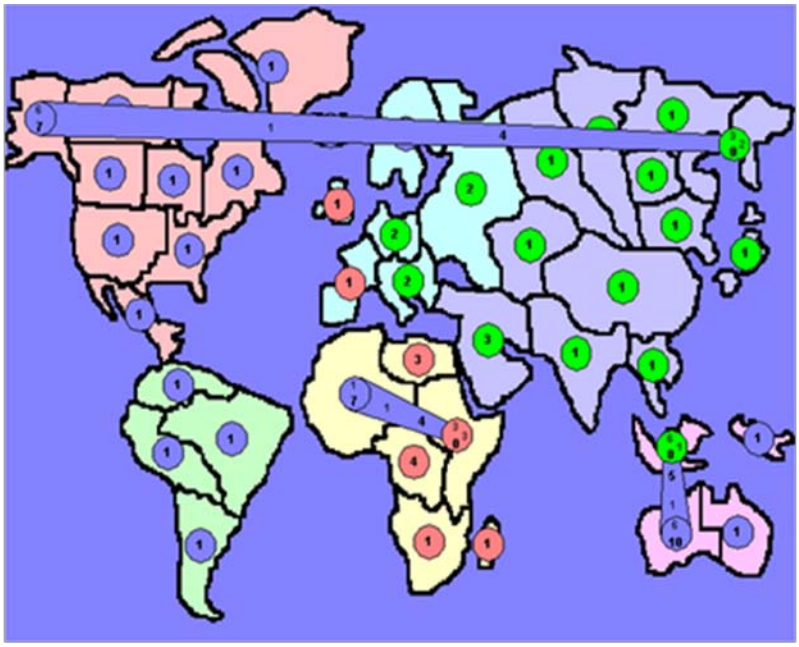

Figure 7.7d. Example Solution for Three Player Game: Turn 4

In turn three, player 2 becomes more aggressive and tries to take over part of Africa and Asia. Player 2 ends up completing his mission but incurs heavy force casualties due to poor dice rolls (Figure 7.7c). This opens the door for player 1 to recapture Asia. In turn four, player 2 attempts to re-deny Asia to player 1 and retake Australia and reinvade Africa. Due to very poor dice rolls, again only the invasion of Africa succeeds (Figure 7.7d). This provided an opportunity for player 1 to take Australia and fortify his Asian position. However, player 3 is left helpless and the reinforcements

are enough for player 2 to eliminate player 3 and deny the Asian continent to player 1 in turn five (Figure 7.7e).

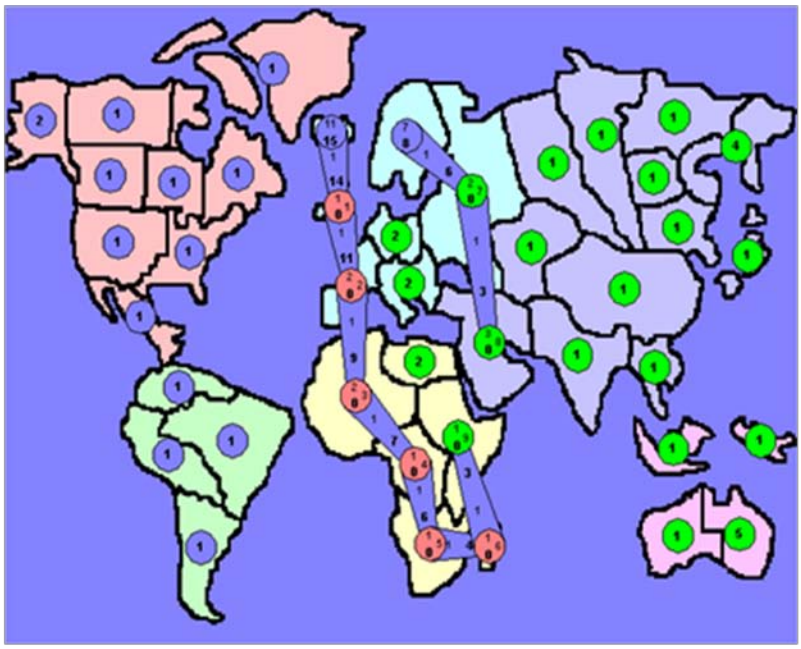

Figure 7.7e. Example Solution for Three Player Game: Turn 5

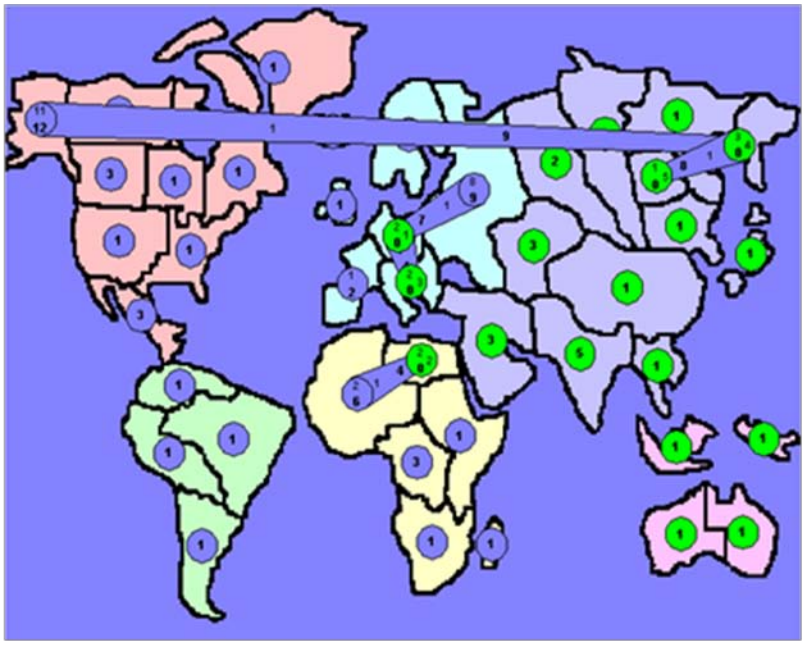

Figure 7.7f. Example Solution for Three Player Game: Turn 6

During turn six, player 1 recaptures Asia and player 2 attempts to take over Europe, Africa and deny Asia again. The mission is successful but poor dice rolls again leave the dynamic planner vulnerable to invasion (Figure 7.7f). During turn seven a similar strategy is generated that tries to do much the same as the previous turn, again poor dice rolling provides too much resistance and only partial success in Asia is achieved (Figure 7.7g).

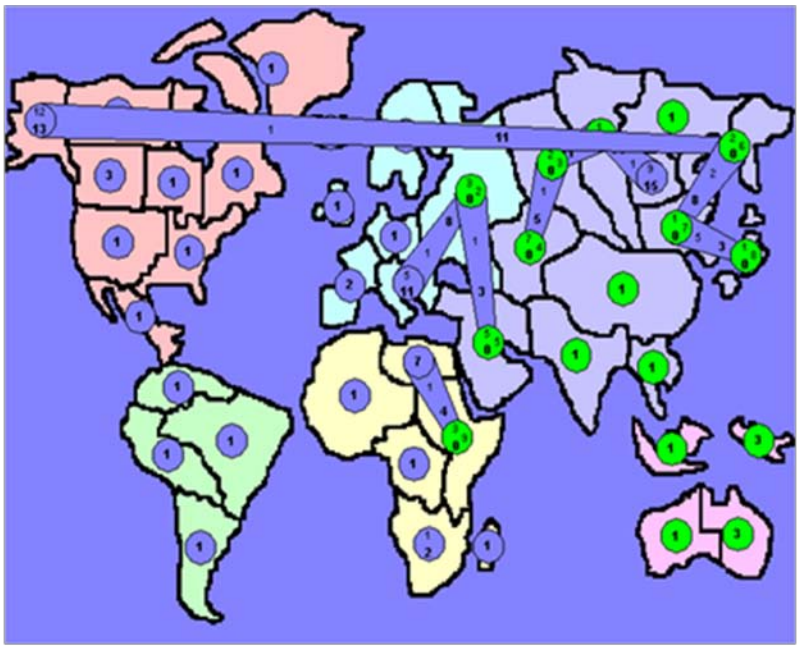

Figure 7.7g. Example Solution for Three Player Game: Turn 7

For turn eight a new strategy is generated that does not want to retake Europe and Africa, but rather invade through south Asia and deny Australia (Figure 7.7h). This proves to be an excellent move and a fatal blow to player 1, because player 1 has only enough reinforcements to retake Australia. As a result, player 1 leaves the majority of his forces there where they are behind the lines and cannot be used effectively. Thus for turn nine, player 2 takes over Africa and Europe and part of Asia with little resistance (Figure

7.7i). Turn ten, not shown here, was the final invasion of player 2 over player 1, which was met with little resistance.

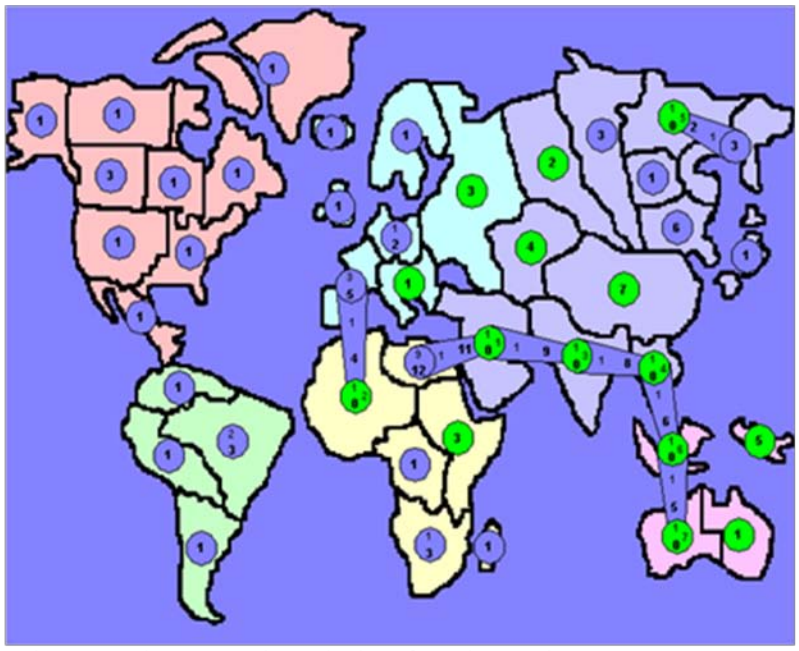

Figure 7.7h. Example Solution for Three Player Game: Turn 8

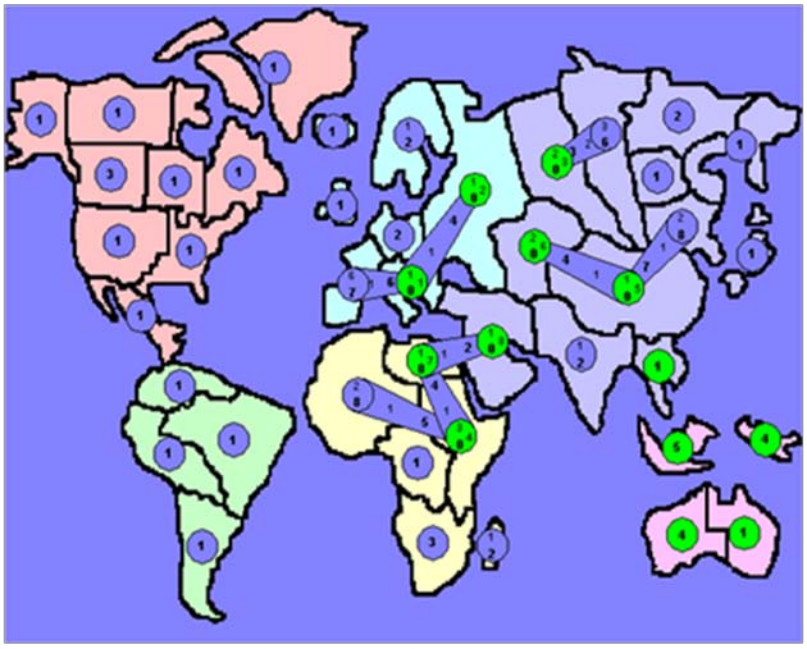

Figure 7.7i. Example Solution for Three Player Game: Turn 9

In summary, player 2, the developed dynamic planner generated moves that were intelligent and generalized well to various situations. In addition, the automated strategies were able to overcome poor dice rolls and still win the game. Similar games were played with similar results for three and more players. However, many more games need to be played in a more automated fashion so that statistical results can prove significant. The game was played 20 times and the dynamic player won 16 of those games or 80% of the time.

### 7.3.1.2 Probability of Successful Plan Completion

As described in Section 7.1.5, completed tasks for the game RISK are territory takeovers, and the probability of successful completion of all projected tasks is the combination of all dependent, intradependent, independent and interdependent battle sequences. A simplified and accurate probability of success can be calculated using all projected leaf nodes of a battle plan. Due to the use of the DBN described in Section 7.1.7, probabilities are propagated forward to the leaf nodes. Thus, for every leaf node that projects a successful task completion, its probability is considered in the overall success of the plan.

For example, given the RISK plan described in Section 7.1.5 and shown in Figure 7.3, this fourteen-step battle plan has probabilities as shown in Table 7.9. Each individual battle probability is given as $\Pr(\tau_{\#})$, and each probability of successful plan completion is given as $\Pr(T)$. Note that the set of successful leaf nodes (i.e., $\Pr(\tau_{\#}) > 0.5$) used to calculate the $\Pr(T)$ changes over the course of planning. This is calculated at each step in a plan by backtracking to determine which nodes are successful leaf nodes (SLN) at each

time increment that an action is selected. For instance, in battle fourteen, SLN = {4, 6, 10, 11, 12, 13, 14}, because battle seven is considered unsuccessful and battles 1, 2, 3, 5, 8, and 9 are no longer leaf nodes at the fourteenth step of the plan (see Figure 7.3). More specifically, the probability of successful plan completion is the product of the SLN (e.g., $\Pr(T) = \Pr(\tau_4)\bullet\Pr(\tau_6)\bullet \Pr(\tau_{10})\bullet\Pr(\tau_{11})\bullet\Pr(\tau_{12})\bullet\Pr(\tau_{13})\bullet\Pr(\tau_{14})$).

Table 7.9. Probability of Successful Plan Completion Example

| Battle # | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| $\Pr(\tau_{\#})$ | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 0.9995 | 1.0000 | 0.1061 |
| $\Pr(T)$ | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 0.9995 | 0.9995 | 0.9995 |
| Successful Leaf Nodes | 1 | 1,2 | 1,2,3 | 1,2,3,4 | 1,2,4,5 | 1,2,4,5,6 | 1,2,4,5,6 |

| Battle # | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|
| $\Pr(\tau_{\#})$ | 0.9968 | 0.9822 | 0.9995 | 0.9078 | 0.9719 | 1.0000 | 0.9967 |
| $\Pr(T)$ | 0.9963 | 0.9790 | 0.9785 | 0.9043 | 0.8817 | 0.8817 | 0.8789 |
| Successful Leaf Nodes | 1,4,5,6,8 | 1,4,6,8,9 | 4,6,8,9,10 | 4,6,8, 10,11 | 4,6,10, 11,12 | 4,6,10, 11,12,13 | 4,6,10,11, 12,13,14 |

The probability of successful plan completion can be used in the fitness function as described using Equation 7.6 in Section 7.2.5.2. The results of using this concept are best shown using an example problem. Consider an endgame problem where one player can finish the game by taking over four remaining players in a single turn. In this example, we consider only the eastern hemisphere of the game board and assume the current player already has control of North and South America. The current player has most of their forces on the four corners of the eastern hemisphere: Iceland, North Africa, Southeast Asia (Siam), and Northeast Russia (Kamchatka). The current player must construct a battle plan that completes the game using these four force areas, including initial turn reinforcements, and additional player takeover reinforcements.

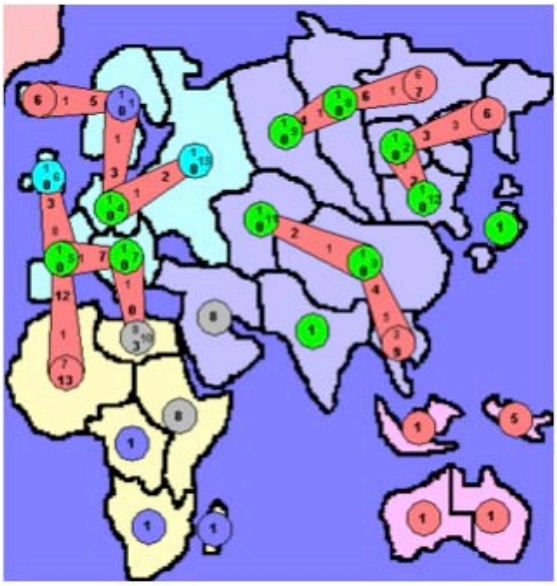

Figure 7.8a. First Takeover

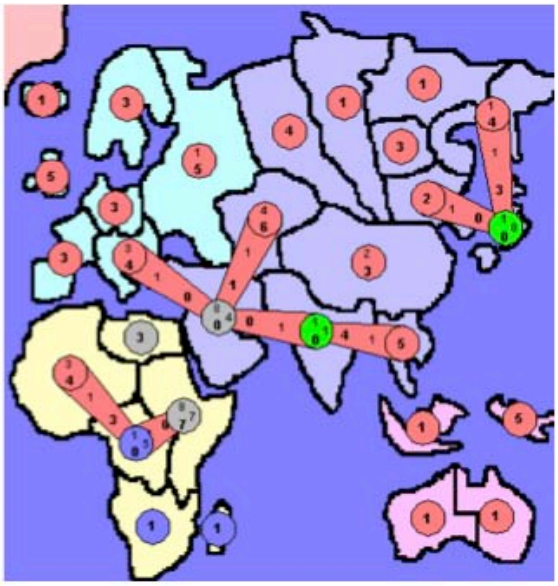

Figure 7.8b. Second Takeover

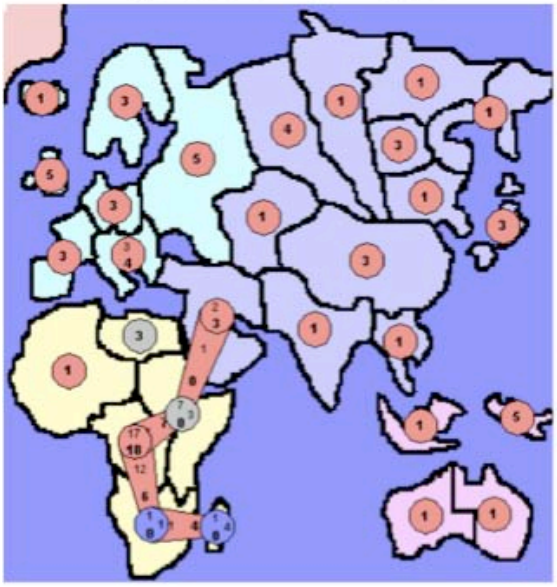

Figure 7.8c. Third Takeover

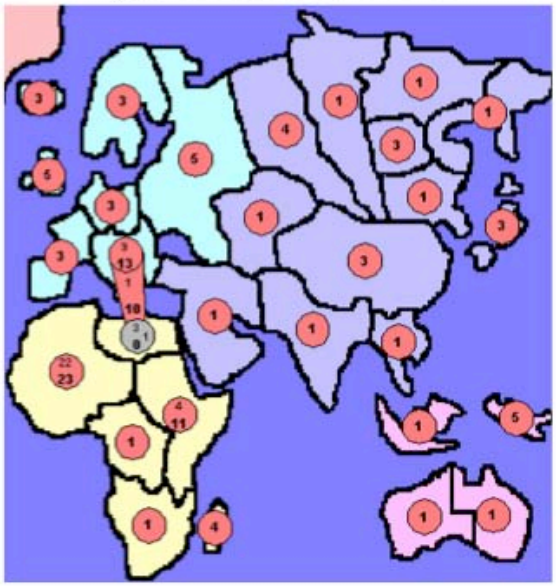

Figure 7.8d. Fourth Takeover

Figure 7.8. Evolving a Plan *Without* the Probability of Success

The search algorithm used to solve this problem is the same as described in Table 7.7 and the sub-goal objectives are the same as shown in Table 7.8. The choice of parameter values for both tables are however different: (1) the number of generations is not 10, but goes on until an endgame completion is reached successfully; (2) selections,

offspring, allocation recombinations, and battle recombinations are set to 8, 8, 1, and 1, respectively; and (3) the sub-goal weights are set at 0.45, 0.33, 0.02, 0.05, and 0.15 as in the order presented in Table 7.8.

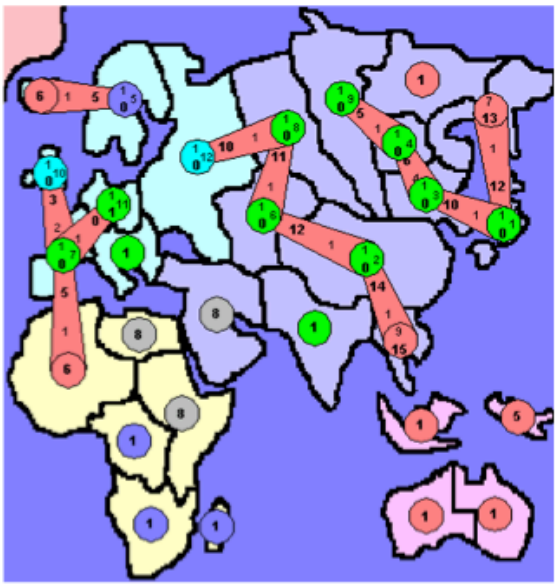

Figure 7.9a. First Takeover

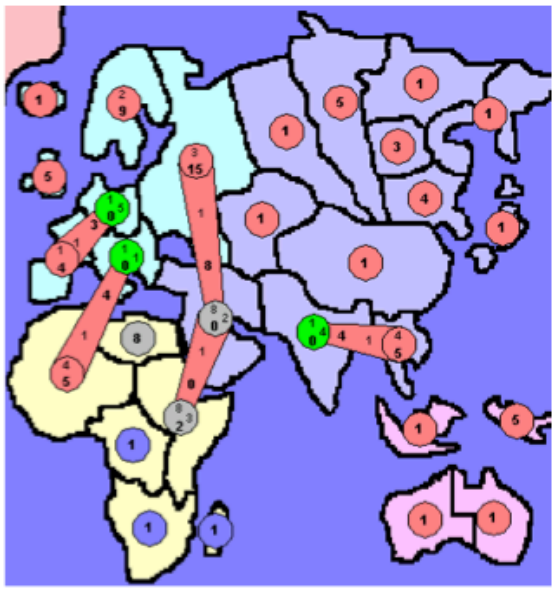

Figure 7.9b. Second Takeover

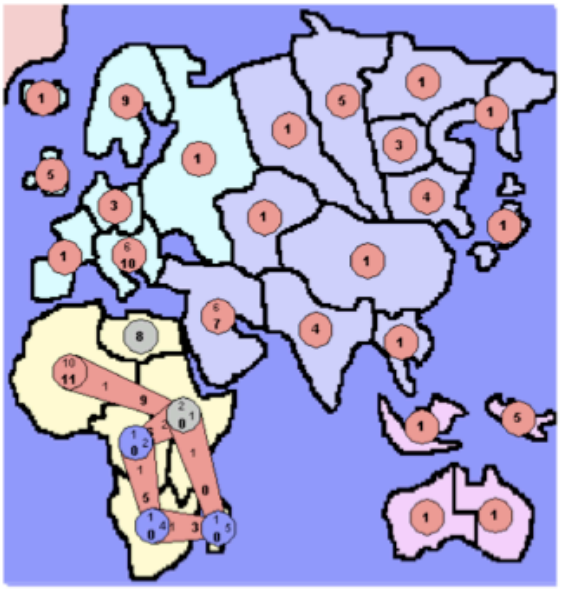

Figure 7.9c. Third Takeover

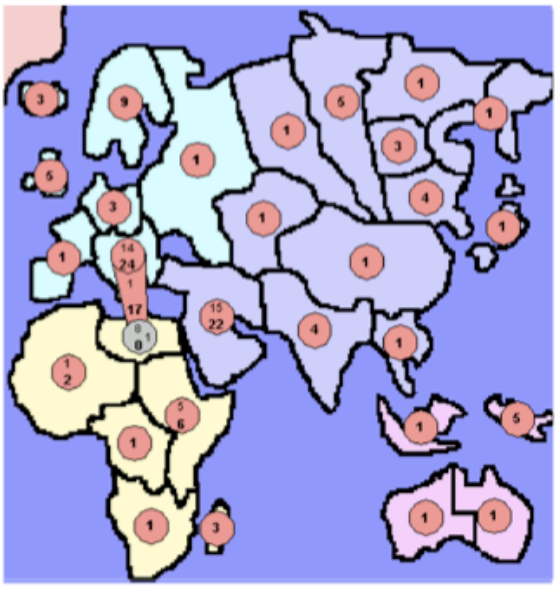

Figure 7.9d. Fourth Takeover

Figure 7.9. Evolving a Plan *With* the Probability of Success

The example problem is executed in two ways using Equation 7.6: (a) one with $\pi$ = 0 (*without* probability of success) and (b) the other with $\pi$ = 1 (*with* probability of success). Two example plans are shown in Figure 7.8a-d and Figure 7.9a-d, respectively. The four takeovers are shown in the order a, b, c and d for both figures. Note that in both Figures 7.8 and 7.9, the takeovers are in the same order (i.e., cyan, green, blue, then gray). However, most battles in Figure 7.8 have projected smaller forces attacking opponent territories, thus resulting in lower success probabilities (shown in Figure 7.10). However, when the success probabilities are taken into account in the fitness function as shown in Figure 7.9, the attacking forces are much larger in most cases.

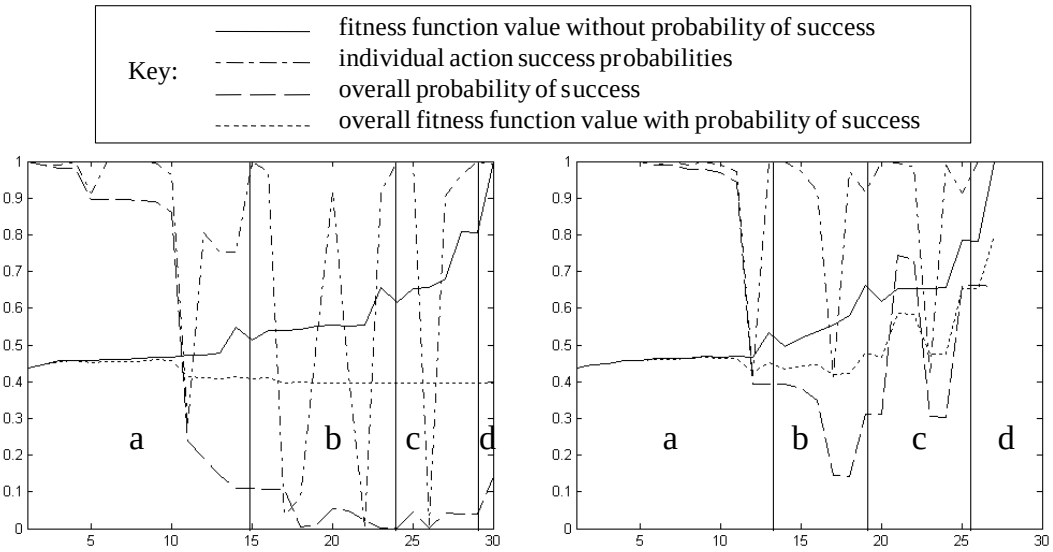


Figure 7.10a. Fitness Function *without* Success Probability

Figure 7.10b. Fitness Function *with* Success Probability

The results are summarized in Figure 7.10. Figure 7.10a and 7.10b illustrate four underlying components that track the difference in success of the two plans. Figure 7.10a represents the plan without the probability of success included, while 7.10b represents the plan with the probability of success. The x-axis represents each action, while the y-axis

either represents probability or fitness. The solid line in both figures represent the value of the fitness function without probability of success, the dot-dashed line represents individual action success probabilities, The dashed lines represent the overall probability of success based on multiplying leaf node probability, while the dotted line represents the overall fitness function value based on including the probability of success.

The most notable features from Figure 7.10a and b are the overall probability of success (dashed lines). For the simple fitness function case, the overall success probability is 14%; while for the case where the probability of success was included, competition drove the probability of success up to 66%. Thus, including the probability of success gives a far better chance of taking over all four opponents in this turn and will probably not require re-planning over the course of the turn.

### 7.3.1.3 Search Comparison

As shown in Section 7.3.1.2 above, the RISK endgame problem is an interesting and difficult problem with an enormous search space. The next logical step is to find an approach that best fits this sort of problem. Thus, this subsection looks at seven algorithms used to search for solutions for the endgame moves of the board game RISK. Comparison of these search methods for the best plan constrained by: (1) using a fixed evaluation function (shown in Table 7.10), (2) fixed time to plan (five minutes), and (3) randomly generated situations that correspond to endgames in RISK with eight remaining players (described in next subsection 7.3.1.3.1: Example Problem).

The search strategies compared are depth-first, breadth-first, best-first, random walk, gradient ascent, simulated annealing, and evolutionary computation. The first three

strategies are deterministic tree search methods described in detail in Chapter 3: Background. The random walk, gradient ascent, and simulated annealing approaches are Monte Carlo search methods defined in general terms in Chapter 3. Their implementation is application dependent, thus these three approaches are described more fully in Subsection 7.3.1.3.2: Monte Carlo Search Methods. The evolutionary computation approach is described in Subsection 7.3.1.3.3: Evolutionary Search. The approaches are compared for each example based on the number of opponents eliminated, plan completion probability, and value of ending position (if the moves do not complete the game). Results are shown in Subsection 7.3.1.3.4: Search Results.

Automated planning for endgame scenarios is presented where the temporal cost of planning is weighed against measured progress toward the goal and the probability of successful completion of the plan. The focus is on problems with no certain path to the endgame, and thus no supervisor is available to train a system to generate the best move. Known or calculable state-transition probabilities are used to prune the enormous planning search-space and provide an expected value to the end-state to aid in a more efficient search and greater plan completion probability. Planning problems that involve opponents require objectives that not only direct the automated player to the endgame, but also to interim positions where it is less vulnerable to opponents. A dynamic plan based on an effective search algorithm appears to be an important step toward solving these types of problems. An efficient search algorithm is required for such problems because the search process is usually the most resource-consuming (time and memory) aspect of the planning process (plan-generation).

### 7.3.1.3.1 Example Problem

RISK endgame scenarios have been chosen for analysis because these situations are the turns with the highest complexity and variability. Solving them provides valuable insight into solving the overall game. Depending on the number of players (i.e., two to eight) and the maturity (i.e., turn) of the game, strategies within the RISK game change. At some point in the game, players will be eliminated, many times in bunches, because elimination of one player often provides resources that can lead to elimination of another, and so on. When this point in the game is reached, it is advisable to eliminate all other players. This situation is the common RISK endgame situation referred to here.

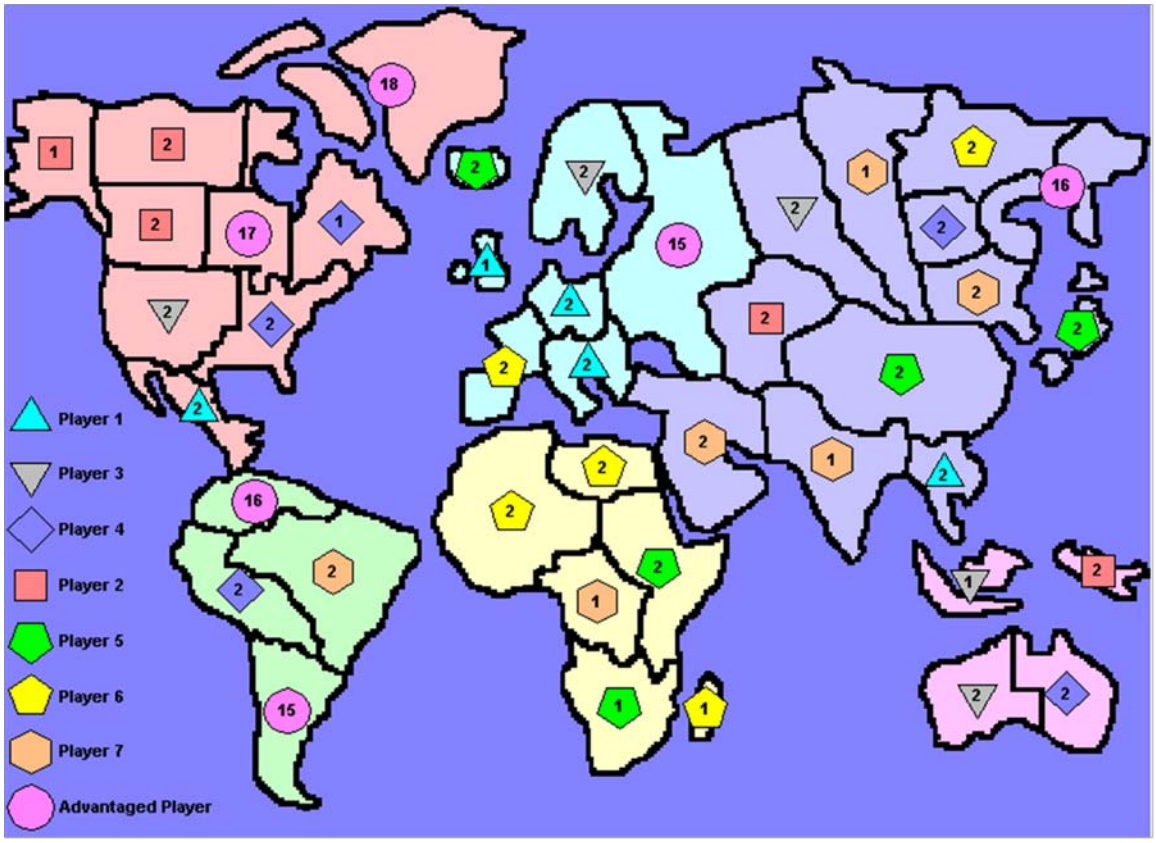


Figure 7.11. Example RISK Endgame Situation

For benchmarking purposes, situations were created that represent possible endgame-scenarios. Sixty endgame situations were generated where one player can win the game in one final well-played turn. More specifically, one player is given approximately 60% of the forces to allocate to their occupied countries (97 forces), while the other seven players split the remaining forces evenly (nine per player). An example is shown in Figure 7.11. The territories with circular symbols are the advantaged player territories, while all other territories represent other players. The different shapes represent different players as shown in the key to the left. For these scenarios, territories on the board are given out randomly in turn, thus each player is given approximately the same number: six players get five territories while the other two get six, equaling the total 42 territories on the board. After the territories are selected, players place units on their territories in turn until all (7 x 9 + 97 = 160) initial allocations are exhausted.

Placing the forces is not random, but based on a tuned fitness function shown in Table 7.10. This leaves the advantaged player with approximately sixteen forces per territory while the others average around two forces. Each player is also given a random number of cards from zero to six, where six is the maximum number any player can hold after their turn is complete. Lastly, the value of turned-in card sets number is initialized to five and subsequent turn-ins will provide resources of 12, 15, 20, 25, etc., as cards are turned in. The objective of the algorithm is to develop a solid plan with which the advantaged player can eliminate the other players from the game, and thus win the game. However, as will be shown, finding a solid plan is not easy given the complex variety of possible moves.

In the fitness function shown in Table 7.10, not all of these objectives are of equal importance, and weighting them is situation dependent. For instance, if only objective 1 is used, no battles would be fought because one would not risk losing any troops on a turn. Conversely if one only used the reinforcement objective 2 then the player would fight to the end, becoming so weakened, that they might not last to their next turn. Balancing these objectives is an issue, and these weightings were tuned by hand based on game simulations played among opponents with different weightings. These objective weightings can be learned from autonomous playing of the game against itself as shown in Section 7.3.3.

Table 7.10. Second Fitness Function

| Sub-Goal to Maximize | Reward Values | Weights |
|---|---|---|
| 1. Overall Strength | $\frac{\text{Individual Force}}{\text{Total Forces}}$ | 0.37 |
| 2. Expected Reinforcements | $\frac{\text{Individual Reinforcements}}{\text{Total Reinforcements}}$ | 0.35 |
| 3. No. of Defensive Fronts | $1 - \frac{\frac{\text{Individual Frontal Territories}}{\text{Individual Total Territories}}}{\frac{\text{Total Frontal Territories}}{\text{Total Territories}}}$ | 0.1 |
| 4. Average Front Strength | $\frac{\frac{\text{Individual Frontal Forces}}{\text{Individual Border Territories}}}{\frac{\text{Total Frontal Forces}}{\text{Total Border Territories}}}$ | 0.06 |
| 5. Logistical Support | $\frac{\frac{\text{Individual Frontal Forces}}{\text{Individual Support Forces}}}{\frac{\text{Total Frontal Forces}}{\text{Total Support Forces}}}$ | 0.06 |
| 6. Smallest Front Strength | $\frac{\text{Individual's Minimum Frontal Force}}{\text{Sum Total of Minimum Frontal Forces}}$ | 0.06 |

### 7.3.1.3.2 Monte Carlo Search Methods

**Random Search Method:**

A good baseline for comparison is random search method, where each action is chosen randomly at every step of the plan until the plan reaches an end. In this case, the end is where no more battles can be fought because resources are misallocated or exhausted, or the plan reaches the end goal of winning the game. Random search is a form of depth first search that does not expand the nodes at each step in the plan. The random search used here is:

1. Generate one plan from beginning to end, choosing random actions
2. Prune plan back to fittest point
3. If generation > 1 compare with stored plan, store fittest
4. If plan meets goal or time expires terminate, otherwise go to 1

Note that this random search method generates each new plan from scratch.

**Gradient Ascent Search Method:**

Gradient search is concerned with ascending up the fitness slope over multiple dimensions to reach the goal. The dimension of the planning space can be seen as each step in the plan. Since the fitness landscape is not a smooth surface in the planning space, gradient search will seldom reach the goal by ascending the fitness at each step in the plan. Thus, a heuristic form of gradient must be used to get more competitive results. Described below is one such method that uses gradient ascent search to find the goal, where each dimension is a plan step:

1. Generate new plans to the end (one in 1$^{st}$ generation), choosing random actions
2. If generation > 1 compare with stored plan

3. Select and store fittest plan
4. Prune back stored plan to each step from beginning to fittest point
5. Use each pruned plan for regeneration
6. If plan meets goal or time expires terminate, otherwise go to 1

**Simulated Annealing Search Method:**

Simulated annealing uses a declining temperature factor to initially allow wide variation in choices then over time the declining temperature restricts the search to only a few choices. This approach overcomes the rough fitness surfaces of many planning spaces, such as observed in RISK. Unlike evolution, simulated annealing is only concerned with modifying one selection, in this case, one plan. After some experimentation, the most competitive simulated annealing approach found uses temperature to determine how far back the plan is pruned. With high temperature, the plan is likely to be pruned back near the beginning. As temperature falls, pruning is more likely to occur near the end of the plan. More specifically, the fitness at time zero is weighted by a ramp function that runs from one to zero over the length of a plan. This ramp function is multiplied by the fitness to determine the new skewed fitness used in this technique. As time goes from zero to 300 seconds, the ramp function slowly transitions to a flat function, thus allowing the true fitness measure to have a greater chance for selection. This ramp method was found to be successful and a ramp that goes beyond a flat slope was found to be less successful. The simulated annealing search method used is:

1. Generate one plan to the end, choosing random actions
2. If generation > 1 compare with stored plan
3. Select and store fittest plan

4. Prune back plan based on fitness and weighting (described above), selected randomly
5. Pruned plan is the beginning plan for next generation
6. If plan meets goal or time expires terminate, otherwise go to 1

#### 7.3.1.3.3 Evolutionary Search Method

All search strategies used are in the family of heuristic search algorithms. Seven search strategies are used to compare and contrast approaches. These approaches include classical methods (heuristic depth-first, breadth-first and best-first search), random search, a form of gradient search, simulated annealing, and our evolutionary search. To provide consistency across comparisons the same DBNs and fitness function are used. To be fair, all search algorithms use a form of alpha-cutoff to avoid dead-end paths when searching. Alpha-cutoff prevents unnecessary search in any direction that cannot exceed its currently found best plan. Normally alpha-cutoff is used in tree search to avoid dead-end branches. For stochastic search, we use alpha-cutoff when the best possible reward given the current probability cannot exceed the best currently realized reward. Alpha-beta cutoff is used in minimax games when considering two players' moves, but since we are only planning a single player's move at this time, alpha-cutoff is a more appropriate method to use.

The classical search methods, random search, gradient search and simulated annealing are described fully in the Chapter 3: Background. The developed evolutionary search method used is very similar to the one defined in Table 7.5, with the inclusion of adding parent selections to the algorithm at two places: after step 1 and after step 3 in Table 7.5. The number of selections, offspring, and recombinations can be chosen at the

discretion of the user, thus, experiments were run to determine the best evolutionary parameters for the RISK endgame problem. Experimental results are shown in Table 7.11.

Table 7.11. Selecting Evolutionary Parameters for Plan-Generation

| Tested Parameters | | | Plans' Average Results | | | | |
|---|---|---|---|---|---|---|---|
| Selections | Offspring | Recombina-tions | Generations | Population per Generation | Planning Time in Seconds | Fitness | Reached Goal Percentage |
| 1 | 1 | 0 | 2,894 | 3.0 | 300 | 0.438 | 0.000 |
| 1 | 2 | 0 | 1,862 | 5.0 | 295 | 0.501 | 0.017 |
| 2 | 1 | 0 | 1994 | 6.0 | 300 | 0.452 | 0.000 |
| 2 | 2 | 0 | 766 | 8.0 | 239 | 0.626 | 0.317 |
| 2 | 3 | 0 | 462 | 10.0 | 230 | 0.644 | 0.400 |
| 3 | 2 | 0 | 471 | 12.0 | 218 | 0.646 | 0.383 |
| 3 | 3 | 0 | 396 | 15.0 | 213 | 0.648 | 0.400 |
| 1 | 1 | 1 | 142 | 21.1 | 168 | 0.673 | 0.600 |
| 1 | 2 | 1 | 44.2 | 32.7 | 170 | 0.671 | 0.567 |
| 2 | 1 | 1 | 71.4 | 31.9 | 166 | 0.658 | 0.583 |
| 2 | 2 | 1 | 41.5 | 48.5 | 200 | 0.663 | 0.533 |
| 2 | 3 | 1 | 27.8 | 67.3 | 170 | 0.669 | 0.633 |
| **3** | **2** | **1** | 23.3 | 68.1 | 237 | **0.679** | **0.683** |
| 3 | 3 | 1 | 20.9 | 87.6 | 183 | 0.668 | 0.583 |
| 4 | 4 | 1 | 14.6 | 154 | 184 | 0.664 | 0.567 |
| 5 | 5 | 1 | 10.4 | 218 | 200 | 0.656 | 0.500 |
| 6 | 6 | 1 | 6.4 | 272 | 198 | 0.666 | 0.550 |
| 7 | 7 | 1 | 6.6 | 370 | 218 | 0.654 | 0.483 |
| 8 | 8 | 1 | 4.7 | 473 | 220 | 0.641 | 0.467 |
| 2 | 2 | 2 | 25.3 | 82.4 | 185 | 0.656 | 0.550 |
| 3 | 3 | 2 | 12.4 | 141 | 193 | 0.666 | 0.583 |
| 4 | 4 | 2 | 8.4 | 223 | 217 | 0.651 | 0.450 |
| 3 | 3 | 3 | 9.0 | 195 | 199 | 0.665 | 0.617 |
| 4 | 4 | 3 | 5.1 | 307 | 218 | 0.652 | 0.467 |

Table 7.11 illustrates the results of testing 24 different parameter combinations. The sample size was 60 for each case, so the results are only approximate. Parameters 3,2,1 for $s_{se}$, $s_{of}$, $s_{re}$ had the greatest fitness and goal percentage. It can also be shown from the table that when recombination equals zero, the plans perform quite poorly. Also, for multiple recombinations, the results are not as good as using one recombination, probably because there are far more plans per generation than the number of generations. It is

likely that better choices for these values can be found, or even that they should vary subject to the conditions of the game.

Note that the number of phases is dependent on how deep the plan goes. A phase is defined as shown in Figure 7.2, where Phase I is the beginning turn phase, Phase II is the allocation phase, Phase III is the battle phase, and Phase IV is the final move phase. A plan for the game RISK resides in either Phase I, II, III, or IV, with the possibility of repeating Phases II and III multiple times when opponents are taken over and cards are turned in. Thus, for a 7 takeover endgame situation, the number of phases can be as many as 15: one phase I and seven each of phases II and III. Phase IV is not important because a final move is not necessary in the RISK endgame. Therefore, based on the algorithm shown above, the number of recombinations per plan is directly proportional to the number of phases in a particular plan. As shown in Table 7.11, when recombination is employed, the population size varies greatly with the number of phases.

### 7.3.1.3.4 Search Results

All search strategies used here are from the family of heuristic search algorithms. Variations on classical search (e.g., depth, breadth and best), random, gradient ascent, simulated annealing, and evolutionary computation are compared to see how they perform in the endgame RISK problem. To put all search algorithms on a level playing field, a common environment and framework are used: (1) a dynamic Bayesian network (DBN) is calculated on the fly for contemplating attack scenarios, (2) the same evaluation function to measure progress toward the goal, (3) a fixed five-minute time constraint for

processing, (4) and run in Matlab Release 14 on a Pentium IV 3.2 GHz processor with one gigabyte of RAM.

Experiments were performed for the seven cases illustrated in Table 7.12. The time limit set for all experiments was 300 seconds (five minutes) as shown in the right-most column of Table 7.12. The darker shaded boxes in Table 7.12 indicate parameters not applicable (N/A) or fixed in value. Some values are adapted based on either time limit for generations or plan length for selections.

Table 7.12. Search Strategies Used and Corresponding Parameters

| Search Strategy | Parameters | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Decision-Tree Search Type | Alpha Cutoff | Generations | Selections | Offspring | Recombina-tions | Trials | Time Limit (seconds) |
| Classical | Breadth-First | Yes | 1 | 1 | 1 | N/A | 5 | 300 |
| | Best-First | Yes | 1 | 1 | 1 | N/A | 5 | 300 |
| | Depth-First | Yes | 1 | 1 | 1 | N/A | 20 | 300 |
| Random | N/A | N/A | adapted | 1 | 1 | N/A | 20 | 300 |
| Gradient Ascent | N/A | N/A | adapted | adapted | 1 | N/A | 20 | 300 |
| Simulated Annealing | N/A | N/A | adapted | 1 | 1 | N/A | 20 | 300 |
| Evolutionary | N/A | Yes | adapted | 3 | 2 | 1 | 20 | 300 |

The selection of five minutes of playing time is obviously arbitrary, and depends heavily on the programming language used and the speed of the machine. For these experiments, five minutes was chosen because it was at a cusp where effective search algorithms could frequently find solutions and ineffective ones could not. It also gave a total execution time for the planned simulations that was tractable. If the algorithms were rewritten in a compiled language and optimized for speed, it is not unreasonable to expect the required time to decrease by as much as a factor of ten. The resulting time of 300

seconds for the automated player to complete its turn should be acceptable to human opponents.

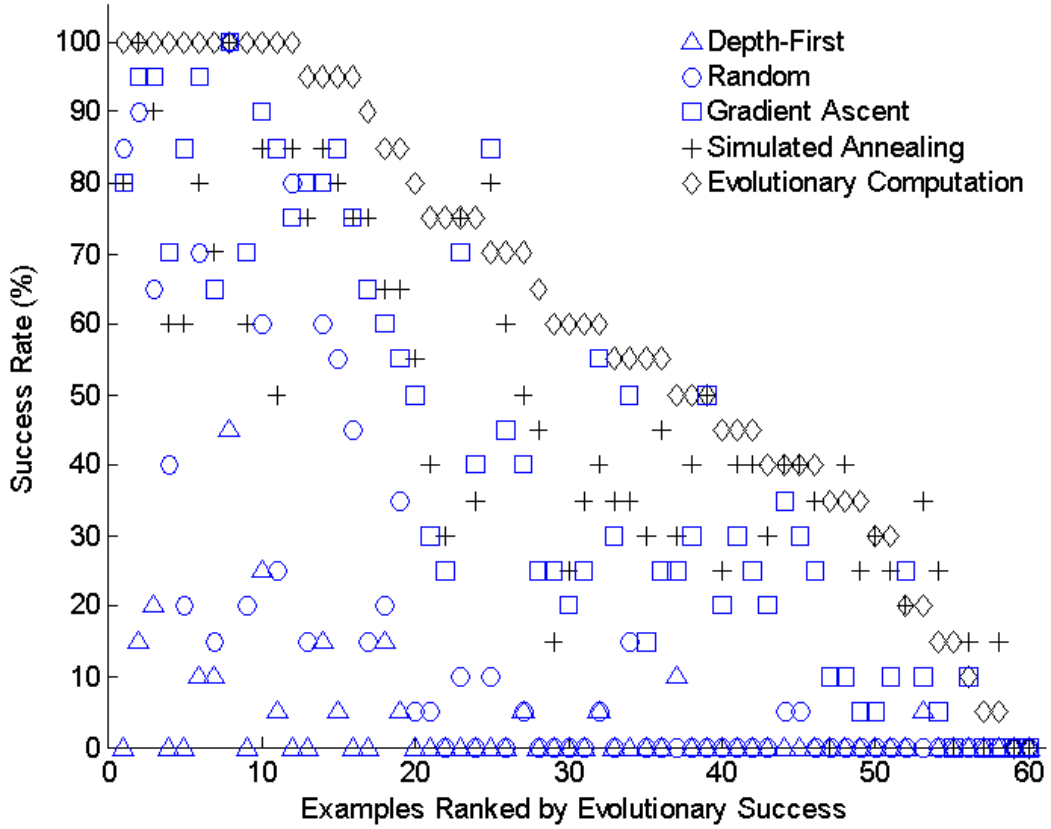


Figure 7.12. Average Success Rate Comparison for All 60 Examples

Figure 7.12 illustrates the success rate of the five best approaches. Breadth-first and best-first searches are not illustrated here, because their success rate was zero for all the example cases. Since all the examples were randomly generated, some of them were simpler than others. Thus, for illustrative purposes, the examples are ordered based on evolutionary success rate for an easier comparison. Each example was run 20 times and the average success rate is plotted in Figure 7.12 from the easiest examples to the most difficult. Note that for the twelve easiest examples the evolutionary approach was able to solve all of them 100% of the time, while simulated annealing and other approaches

could only solve at most two of them consistently. From Figure 7.12, one can see that simulated annealing is better than the evolutionary approach in only six cases, while in all other cases evolutionary computation is better or the same as the simulated annealing method. Gradient ascent was the best in only three cases and random search and depth-first search were never better than the evolutionary approach.

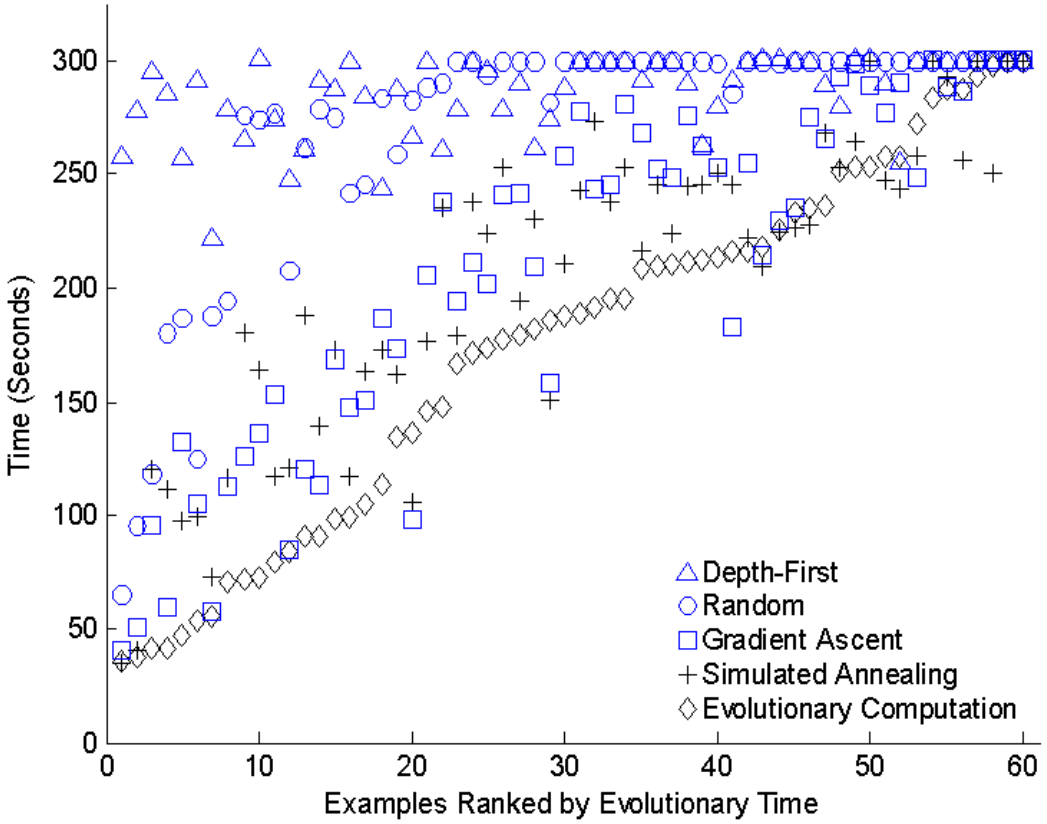


Figure 7.13. Average Search Completion Time Comparison for All 60 Examples

Another important result is the average time taken for each example. The top five approaches are illustrated in Figure 7.13 where the examples are ordered by average evolutionary time. Again, breadth- and best-first searches are not plotted because they never reached success and therefore always terminated at the time limit of 300 seconds. The evolutionary approach is predominantly better than the others, however the simulated

annealing approach was faster in ten cases and the other approaches were better in only four cases. Note that depth-first search either did very poorly or very well depending on the example and was overall the poorest approach.

Table 7.13. Search Results Summary

| Search Method \ Plans' Average | Generations | Population per Generation | Reward | Completion Probability | Fitness | Fitness Standard Deviation | Number of Takeovers | Reached Goal Percentage | Planning Time in Seconds |
|---|---|---|---|---|---|---|---|---|---|
| Breadth-First | 1 | **9281** | 0.404 | **0.991** | 0.404 | **0.008** | 0.00 | 0.00 | 300+ |
| Best-First | 1 | 8031 | 0.417 | 0.933 | 0.411 | 0.018 | 0.24 | 0.00 | 300+ |
| Depth-First | 1 | 6434 | 0.476 | 0.655 | 0.477 | 0.046 | 2.62 | 3.25 | 286 |
| Random | 1577 | 1 | 0.933 | 0.429 | 0.592 | 0.045 | 6.23 | 16.3 | 271 |
| Gradient Ascent | 64.6 | 45.1 | 0.973 | 0.467 | 0.645 | 0.057 | 6.68 | 42.3 | 209 |
| Simulated Annealing | **1730** | 1 | **0.974** | 0.475 | 0.653 | 0.057 | **6.71** | 46.2 | 204 |
| Evolutionary Comp. | 23.3 | 68.1 | **0.974** | 0.506 | **0.671** | 0.054 | 6.70 | **60.9** | **175** |

The average performance of all seven approaches is summarized in Table 7.13. The approaches are listed from worst to best with the best performance for each category in bold for each column. Breadth and best-first search, which had very limited success, were only performed over five trials. All other searches were performed 20 times. The goal for each search was to find a plan that wins the game with a greater then 50% probability within the 300-second time limit. The average reward and fitness for all initial random situations was 0.378. Since fitness is a measure of change in reward multiplied by the completion probability as described in Section 5.4, one can see that the classical searches had higher completion probability but very little improvement in reward and consequently, fitness. In other words, the classical approaches found short low risk plans that made little consideration for ending the game. This is also seen in the takeover column, where the three classical search methods achieved between zero and three takeovers on average. In contrast, all the other approaches achieved excellent

takeover rates, averaging between six and the goal number, seven. The large difference among these top-four approaches is found in the two far right columns. These two columns measure performance and speed, respectively. The evolutionary approach had a 31.8% increase in performance and a 14.2% increase in speed over its best competitor, simulated annealing.

### 7.3.2 Planning Cycle

The planning cycle not only includes plan-generation as described in Section 7.3.1, but also includes plan-execution and plan-assessment. This iterative planning cycle is summarized in Table 7.14. The algorithm for steps 1.1, 1.2, and 1.3 are identical except for the number of generations (i.e., $s_{es}$, $s_{rs}$, and $s_{cs}$) and are described in Table 7.5. As shown in Table 7.14, the inclusion of plan-execution and -assessment provides an opportunity for the planner to change plans after each task.

Results in this section will demonstrate the need for task feedback within the planning-cycle, and the impact of this feedback on planning parameters. Experimental results are examined in two areas: planning cycle search strategies and risk-aversion. Plan-generation creates a plan for execution, but based on early execution results, a plan may need further search. Planning cycle search strategies are the inclusion of these three search parameters $\{s_{es}, s_{rs}, s_{cs}\}$ described in Section 7.2.5.1 Search and Select Plans. Results here demonstrate the need for using all three parameters versus using only one or two of them. In the RISK game application, the risk aversion coefficients described in Section 7.2.4 Generate Random Plans ($r_Y$) and Section 7.2.6 Assess Plans ($r_Q$) are assumed equivalent for simplicity. As shown in Table 7.14, the expectations acceptance

threshold coefficient ($r_Q$) is used to determine whether to replan $s_{cs}$ generations if all remaining tasks are projected to be successful, or to replan $s_{rs}$ generations if any remaining task is projected to fail. The results here will show that the strength of a player varies with different risk aversion coefficient values.

Table 7.14. Pseudo-Code Representation of Planning Cycle

| Planning Cycle Tasks (in order unless otherwise specified) |
|---|
| 1. Plan-generation (generate plan for entire turn) |
| 1.1. Beginning turn: loop plan search by evolving $s_{es}$ generations: *go to step 2* |
| 1.2. After projected task failure: loop plan search by evolving $s_{rs}$ generations: *go to step 2* |
| 1.3. After projected task success: loop plan search by evolving $s_{cs}$ generations: *go to step 2* |
| 2. Plan-execution (execute actions until task is completed or fails) |
| 2.1. If plan empty: ***stop cycle and pass to next player step 1*** |
| 2.2. Else if action can be completed, execute action and truncate plan |
| 2.2.1. If task is successful and plan empty: *go to step 1* |
| 2.2.2. Else if task successful: *go to step 3* |
| 2.2.3. Else if task fails, set plan to null: *go to step 1* |
| 2.2.4. Else: *repeat step 2* |
| 2.3. Else set plan equal to null: *go to step 1* |
| 3. Plan-assessment (set plan-remainder equal to plan and reenact plan-remainder) |
| 3.1. If plan-remainder empty: *go to step 1* |
| 3.2. Select first plan-remainder action and truncate plan-remainder |
| 3.2.1. If action achieves expectations ($Q \bullet r_Q \geq Q''$): *repeat step 3* |
| 3.2.2. Else a projected task failure and set plan equal to null: *go to step 1* |

### 7.3.2.1 Search Strategies

To demonstrate the usefulness of including all three search parameters in plan-generation (i.e., $s_{es}$, $s_{rs}$, and $s_{cs}$) as described in Table 7.14, a simulation was performed to compare these parameters. Four players were used: one player used only one necessary parameter {$s_{es}$}, the second player used two parameters {$s_{es}$, $s_{cs}$}, the third player used two different parameters {$s_{es}$, $s_{rs}$}, and the fourth player used all three parameters {$s_{es}$, $s_{rs}$,

$s_{cs}$}. The parameters not used were set to zero, and the parameters used were set to equal values, provided the time per turn was approximately the same.

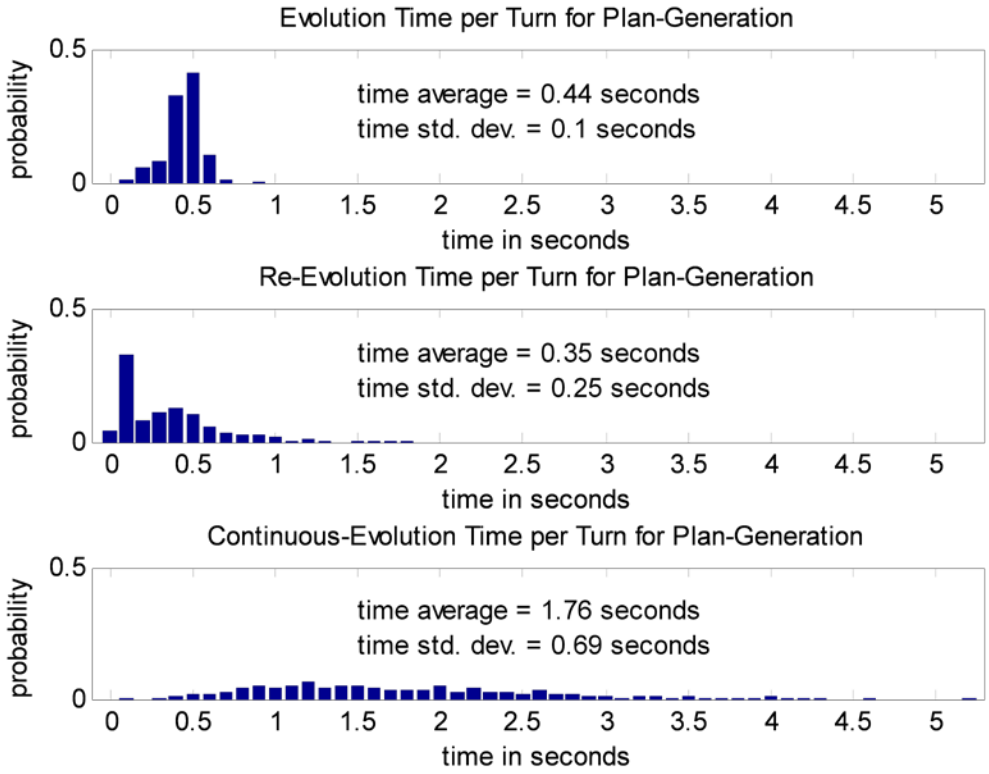


Figure 7.14. Time per Turn Generation for Three Search Parameters {$s_{es}$, $s_{rs}$, $s_{cs}$}

Figure 7.14 provides a distribution of the time costs per turn when using each of the three search parameters {$s_{es}$, $s_{rs}$, $s_{cs}$}, with all other remaining parameters set as described in Section 7.3.1.3.1. Note that time to search at the beginning of a turn (top graph) has the smallest standard deviation, because this type of search is always used only once per turn. Note that when tasks are projected to fail (middle graph) it has the smallest time average, because this case requires a plan failure and is thus not used in every turn. Also, note that continuous-evolution occurs after every successful task

completion, so the time impact is the greatest (bottom graph). In addition, since turns can vary largely in task length, the standard deviation is also the largest.

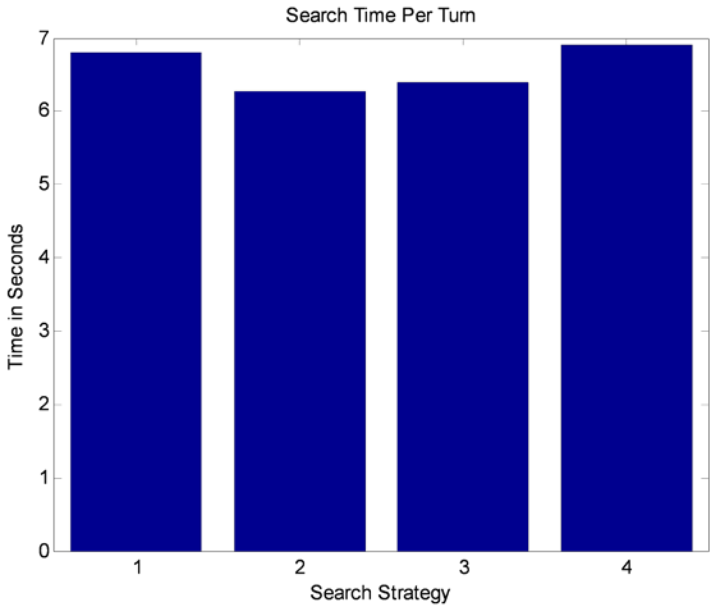


Figure 7.15. Time per Turn for Four Different Strategies

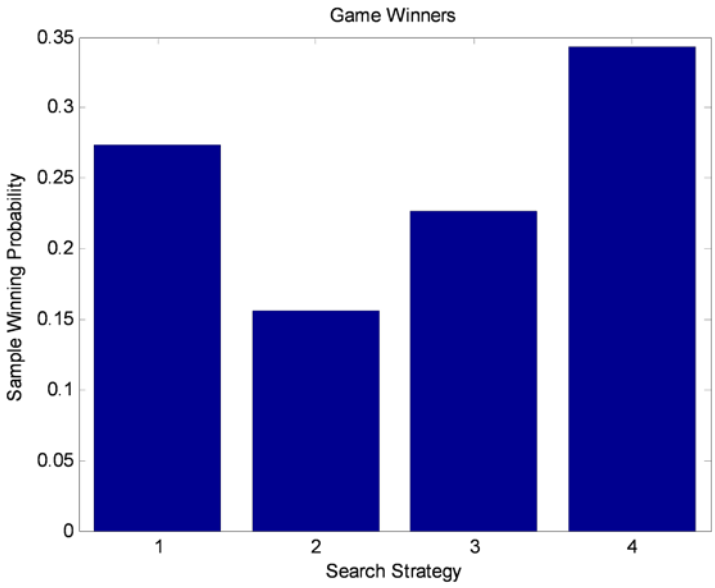


Figure 7.16. Winning Probability for Four Different Strategies

To compare the value of these three parameters, each of the four players described above was used twice in each of 128 eight-player games, and each of the parameters was set to give a similar average search time per turn. Player one strategy was set to: $s_{es}$ = 16, $s_{rs}$ = 0, and $s_{cs}$ = 0; player two strategy was set to: $s_{es}$ = 4, $s_{rs}$ = 0, and $s_{cs}$ = 4; player three strategy was set to: $s_{es}$ = 9, $s_{rs}$ = 9, and $s_{cs}$ = 0; and player four strategy was set to: $s_{es}$ = 5, $s_{rs}$ = 5, and $s_{cs}$ = 3. The time-per-turn for these four player strategies is shown in Figure 7.15. Note that the time per turn is approximately the same for all parameter settings.

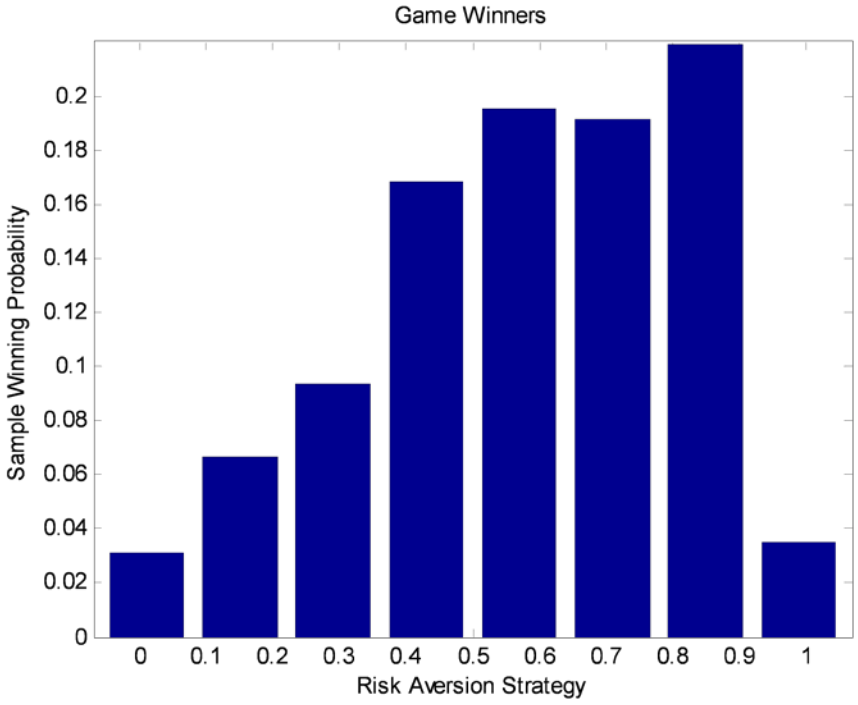


Figure 7.17. Winning Probability for Eight Risk Aversion Strategies

Over the course of 128 games, the winning performance was not equivalent and is shown in Figure 7.16, respectively. The figure shows that searching only at the beginning of a turn (strategy one) is an important factor for winning, but having all three parameters greater than zero performed the best (strategy four). Note that including re-

planning after a projected task failure increased the winning performance by more than a factor of two (i.e., comparing strategy two with four).

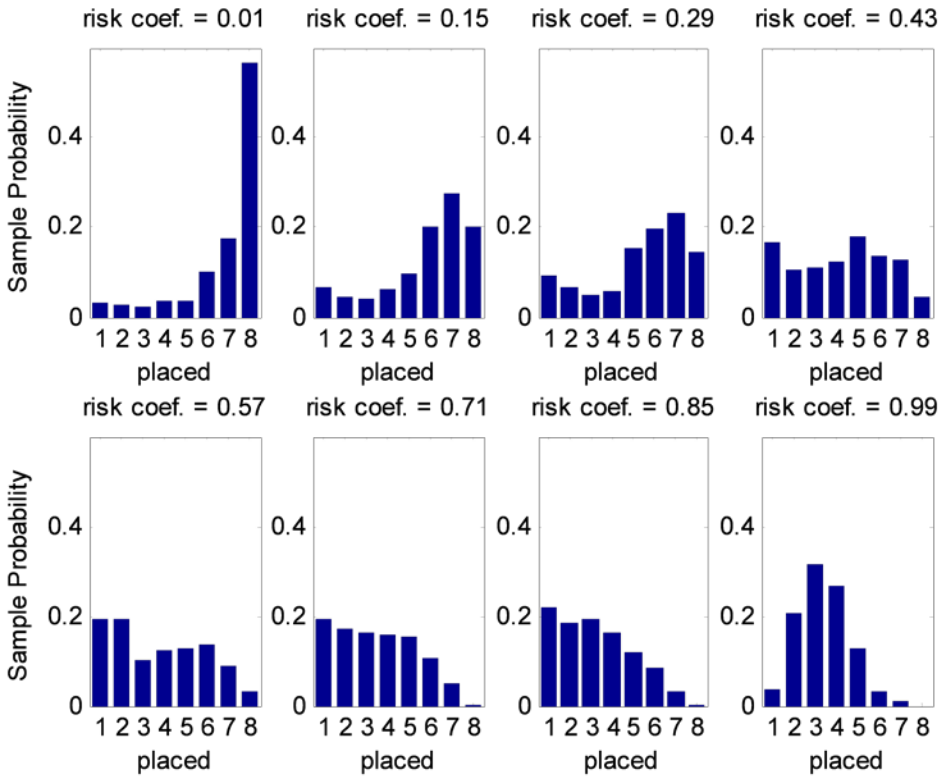


Figure 7.18. Sample Probability of Order Placement Based on Risk Aversion Strategy

### 7.3.2.2 Risk Aversion

For the game RISK, the risk aversion coefficients can vary from zero to one and determine each action's outcome ($r_Y$) and re-planning threshold ($r_Q$). As mentioned above, for simplicity it is assumed that $r_Y = r_Q$. As described in Section 7.2.4, if risk aversion equals zero then the player will always fight, because the player assumes that every battle will be won. If risk aversion equals one then the player will never fight, because the player assumes that every battle will be lost. Thus, the best value(s) of risk aversion are probably somewhere between zero and one. To determine good values,

eight values between zero and one were compared. These chosen values were equally spaced from 0.01 to 0.99 and are: {0.01, 0.15, 0.29, 0.43, 0.57, 0.71, 0.85, 0.99}. Using an eight-player game, 256 games were simulated where each player uses one of these eight risk aversion values. The player order and the player to start are scrambled to avoid redundant games. Figure 7.17 shows the sample probability mass function of winning the game. Note that having a value greater than expected value ($r_Y$, $r_Q$ = 0.5) performed better, and $r_Y$, $r_Q$ = 0.85 performed the best overall.

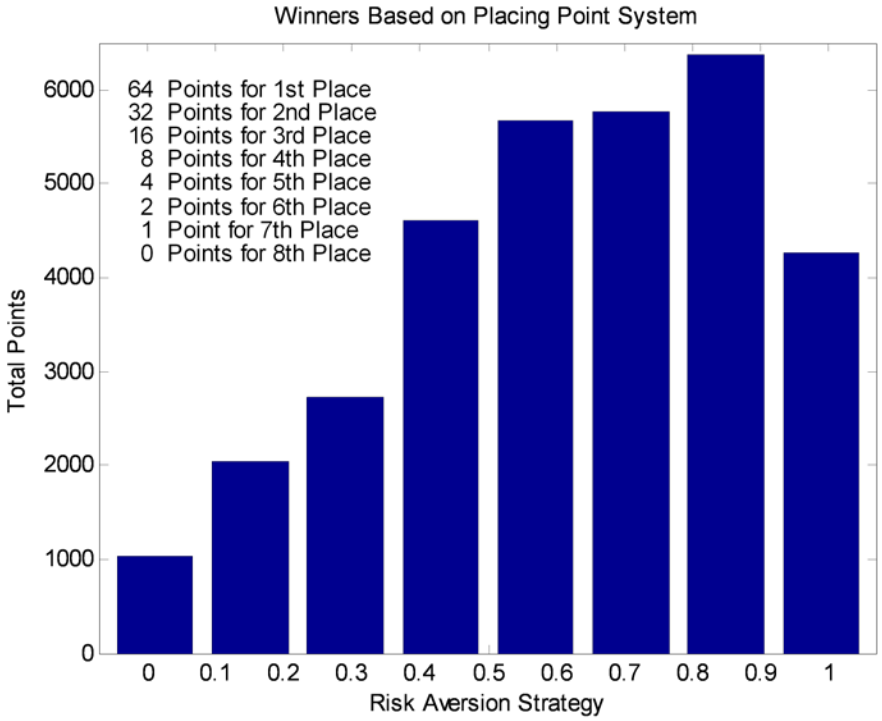


Figure 7.19. Point Totals for Eight Risk Aversion Strategies

Another interesting result from the simulation is shown in Figure 7.18. Here, the winning rank of each player is shown (1 is the winner, 8 is the first eliminated). Notice that the highest risk player $r_Y$, $r_Q$ = 0.01, usually comes in last place, because the player depletes his forces most quickly and is thus most likely to be eliminated first. Conversely,

the lowest risk player $r_Y$, $r_Q$ = 0.99, never comes in last place, because this player stacks his forces and is seldom attacked early in a game. If a winner-take-all strategy were not used as shown in Figure 7.17, and placement did matter in each game, then the winning points for players might be as shown in Figure 7.19. This reward strategy is similar to a PGA golf tournament payouts, where the first player receives twice as many points (dollars in golf) as the second player; the second player receives twice as many as the third player; and so on. In comparison to Figure 7.17, being highly risk averse ($r_Y$, $r_Q$ = 0.99) pays more dividends under this point system strategy.

### 7.3.3 Training RISK Players

The approach taken here trains RISK players using a tournament strategy, where iterative loops of rounds, games, player's turns, and planning cycles are used to choose the best players and further improve their abilities through modifying their parameters. There are four iterative loops within a tournament that lead to training players:

Tournament (described in Section 7.3.3)
Loop Rounds (described in Section 7.3.3)
Loop Games (described in Section 7.3.3)
Loop Players' Turns (i.e., game rules: described in Section 7.1.3)
Loop Planning Cycle (described in Section 7.3.2)
Plan-Generation (described in Section 7.3.1)
Plan-Execution (described in Section 7.3.2)
Plan-Assessment (described in Section 7.3.2)

The RISK experiment here considers a single tournament. In the future, many tournaments should be considered to validate results: this is addressed in Chapter 10: Discussions and Future Work. This tournament is composed of 100 rounds or tournament generations. Each round is defined to have 32 unique players, and except for the first

round, each player is derived from results of the previous round. The first round players are described in Section 7.3.3.1: Player Parameters and Game Phases. The details of choosing and modifying players are described in Section 7.3.3.2: Evolutionary Algorithm. Here is how a tournament rounds work. Each player randomly draws a game with other players, such that each player plays all other players at least once in an eight-player game. This draw gives a range in the number of games played in each round. The number of games played ranged from 36 to 47 games with an average number around 43 games per round. All players play at least 7 games, at most 18 games, and on average 11 games per round. Also due to random draw, each player plays each other player between one and eleven times with an average of 2.4 times.

In the following subsections, the problem and results will be described in five areas. First, the problem is described further by choosing the parameter sets to be trained and game phases (i.e., beginning, middle and end game) with associated linear transition weights. Secondly, the evolutionary algorithm used at the end of each tournament round is described with associated results. Thirdly, each player in every game is constrained in planning time and turns allowed, and the constraints and associated results are shown and described. Then, the learning progression results are shown, analyzed and described. Finally, game-phase dependent strategies are compared to fixed strategies and results are shown.

### 7.3.3.1 Parameters and Game Phase

To train RISK players, there are many choices that can be considered within an ADP&E framework. These variable choices fall in one of three categories: search,

selection, and goals. Search parameters include numbers of search: generations to start turn ($s_{es}$), generations after task failure ($s_{rs}$), generations after task success ($s_{cs}$), selections per generation ($s_{se}$), offspring per generation ($s_{of}$) and recombinations per generation ($s_{re}$). Also, the portion of used resources ($u_R$) restricts search. The selection parameters considered here are the importance of diversity in plan selection ($d$), importance of probability of successful plan's completion ($\pi$), and risk aversion ($r_Y$, $r_Q = r$) in choosing outcomes ($r_Y$) and achieving expectations ($r_Q$). Goal parameters include ten weights, one for each of the ten selected sub-goals shown in Table 7.6. All these parameters combine to make twenty ($s_{es}$, $s_{rs}$, $s_{cs}$, $s_{se}$, $s_{of}$, $s_{re}$, $u_R$, $d$, $\pi$, $r$, $w_1$, $w_2$, $w_3$, $w_4$, $w_5$, $w_6$, $w_7$, $w_8$, $w_9$, $w_{10}$). In addition, a beginning game, a middle game, and an endgame strategy are considered and each of these game phases requires a separate set of parameters. Thus, sixty parameters were used for each player in this experiment with twenty parameters for each game phase. See Table 7.15 below for a brief summary of all sixty parameters.

Table 7.15. Summary of All Parameters Used

| Seven Search / Three Selection Parameters | | | | Ten Goal Parameters | | | |
|---|---|---|---|---|---|---|---|
| Parameter Type | Begin. Game (Phase 1) | Middle Game (Phase 2) | End Game (Phase 3) | Parameter Type | Begin. Game (Phase 1) | Middle Game (Phase 2) | End Game (Phase 3) |
| Start Turn Generations | $s_{es}^1$ | $s_{es}^2$ | $s_{es}^3$ | Ten Sub-Goal Weights | $w_1^1$ | $w_1^2$ | $w_1^3$ |
| Task Failure Generations | $s_{rs}^1$ | $s_{rs}^2$ | $s_{rs}^3$ | | $w_2^1$ | $w_2^2$ | $w_2^3$ |
| Task Success Generations | $s_{cs}^1$ | $s_{cs}^2$ | $s_{cs}^3$ | | $w_3^1$ | $w_3^2$ | $w_3^3$ |
| Selections per Generation | $s_{se}^1$ | $s_{se}^2$ | $s_{se}^3$ | | $w_4^1$ | $w_4^2$ | $w_4^3$ |
| Offspring per Generation | $s_{of}^1$ | $s_{of}^2$ | $s_{of}^3$ | | $w_5^1$ | $w_5^2$ | $w_5^3$ |
| Recombinations per Gen. | $s_{re}^1$ | $s_{re}^2$ | $s_{re}^3$ | | $w_6^1$ | $w_6^2$ | $w_6^3$ |
| Portion of Resources Used | $u_R^1$ | $u_R^2$ | $u_R^3$ | | $w_7^1$ | $w_7^2$ | $w_7^3$ |
| Diversity Factor | $d^1$ | $d^2$ | $d^3$ | | $w_8^1$ | $w_8^2$ | $w_8^3$ |
| Success Probability Factor | $\pi^1$ | $\pi^2$ | $\pi^3$ | | $w_9^1$ | $w_9^2$ | $w_9^3$ |
| Risk Aversion Factor | $r^1$ | $r^2$ | $r^3$ | | $w_{10}^1$ | $w_{10}^2$ | $w_{10}^3$ |

The first players for round one were chosen to be the simplest possible. Table 7.16 shows the beginning game phase search and selection parameters for the first 10 players in round one ($\theta_1$). The search parameters for the numbers of search: generations after task failure ($s_{rs}$), generations after task success ($s_{cs}$), and recombinations per generation ($s_{re}$) were set to zero, because they are useful but not necessary. The search parameters for the numbers of search: generations to start turn ($s_{es}$), selections per generation ($s_{se}$), and offspring per generation ($s_{of}$) were each set to one, because this is the minimum number allowed. This configuration gives a random walk result in plan-generation, where only one plan is generated and the length of the plan is dependent on the portion of used resources ($u_R$). The final search parameter, portion of used resources ($u_R$), as well as all three selection parameters: importance of diversity in plan selection ($d$), importance of probability of successful plan completion ($\pi$), and risk aversion ($r_Y$, $r_Q = r$) are set as random numbers in the range zero to one. Note that these search and selection parameters for all 32 round one players, as well as for both middle and endgame parameters, are chosen in the same way.

Table 7.16. Beginning Game Phase Search and Selection Parameters For First 10 players in Round One

| | Search Parameters | | | | | | | Selection Parameters | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Player No.** | $s_{es}^1$ | $s_{rs}^1$ | $s_{cs}^1$ | $s_{se}^1$ | $s_{of}^1$ | $s_{re}^1$ | $u_R^1$ | $d^1$ | $\pi^1$ | $r^1$ |
| 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0.9501 | 0.4451 | 0.8180 | 0.2844 |
| 2 | 1 | 0 | 0 | 1 | 1 | 0 | 0.2311 | 0.9318 | 0.6602 | 0.4692 |
| 3 | 1 | 0 | 0 | 1 | 1 | 0 | 0.6068 | 0.4660 | 0.3420 | 0.0648 |
| 4 | 1 | 0 | 0 | 1 | 1 | 0 | 0.4860 | 0.4186 | 0.2897 | 0.9883 |
| 5 | 1 | 0 | 0 | 1 | 1 | 0 | 0.8913 | 0.8462 | 0.3412 | 0.5828 |
| 6 | 1 | 0 | 0 | 1 | 1 | 0 | 0.7621 | 0.5252 | 0.5341 | 0.4235 |
| 7 | 1 | 0 | 0 | 1 | 1 | 0 | 0.4565 | 0.2026 | 0.7271 | 0.5155 |
| 8 | 1 | 0 | 0 | 1 | 1 | 0 | 0.0185 | 0.6721 | 0.3093 | 0.3340 |
| 9 | 1 | 0 | 0 | 1 | 1 | 0 | 0.8214 | 0.8381 | 0.8385 | 0.4329 |
| 10 | 1 | 0 | 0 | 1 | 1 | 0 | 0.4447 | 0.0196 | 0.5681 | 0.2259 |

Table 7.17. Beginning Game Phase Reward Parameters
For First 10 players in Round One

| | Goal Parameters | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Player No.** | $w_1^1$ | $w_2^1$ | $w_3^1$ | $w_4^1$ | $w_5^1$ | $w_6^1$ | $w_7^1$ | $w_8^1$ | $w_9^1$ | $w_{10}^1$ |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

The goal parameters for the first players for round one were chosen to be at the extreme corners of the space and orthogonal to each other. Table 7.16 shows the beginning game phase goal parameters for first 10 players in round one. The goal parameters are ten weights that match the ten selected sub-goals shown in Table 7.17, respectively. One of the ten weights is selected to be one, while all other weights are set to zero for each player. Note that these goal parameters for all 32 round one players as well as for both middle and endgame parameters are chosen in the same way.

A natural measure of game phase is the number of players left in the game during plan-generation. Thus, when all eight players are in a game, this is considered the beginning game phase; when five players are remaining, this is considered the middle game phase; and when two players are remaining, this is considered the endgame phase. Also, transitions between these phases are modeled as a linear change in strategy between the two closest phases. Figure 7.20 illustrates the transition through the three phases of a game, where a weight is multiplied by each of the twenty parameters to produce the strategy for the current number of remaining players. Therefore, when six or seven

players are remaining, the strategy combines beginning and middle game strategies, and when three or four players are remaining, the current strategy combines values of middle and endgame. Note that in this process six of the seven search parameters ($s_{es}$, $s_{rs}$, $s_{cs}$, $s_{se}$, $s_{of}$, $s_{re}$) are rounded to the nearest integer to make them valid.

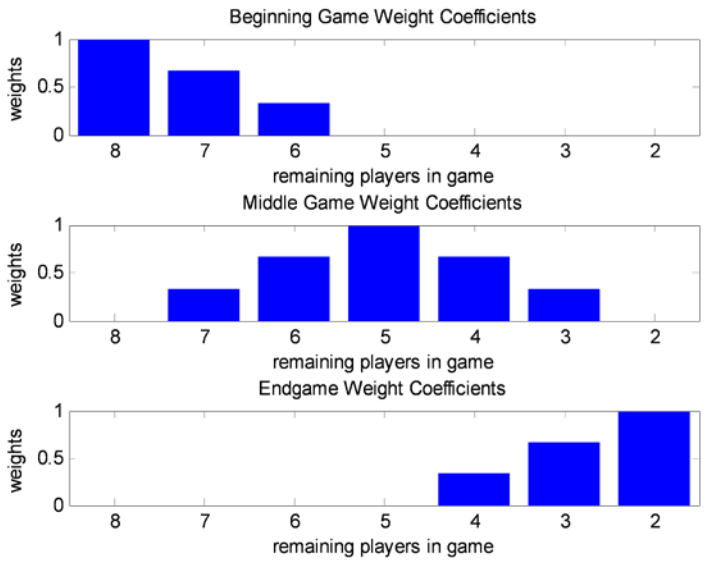


Figure 7.20. Beginning, Middle and End Game Weights for Strategy Transition

### 7.3.3.2 Evolutionary Algorithm

As mentioned above, the evolutionary algorithm used considers 32 players per generational round. The reward for each player is scored based on a winner-take-all approach. Only the winner of each game receives a reward, while all other players in each game receive a punishment. An average player in an eight player game should win one-eighth of their games, thus a point system was used that gives seven points for the winner and minus one point for each loser, totaling zero points for each completed game. For games that cannot complete within the 70-turn limit, all players receive minus one

point. This point system was used to provide near equal opportunity for players who played fewer games and have the same chance for points as other players.

At the end of each tournament round, the next generation of 32 players is composed of five groups. The first group is the top eight players of the previous tournament round based on the highest score. The second group is the top six players with mutated search parameters. These six players have modified search parameters by -1, 0, or +1 with a one-third probability of choosing any of these modifications for each of the 18 parameters. Also, all three projected resource constraint coefficients are changed for each of these six players via a normal distribution with zero mean and standard deviation equal to 0.1. The third group is the top six players with mutated selection parameters. These six players have modified selection parameters for all nine parameters for each of these six players via a normal distribution with zero mean and standard deviation equal to 0.1. The fourth group is the top six players with mutated goal parameters. These six players have modified goal parameters (sub-goal weights) for all 30 parameters for each of these six players via a normal distribution with zero mean and standard deviation equal to 0.1. The final group is the top six players recombined with another randomly selected player. The randomly selected player is chosen from a pool of players with a chance of selection proportional to the score of the player. Recombination takes the average of each parameter pair of both players, and the result is the new parameter value. Thus, eight (group one) plus four times six (groups two to five) gives 32 unique players for the next round.

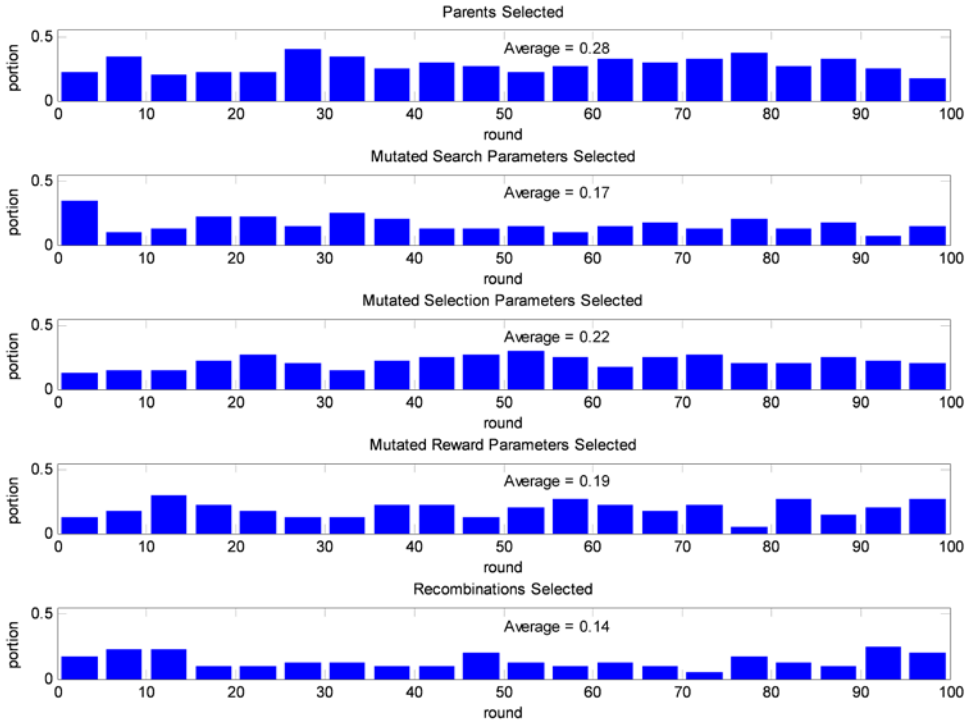


Figure 7.21. Type of Genome Selected for Next Round (Tournament Generation)

The portion of the five groups being selected as a top eight player for the next round is shown in Figure 7.21. This figure shows the average performance of each group over five round increments, and the overall average of each parameter set is labeled in the center of each graph. As expected, the parents tend to have a greater survival rate than the mutated and recombined players (i.e., offspring). However, on average, 72% of the new players are different from the previous round. Note that mutation of selection parameters shows the next greatest utility overall and recombination shows the worst performance. As expected, the first five rounds show the greatest variation in search parameters, as players are learning to take more game time in planning their turns.

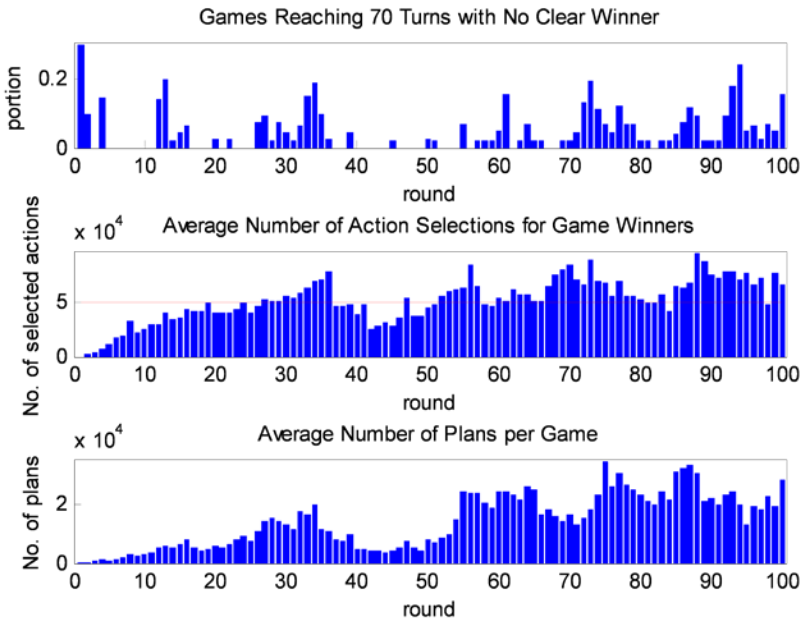


Figure 7.22. The Impact of Limiting Action Selections per Game

### 7.3.3.3 Player Constraints

Due to the nature of the RISK game, two constraints were enforced on each player. First, since a RISK game can conceivably go on indefinitely if all eight players are poor quality, the number of turns was restricted to 70. This first constraint of 70 turns was chosen, because it is far beyond the number of turns necessary to win the game RISK with competent players. Experiments have shown that very competent players usually complete a game within 25 to 30 turns, given the enormous number of forces accumulated at that point in the game [13] [14]. For games that exceed 70 turns, the game is stopped with no winner, and all players receive minus one point for participating. Secondly, a turn in RISK takes search as in a chess game, thus as in chess, a player is restricted to the extent of search allowable [15]. To avoid computational restrictions based on processor speed, the number of contemplated actions over the course of a game

was restricted to 50,000. This second constraint of 50,000 was chosen arbitrarily to restrict the entire length of a game to be less than 10 minutes computer time on a 2.4 GHz Intel Core 2 Duo Apple computer with 4 GB 667 MHz DDR2 SDRAM. This time limit only provides a few seconds for a player to generate a plan per turn and is far too little time for a quality player to evolve, but sufficient to get interesting experimental results. Once a player exceeds 50,000 contemplated actions in a game, parameters are set to a random walk mode (i.e., $s_{es} = 1$, $s_{rs} = 0$, $s_{cs} = 0$, $s_{se} = 1$, $s_{of} = 1$, and $s_{re} = 0$), where a player has minimal search capability as used in the first round.

In limiting the number of turns per game and contemplated actions per player, the search parameters are forced to compensate for these restrictions. Figure 7.22 illustrates three important consequences of restricting both turns in a game and contemplated actions per player. The first graph in Figure 7.22 shows the proportion of games per generation that failed to complete within 70 turns. In the first few rounds (one to five), the main reason the games did not complete is due to the players' incompetence. For all remaining rounds, the intermittent game completion failures are due to players exceeding the 50,000 contemplated actions. The average contemplated actions for the game winners are shown in the second graph in Figure 7.22. This ceiling, shown as the red line in the second graph, changes competent players into incompetent ones due to instantiating the random walk search. Note that more games exceed 70 turns (graph one) when game winners exceed 50,000 contemplated actions (graph two). Graph three shows a different trend. Graph three shows that the average number of plans does not have to be directly related the average number of generated actions. In the first forty generations, the trends look similar, but after generation forty, the approach adjusts the parameters such that

fewer actions can give shorter plans. Thus, parameters are changed to give fewer actions per plan. The results in the next subsection give a reason for this trend.

### 7.3.3.4 Learning Progression

Learning progression can be measured in several ways. Here, we examine learning progression in three major areas compared across all rounds: (1) the average parameter sets; (2) the differences in parameter sets; and (3) player performance.

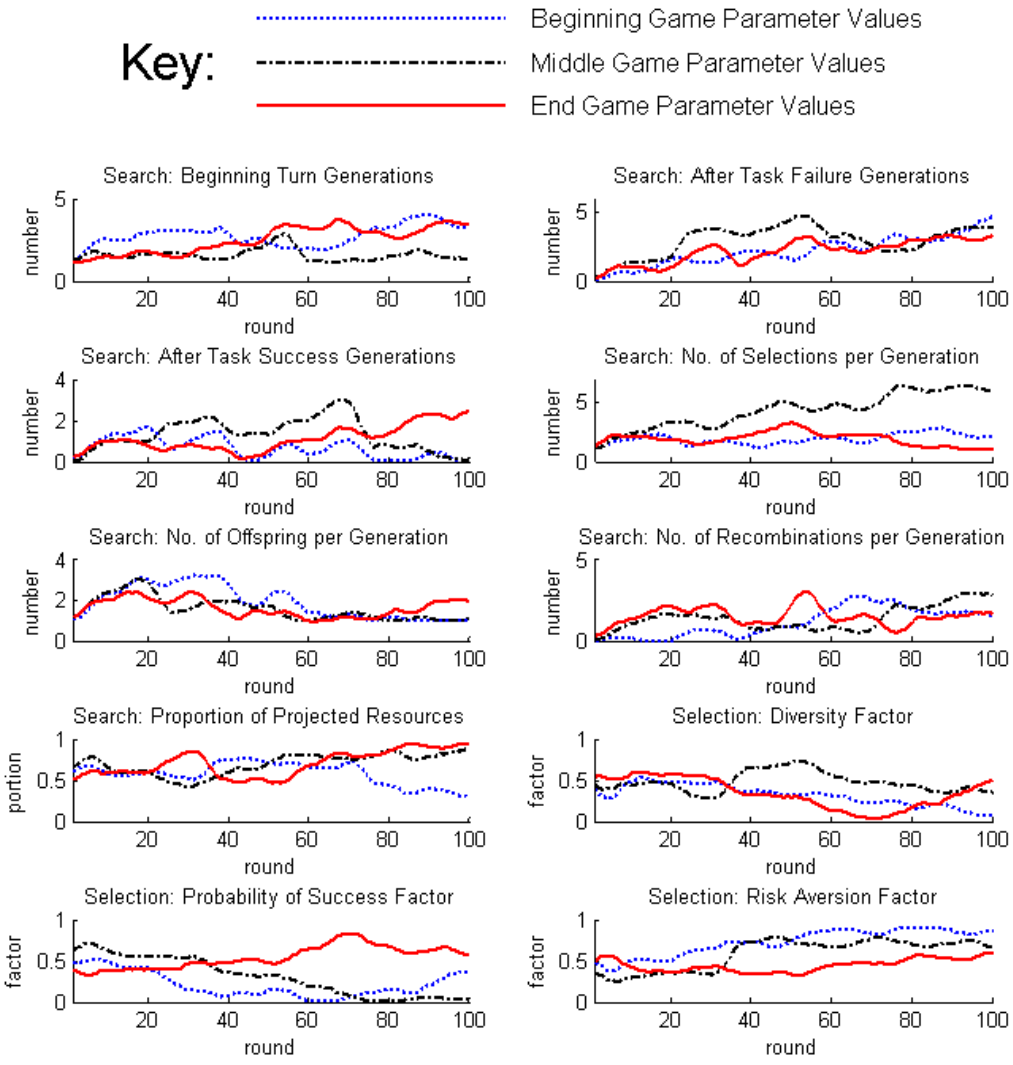


Figure 7.23. Progression of Search and Selection Parameters

The average parameter sets for each round are shown in Figures 7.23 and 7.24. The blue dashed-lines indicate the beginning-game parameters, the black dot-dashed lines indicate middle-game parameters, and the red solid-lines indicate endgame parameters. Over the hundred generations, it is clear to see that the search and selection parameters shown in Figure 7.23 have not converged at all. This is a strong indication that the 50,000 contemplated actions limitation is too small and that a hard change in search parameters to random walk make convergence of search and selection parameters difficult.

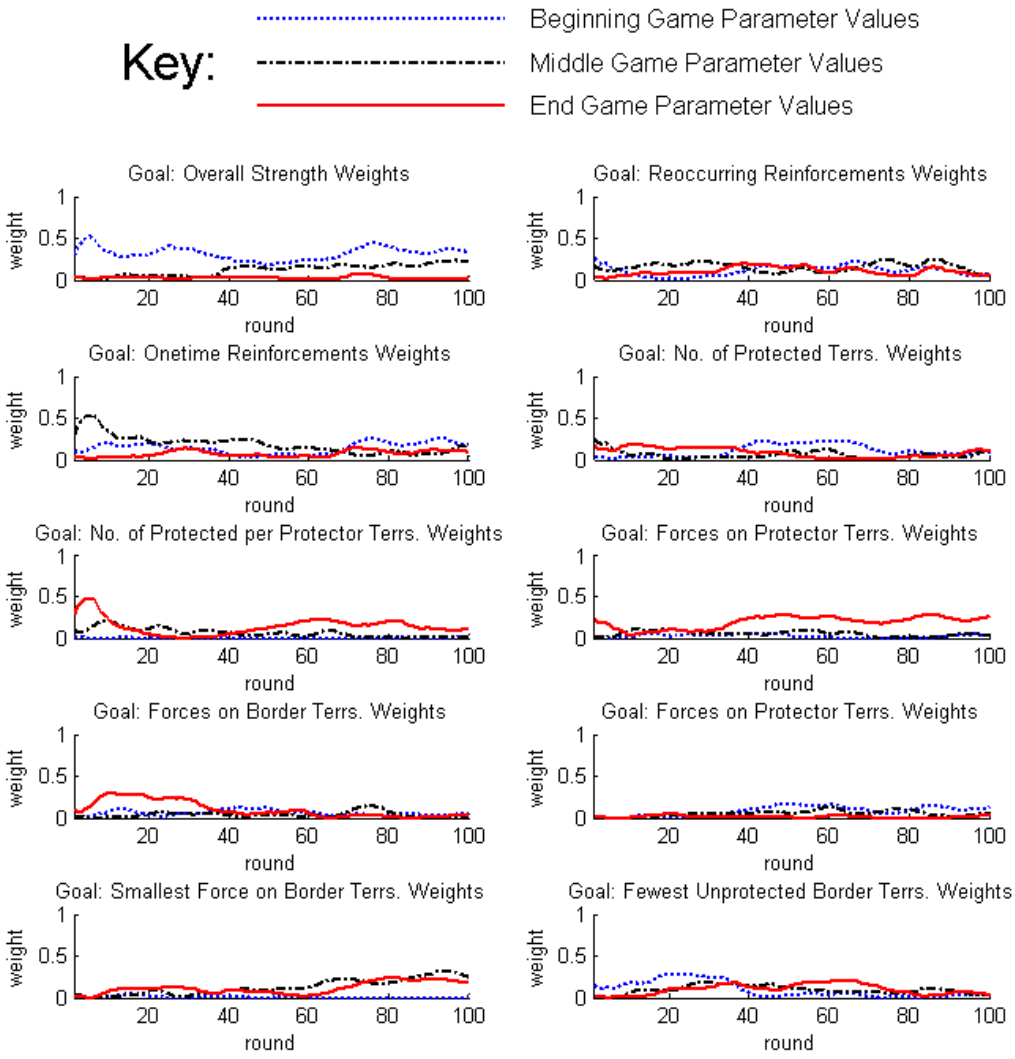


Figure 7.24. Progression of Reward Parameters

On the other hand, the goal parameters shown in Figure 7.24 are still moving from round to round, but their changes are fewer, and many parameters show some consistency, such as overall strength, number of protected vs. protector territories, and forces on protector territories. Protected territories are territories that do not border any other player's territories, and protector territories border another player's territory. As shown in the difference in red, black and green lines, overall strength matters less and less as games progress. Contrarily, the number of protected per protector territories, and forces on protector territories are only important near the endgame and not important otherwise.

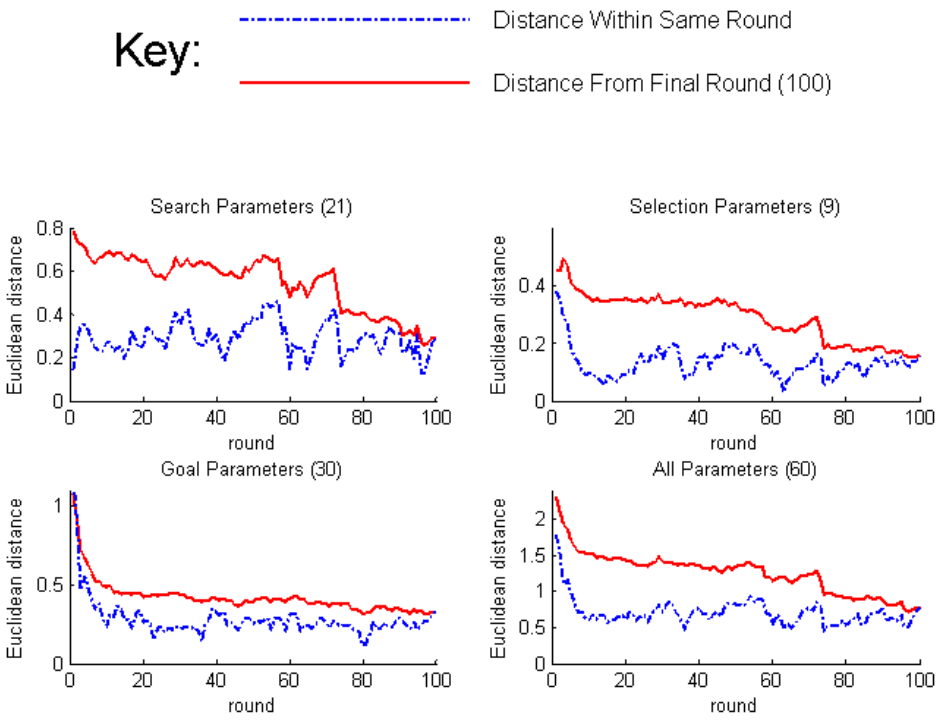


Figure 7.25. Change in Parameters Based on Euclidean Distance

The differences in parameters sets measured in normalized Euclidean distance are shown in Figure 7.25. A normalized Euclidean distance was calculated based on the

range of parameters. Most parameters ranged from zero to one and did not require normalizing. However, six of the seven search parameters ($s_{es}$, $s_{rs}$, $s_{cs}$, $s_{se}$, $s_{of}$, $s_{re}$) were normalized by subtracting their corresponding allowed minimal value (1, 0, 0, 1, 1, 0) and dividing by their corresponding range (based on 50,000 action gaming constraint). For instance, the selections per generation parameter ($s_{se}$) ranges from 1 to 7, thus the values are reduced by one, then divided by six. After normalization, the Euclidean distance for each parameter was determined in the following way. First, add the squared distance of every pair of parameters within the group of the top eight selected players. Then, take the square root of this sum and divide by the number of pairs (8 x 7 = 56).

The four graphs shown in Figure 7.25 divide the distances into three groups: search, selection, and goal parameters, and the fourth graph shows the composite result. The blue dot-dashed lines in Figure 7.25 shows the distance among players within the same round, and the red solid-lines in Figure 7.25 shows the distance between that round and the last round (note that at generation 100, both red and blue lines meet). Thus, the blue dot-dashed line shows how different the players are in each game round, and the red solid-line indicates how much drift the parameters are undergoing across rounds.

All four graphs in Figure 7.25 indicate that the blue dot-dashed lines are sufficiently noisy to show differences in players in a single round, indicating that the players are not converging to a single point. Also, the red solid-lines in all four graphs indicate that the player parameters are still moving from the original location, and thus not oscillating in the space. Also, the top-left graph in Figure 7.25 shows that the search parameters are not converging at all, and is a strong indication that the limiting of 50,000

contemplated actions is not working well. The top-right graph in Figure 7.25 shows that the selection parameters are also not converging well, and is probably influenced greatly by the same constraints. The red line in the lower-left graph in Figure 7.25 indicates that the goal parameters are converging reasonably well due to the decreased slope of the red line. In the lower-right graph in Figure 7.25, the overall convergence of the parameters is shown to be far from complete, thus another approach that avoids hard constraints needs to be studied further. Future considerations are described in Chapter 10: Discussions and Future Work.

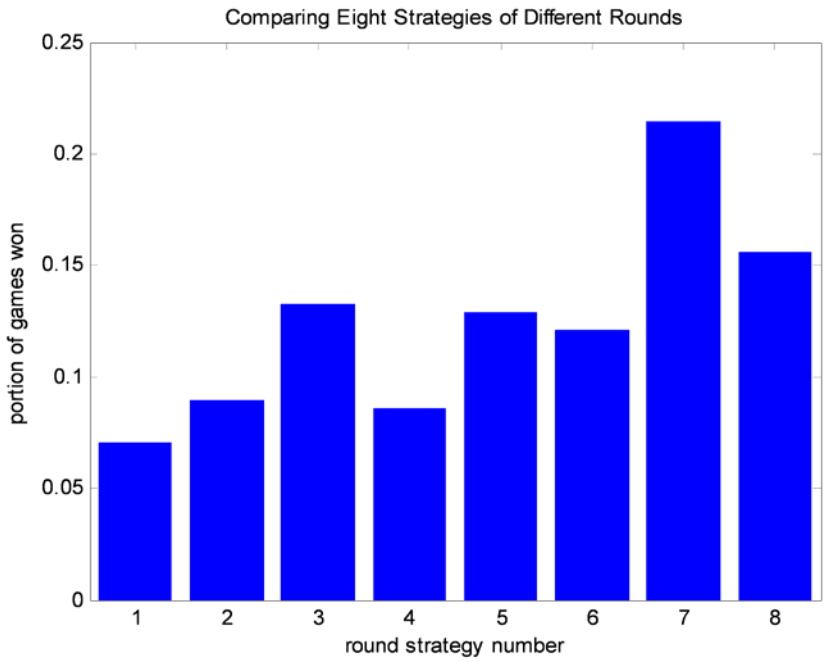


Figure 7.26. Progression of Player's Winning Strategy

Player performance across all the rounds is shown in Figure 7.26. Since there can only be eight players in a game and there were 100 rounds of training, the eight players were selected from the best performers across sets of rounds. The first four rounds were not used, but all subsequent rounds were used in the following way. The best players for

each round in the bands 5 to 16, 17 to 28, 29 to 40, 41 to 52, 53 to 64, 65 to 76, 77 to 88, and 89 to 100 were used. Player one used the average parameter values for the first band (5 to 16), the second player used the average parameter values for the second band (17 to 28), and so on for all eight players. These players played each other 256 times in random order and with a random player selected to start each game. The general trend shows an increase in performance, where the seventh player performed best overall. Note the decrease in performance in player four compared to player three due to the decrease in action selections allowed as shown in Figure 7.22 in generations 41 to 52.

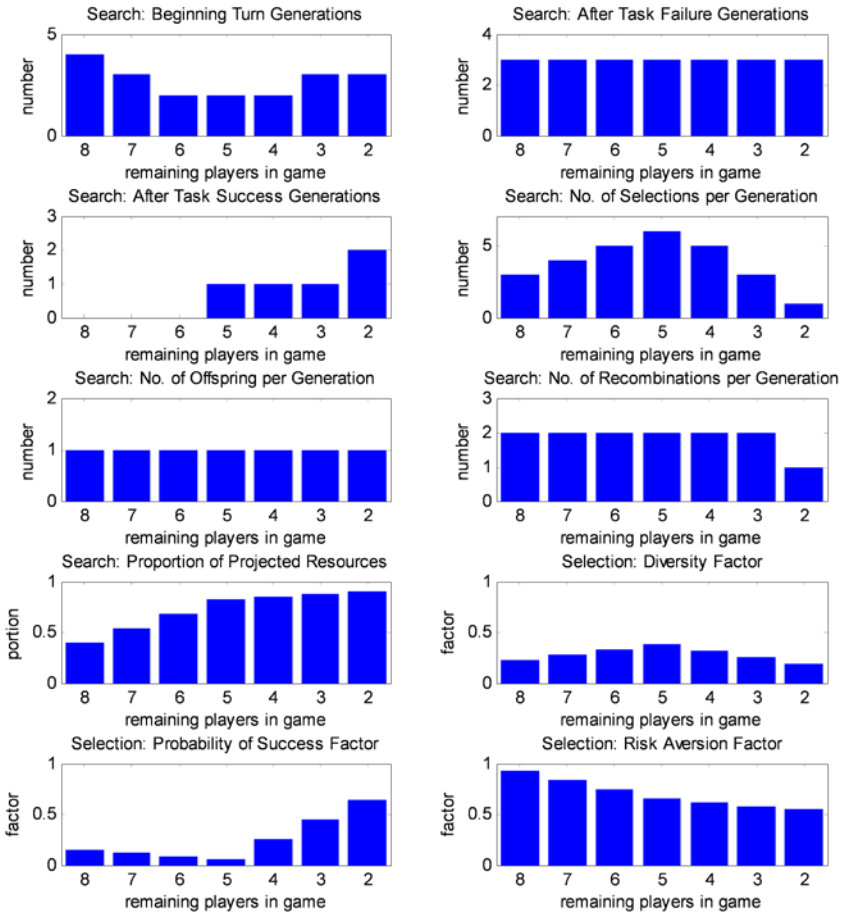


Figure 7.27. One Top Strategy's Search and Selection Parameter Values as a Function of Remaining Players in Game

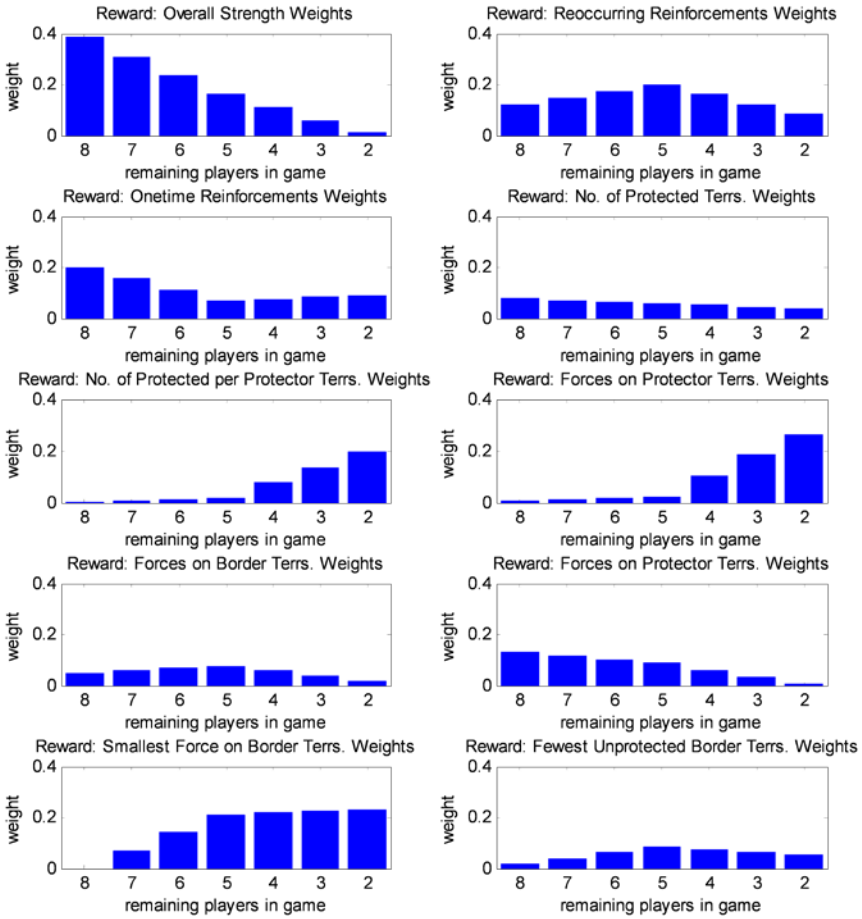


Figure 7.28. One Top Strategy's Reward Parameter Values as a Function of Remaining Players in Game

### 7.3.3.5 Comparing Game-Phase Dependent Verses Fixed Strategies

Mixed strategies are often used for variability in choosing actions over the course of a game [16] [17]. In the case of RISK, a game-phase dependent strategy is used, and refers to having different parameter values over the course of a game. Given that the players on average performed best in generations 77 to 88, as defined by player seven above, we have illustrated what the parameters of this player are over the course of a game. Figures 7.27 and 7.28 show all player parameters and how they change in

accordance with game phase (defined as players left in game). As shown in the figures, only a few parameters remain constant over the course of a game.

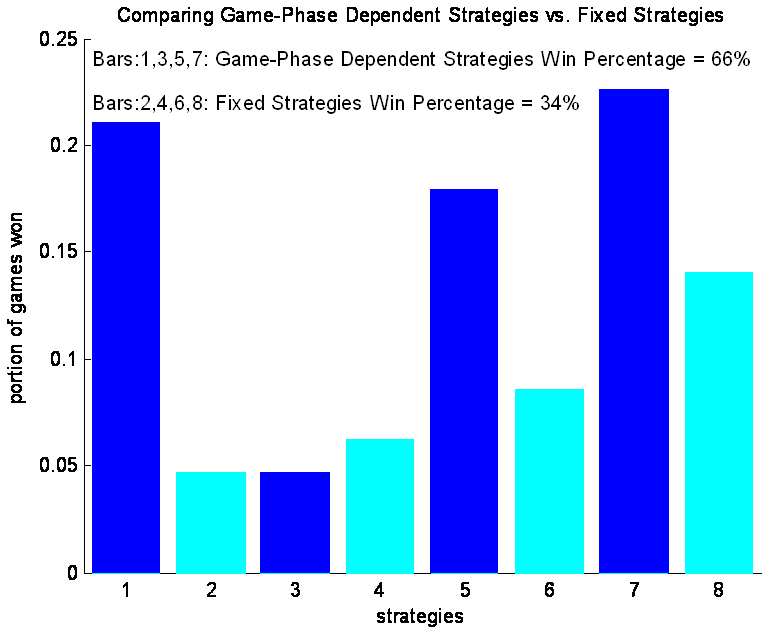


Figure 7.29. Comparing Game-Phase Dependent Vs. Fixed Strategies

To compare the benefit of having a beginning, middle and endgame strategy in comparison to having a single strategy throughout the game, a tournament of 128 games was run. Players 3, 5, 6, and 7 were used from the previous experiment and assigned as players 1, 3, 5, and 7, respectively. In addition, four other players were added, having a single strategy throughout the game. These players used average parameters of the players 3, 5, 6, and 7 described above. Thus, parameters for the beginning, middle and endgame were averaged and assigned to all those parameters. These additional four players were assigned as players 2, 4, 6, and 8, respectively. Figure 7.29 shows the results of the 128 game simulations. The blue bars indicate the players with game-phase dependent strategies and the cyan bars indicate the players with fixed strategies. As

expected, player seven did the best, but player one did surprisingly well. In all cases, the game-phase dependent strategies did far better than the fixed strategies, and overall the game-phase dependent strategies won two-thirds of all games.

The text in Chapter 7 contains material previously presented in three publications: Lecture Notes in Computer Science: Applications of Evolutionary Computation, IEEE Transactions on Evolutionary Computation, and IEEE Congress on Evolutionary Computation. I was the primary researcher and author, and the coauthor directed, and/or supervised in the research, which forms the basis of the chapter.

**References:**


1. D. Shapiro. "Risk: The Evolution of a Game". *The Games Journal*. December 2002.

2. S. Blatt. "RISKy business: An in-depth look at the game RISK". *Undergraduate Math Journal* **3** (2), 2002.

3. B. Tan. "Markov chains and the RISK board game". *Mathematics Magazine* **70** (5): December 1997 pp. 349–357.

4. J. A. Osborne "Markov Chains for the RISK Board Game Revisited," *Mathematics Magazine* **76** (2): April 2003, pp. 129–135.

5. C. E. Clark. “The PERT Model for the Distribution of an Activity Time,” Operations Research 10(3), 1962, pp. 405-406.

6. N. L. Johnson,, S. Kotz, and N. Balakrishnan. *Continuous Multivariate Distributions: Models and Applications*, John Wiley & Sons, 2000.

7. Z. Ghahramani. “Learning Dynamic Bayesian Networks,” Lecture Notes In Computer Science, Vol. 1387, 1997 pp. 168-197.

8. J. Vaccaro, C. Guest, “Evolutionary Bayesian Network Dynamic Planner for Game RISK,” *Lecture Notes in Computer Science: Applications of Evolutionary Computation, EvoSTOC Proceedings*, Coimbra, Portugal, April, 2004, pp. 549-560.

9. J. O. Berger. *Statistical Decision Theory and Bayesian Analysis*. Springer Series in Statistics (Second ed.). Springer-Verlag, 1985.

10. A. Hald. "On the history of maximum likelihood in relation to inverse probability and least squares". *Statistical Science* **14** (2): 1999. pp. 214–222.

11. O. Kallenberg. *Foundations of Modern Probability*, 2nd ed. Springer Series in Statistics. 2002.

12. T. Kohonen, *Self-Organizing Maps*, Springer-Verlag Berlin Heidelberg, 1995, pp. 1-50.

13. J. Vaccaro, C. Guest, "Learning Multiple Search, Utility and Goal Parameters for the Game RISK," *IEEE Congress on Evolutionary Computation*, Vancouver, Canada, July 2006, pp. 4351-4358.

14. J. Vaccaro, C. Guest, "Planning an Endgame Move Set for the Game RISK: A Comparison of Search Algorithms," *IEEE Transactions on Evolutionary Computation, Vol. 9, No. 6*, December 2005, pp. 641-652.

15. R. Levinson, F. H. Hsu, J. Schaeffe, T. A. Marsland, & D. E. Wilkins, "The role of chess in artificial-intelligence research," *ICCA Journal*, V. 14, N. 3, 1991, pp. 153-161.

16. A. Rubinstein. "Comments on the interpretation of Game Theory", *Econometrica*, July, 1991 (Vol. 59, n°4)

17. J. Haranyi. "Games with randomly disturbed payoffs: a new rationale for mixed-strategy equilibrium points", *Int. J. Game Theory* **2**: 1973. pp. 1–23.

# Chapter 8

# Urban Search and Rescue Operation Application

## 8.0 Abstract

The problem of interest here is simulation of an urban search and rescue operation (US&RO). This chapter will describe the mission simulation, provide implementation details, and demonstrate results. The simulation's architecture, constraints, variables, and parameters will be described in terms of the ADP&E goal-directed hierarchy introduced in Chapter 6. The implementation details are described in the nine-step design, build, and test (DB&T) strategy. Results are presented in three areas of value: plan fitness, planning cycle assessment, and planner utility. The hierarchy, DB&T, and value function definitions and details are described in Chapter 6.

## 8.1 General Hierarchical Description

As described in Chapter 6, the goal-directed hierarchy is composed of ten levels: (1) tournaments ($\Omega$), (2) rounds or competitions ($\Theta$), (3) games or missions ($G$), (4) players or planners ($\Psi$), (5) agents ($\Gamma$), (6) plans ($P$), (7) tasks ($T$), (8) actions ($A$), (9) outcomes ($Y$), and (10) observations ($O$). Just as the case of the RISK application described in Chapter 7, tournaments and rounds are only implemented after the US&RO application was fully built and tested for a single simulation. Thus, the order of the US&RO description is levels 3 through 10 first, followed by level 2, and finally level 1.

### 8.1.1 Missions or Games (*G*)

Planning, executing and assessing an US&RO is best understood by describing the implemented example. The extent of the planning space is illustrated in Figure 8.1. The problem is the evacuation of a small city, approximately 4 km x 5 km. Within these 20 square km, there are 20,000 stranded people, twenty agents of various types (boats, busses and helicopters) available, an unknown water level, randomly placed roadblocks, bases to drop off people, and building locations where people require rescue. More specifically, there are 3649 buildings having over 12,000 floor locations, 2135 bus stops, over 900 boat stops, survey and rescue helicopters that can stop at any building, four upstream boat bases, and a bus and helicopter base.

Note that unlike RISK, the US&RO mission simulation is set up as cooperative with only one planner and no disruptive agents, such as looters or scavengers. Thus, there is only one player of this game (i.e., mission), and it is the planner, executor, and assessor all in one. Note also that the player directs many agents that have some degree of autonomy. The objective is simple and measurable: to get as many people as possible out alive. The model environment is partially observable. Many elements are unknown a priori, such as the water level, where people are, where injured are, where unsupplied people are, where roads are blocked, and what locations are traversable by water, road or only by air.

The mission is made up of two major component models: (1) terrain interconnections, and (2) stranded people. The terrain is of an urban city model supplied by information within a Compact Terrain Data Base (CTDB). An approach was designed

and implemented that automatically generates a terrain interconnection model from a CTDB, described in Section 8.1.1.1. The stranded people are placed as described in Subsection 8.1.1.2.

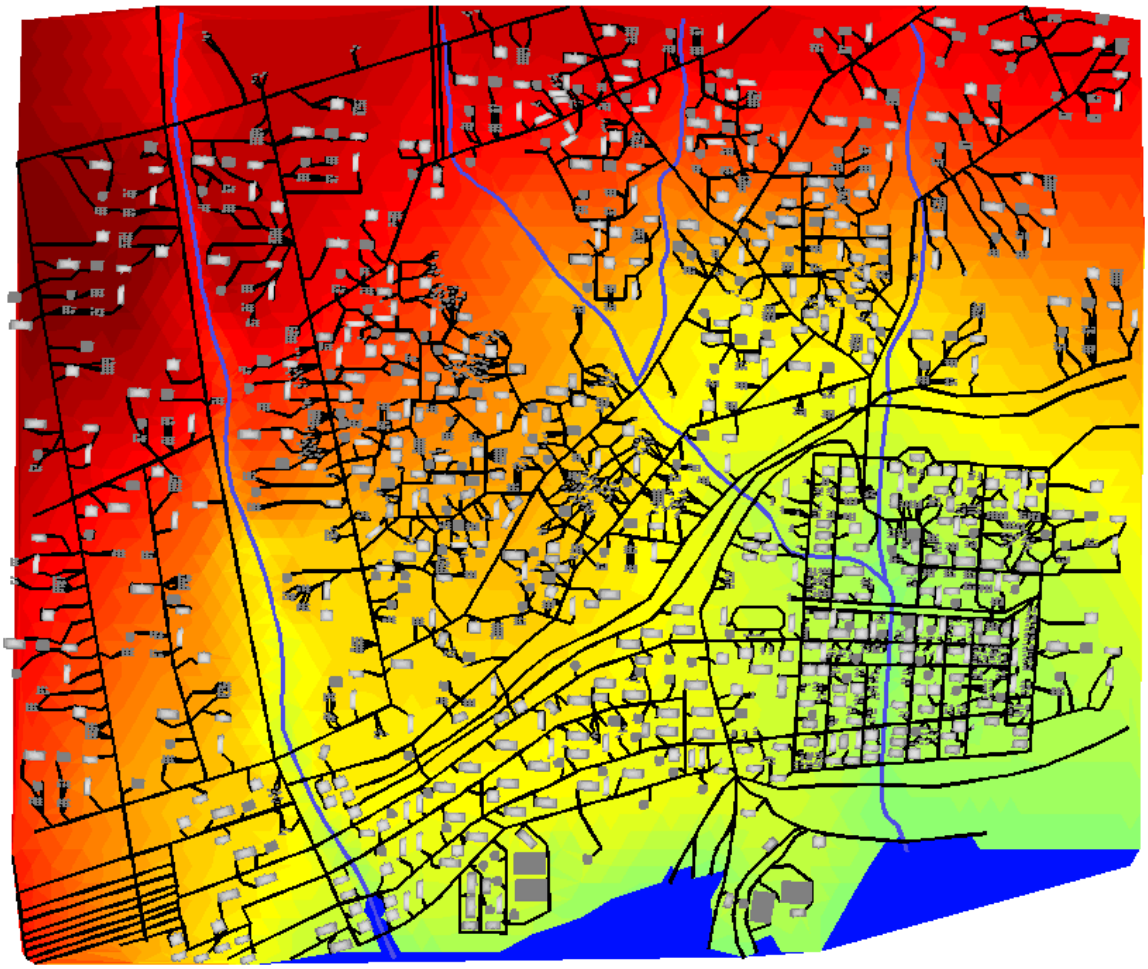

Figure 8.1. City Under Normal (No Flood) Conditions

### 8.1.1.1 Terrain Model

CTDB represents three area types in urban terrain: natural terrain, developed terrain and Multi-Elevation Structures (MES) [1]. Natural terrain can be soil (3D triangles), water (2D polygons), trees (2D points), and canopies (2D polygons). Developed terrain includes roads, railroads, and canals, which are all defined as 2D linear

shapes. MES data includes buildings, monuments, and walls, which are typically defined as 2D polygons with height. More specific descriptions related to this US&RO application are given later in this section.

For all CTDB data, urban terrain is gridded into square blocks (e.g., 256x256 meter squares), which divides many two-dimensional, and three-dimensional shapes into separate parts. This presents a problem for reconstructing the data into gaming objects, such as keeping buildings together, and roads, rivers and lakes properly connected. Since our goal is not for human use, but for computer simulations and gaming, the visual aspects are less important than the object information. Because of these grid blocks, creating objects requires fusing fragmented shapes. Drawing and filling these shapes fuses them to create objects. After objects are created, they form a basis for determining waypoints and connectivity.

The problem domain of interest here is parsing the CTDB into a discrete and compact format for gaming individual actions, while retaining the variability and continuity aspects of the terrain data. The variability is retained through creating waypoints at strategic locations, and continuity is retained through connecting the waypoints in a manner that reflects their relationship. Having only waypoints and connectivity does not restrict choices for the planning options of computer algorithms.

A compact and computable representation is formulated through recomposing the CTDB into a network form, using waypoints and connections. The current CTDB form is represented as a virtual 3D world, which human players can navigate through with high visual acuity. For computer simulation purposes, there is no need to have a full

continuous and uniform representation; a tractable abstract representation is sufficient. Waypoints and connections provide a sufficient representation. Though the concept is simple, the real challenge is developing a useful model that takes into account many different possible features. For the example US&RO application, we are only concerned with these features: roads, rivers, buildings, open water and land areas, and terrain elevation.

Roads and rivers are relatively simple to recompose, because they are given in CTDB as 2D linear segments with a corresponding width. Given the linear representation, every curve requires many segments, while straight runs may contain only one segment. Having both ends of every segment become waypoints shared with all connecting segments creates waypoints and connections. Connections are simply the segments, waypoints are points where the direction deviates or an intersection exists. This approach works in most cases, but it is important to make sure that all segment intersections are included, because many times they are not represented explicitly. In such cases an additional waypoint is inserted at each intersection automatically based on proximity of crossing paths.

Rivers are a special case because they can be traveled by boat, but their banks can also be traveled by boat when they are flooded. Riverbank waypoints are generated in most cases by simply creating waypoints paralleling the river at a distance of half its width on both sides of the river. However, this does not work in the case of river intersections. These intersections are completed through adding the closest point between intersecting river segment that is beyond both rivers' widths.

Waypoints and connections for buildings are implemented in three areas: interior, nearby exterior, and exiting the building. Given that many buildings are not single CTDB objects, but a combination of 3D volumes, merging is required. This is done using a technique of projecting all overlapping or connected volumes to the x-y plane (the ground) and then filling all internal pixels to determine the total footprint of the building object. Figure 8.2 illustrates this process.

To start, a building 3D volume is chosen from a building volumes list (all building volumes), and all buildings sharing vertices with the chosen building volume are selected. All those selected are eliminated from the list of unprocessed building volumes. Choosing building volumes is iterated until there are no more shared vertices. After all volumes are selected, the largest distance among all pairs of vertices of the selected building volumes is computed, and a square image canvas with edge length twice that distance is generated. The edge-length is twice to avoid filling the image on the exterior side, which would invert the building image. The canvas grid is initially filled with all zeros and the building coordinates are normalized to ensure the building points can be placed in the middle of the canvas. In other words, the mass center of the volume is normalized to the middle of the canvas. The offsets are stored for recomputing the building edges once the building volumes are merged into a single building. Next, each building volume is drawn and filled on the canvas with value one. Next, all interior building vertices are filtered out through keeping only those volume vertices that intersect an outer pixel of value zero. Figure 8.2 shows the image of building volumes, where the single interior vertex is labeled as a ‘triangle’ and the exterior vertices are labeled as ‘stars’ and ‘squares’. All ‘triangle’ coordinates are assumed as interior points and

removed. In Figure 8.2, the gray vertical and horizontal lines are gridlines passing through the building, giving four separate building volumes that must be merged to form a single building.

Remaining vertices are now reconnected such that they follow the outer wall. This is accomplished through cycling all connecting vertex pairs from all volumes, and keeping those pairs that have both exterior vertices. For illustrative purposes, Figure 8.2 shows all the vertex pairs by placing a ‘diamond’ on either side of the center connections. Those walls with a diamond outside of the wall are kept. The heavier shaded walls in Figure 8.2 represent the kept walls.

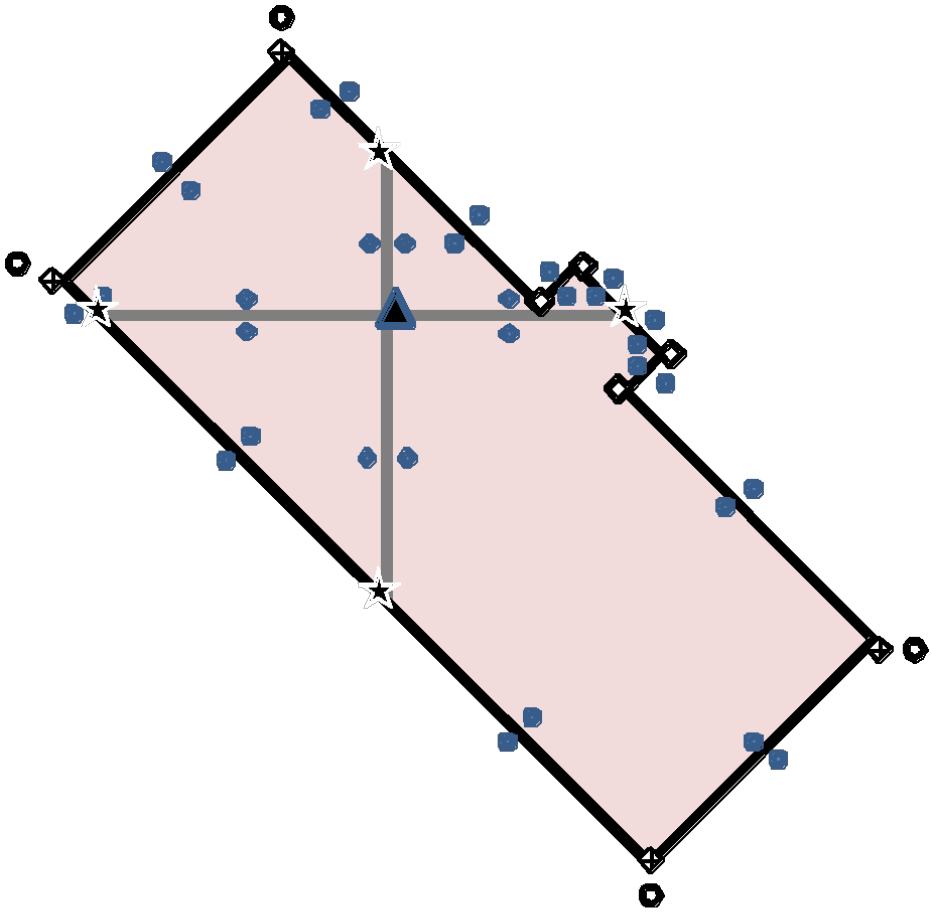

Figure 8.2. Merging Building Volumes into One Object

There are still some vertices that are in the middle of walls that can be eliminated without changing the outline of the building. Now that we have a complete perimeter of the building, the ‘middle of wall’ vertices can be eliminated through taking out all vertices that have the same projected angle from both sides. ‘Middle of wall’ vertices are represented as ‘stars’ in Figure 8.2 and are eliminated.

We now have a building object with at least three connecting vertices, but often times more (i.e., Figure 8.2 ‘square’ symbol represents a sufficient set of perimeter vertices). This process is repeated from the first step until all building volumes within the CTDB are processed.

For simplicity, there are only four exterior waypoints near each building, unless only three vertices define the entire building object. These near waypoints are chosen starting from the most outer vertices of the building by projecting them an additional two meters away from the building center. The perimeter vertices described above are used to determine these waypoints.

First, the most outer vertices (corners) are chosen as follows: the first corner is the farthest (i.e., distance) from the center, the second corner is farthest from the first corner selected, the third corner is farthest from the first and second corners combined distance and the fourth corner is farthest from the combined first, second and third corners. In Figure 8.2, the chosen corners are those with an ‘x’ in the ‘square’. The nearby exterior building waypoints generated from these for extreme corners are shown as ‘circles’ in Figure 8.2. These waypoints are connected in such a way that they span the minimal distance, and thus, do not cross paths and form a close approximation to a path that

follows the circumference of the building. Some buildings are very near one another, thus near building waypoints from multiple buildings are clustered if they are less than three meters apart. In other words, these nearby waypoints from multiple buildings are combined as one, while retaining their previous connectivity.

In a CTDB, buildings often come with floor plans, but for this US&RO application, we assume only two interior waypoints per floor, and two on the rooftop. One point represents an observable location, while the other represents a hidden location. All pairs of waypoints on each floor are connected. Connection between floors, exiting the building, and going to the roof are only accessed through the hidden waypoints. Both hidden and observable waypoints are placed near the center of each floor.

Building exit connections provide access to the nearest road (or equivalent water waypoint) for US&RO evacuation purposes. These connections are found through taking each building corner waypoint and determining the road nearest to each, then selecting the shortest of these lengths to be the building sidewalk access. Buildings that share the same road waypoint are clustered together and considered as a single bus or boat stop. The stops are used in planning evacuations for each clustered set of buildings.

Open land and water area waypoints are necessary for regions where there are no roads or buildings. For simplicity, a hexagonal grid (75 meters edge length) of waypoints is projected on the 2D image of roads, water bodies, and buildings to provide a complimentary set of waypoints. All waypoints projected to be within fifteen meters of another existing waypoint, on a road, in a water body, or within a building are rejected. All other projected waypoints are kept as open area waypoints.

Connections to complete the model are implemented as follows. All waypoints, excluding interior building waypoints are connected using Delaunay triangles [2]. Next, all Delaunay connections are projected onto the same 2D image used above. All connections that cross over roads, rivers or buildings are rejected and the remaining connections are kept. All connections discussed earlier (roads, rivers, buildings, etc.) are added to this connection model, and redundant connections are removed. This completes the surface model connectivity.

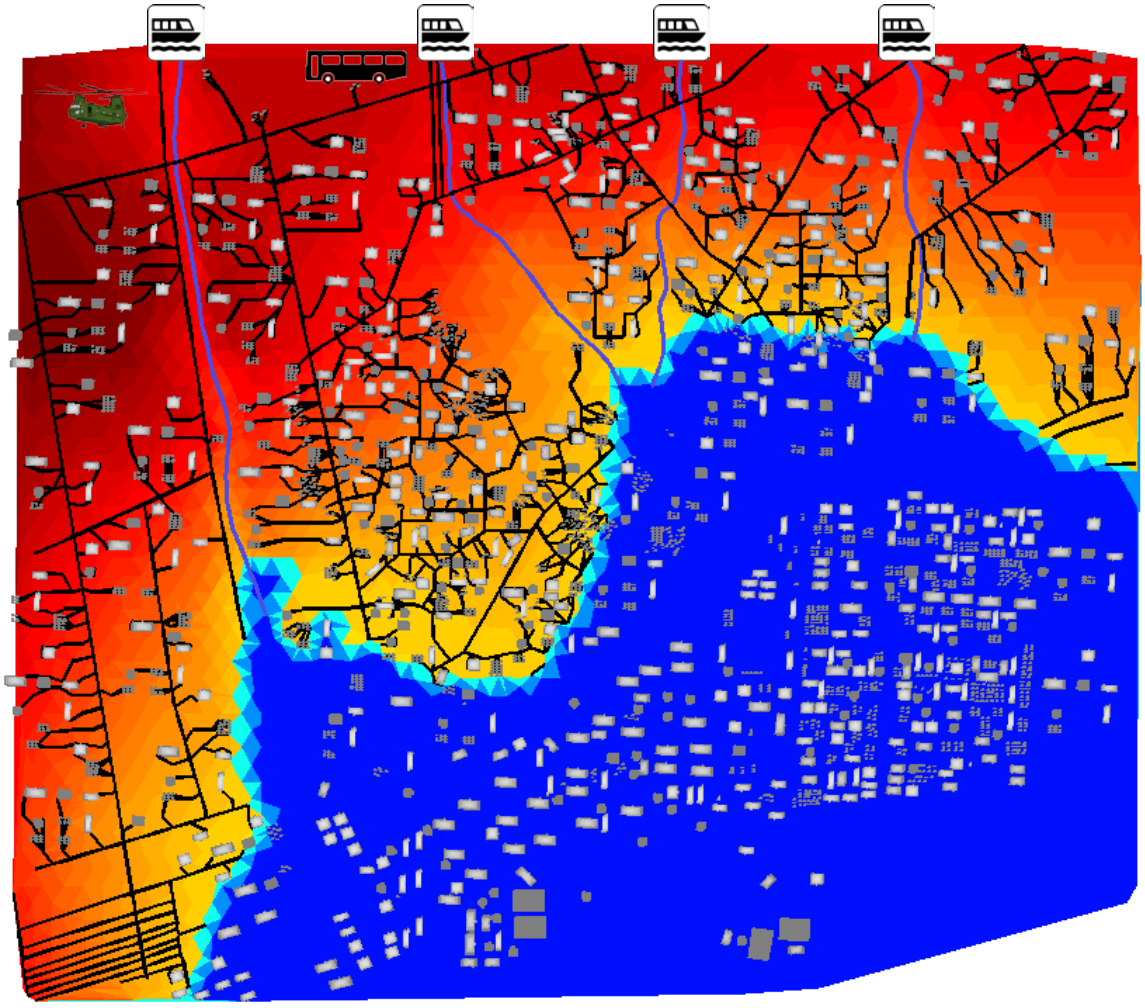

Figure 8.3. City Map After a Hurricane or Tsunami Disaster

Terrain elevation is also supplied within the CTDB and required in the model, because water level determines which surface waypoints can be traveled by land or by water. Figure 8.3 illustrates the city map with the highest flood level, and the one examined in Section 8.3: US&RO Results. In the example model, boats, busses and helicopters are used. Boats can traverse all waypoints below the waterline and busses can travel only road connections above the waterline. Connections to and from buildings adapt to the nearest floor not underwater.

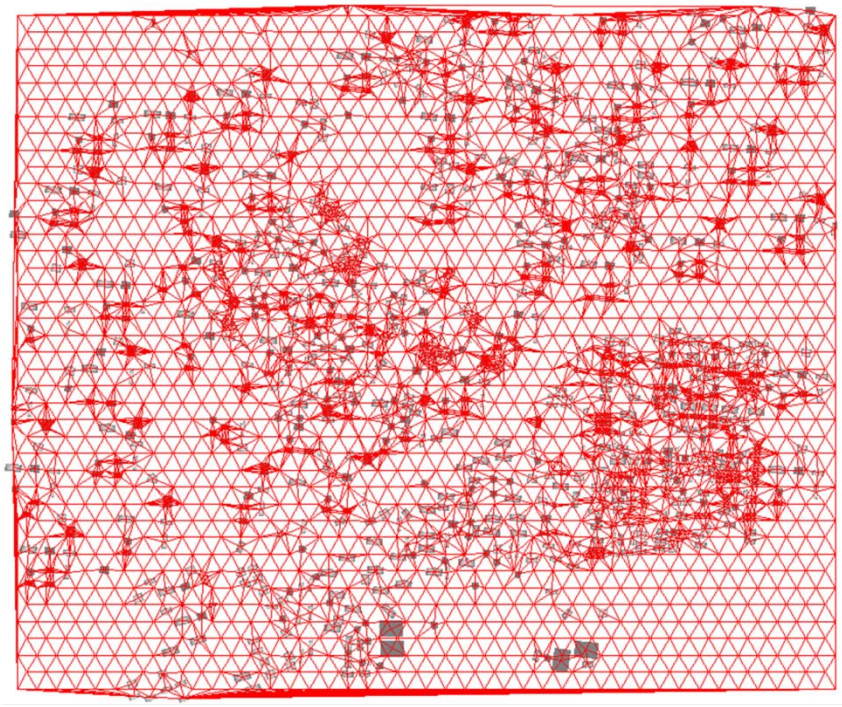

Figure 8.4. City Helicopter Routes Under Any Conditions

Helicopter paths require an air movement model. For simplicity, a sky waypoint is placed above each building at a constant elevation slightly above the tallest building height. Connections are made between each sky waypoint and to the building rooftop observable waypoint below. In addition, a hexagonal grid (100 meters edge length) of sky waypoints is added if the waypoint is not within fifteen meters of an existing sky

waypoint. All sky waypoints are connected with one another via Delaunay triangles, but this time all connections are kept. A helicopter base is added at the most strategic (i.e., as deemed by an expert) open area waypoint and connected to the nearest sky waypoint. Figure 8.4 illustrates the sky waypoint connections overlaid on the city map. The helicopter base is in the upper-left corner as shown in Figure 8.3.

Figure 8.5a. City Water Routes
Under Normal Conditions

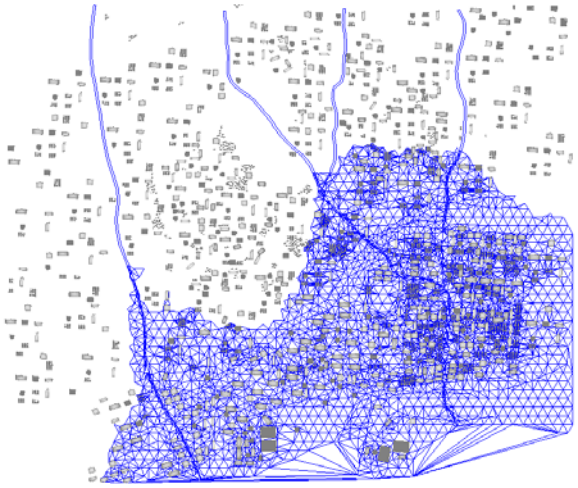

Figure 8.5b. City Water Routes
Under Maximum Flood Conditions

Figure 8.6a. City Road Routes
Under Normal Conditions

Figure 8.6b. City Road Routes
Under Maximum Flood Conditions

Waypoint type is used to segment the simulation into a discrete choice model for multiple agents. Thus, agents of specific types are restricted to a particular set of waypoints, and paths. Further, agent choices are economized by clustering waypoints before preprocessing shortest paths. Experiments have shown that minimizing choices is important for efficiently computing the shortest paths for each waypoint type [2]. Figure 8.5a and b illustrate two maps, the first showing the water connectivity under normal conditions and the second showing the water connectivity under maximum flooded conditions. Figure 8.6a and b illustrate two maps, the first showing the road connectivity under normal conditions and the second showing the road connectivity under maximum flooded conditions. To ensure that maximum connectivity is used for busses and boats cases, the shortest path algorithm assumes that roads are under normal conditions (Figure 8.6a) and that waterways are under maximum flooded condition (Figure 8.5b).

### 8.1.1.2 Stranded People

The 20,000 stranded people are inside buildings, and do not move unless rescued. All people are randomly placed as follows: first, a random floor location is picked and one to ten individuals are placed in that location (average 5.5). Both selection processes are random using a uniform distribution [3]. This two-step process continues until all 20,000 people are allocated. All floors have a visible and hidden location from outside view; thus some people can be seen from helicopters while some cannot. The current setting is that 25% of locations have people at observable waypoints within buildings. Note that stranded people are not placed on flooded floors of any building.

In addition, people are divided into three categories: supplied, unsupplied, and injured. The current settings are that half of the locations are supplied and half are not. Also, currently 5% of the locations have an injured individual. Thus, there are about 10,000 supplied and 10,000 unsupplied individuals on average and about 180 injured individuals (20000/5.5 x 0.05). The times for survival of unsupplied and injured individuals are selected from the following density functions. For unsupplied, a natural lognormal distribution is used [3], with the mean (m) = 3.9 (exp(3.9) = 50 hours) and the standard deviation (sd) = 0.11 (providing a range of approximately 33 to 72 hours). For injured, a normal distribution is used [3], with the m = 16 (hours) and sd = 7 (providing a range of approximately 1 to 40 hours). Values calculated under zero hours for injured individuals are reset to one hour, thus giving the injured at least some minimal lifespan. Unsupplied and injured distributions are shown in Figure 8.7a and 8.7b, respectively.

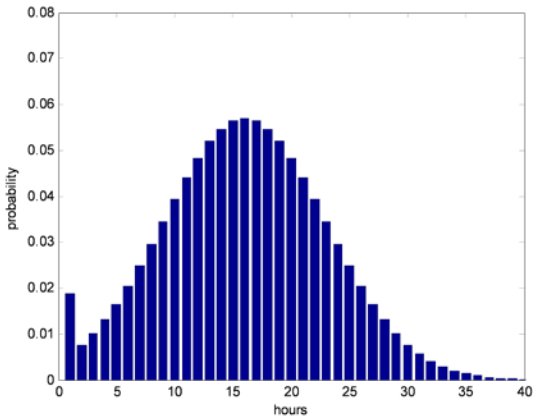


Figure 8.7a. Probability Mass Function for Survival Times of Injured People

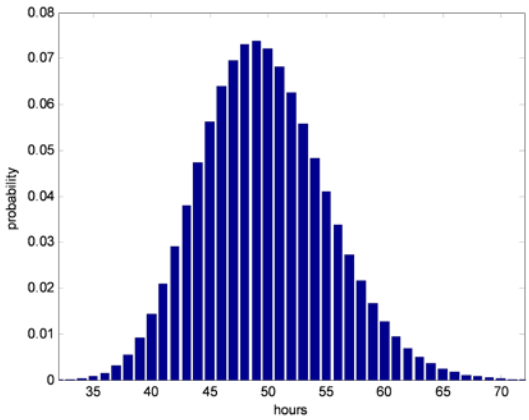


Figure 8.7b. Probability Mass Function for Survival Times of Unsupplied People

### 8.1.2 Planners or Players ($\Psi$)

The player is the centerpiece of the approach. The player has a set of control parameters that determine its nature for generating, executing, and assessing plans. For a single player cooperative game, as is the case here, the player is the planner and determines many aspects of the core planning cycle. Each action within a plan is chosen according to the player's characteristics. For the problem here, the player's characteristics are defined by parameters in four areas: (1) model update time, (2) expectation acceptance threshold for re-planning tradeoff, (3) number of each agent type, and (4) nine decision features described in Table 8.1.

Table 8.1. Decision Features with Corresponding Vehicle Type and Value Range

| Decision Feature | Vehicle Type | Value Range |
|---|---|---|
| (1) leg distance | All vehicles | (0, 6500) meters |
| (2) distance from nearest base | All vehicles | (0, 6500) meters |
| (3) distance to nearest same type vehicle | All vehicles | (0, 6500) meters |
| (4) number people present | All vehicles | (unknown, 1, max) people |
| (5) unsupplied present | All vehicles | (unknown, 1, max) people |
| (6) injured present | All vehicles | (unknown, 1, max) people |
| (7) bldg spotted with people | All vehicles | (0, 1) fraction of bldgs per stop |
| (8) estimated unsupplied survival time | All vehicles | (-100, 100) hours |
| (9) destination visited by boat or bus | Helicopters Only | (No, Yes) |

First, for partially observable environments, the model update is paramount, because it refreshes the virtual-state model to more closely resemble the true real-state model of the environment. This refresh comes from information collected by the sensor input of the agents. This refresh gives the planner the opportunity to better predict

outcomes of future actions. For the current application, this model update parameter was not adapted but set to a reasonable value. Specifically, every time a vehicle returned to base (about 20 minutes in real-world-model time), all agents were required to report all of their newly acquired sensor data. This assumption was necessary, because our experiments showed that not updating the virtual model frequently enough caused worse results than randomly searching with continual updates. This result is described in Section 8.3.2.

Another important parameter for the player is the expectation acceptance threshold ($r_Q$) ($0 <= r_Q <= 1$) for meeting expectations. This expectation acceptance threshold trades time to plan against the desire to re-plan. This competition is based on whether a plan is meeting expectations as assessed in plan-assessment. When the expectation acceptance threshold is high, the tendency is to re-plan often; when expectation acceptance threshold is low, the tendency is to stick to the current plan and thus avoid the time for re-planning. More specifically, if the expectation acceptance threshold is set to one, then expectations must be met completely or the player will re-plan. If the expectation acceptance threshold is set to zero, then the player will only re-plan when absolutely necessary. Results from this tradeoff are described in Section 8.3.3.

Third, the relative quantity of each agent type used to best solve a partially observable problem is unknown and therefore made adaptable. For the problem at hand, experiments have shown that providing an unrestricted number of agents will lead to minimal loss of life and therefore trivializes the problem. Therefore, as in the real-world, resources are limited, and in this particular case, only twenty agents are allowed. For the

US&RO problem, there are four vehicle types used: boats, busses, survey, and rescue helicopters. Thus, four parameters are used to indicate number of each vehicle type. Results from this tradeoff are also described in Section 8.3.3.

Fourth, the nine decision features shown in Table 8.1 were used because these features are quickly calculable for each possible action, and appear relevant to choosing actions. Given that at each time step an agent has one of thousands of destinations (3649 buildings) to choose from and plans may have lengths in the dozens of actions, a tree search method is impractical. Also, one does not want to restrict an agent to follow a deterministic rule set based on any specific feature or combination of features, because this severely limits the search space of possible choices. Thus, a new method was developed which does not restrict the search space. It combines a Monte Carlo (MC) method [4] for choosing actions and weighted Beta distributions (weighted BDs or WBDs) method [5] for representing decision features.

For our purposes, MC represents taking a random choice from many possibilities, and the WBDs represent the density functions of actions from which to make those choices. A WBD required for each feature and agent type has three parameters: $w$, $\alpha$, and $\beta$. The probability mass function of the WBD is as follows:

$$f(x; w, \alpha, \beta) = w \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} x(1-x)^{\beta-1} \qquad \text{(eq. 8.1)}$$

where $\alpha$ is the alpha parameter, $\beta$ is the beta parameter, $w$ is the weight parameter, $x$ is a vector of feature values corresponding to all possible actions, and $\Gamma$ is the Gamma function [5].

From Table 8.1, there are 34 WBDs required (two vehicle types times eight features plus two vehicle types times one feature), and thus, 102 WBD parameters (three parameters for each WBD times 34 WBDs) for this US&RO problem. The three WBD parameters are weight, alpha and beta; all initial values are randomly chosen. For each agent, the weights are greater than or equal to zero and sum to one. The alpha and beta parameters are restricted to be greater than zero and provide a variety of possible distributions as shown in the results Section 8.3.3. Another reason WBDs are used is that they do not restrict the search space if the initial parameters are particularly set. For instance, if all WBDs have alpha and beta parameters set to {1, 1}, then the probability of selecting any action is equally likely at every time step, independent of the weightings.

The process of using MC and WBD together is as follows. First, for each agent under consideration, a mass function is calculated for each feature type for all possible actions (Equation 8.1 without weight *w*). The mass function is a vector of length equal to the number of possible actions and each mass represents its particular influence in action selection. Before *w* can be applied, each mass function is normalized to sum to one to represent a probability mass function. Next, each mass function is multiplied by the corresponding weight (*w*) assigned to that agent and feature. Finally, all resulting mass functions corresponding to each agent are summed. This new mass function, which holds the accumulated influence of all features for each agent, is normalized and expressed as a distributed mass function (0, 1) to form a reference lookup table for each action. At this point the selection process begins. First, a random number is chosen from a uniform density (0, 1), and this random value chooses the action by looking up the corresponding action linked to that number in the distributed mass function table. Note that, the greater a

feature's weight, the greater its influence, and the BD settings focus on influential feature values.

In summary, the planner controls the planning with 107 parameters. There is an expectation acceptance threshold, four vehicle type parameters, and 102 WBD parameters. More specifically, since the boat and bus agents use eight of the nine features, there are 24 WBD parameters (8 weights, and 16 alphas and betas) for both boat and bus type vehicles. Since helicopters use all nine features, there are 27 WBD parameters (9 weights, 9 alpha, and 9 beta) for both survey and rescue helicopter type vehicles. Thus, the total number of parameters is 107 (1 + 4 + 24 + 24 + 27 + 27).

### 8.1.3 Agents ($I$)

Agents are the instruments of the plans. Agents contain various actuators and sensors. For the US&RO problem, the agents are vehicles used in the model environments. There are four types of vehicles used: boats, busses, and survey and rescue helicopters. The agents' actuators and sensors are shown in Tables 8.2 and 8.3, respectively. These tables illustrate all the agent categories and their associated characteristics. Table 8.4 shows the agents' temporal costs associated with each planned action. Each agent has a separate plan and each plan is defined to include for each chosen action: a time estimate, location, fuel level, passenger list, and speed. Agents are specific to the problem domain, so a more detailed description is in Section 8.2: Implementation Strategy.

Planning has four types of agents at its disposal: boats, busses, rescue helicopters, and survey helicopters. All possible movement paths for vehicles, building access and

movement within buildings are pre-calculated. There are 31,115 movement locations called waypoints, and there are five types of waypoints used for movement: roads for busses, water for boats, sky for helicopters, building accesses for all vehicles, and floors within buildings.

Table 8.2. Agent or Vehicle Actuators

| Agent or Vehicle Actuators | Boats | Busses | Rescue Copters | Survey Helicopters |
|---|---|---|---|---|
| **Purpose** | Search Bldgs; Rescue People | Search Bldgs; Rescue People | Search Bldgs; Rescue People; Drop Supply | Survey Road and Water Access; Spot People; Drop Supplies |
| **Travel speed** | 40 km/hr | 50 km/hr | 100 km/hr | 45 km/hr |
| **Move** | Water | Road | Sky | Sky |
| **Range** | 4 hrs | 8 hrs | 1.5 hrs | 1.5 hrs |
| **Capacity** | 8 people | 50 people | 10 people | n/a |
| **Supplies** | n/a | n/a | 1 Drop | 12 Drops |

Table 8.3. Agent or Vehicle Sensors

| Agent or Vehicle Sensors | *Boats* | *Busses* | *Rescue Helicopters* | *Survey Helicopters* |
|---|---|---|---|---|
| ***Route Access via Trial*** | YES | YES | n/a | n/a |
| ***Route Access via Sight*** | NO | NO | LIMITED | YES |
| ***Survivors via Search*** | YES | YES | YES | NO |
| ***Survivors via Search*** | NO | NO | LIMITED | YES |

The connectivity of the model is assumed different for each vehicle type. Busses can travel on any road, but many roads are flooded and some are blocked (ten randomly placed roadblocks per simulation). Boat planning assumes the city is flooded at the highest stage (see Figure 8.3 or 8.5b), where all low-lying areas can be traversed. In the case used, boats can traverse to all low-lying areas. Helicopters paths are unaffected by

water level. Helicopters are required because some places cannot be reached by bus or boat. Each vehicle also has many other particular attributes. Tables 8.1 to 8.4 define all the relevant attributes of the vehicles (agents).

Table 8.4. Time Constraints and Delays

| Time Constraints / Delays | *Boats* | *Busses* | *Rescue Copters* | *Survey Copters* |
|---|---|---|---|---|
| *Floor Search* | 300 sec | 200 sec | 600 sec | n/a |
| *Building Evacuation* | 40 sec/ person | 20 sec/ person | 120 sec/ person | n/a |
| *Road Blocks* | 600 sec | n/a | n/a | n/a |
| *Backtrack* | 60 sec | n/a | n/a | n/a |
| *Bad Plan* | 10 sec/ plan | n/a | n/a | n/a |
| *Drop Off People* | 300 sec | 300 sec | 300 sec | 300 sec |
| *Drop Off Supplies* | n/a | n/a | 300 sec | 300 sec |
| *Turn Around* | 30 sec | n/a | n/a | n/a |
| *Building Submerged* | n/a | 120 sec | 120 sec | n/a |
| *On Base No Orders* | 1200 sec | 1200 sec | 1200 sec | 1200 sec |
| *Off Base No Orders* | 120 sec | 120 sec | 120 sec | 120 sec |
| *Reinitialize Plan* | 600 sec | 600 sec | 600 sec | 600 sec |

All agents require time to execute tasks, and Table 8.4 shows all timed tasks except for movement times, which are dependent on speed and distance traveled (see Table 8.2). Note that in Table 8.4 helicopters hovering over buildings require more time to search buildings than busses or boats, and busses are by far the quickest. Note also that busses evacuate the buildings the fastest per person, because they do not have to deal with water or hovering in midair. Busses are the only vehicle type that experience

problems with roadblocks and dead-ends due to water level. Busses thus have additional time penalties associated with making plans to retrace paths back to previously traversable areas. Only helicopters drop off supplies.

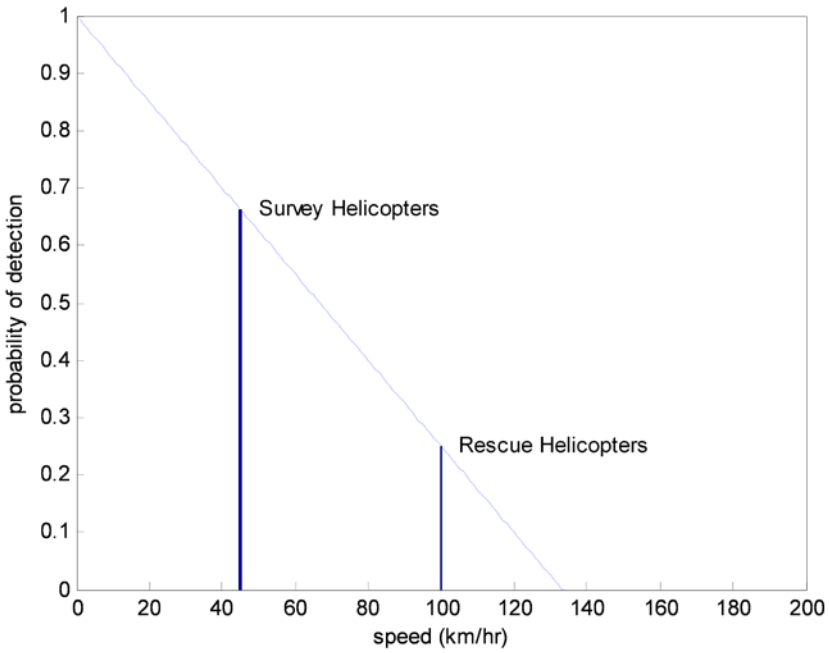


Figure 8.8. Probability of Detecting People and Passageways

Tables 8.2 and 8.3 describe the agents' actuators and sensors, respectively. Note the difference in purpose between the two helicopter types and the speed differences among all types of vehicles. Busses have the highest capacity and range, while survey helicopters have visual capabilities to spot people and check land or water movement access. Rescue helicopters have limited visual abilities because the increased speed reduces the probability of seeing something. Survey and rescue helicopters have different probability in seeing people or passageways, and it is based on speed as shown in Figure 8.8. Based on the helicopters speeds 45 km/hr and 100 km/hr, the survey and rescue helicopters can see waypoints below with the probability of 0.66 and 0.25, respectively.

Survey helicopters primarily survey for open movement paths and drop off supplies; they do not search or evacuate buildings, because this is considered too time consuming for this type of helicopter.

### 8.1.4 Plans (*P*)

Unlike in RISK, plan search for the US&RO problem is limited in depth, because from experience, long-term plans are often discarded due to new observations in the environment. Long plans cannot keep up with these observations, so applying a cost to the planning activity limits re-planning to a single vehicle roundtrip (as shown in Section 8.3.2). Also, unlike the RISK application, US&RO has many simultaneous and time-dependent actions, which need to be managed to ensure that many planning and re-planning strategies for many agents can be contemplated without scheduling conflicts. For instance, the planner does not plan to send multiple vehicles to the same location at the same time. Thus, planning, execution, and assessment occur over multiple agents simultaneously. More specifically, throughout the planning cycle, the next action to be chosen is for the vehicle that the nearest in time completed action.

For US&RO, plans are considered as interdependent sequences of actions with one sequence per agent. Due to computational constraints and given only partial state information, searching multiple plans as in RISK is infeasible. Thus, plans are generated as described in Section 8.1.2, where actions are chosen based on the WBD parameter sets. These parameters are learned as described in Sections 8.1.9 and 8.1.10 and results are shown in Section 8.3.3.

### 8.1.5 Tasks (*T*)

The US&RO goal is to evacuate as many people as possible while minimizing casualties. Thus, a completed task occurs every time people are rescued from a building. The degree of completing a task is graded on how well the projected or anticipated rescue matches the real rescue operation. Points are assigned to each person rescued, and observing if the point total is met to some degree of acceptance is considered as meeting expectations.

Since there are injured, unsupplied, and supplied individuals, where the injured or unsupplied could die at anytime (Figure 8.7a and b), it is more important to evacuate injured and unsupplied first. Also, injured people tend to need more immediate attention (compare Figures 8.7a to 8.7b), so it is even more important to get injured people out of the buildings sooner. Therefore, a point system was created that gives three points per injured person, two points per unsupplied person, one point per supplied person and zero points per dead person (casualty) retrieved. A threshold (an acceptable prediction error in meeting expectations) of what percentage of points is acceptable before re-planning is shown in Section 8.3.3.

### 8.1.6 Actions (*A*)

For US&RO application, multiple agents can travel by roads, waterways, and sky to survey surface routes, spot or supply people within buildings, and search and rescue people from buildings. Each agent has: a purpose, a travel speed, a set of movement waypoints, a range, a passenger capacity, and can possibly drop supplies. Nominal parameter values are shown in Table 8.2, but travel speed, range, capacity, and number of

supplies are modifiable. Each vehicle is an agent, and each agent type: boats, busses, and survey and rescue helicopters have different options. The options select a set of actions, which formulate a plan, and each vehicle is given its own plan. The number of vehicles chosen is arbitrary, but for our simulation, twenty total vehicles of various types were chosen.

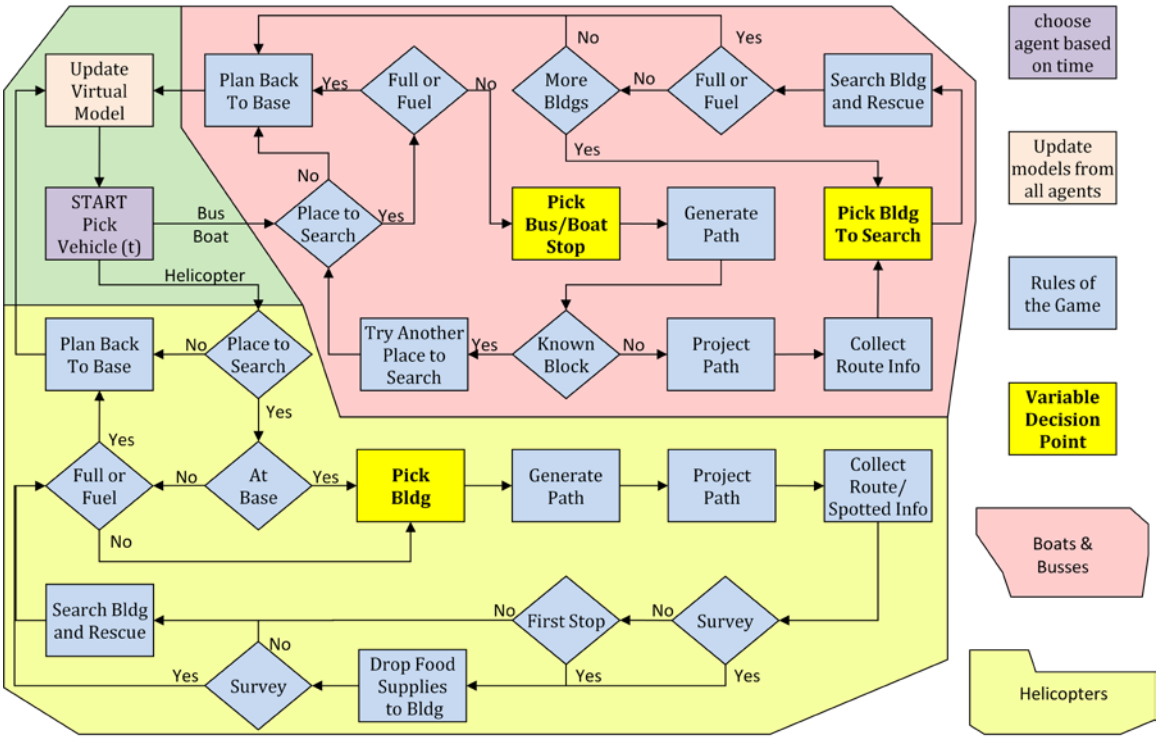


Figure 8.9. Urban Search and Rescue Functional Block Diagram

Actions for planning are based on the functional diagram shown in Figure 8.9. Available actions depend on the vehicle type and state of the vehicle. Boats and busses can choose any boat or bus stop as a destination, respectively. The three yellow blocks in the diagram indicate where the variable choices are decided for all vehicle types. Boat and bus stops may serve multiple buildings, thus plans for agents need to also select which buildings at each stop to visit. It is assumed that a boat or bus does not leave a stop

until they are either full of passengers, or all buildings at that stop have been searched. Rescue helicopters can choose any specific building for search and rescue, and they drop a single supply package when they search the first building that has people. Survey helicopters survey roads, and spot or supply people within buildings. All vehicles return to base when their fuel time is spent, or the vehicle is full of passengers (see Table 8.2).

As actions are taken, the environment responds with time penalties, which are determined by travel speed shown in Table 8.2, or other time delays as shown in Table 8.4. Plans are made based on real known observation models combined with approximated (unobserved) models of the environment. This was described more thoroughly in Section 8.1.8: Observations. Each vehicle's plan is considered complete when a return to base is included in the plan. Planning beyond this point has proven fruitless, because the real models differed greatly from observed models for long-term plans. Results on model update are shown in Sections 8.3.2.

### 8.1.7 Outcomes (*Y*)

There are five types of outcomes in the ADP&E application of US&RO: (1) people rescued including type (i.e., injured, etc.); (2) buildings searched; (3) buildings evacuated; (4) the time a vehicle reaches a planned destination; and (5) the time a vehicle is predicted to reach base for a new assignment. For buildings that have been reached via any vehicle except supply helicopters, the building is searched and the vehicle is loaded with as many passengers as possible. The non-evacuated people are now known for the next visit. The number of people rescued and type is only known after a building is searched, thus an approximated number of people are the expected outcome (based on

those missing from the 20,000) so that a building would be visited in the first place with some positive expectation. A time a vehicle is predicted to reach a destination depends on two major factors: route blockage in the bus case (via water or downed tree), and how much time did the previous action take differ from what was expected. For instance, if more or less passengers were acquired then more or less time was taken, respectively. Thus, outcomes for US&RO are considered straightforward for areas where the environment has been observed, but for unobserved portions of the environment, the outcomes can vary a great deal and propagate their error to future actions.

### 8.1.8 Observations (*O*)

Model observations include roads, waterways, building content (i.e., people), and total people rescued including casualties. Road models determine which roads are open to bus travel. Water models determine which waterways are open for boat travel. Buildings include models that contain people of four types: injured, unsupplied, supplied, and dead.

There are two models for each observation type. One is the real or physical state, and the other is the assumed or known state. The known state models are initialized to the assumed values. The road model assumes all roads are open, and the water model assumes all low-lying areas are flooded and traversable by boat. The building observed model assumes the person quantities are unknown for every floor of every building until observed. As they are found, injured and unsupplied people are on a time clock and are considered casualties if not reached in time.

The known model learns real model information by tactile and visual sensing. Tactile sensing of travel routes is accomplished with boats and busses simply by

traversing each connection and visiting each waypoint. As boats and busses follow paths, they discover whether they are open. Helicopters also can view surface travel routes. Helicopters flying over surface waypoints can see if they are water or land, and slower helicopters see more as shown in Figure 8.8. For our application, survey helicopters travel much slower and have a better chance (i.e., probability) of seeing waypoint conditions below than higher speed rescue helicopters.

Tactile and visual sensing is also implemented for observing people. People are assumed to not travel without being rescued. The only visual sensing from afar is by helicopters. People can be spotted, but their type cannot be identified, such as injured, unsupplied, etc. All tactile observations of people are done through searching buildings. Boats, busses, and rescue helicopters visiting a building will search it on the first visit. This will identify each building floor contents, and as many people as possible will be evacuated in the process. Travel times within buildings, and other time costs are shown in Table 8.4. Rescue capacities and other possible actions for each vehicle type are identified in Table 8.2. In summary, models have incomplete information a priori, which includes blocks in roads and water, and building content. Each model has either a real or an observed copy of this information. Depending on the planning phase, the observed or real copy is used. Plan-generation and -assessment use the observed copy stored within the planning-model, and plan-execution uses the real copy stored within the real-world model.

### 8.1.9 Rounds or Competitions ($\mathcal{Q}$)

The rounds tier level is used to compare the accumulated results from all planners for each generation of tournament play. For the US&RO application, each generation or round is a sixty game competition with twenty players. Each game played uses a different randomly generated environment. Thus, each player plays out multiple games to determine its general effectiveness for different partially observable environments. Each player plays the game three times. The score for each player is the worst case, which is the maximum number of casualties of the three games. The players' planning capabilities are adapted using the following handcrafted GA:

- Top 4 performers return for next generation
- Next 4, mutate decision feature weights, vehicle number, and expectation acceptance threshold of top 4
- Next 4, mutate the BD parameters of top 4
- Next 4, crossover decision feature weights, vehicle number, and expectation acceptance threshold
- Next 4, crossover the BD parameters

For each top player selected, the weights, expectation acceptance threshold, and vehicle number are mutated independently. Each weight is adjusted by adding a random variable selected from a normal distribution with zero mean and standard deviation 0.2. The weight is adjusted not to be negative (shifted up to zero) or exceed one (shifted down to one). The weights are then renormalized to sum to one.

The expectation acceptance threshold for each top player selected has a 50% probability of being changed. If selected to change, it is adjusted by adding a value selected from a normal distribution with mean zero and standard deviation 0.05 and the result is readjusted not to be negative or exceed one. Each number-of-vehicle parameter

is also adjusted with a 50% probability. Each vehicle type has the probability of increasing or decreasing provided the number is greater than zero for busses, boats, and survey helicopters, and one for rescue helicopters. There needs to be at least one rescue helicopter to ensure that the simulation finishes, because rescue helicopters are the only vehicle that can reach every building and pick up all stranded people. The sum of all vehicle types is equal to twenty, so the change in vehicle number is normalized to match twenty total vehicles.

The mutation of the BD parameters is a special case, because the values must only be greater than zero. Therefore, a lognormal density function is used to pick the adjustment in the alpha and beta parameters. The mean of the density function chosen is two divided by the generation number and the standard deviation is the square root of this calculated mean. This lognormal density produces an adjustment in the BD parameters to be very large at first and decrease with each subsequent generation. Results showed this to be effective in providing enough variability in the BD densities.

For all crossovers, a player is picked based on how well it did with respect to other players. Players with better than average performance are put into a pool with probability of selection proportional to its value above the average. Each of the top 4 performers (top players) is combined with a randomly selected player from the pool. The new players are initially a copy of the top four players. Crossover is performed through choosing each parameter of these players and giving a 50% chance of acquiring a new parameter value from the other randomly selected player.

### 8.1.10 Tournaments ($\Omega$)

A tournament runs many rounds until the parameters show a satisfactory level of convergence, or the capability of the planner seems to show little progress in improvement. Section 8.3.3 Tournament Planner Training shows measures of efficacy of a planner, and the measures of convergence of all planner parameters.

### 8.1.11 Summary

In summary, the ADP&E framework was built to scale efficiently. The core planning cycle is not concerned with the agent or any other factors above it in the hierarchy. This allows a separation among the tiers, so that plans can be generated in the same way for all agents. In other words, the core cycle is only concerned with processing actions and is not concerned with the tiers above. Agents in the same way are not concerned with the player giving the orders, only that the orders were given and need to be carried out. Players have control parameters that make them versatile and adaptive. Games are the completion of a large complex task, and it is imperative that the game is set up to always finish (having at least one rescue helicopter). Rounds are a strategy level above the game that compares game outcomes of multiple planners to improve the utility of planner parameters in generating plans, executing actions, updating models, and reassessing outcomes. A tournament captures the set of rounds and measures the effectiveness of converging on planners that reach an acceptable level of performance.

## 8.2 Implementation Strategy

The implementation strategy follows the nine-step DBT approach described in Chapter 6. The approach for the simulation of an US&RO is divided into these nine

incremental steps: (1) **execute arbitrary actions**; (2) **update planning-model**; (3) **choose and execute single actions**; (4) **generate arbitrary multi-action plans**; (5) **generate and choose plans**; (6) **assess plans**; (7) coordinate **core planning cycle** for a single agent's operation; (8) coordinate multiple agents to complete **a single game or mission** evacuation; and (9) implement **a complete tournament** as a meta-learning strategy for training planners to autonomously improve performance.

### 8.2.1 Execute Arbitrary Actions

**Execute arbitrary actions** implementation requires four building blocks: simulation or game model, simulation state, planning state, and action choices as shown in Figure 6.7 in Chapter 6. For the simulation of an US&RO, the simulation model is a function of the terrain model, waypoint connectivity, agents or vehicles available, and simulation constraints shown in Tables 8.1 to 8.4. The simulation state is a function of the agents and the environment. State variables include: (1) agents' location, passengers, food supplies, fuel available, and acquired observations of the environment, (2) which buildings have been searched and/or evacuated, which roads and waterways are available, and where have people been found that are injured, unsupplied, and supplied. Every agent is somewhere in the functional diagram shown in Figure 8.9. As shown in Figure 8.9, each agent is a function of their current state.

Before discussing how random actions are chosen, very fast planning simulation is enabled by pre-computing all shortest paths for every pair of destination waypoints for boats, busses, and helicopters. This enables a quick lookup for each action chosen so that the simulation can run much faster than real time. Boat and bus paths can also be

computed for different water levels as well, but for the example experiment only one water level was used. For simplicity, the positions of boat and bus stops are the same, whether they are above or below the waterline. Each vehicle requires a base to drop off rescued people, so boat bases are put at the highest point of each river, a bus base is placed at a high elevation road waypoint, and a helicopter base is placed as described earlier (Figure 8.3).

Preprocessing for shortest paths is done for all road waypoints for busses, all water waypoints for boats, and all sky waypoints for helicopters [6]. This produces two matrices for each domain: a path matrix which indicates the link one must take to go from any waypoint to another; and a distance matrix which indicates the distance between every pair of waypoints. The number of matrix lookups for a trip is the number of waypoints traversed on a path between the starting and ending waypoints.

In addition to preprocessing shortest path, line-of-sight for helicopters is also preprocessed. Line-of-sight for boats and busses are not used here, because they are designed to rely heavily on tactile (i.e., information at current waypoint) sensors. Helicopters, on the other hand are not in direct contact with roads, water, or buildings, thus, they require a line-of-sight mapping to each. For simplicity, line-of-sight can only observe nearby waypoints, and is further restricted based on speed of the helicopter as described in Section 8.1.3. Flying over roads and water, a helicopter can verify if the surface is available for boats or busses. Flying over buildings, a helicopter passing slowly enough, can verify if individuals are in the buildings. The probability of spotting people is inversely proportional to the helicopter speed as shown in Figure 8.8.

As shown in Figure 6.7, the building blocks require four functional interactions for executing random actions: (2.1) *check actions are executable* (i.e., $P' = P_{\Xi}(P_Z, \gamma, M_{\psi}, X_P)$); (2.2) *execute actions* (i.e., $\{A_1, P'\} = A_{\Xi}(P', t)$); (2.3) *realize outcomes* (i.e., $Y' = Y_R(A_1, \gamma, M_{\Xi}, X_{\Xi})$); and (2.4) *sense response* (i.e., $O' = O_{\Xi}(Y', \gamma, M_{\Xi}, X_{\Xi})$). The four functions here are tailored for the simulation of an US&RO and described in detail below.

The *check actions are executable* function requires four inputs: (1) the selected plans ($P_Z$); (2) the corresponding agent under consideration ($\gamma$); (3) the planning-model of the current planner ($M_{\psi}$); and (4) the current state of the corresponding plan ($X_P$). In the **execute arbitrary actions** implementation, the selected plan is a single action for each agent as generated from the functional block diagram shown in Figure 8.9. As a plan is executed, each next agent under consideration depends on the time criteria described in Section 8.1.6: Actions. In the **execute arbitrary actions** implementation, the planning model is considered equivalent to the real-world model, thus the entire model is assumed fully observable for this exercise. This *check actions are executable* function outputs a single-action plan ($P'$) for each agent, which demonstrates and tests that the rules of the game are properly implemented.

The *execute actions* function requires two inputs: (1) plan-remainder ($P'$ described above), and (2) time ($t$). Each action for all agents is executed in time order depending on the calculated time delays using Tables 8.2 and 8.4. For *execute actions* function, the plan-remainder is null. This function outputs two separate variables, the first action ($A_1$) and the new plan-remainder ($P'$). The *execute actions* function is the same as the process

3.1 shown in Table 6.5, check plans can be assessed function. After plans are sequenced together, the plan-remainder will be adapted as described for process 3.1 described later.

The *realize outcomes* function plays out a single action per time step in the real-world simulation of an US&RO. This function uses four already described variables: (1) first time-ordered actions ($A_1$); (2) corresponding agent under consideration ($\gamma$); (3) the real-world or environmental model ($M_\Xi$); and (4) the current state of the environment ($X_\Xi$). This function outputs realized outcomes ($Y'$), which are the traveling of agents, searching and rescuing of people out of buildings, and acquisition of knowledge of pathways and people locations.

The *sense response* function observes each action's outcome and modifies the current state to match this observation. This function uses four already described variables: (1) realized outcomes ($Y'$); (2) the corresponding agent under consideration ($\gamma$); (3) the real-world or environmental model ($M_\Xi$ described above); and (4) the current state of the environment ($X_\Xi$ described above). This function outputs realized observations ($O'$), which are a one to one mapping with all realized outcomes ($Y'$).

### 8.2.2 Update Planning-Model

In the update-planning model implementation, the automated planner becomes more aware of its surroundings, and thus can better plan agent activity to take into account this new acquired knowledge. For the US&RO application, the real-world model is only partially observable, thus updating the planning-model is an important part of implementing an ADP&E framework. The model changes occur in both the pathways to

the buildings (water or road) and the building content (how many people are there and of what type).

Updating the planning-model uses the *feedback state* function (2.5: $X_U$). This function uses three input variables: (1) a set of agents under consideration for planning ($\Gamma$); (2) the observations ($O$ described above); and (3) the task update coefficient ($\tau_U$). The set of agents are the vehicles available to the planner. The agents' locations, passengers, fuel capacity, and acquired knowledge are modified based on realized observations. The task update coefficient is a single completion of a roundtrip from base to base for each vehicle. This gives the planner the opportunity to change their plan for any of the agents based on all the newly acquired knowledge from all agents. In other words, all agents are required to report their findings each time any agent reaches base. The output of this function is a new state representation for the internal planning-model (i.e., $X_\psi = X_U(\Gamma, O', t_U)$). All new plans will use this state as a starting point for generating new plans.

### 8.2.3 Choose and Execute Single Actions

**Choose and execute single actions** implementation of the simulation of an US&RO requires the final building block, the value function (V), as shown in Figure 6.10. The value function includes all control parameters of a planner. In choosing and executing actions, only a small portion of these parameters is introduced. These parameters are those used in selecting individual actions. The choosing of actions for the simulation of an US&RO is done as described in Section 8.1.6: Actions. Selecting amongst all the possible choices available is based on the functional diagram (Figure 8.9)

and current state of each agent. This functional diagram is used as the *determine action choices* function based on the current state (i.e., 1.2: $A = \mathrm{I}_{\mathrm{A}}(\gamma, M_{\Psi}, X_P)$). *Select actions* function is done using a Monte Carlo method (i.e., 1.3: $A_Z = \mathrm{Z}_{\mathrm{A}}(\Gamma, A, H, R, C_A, t)$). For the simulation of an US&RO, action selection requires information on all of the variables: agents ($\Gamma$), action choices ($A$), selected features ($H$), each agent's available resources ($R$), each action's temporal costs ($C_A$), and time ($t$). More specifically, choosing which agent to choose an action for requires previous action temporal costs, time delays, and resources available. Choosing a particular destination for the selected agent requires the selected features (as described in Section 8.1.2 using WBDs) and action choices available. This implementation tests the coordination of using plan-generation with plan-execution.

### 8.2.4 Generate Arbitrary Multi-Action Plans

**Generate arbitrary multi-action plans** implementation for the simulation of an US&RO is equivalent to generating multiple actions for each agent until a plan for each is complete. A completed plan is one which has an agent return to base based on either dropping off all supplies, being filled with passengers, or running low on fuel. Each action is chosen as described in section 8.1.6. However, the plans are generated using the planning-model instead of the real-world model. Many model features in the planning-model are unknown, but their state values are predicted based on the process described in Section 8.1.8: Observations. These model features are required for processing the interactions of model and state for each action. The interaction of model and state process

has four sub-process functions: (1.4.1) *predict outcomes*, (1.4.2) *select outcomes*, (1.4.3) *observe state changes*, and (1.4.4) *update plan's model state* as shown in Table 6.5.

For the simulation of an US&RO, the *predict outcomes* function (i.e., $Y = K_Y(A_Z, \gamma, M_\Psi, X_P)$) uses the selected action ($A_Z$) described above and calculates the outcomes ($Y$) based on using the planning-model state and information shown in Tables 8.1 to 8.4. The *select outcomes* function (i.e., $Y_Z = Z_Y(Y, r_Y)$) is not required for the US&RO application, because there is a one-to-one mapping of outcomes to actions. The *observe state changes* function (i.e., $O = O_X(Y_Z, \gamma, M_\Psi, X_P)$) uses the selected outcome ($Y_Z$) as the observable state transition ($O$) for each action. Finally, the *update plan's model state* (i.e., $X_P = X_U(\Gamma, O, \tau_U)$) uses the observations ($O$) to update the planning-model state ($X_P$) for each action chosen within a plan. The task update coefficient ($\tau_U$) is the completion of a roundtrip for each vehicle agent. The task update coefficient is not the actual number of completed building visits, because that number for each vehicle is dependent on passengers found, traversable pathways, and fuel capacity. The tradeoffs in task update coefficient are discussed further in Subsection 8.3.2: Planning Cycle.

### 8.2.5 Generate and Choose Plans

**Generate and choose plans** implementation uses the full capacity of plan-generation (processes 1.1 – 1.5 described in Sections 8.2.3 and 8.2.4) for a single planner choosing plans for multiple agents simultaneously. More specifically, **generate and choose plans** requires the entire plan-generation processes: (1.1) a *search and select plans* function, (1.2), (1.3), (1.4), and (1.5) an *evaluate plan fitness* function. The *search and select plans* function includes the use of fitness to choose plans, and a *forecast*

*expectations* function to measure the completed tasks for each plan. The *evaluate plan fitness* function includes SME in choosing the features considered important to planning. Forecasted expectations are already described in Section 8.1.5: Tasks.

Unlike RISK, plans are not searched but each action is chosen based on current perceived environmental features as shown in Table 8.1. Table 8.1 shows the list of features pertinent to planning (one to nine) and their applicability in making each planning decision. Note that all decisions involve going to a particular stop or building. To simplify the experiment, all decisions to go back to base are automated: if a vehicle is full of passengers or low on fuel, the vehicle returns to base, irrespective of planned activity. Boats and busses have features that determine which stop or building to go to, because each boat or bus stop may have multiple buildings at that location (i.e., 1 to 5). Thus, the planner must first decide which stop the bus or boat should go to, and then decide in which order to visit each building at that stop, using the same decision features. Depending on their nature, the planner's decision features shown in Table 8.1 are in four different formats discussed below.

First, decision features one, two and three are measurements of distance. Thus these decision features have a range of values (i.e., zero to 6500 meters). "Action Distance" is for a particular leg of the trip, and "Nearest Vehicle Distance" pertains only to same type vehicles.

Second, since many stops and buildings are unsearched and have varying numbers of people left behind within a building once searched, decision features four, five, and six are multi-valued. Specifically, for unsearched buildings, the smallest value is used and

for searched buildings the range of values used is distributed as $\log_{10}(PLB+1)$, where PLB is the number of people left behind. Feature seven is also multi-valued. For feature seven, all stops and building locations are assigned zero unless people are spotted. For instance, if a helicopter spots a person at a building, then the value assigned to that building is one. However, for bus and boat stops a one is not necessarily assigned. For a stop, a fraction is assigned which represents the number of buildings spotted with people at that stop over total buildings at that stop.

For feature eight, survival time is the projected amount of time available for unsupplied or injured people before they become casualties. Thus, an estimated time is given for all stops and buildings, and then, as people are discovered, their true lifespan time is used to update the feature. It is presumed that all unsupplied and injured people will become casualties within 75 hours (described above), thus 75 hours is assumed for all unsearched locations. This feature value changes with time. People not retrieved in time will have a negative lifespan. Thus, they will be dead if their building was previously searched (thus their survival time is known), and presumed dead for an unsearched building. Buildings where supplies are dropped add 48 hours to the lifespan for unsupplied people in those buildings. Injured people must be rescued to survive, because supplies do not extend their lifetime.

Fourth, feature nine is a simple binary value. This feature is initially valued zero for all buildings. If a boat or bus has searched a building, then the value for that building is set to one, otherwise the value remains zero.

All features are normalized to be between zero and one for later use. For decision features one, two and three, the largest diagonal distance across the city is used to normalize the decision features (6500 meters). Subtracting their minimum value and dividing by their range normalizes all feature values. For example, for feature eight, all values are kept between -100 and 100, thus, 100 is added and the result is divided by 200.

The number of vehicles used is fixed per game at twenty. The number of vehicles of each type is left to the planner with the restriction that a minimum of one rescue helicopter is in the environment at all times to ensure that the game will run to completion. One complete game run is the evacuation of all buildings and a return of all vehicles to their bases.

### 8.2.6 Assess Plans

**Assess plans** implementation is the DB&T of the plan-assessment phase. Planning without assessment provides no feedback on the value of a planner. In other words, a planner's decisions are only as good as real results indicate, which the assessor takes into account. Three of the four processes are already complete as described in Subsection 6.2.3, indicating that processes 3.1, 3.2, and 3.3 are duplicates of processes 2.1, 2.2, and 4.1, with minor modifications, respectively. At this point, the fourth and only process not to be implemented is process 3.4, the *assess expectations* function. This function has three sub-processes identified in Table 6.4 (3.4.1-3.4.3). The first of these three processes is *predict new expectations* function (3.4.1: $K_Q$). This function operates the same as the *forecast expectations* function of plan-generation (1.3: $K_Q$) described earlier.

The second of the three sub-processes is the *compare expectations* function (3.4.2: $E_Q$). As described above, expectations are a positive points value based on completed building visits (people rescued). If original expectations multiplied by expectation acceptance threshold ($r_Q$) is greater than or equal to the new expectations (i.e., $Q \bullet r_Q \geq Q''$), then the compare expectations function outputs the new expectations, otherwise the output is set to zero (i.e., $Q = E_Q(Q, Q'', r_Q)$). As in RISK, the third and final sub-process of *assess expectations* (3.4) is the *initiate re-planning* function (3.4.3: $P = I_R(P'', Q)$). The new plans are equal to the plan-remainder if the expectations are greater than zero ($Q > 0$); otherwise, the plans are reset to null. This operates the same as in RISK, except that expectations here are a point system and for RISK, it is the predicted number of conquered territories.

### 8.2.7 Core Planning Cycle

Planning without execution and assessment provides no feedback on the value of a planner. In other words, a planner's decisions are only as good as the real results indicate. To incorporate execution and assessment in the planning process, the model was expanded into three modular processes that handle planning differently. Figure 6.6 illustrates these cyclical processes. The **core planning cycle** implementation is the coordination of three modular processes plan-generation, plan-execution, and plan-assessment. Plans, models and expectations are objects shared within the planning cycle. Plan-generation contains the interface to the planner, where the user can choose amongst the available parameters to generate plans. Plan-execution and plan-assessment are implemented explicitly and interdependent of the planner. Each module contains its own

plans, models and/or expectations, and each module shares two of the three objects with one another as described in Figure 6.6. All of three pieces have been implemented at this point. The build and test challenge here is to share and pass the plans, model updates and expectations within the cycle.

### 8.2.8 A Single Game or Mission

**A single game or mission** simulation is not considered complete until all buildings are cleared in the real 'building content' models and all 20,000 stranded people are delivered to bases (Figure 8.3). To ensure all buildings can be searched and cleared, at least one rescue helicopter is used to rescue people from land islands created by the flooding, where boats and busses cannot reach. Also, the observation models are updated frequently (here about every 20 minutes real-world time), so that resources are not sent to same buildings again and again. Lastly, busses can be trapped in the environment, because shortest path return routes are flooded in the real model. In this case, busses are automatically planned to retrace steps back to base when full of passengers or low on fuel. When properly implemented, the build and test of these additional features make sure that each simulation runs to completion autonomously.

### 8.2.9 A Complete Tournament

Implementing **a complete tournament** requires first implementing tournament rounds, such as described in Section 8.1.9. This methodology was handcrafted to fit the problem. Many other approaches could be considered, however Section 8.3.3 shows that this approach can learn a good ADP&E strategy for an US&RO mission application.

Chapter 10: Discussion and Future Work will discuss other possible methods for solving similar problems.

## 8.3 US&RO Results

Results are shown in three areas: (1) plan-generation, (2) the integration of plan-execution and plan-evaluation into a planning cycle, and (3) the complete orchestration of tournament play for the training of US&RO planners. The results are in chronological order of development, and built hierarchically from the bottom up. First, plan-generation results will concentrate on the modeling and simulation tradeoffs in choosing actions for plans. Second, the integration of the planning cycle will demonstrate the tradeoff of the cost of planning with the need to update the planning model with new observations and possibly re-plan. Finally, training of US&RO planners will demonstrate an approach for improving planners through an autonomous offline tournament competition. Sections 8.3.1, 8.3.2 and 8.3.3 will describe the results for plan-generation, planning-cycle, and tournament play, respectively.

### 8.3.1 Plan-Generation

There are four main features described below that went into implementing the US&RO model. The best way to show the impact of considering these features is through example results. For simplicity, this example assumes there are no unsupplied or injured present in the city model and all people are supplied. Each feature is exemplified below by showing results that pertain to that feature. Results are based on playing out sixty games using a planner that picks random actions for each plan. If an agent plan

encounters conditions that prevent its completion, a random, feasible, extension to the plan is provided.

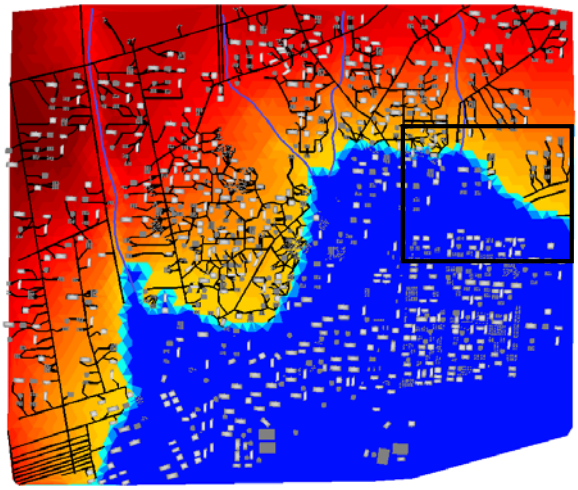

Figure 8.10a. City Under Maximum Flood Conditions

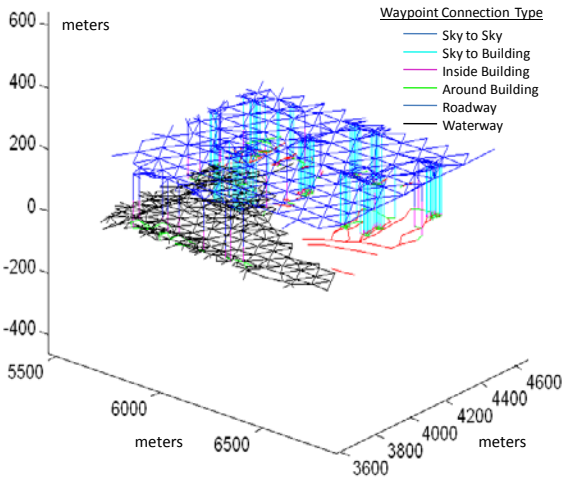


Figure 8.10b. Waypoint Connections: Three-Dimensional Cross Section

### 8.3.1.1 Feature One: Compact and Computable

To better understand the order of magnitude of this US&RO problem, Figure 8.3 shows a top-down picture of the terrain model used. The city is approximately 4 km x 5 km. Normally, processing and computation is done over the terrain mapped in Figure 8.3, instead of the waypoints and connections derived from the CTDB data and illustrated in Figures 8.4, 8.5 and 8.6. Using waypoints and connections reduces the complexity of the space substantially by eliminating the waste of computing over every cubic meter of ground, water, and airspace, and instead, computing action paths based on connected waypoint types. Within these 20 square km, there are 20,000 stranded people, twenty agents of various types, an initially unknown water level, roadblocks, building locations where people require rescue, and bases to drop off people. Specifically, there are 3649 buildings with over 12,758 floor locations, 2135 bus stops, 902 boat stops, four upstream

boat bases, and a bus and helicopter base. There are 31,115 total waypoints for this application, instead of millions of cubic meters to compute on.

A cross-sectional view of a portion of the city is shown in Figure 8.10b. This view shows all waypoints and connections for a small grid section of the city as shown by the small square section outlined in Figure 8.10a. Sky waypoints are shown on top and building connections, roads and waterways are shown below.

### 8.3.1.2 Feature Two: Very Fast Simulation

Due to the shortest path lookup tables described in Section 8.1.1.1, single actions for a plan are not waypoint to waypoint, but from origin to destination. This feature allows for very fast simulation speeds. The computation times are results from software developed in Matlab® particularly for this simulation, and are performed on a MacBook Pro running Matlab in a Microsoft Windows XP BootCamp environment. Complete US&RO simulations were run in an average of 7.66 minutes computer time, corresponding to 5525 minutes on average of real-world simulation time. That is a speed up of over 700 times.

### 8.3.1.3 Feature Three: Environment Observation Models

Over the course of one example game, Figure 8.11 shows the changes in the observation models. The figure shows the accumulation of route information via both land and water, and the proportion of buildings searched and cleared, and people rescued over time.

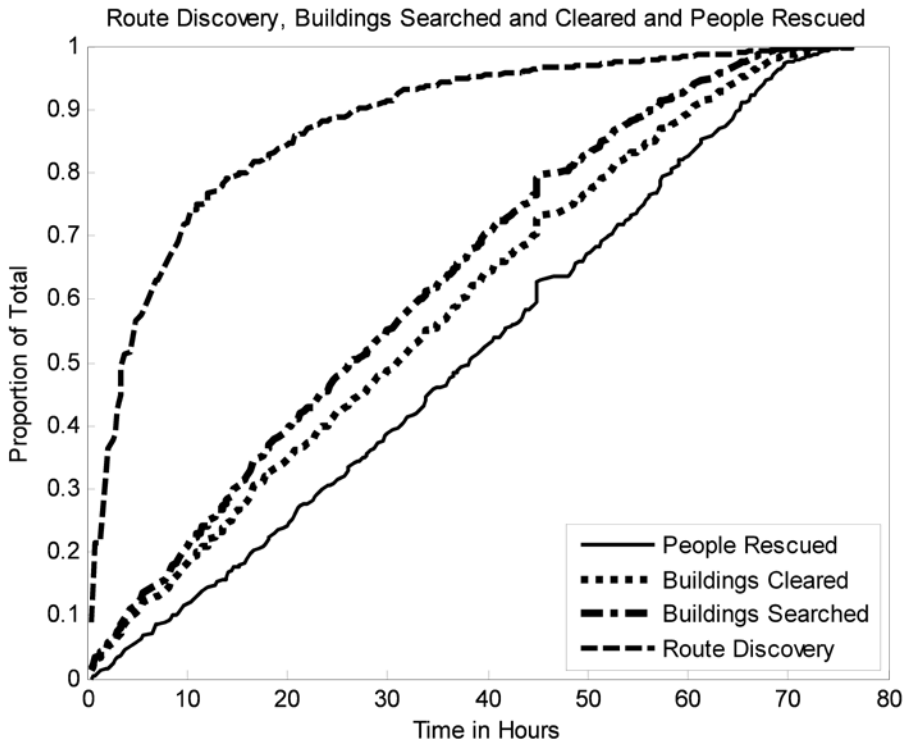


Figure 8.11. Observation Models

### 8.3.1.4 Feature Four: Multi-Agent Planning

Over the course of sixty example games, there were twenty agents used in each game of various types. Over the course of an entire game, agents had plans of average length equal to about sixteen thousand visited waypoints. That is over nineteen million traversed waypoints for each of the twenty vehicles in sixty games.

## 8.3.2 Planning Cycle

The planning cycle includes all three phases of planning: plan-generation, plan-execution, and plan-assessment. The inclusion of plan-execution and plan-assessment provides the opportunity to include tradeoffs among the cost of planning, length of plans, frequency of model updates, and when to re-plan. Re-planning is based on how well a

plan is executing and how the model updates are affecting predicted expectations. These four areas were tested to determine a tradeoff region that "makes sense" prior to the training of planners as shown in Section 8.3.3. These areas could have been learned as well, and Chapter 10: Discussions and Future Work will address this issue.

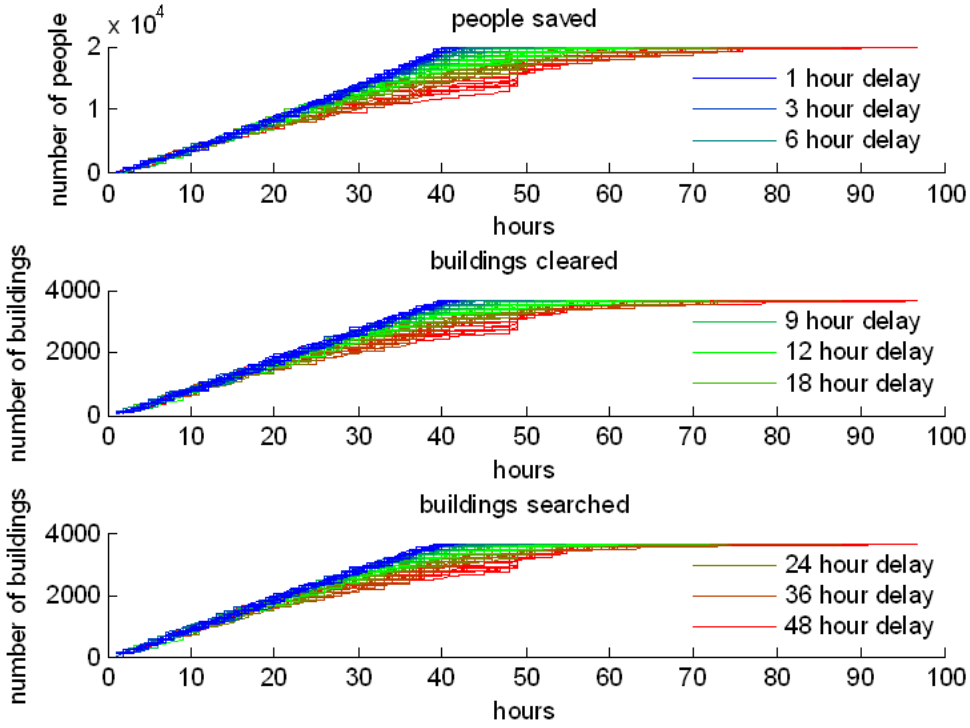


Figure 8.12. Model Update Frequency Variability Test Results

The cost of planning is defined here as how much time a planner needs to make a new plan for a vehicle or agent. The length of a plan is measured as roundtrips per each vehicle. To keep a standard measure for these four areas of tradeoff, the frequency of model updates is also measured on a roundtrip basis. When to re-plan is based on how well the expectations (described in Section 8.1.2) are predicted for each plan of each vehicle. Re-planning of each vehicle is considered independent of other vehicles, and

thus is assumed to be multi-threaded. In other words, vehicles can be tasked simultaneously, and thus not delay or interfere with the tasking of other vehicles.

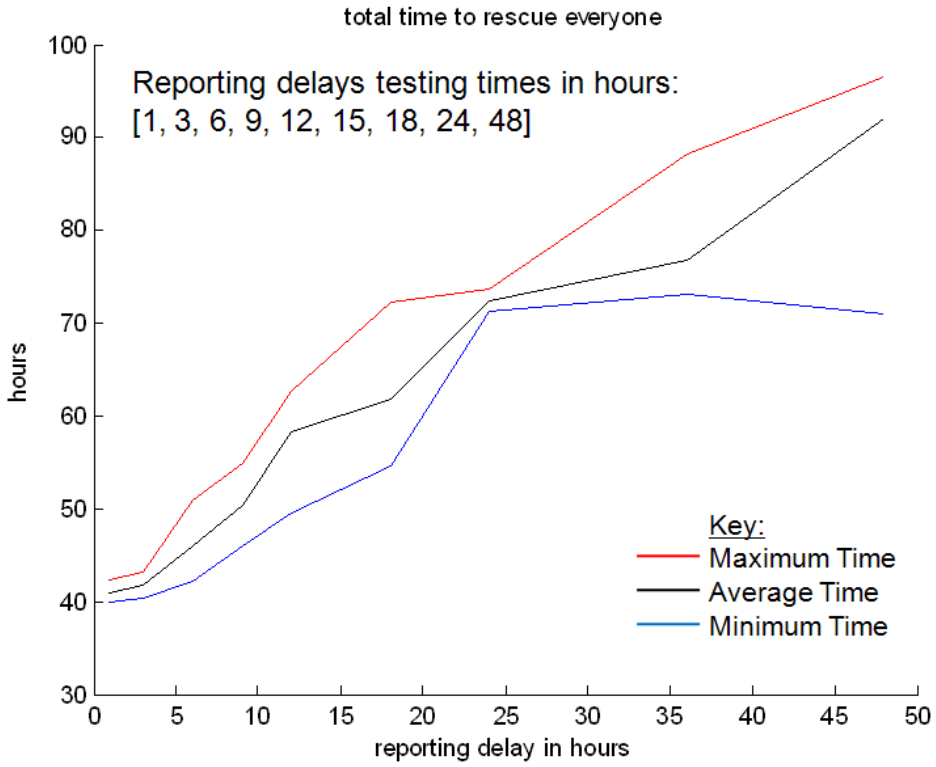


Figure 8.13. Summary of Model Update Frequency Variability Test Results

The first measure tested was the frequency of model updates. As described in Section 8.3.1, the same random tasking was done for twenty vehicles with differing model update frequencies. In this case, update frequencies were tested for ten cases for 1, 3, 6, 9, 12, 15, 18, 24, and 48 hours. Figure 8.12 shows the plots of all nine update frequencies for all ten cases of each. The graph at the top indicates the number of people saved as a function of time, the second graph shows the buildings cleared as a function of time, and the third graph shows the buildings searched as a function of time. Note that all plots look very similar with the longer model updates leading to longer times to evacuate. Figure 8.13 provides a summary graph that shows the extremes of each model update

frequency. As expected, the more often the model is updated the more efficient the evacuation. The savings in evacuation time was 66% between 1 and 48 hour reporting delays. Therefore, a very fast model update was chosen. To be consistent with using vehicle roundtrips as a time marker, the update frequency was assigned to be when every vehicle reached base, which turns out to be around every 20 minutes.

The cost of planning was tested in two cases: 20 minutes and 10 minutes per vehicle. Both costs were tested by training planners as described in Section 8.1.9 and 8.1.10. The 20 minute case is shown in Figure 8.14, and the 10 minute case is used in the training example shown in Section 8.3.3 and discussed there. In the 20 minute case at the $14^{th}$ round and beyond the expectation acceptance threshold drops to zero indicating that the time to plan is too costly for any re-planning of any vehicle. Therefore, 20 minutes was considered too high a cost to re-planning a new vehicle, since it is better to keep a vehicle active in a bad plan than to have it wait 20 minutes before returning to action. As will be shown in Section 8.3.3, having only a 10-minute delay in re-planning is far more effective, because a non-zero value is converged upon when considering re-planning.

The length of a plan is measured in round trips by each vehicle or agent. As shown in Figure 8.14 above, if the cost of planning is estimated at twenty minutes (or ten minutes), scheduling only one or two actions (stop or building visits) ahead at that cost is meaningless. Also, as shown in Table 8.4, the communication delay for updating the planning model for each vehicle is five minutes. If each vehicle calls in after every few actions the delays become too accumulative for the vehicle to be efficient. Therefore, a roundtrip seemed the most logical choice in choosing plan length as a minimum length.

To see what constraint should be placed on a maximum plan length, two experiments were conducted using a plan length of one roundtrip and another using a plan length of two roundtrips. The re-planning of the plans with two roundtrips was far more likely to re-plan given the new observations in the environment over that length of time. Thus, using two roundtrips was considered too costly given the re-planning costs were more than twice those of having a single roundtrip plan.

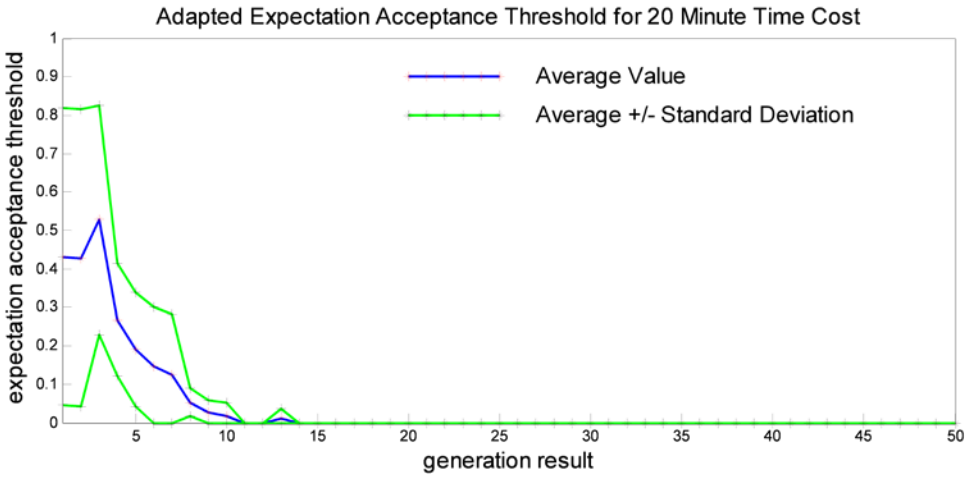


Figure 8.14. Training Expectations Acceptance Threshold Coefficient

### 8.3.3 Training US&RO Planners

Rounds and tournaments for training US&RO planners are described in Sections 8.1.9 and 8.1.10, respectively. Results are illustrated in five areas: (1) expectation acceptance threshold, (2) number of vehicle types, (3) decision feature weights, (4) decision feature beta density functions, and (5) evacuation progress in saving lives. All results are for 100 generations of tournament play.

Expectation acceptance threshold is used to trade off time to plan against expected outcomes. Time to plan was fixed at ten (predicted) minutes per plan per agent.

Expectations were based on a point system where an agent was given three points per injured person rescued, two points per unsupplied person, one point per supplied person and zero points per dead person rescued. The expectation acceptance threshold is shown in Figure 8.15 in the top graph. The final converged value is around 0.2, thus, if a plan achieved 20% or more of its expected points, re-planning was not initiated. In another experiment described in Section 8.3.2, a 20-minute penalty was given for generating each plan. This resulted in a zero expectation acceptance threshold, meaning the time to plan was too costly to ever re-plan unless necessary.

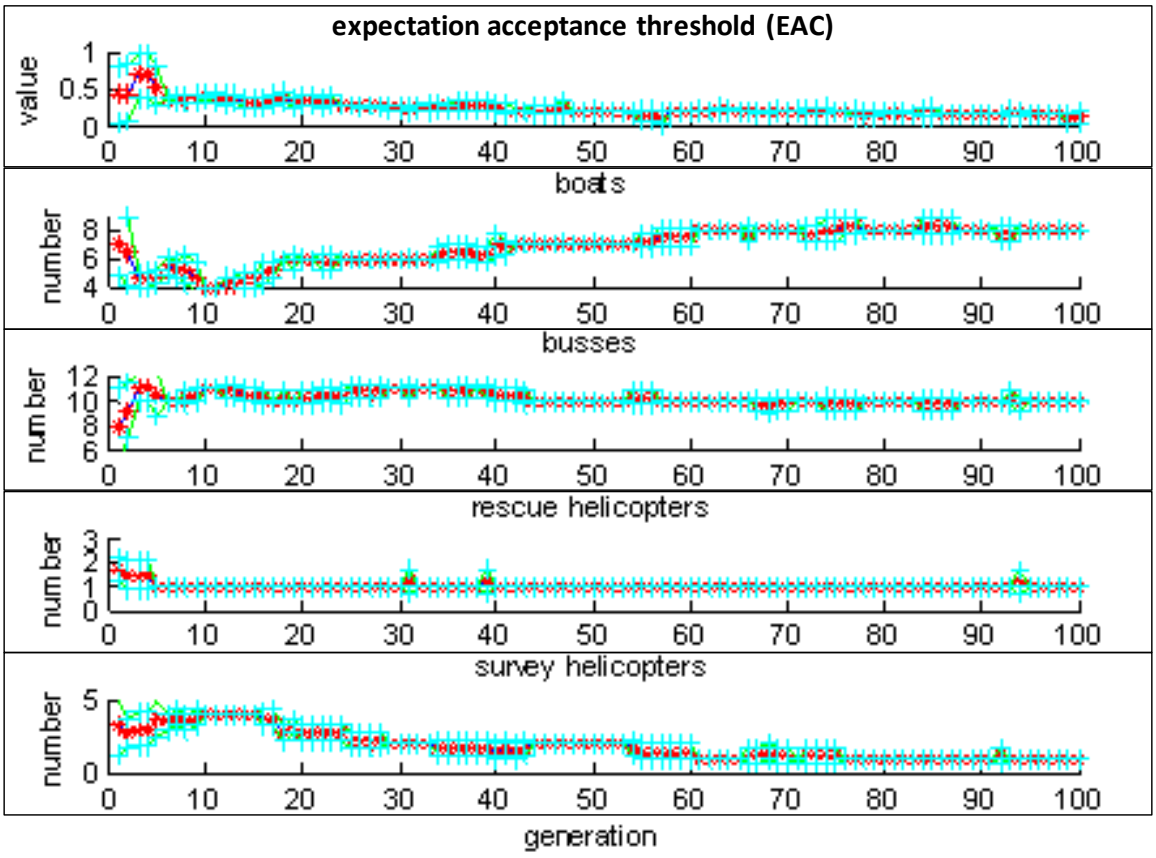


Figure 8.15. Expectation Acceptance Threshold and Number of Vehicle Types

The number of vehicle types is shown in Figure 8.15 located in the four lower graphs. In later generations, boats number around eight, busses number around ten, rescue helicopters number around one, and survey helicopters number around one.

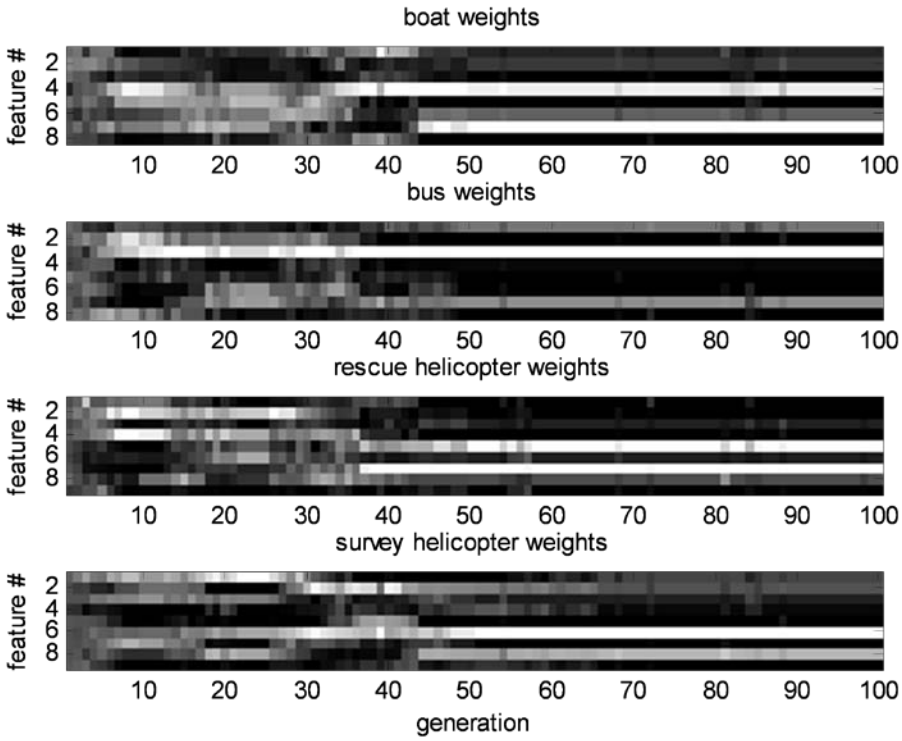


Figure 8.16. Decision Feature Weights

Decision feature weights are shown in Figures 8.16 and 8.17. Figure 8.16 presents the change in weights over tournament generations. In all image figures (8.16, 8.18, 8.19, 8.20, and 8.21), the lighter shade indicates a higher value. Figure 8.17 illustrates the final weights observed in generation one hundred. The three black bars indicate the three largest weights. The feature numbers on the left of Figure 8.16 and on the bottom of Figure 8.17 match those displayed in Table 8.1.

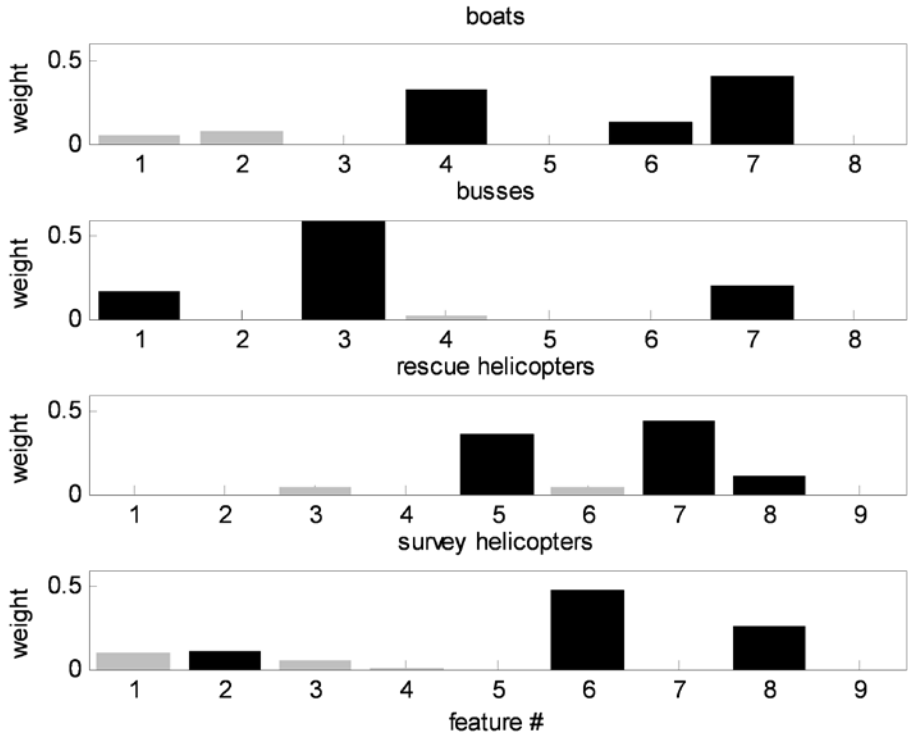


Figure 8.17. Final Decision Feature Weights

The decision feature beta density functions shown in Figures 8.18, 8.19, 8.20, 8.21 and 8.22 are weighted proportional to the black bars in Figure 8.17. Figures 8.18 through 8.21 shows the top weighted feature density functions across all forty rounds, while Figure 8.22 shows the final round's top feature weighted density functions. Note that sometimes the shading goes dark in Figures 8.18 through 8.21. This indicates that the weight was low for that round (Figure 8.16) and had little influence. Figure 8.22 provides a good summary of influential features. Each graph directly corresponds to a black bar in Figure 8.17.

The feature densities for boats are in the top row of graphs in Figure 8.22. All three graphs indicate that boats are faster at searching buildings than helicopters, so they

are sent to buildings that have unknown number of people, unknown number of unsupplied people, and where few people have been spotted, respectively.

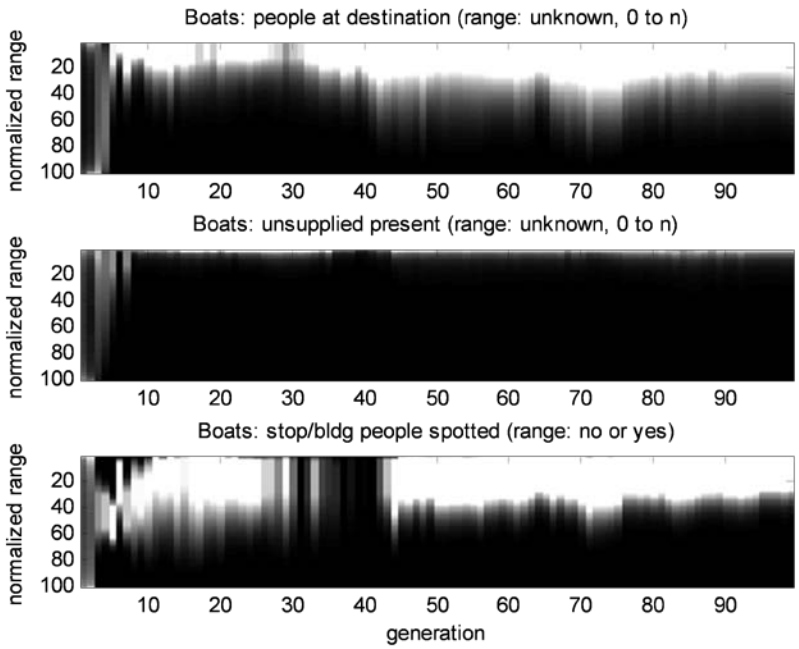


Figure 8.18. Boat Features Weighted Beta Densities

The feature densities for busses are in the second row of graphs in Figure 8.22. The first graph indicates that busses prefer to travel short distances between stops and the second graph strongly indicates a need to choose actions near other busses. Since busses encounter many roadblocks due to trees and water, staying near other busses increases the chances of finding an open path. The third graph in row two slightly indicate an additional preference to go where few people have been spotted.

The feature densities for rescue helicopters are in the third row of graphs in Figure 8.22. The three graphs indicate a preference to go to buildings where people are injured, where no one has been spotted, and where people still have supply time remaining.

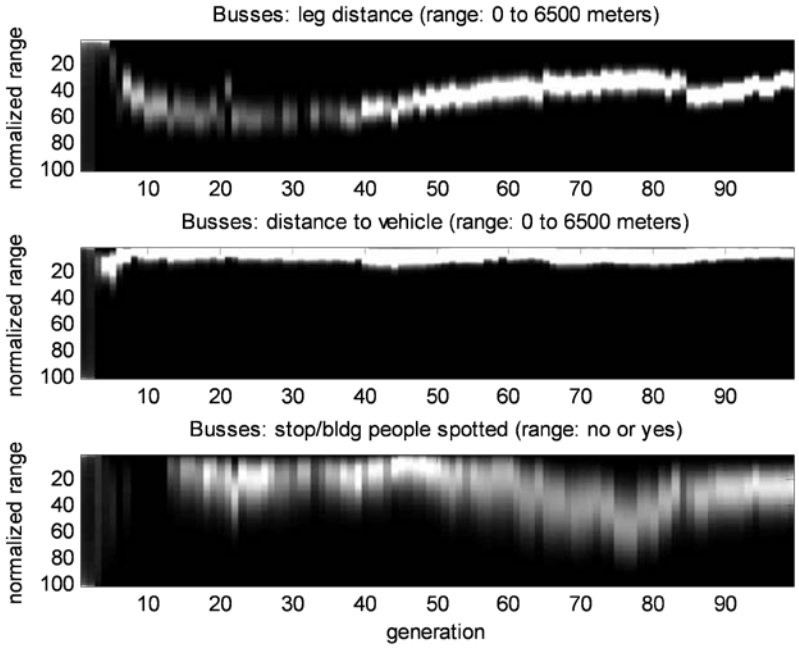


Figure 8.19. Bus Features Weighted Beta Densities

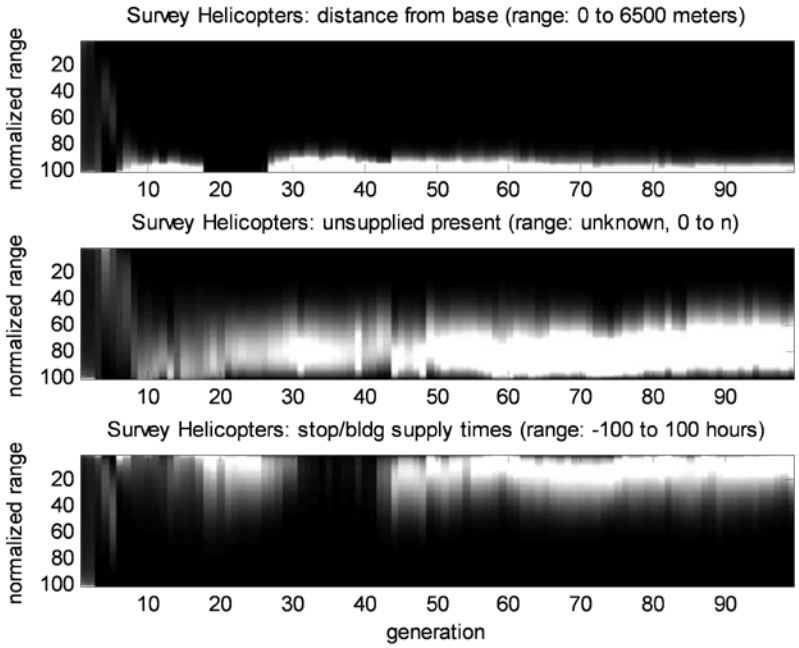


Figure 8.20. Rescue Copter Features Weighted Beta Densities

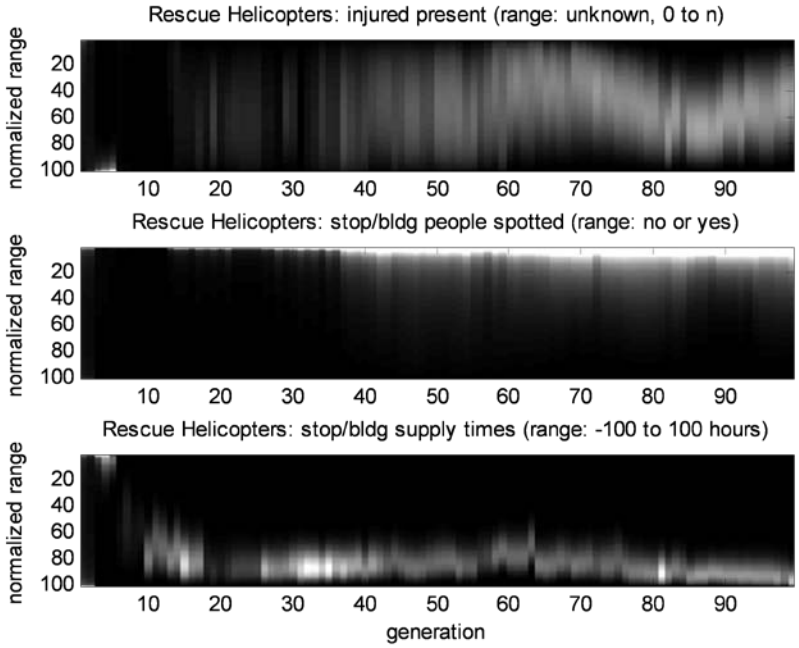


Figure 8.21. Survey Copter Features Weighted Beta Densities

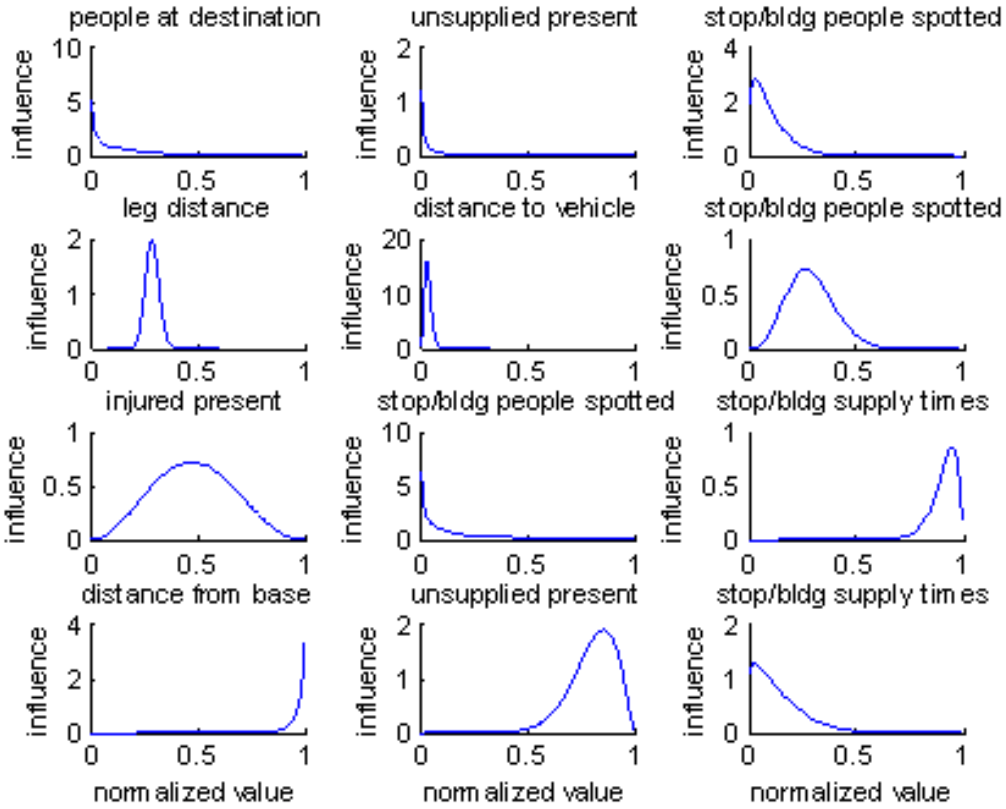


Figure 8.22. Largest Feature Weighted Beta Densities

The feature densities for survey helicopters are in the final row of graphs in Figure 8.22. The three graphs indicate a preference to go far from the base, to go where people require supplies, and to go where people have little survival time left. Since the busses are fast and cannot go far from the base due to water levels it makes a lot of sense to first supply people far from the base. Also, getting people supplies in time is critical to their survival.

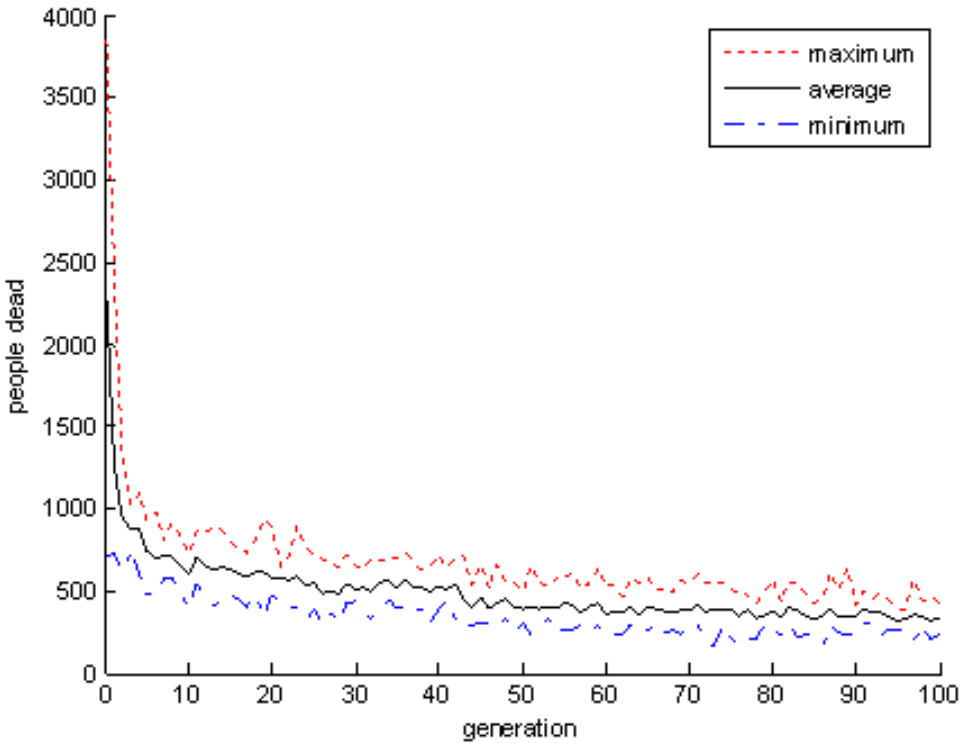


Figure 8.23. Evacuation Progress of Top Four Performers

The evacuation progress of the top performers is shown in Figure 8.23. Evacuation effectiveness improves drastically over the first few generations and then slowly after that.

The text in Chapter 8 contains material previously presented in three publications: Lecture Notes in Computer Science: IEEE World Congress on Computational

Intelligence, IASTED Applied Modeling and Simulation Conference, and Modelling, Simulation and Optimization (In Tech Publishing, Chapter 25). I was the primary researcher and author, and the coauthor directed, supervised and/or participated in the research, which forms the basis of the chapter.

**References:**


1. T. Witte, “A Survey of 3-D urban mapping and visualization capabilities from an army perspective,” *Proceedings of the ISPRS Joint Conference, V. XXXVI,* March, 2005.

2. M. Isenburg, Y. Liu, J. Shewchuk and J. Snoeyink, “Streaming computation of Delaunay triangulations,” *ACM Transactions on Graphics 25(3),* July 2006, pp. 1049-1056.

3. M. Evans, N. Hastings and B. Peacock. “Statistical Distributions” - Second Edition. *John Wiley and Sons, Inc.* 1993.

4. N. Metropolis; S. Ulam. "The Monte Carlo Method". *Journal of the American Statistical Association* **44** (247). 1949: pp. 335–34.

5. S. Nadarajah, S. Kotz, “Multitude of beta distributions with applications”, *Statistics: A Journal of Theoretical and Applied Statistics*, Volume 41, Issue 2 April 2007, Pages 153-179.

6. K. M. Chandy, J. Mistra, “Distribued computation on graphs: shortest path algorithms,” *Communications of the ACM, V. 25, No. 11,* November, 1982, pp. 833-837.

# Chapter 9

# Design and Implementation of Future Applications

## 9.0 Abstract

Combining the general approach described in Chapter 6 with specific application details described in Chapters 8 and 9 provides an opportunity to examine what it takes to build future planning applications. Chapter 6 provides a detailed description of the general goal-directed architecture, while Chapters 7 and 8 help define all component variables, functions, and parameters for specific ADP&E applications. This chapter is intended to join the lessons learned from Chapters 6 through 8 into a concise and more abstract view of the ADP&E system. This chapter will provide more engineering than science, because building future systems depends more on standardizing the approach than pursuing future modifications. Future extensions will be discussed in Chapter 10: Discussion and Future Work.

The complexity of the problem domain and solution space warrants a detailed description for designing and implementing future applications. This process is described in three incremental steps: (1) define and design classes, and build and instantiate all interacting objects for each hierarchical level from missions to observations (i.e., levels 3 to 10), (2) describe the nine step design, build and test (DB&T) strategy, which includes the step-by-step implementation of the ADP&E architecture with corresponding

functional components, and (3) summarize the inclusion of all functional components with corresponding attributes.

## 9.1 Classes and Objects

All hierarchical levels, except tournaments ($\Omega$) and rounds ($\Theta$), require in-depth knowledge of the application domain to properly define and design classes, and build and instantiate objects [1]. This section describes the classes and objects required at levels three through ten in the hierarchy. The following description includes the definitions of each hierarchical level in terms of classes and objects. Also included are class and object examples for each level, and from each of the two previously implemented applications described in Chapters 7 and 8. Finally, for each level including classes and objects, there is a short description of the choices available for future applications.

Hierarchical level three is missions or games ($G$). A mission or game requires an environment, which includes the rules of the game, a feature space with all physical and temporal constraints, and the required number of planners (or players) and their types (e.g., cooperative or non-cooperative). The mission also requires completion goals or termination criteria to know when to stop a mission or game. For the RISK game, the goal is to conquer the world by occupying all 42 territories. In addition to 42 territories, the world environment also includes 166 battle possibilities, 44 cards, and an unlimited number of possible battling forces [2]. For the US&RO mission, the goal is to evacuate a severely flooded city with 20,000 stranded people. The environment includes bases, waterways, roadways, skyways, buildings, people, and vehicles. There are over 31,000 waypoints, over 4,000 buildings, 20,000 people, and 20 vehicles [3]. Therefore, future

applications can support a complex mission or game in very large, uncertain environments and can handle multiple planners of various types.

Hierarchical level four is the planners ($\Psi$) or players. The planners are the organizers or orchestrators of the plans. The planners control and coordinate all the agents from a centralized planning perspective. The planner can choose the number of agents, agent types and agent grouping in planning tasks. For the RISK game, there are 2 to 8 players (planners) that are non-cooperative, and the game is turn-based. Each turn is rich with many possible moves, such as allocating forces, battling forces, moving forces and turning in cards. For the US&RO mission, there is only a single cooperative planner in a partially observable environment. The planner controls and coordinates vehicle resources simultaneously to evacuate all 20,000 stranded persons. Future applications can be built to support multiple planners in cooperative and non-cooperative environments, and support many parameters in many areas (e.g., search, selection, reward, risk) for versatility in planning.

Hierarchical level five is the agents ($\Gamma$) or resources. The agents are the actors used to implement plans of actions. Agents can be of different types, of different number, have a variety of actuators and sensors, be resource constrained, can have multiple contingency plans or a single plan, and can have a varied plan length depending on resources or task completions. For the RISK game, there are two types of resources: cards ranging from zero to eleven in number, and forces ranging from zero to hundreds in number. Card agents are considered trivial due to their strict rules of the game, but force agents are extremely important given they can be allocated, battled and moved. For the

US&RO mission, twenty vehicle agents are used of four different types: (1) busses, (2) boats, (3) rescue helicopters, and (4) survey helicopters. All busses, boats and helicopters have an appropriate set of actuators and sensors (as shown in the tables in Chapter 8). Future applications can support variable numbers of agent classes and agent type classes, and instantiated agents can vary greatly in role (actuators and sensors) and number.

Hierarchical level six is the plans (*P*) or courses of actions. The plans are the recipes used to control and coordinate agent resources. Plans are generated using search and plan selection, and encapsulate who, what, where, when, why and how at each action step in a plan. The who represents the agents under consideration, the what represents the action, where represents the action locations, when represents the timestamp or action order, why represents the reward change (can be a reward range for multiple outcomes) based on taking that action, and how represents the actuators and sensors involved and the resources expended. Plans take computational resources to generate, so the number and length of plans generated vary with application. For the RISK game, plan lengths are constrained to a single player's turn, and are resource constrained based on expending only a portion of force agents in each plan. In searching for plans, a RISK player generates multiple plans to compare and a variable number of completed tasks per plan. For the US&RO mission, plan length is restricted to a single roundtrip of all vehicles (agents). Within each plan, rescuing people is considered as a variable degree of task completions. Future applications can support variable length plans, and include who, what, when, where, why and how for each contemplated action. Plans can support a variable number of agents, number of tasks, plan length, and amount of resources consumed.

Hierarchical level seven is the tasks ( *T*) or expected plan results. Tasks are measured as expectations with associated success probabilities. For the RISK game, a task is considered as the successful conquering of a territory. These tasks carry a probability of success value, and an acceptance threshold to trigger re-planning if probability falls below this trigger value. For the US&RO mission, a task is considered in terms of the number of surviving people rescued in a vehicle roundtrip. Tasks are measured based on this number of people rescued and their type: injured, unsupplied and supplied. Future applications will use task classes as a measure for progress when measuring plan success probabilities. Measures of progress can include completed objectives (e.g., conquering territory), a probability of success, risk, and reward (e.g., point system for rescuing people).

Hierarchical level eight is the actions (*A*) or available options. Actions are considered as all options available for all agents including their actuators and sensors. Note that this list of options can change depending on the projected state of the environment or planner under consideration. For the RISK game, actions are turning-in cards, reinforcing territories, and battling and moving forces. During plan generation, each action is chosen randomly to avoid a biased plan search. However, action choice can also be learned in future versions. For large combinatorial problems addressed here, contemplating actions carries a temporal cost and is restricted in number, so that search does not run on indefinitely. For the US&RO mission, actions are travel to/from buildings, search buildings, rescue people, drop supplies, and communicate observations to the planner. Actions are chosen by feature conditions that are adjustably weighted, and each action has a delayed cost based on vehicle type, people involved, and distance

traveled. In future applications, actions can include temporal costs of contemplation and execution. Actions are options that can be searched individually or as sets within multiple plans.

Table 9.1. Implementation Tasks for Hierarchy Levels Three through Ten

| Hierarchy Level | Implement-ation Tasks | Application Domain | Brief Description |
|---|---|---|---|
| 3. Missions (main goal) | Design/ Define Classes | RISK Game | conquer world by occupying all 42 territories |
| | | US&RO Mission | evacuate a city that is severely flooded (20,000 people) |
| | | Future Applications | can support a complex mission or game in a very large uncertain environment |
| | Build/ Instantiate Objects | RISK Game | has 42 territories, 166 battle possibilities, 44 cards, unrestrained number of forces |
| | | US&RO Mission | supports over 31,000 waypoints, over 4,000 buildings, 20,000 people, 20 vehicles |
| | | Future Applications | handles requirements for very large uncertain environments with multiple planners |
| 4. Planners (orches-trators) | Design/ Define Classes | RISK Game | 2 to 8 players in this non-cooperative turn-based game |
| | | US&RO Mission | a single planner works in a cooperative environment |
| | | Future Applications | can support multiple planners in cooperative or non-cooperative environments |
| | Build/ Instantiate Objects | RISK Game | each turn is rich with many possible moves in an attempt to win the game |
| | | US&RO Mission | planner controls and coordinates resources simultaneously to meet mission goals |
| | | Future Applications | supports many parameters (e.g., search, selection, reward, risk) for versatility |
| 5. Agents (actors) | Design/ Define Classes | RISK Game | (1) cards ranging in number from 0 to 11, (2) forces ranging in number 0 to 100s |
| | | US&RO Mission | twenty total: (1) busses, (2) boats, (3) rescue helicopters, (4) supply helicopters |
| | | Future Applications | can support variable number of agents and agent types |
| | Build/ Instantiate Objects | RISK Game | card agents are trivial, but force agents can be reinforced, battled, and moved |
| | | US&RO Mission | busses, boats, and helicopters have an adjustable set of actuators and sensors |
| | | Future Applications | agents can vary greatly in role (actuators and sensors) and number |
| 6. Plans (recipes) | Design/ Define Classes | RISK Game | plan length is a single player's turn with resource constraints |
| | | US&RO Mission | plan length is a single roundtrip for all agents |
| | | Future Applications | can support variable length and includes who, what, when, where, how and why |
| | Build/ Instantiate Objects | RISK Game | search generates multiple plans to compare, and variable number of tasks per plan |
| | | US&RO Mission | each agent has a plan with a variable degree of task completions |
| | | Future Applications | plans can be variable with respect to agents, number of tasks, plan length, etc. |
| 7. Tasks (expecta-tions) | Design/ Define Classes | RISK Game | a task is a successful conquering of a territory |
| | | US&RO Mission | a task is considered in terms of the number of people rescued in a roundtrip |
| | | Future Applications | tasks are a measure progress to evaluate plan success |
| | Build/ Instantiate Objects | RISK Game | tasks carry a probability of success value, and an acceptance threshold as a trigger |
| | | US&RO Mission | tasks are measured based on people rescued: injured, unsupplied, etc. |
| | | Future Applications | tasks are a measure of progress: includes probability of success, risk, and reward |
| 8. Actions (ingred-ients) | Design/ Define Classes | RISK Game | actions are turning-in cards, reinforcing territories, battling and moving forces |
| | | US&RO Mission | actions are travel, search, rescue, supply, and communicate observations to planner |
| | | Future Applications | actions are options that can include temporal costs of contemplation and execution |
| | Build/ Instantiate Objects | RISK Game | each action is chosen randomly to avoid a biased plan search |
| | | US&RO Mission | actions are chosen by feature conditions that are adjustably weighted |
| | | Future Applications | actions are options that can be searched individually or as a set in multiple plans |
| 9. Out-comes (predic-tions) | Design/ Define Classes | RISK Game | multiple possible outcomes per action including probability for each |
| | | US&RO Mission | many possible outcomes due to partially observable environment |
| | | Future Applications | multiple outcomes per action is the reflection of an uncertain environment |
| | Build/ Instantiate Objects | RISK Game | outcomes are predicted by using a risk aversion coefficient, which accounts for risk |
| | | US&RO Mission | unknown outcomes are predicted based on whose missing (i.e., people) |
| | | Future Applications | unknown outcomes can be predicted in accordance to risk or due to known info |
| 10. Observ-ations (new Info) | Design/ Define Classes | RISK Game | can observe everything except cards held in opponent hands and future dice rolls |
| | | US&RO Mission | must sense roadways, waterways, and building contents |
| | | Future Applications | observations drive sensor activity to keep plans on track |
| | Build/ Instantiate Objects | RISK Game | observations are on dice rolls, and updated after each battle is complete at no cost |
| | | US&RO Mission | vehicles gather building content and route info, such as waterways by boat, etc. |
| | | Future Applications | observations are done in many ways and model update can incur temporal costs |

Hierarchical level nine is the outcomes (*Y*) or predicted effects. Outcomes are considered as all possible effects for each particular action, under the current or projected circumstances (state), and each particular set of agents under consideration. For the RISK game, every battle action can have a set of possible outcomes, including a probability for each outcome. Outcomes are predicted based on the forces in conflict and by using a risk aversion coefficient, which accounts for the risk the player is willing to take (described more fully in Chapter 7). For the US&RO mission, there are many possible outcomes due to the partially observable environment. Unknown outcomes are predicted based on the number of unsearched floors in all buildings and based on the people still missing (unaccounted for). Future applications can handle multiple outcomes per action as reflected in a stochastic environment, and unknown outcomes can be predicted in accordance to risk (e.g., risk aversion) or due to known information (e.g., missing persons).

Hierarchical level ten is the observations (*O*) or newly discovered environmental state information. Some real world state information may be unknown due to partial observability, and observations are used to discover this unknown information and update the planner's state to reflect this newly acquired knowledge. For the RISK game, everything can be observed except the cards held in each opponent's hand and future dice rolls. Observations on dice rolls are updated after each territory battle is complete and at no temporal cost to the planner. For the US&RO mission, agents must sense roadways, waterways, and building contents for missing people. Vehicles gather building content and route information through their sensors, such as waterways for boats and roadways for busses, and all routes can be partially viewed by helicopters. Information is passed on

to the planner with a temporal communication cost. Future applications can use observations to drive sensor activity to keep plans on track. Observations can be done in many ways, and model updates can incur temporal costs.

Table 9.1 summarizes all define and design classes, and build and instantiate objects for each hierarchical level. Note that the last column in Table 9.1 gives a brief summary description for all objects and classes in hierarchical levels 3 through 10 for both applications described in Chapters 7 and 8, and in a general form for future applications.

## 9.2 Design, Build and Test Implementation Procedure

As introduced in Chapter 6, the design, build and test (DB&T) implementation procedure is a nine-step process: (1) **execute arbitrary actions**; (2) **update planning-model**; (3) **choose and execute single actions**; (4) **generate arbitrary multi-action plans**; (5) **generate and choose plans**; (6) **assess plans**; (7) coordinating the full **core planning cycle**; (8) coordinating multiple planners for **a single mission or game**; and (9) orchestrating **a complete tournament** as a meta-learning strategy for training planners. In Chapter 6, the nine-step DB&T process was only described from the planning cycle perspective, and did not include the goal-directed training functions of ‘hierarchical control chain’ (HCC), nor the goal-directed training functions ‘learning and analysis’ (L&A). These training functions were introduced in Chapter 6. Here we describe the whole system with all 33 functional components. The functional components not fully described in Chapter 6 will be described fully here; otherwise refer to Chapter 6 for more details.

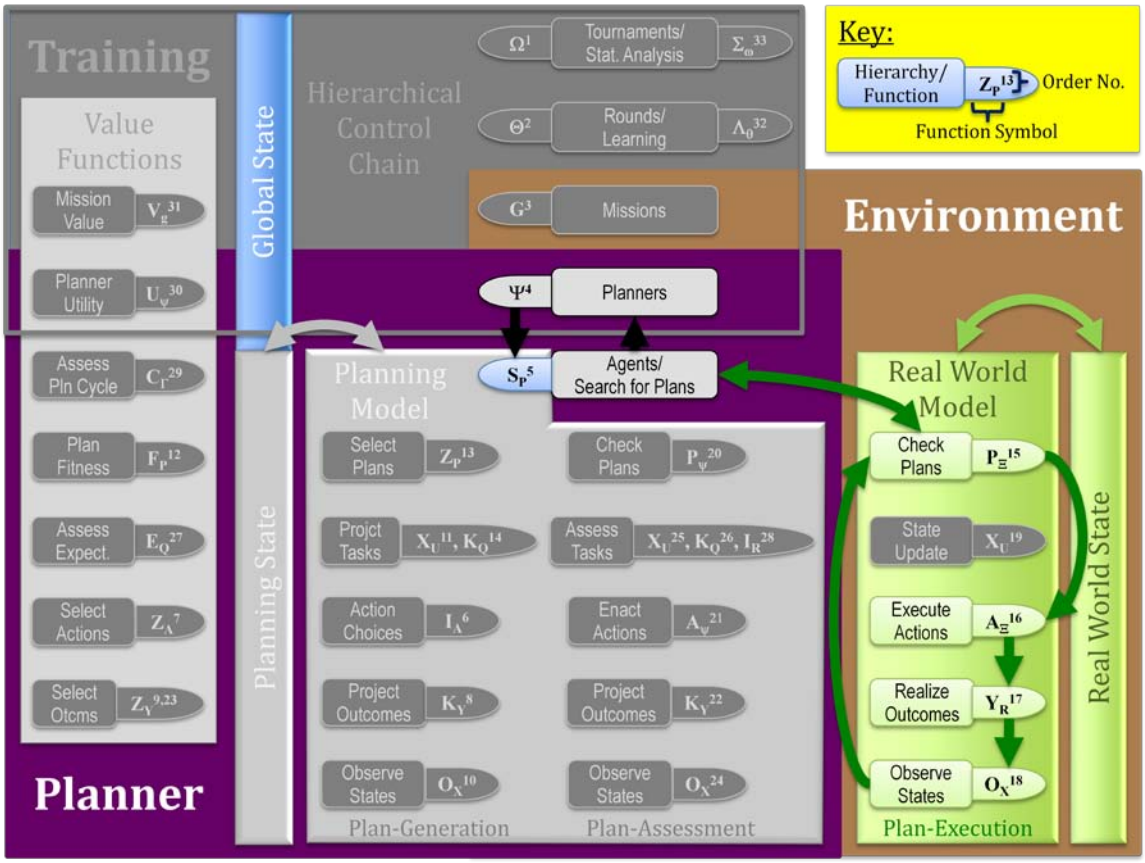


Figure 9.1. DB&T Step 1: Execute Arbitrary Actions

Step one of DB&T, **execute arbitrary actions**, is shown in Figure 9.1. As a first step, the global state is implemented to initiate the monitoring of the HCC structure. The only function in this structure is the fourth function, the *planner* function ($\Psi^4$). Only a single planner needs to be implemented at this time, unless the planners have completely different roles and thus different parameter sets. If the planners have completely different roles, then each planner can be designed separately. Thus, complete DB&T steps one through seven separately for each planner, and then combine these planners in step eight. In this first step, only a small part of the planner is implemented, the *search for plans* function ($S_P^5$). Only random or arbitrary actions need to be searched at this time, so this function is not implemented fully until step four. After implementing this simplified

*search for plans* function, four out of five of the plan-execution functions can be implemented. The *check plans are executable* ($P_{\Xi}^{15}$), *execute actions* ($A_{\Xi}^{16}$), *realize outcomes* ($Y_R^{17}$), and *observe states* ($O_X^{18}$) functions are implemented here. These five functions are described in detail in Chapter 6. When implemented, all five of these functions should be able to operate iteratively and indefinitely and in the order shown by the arrows in Figure 9.1.

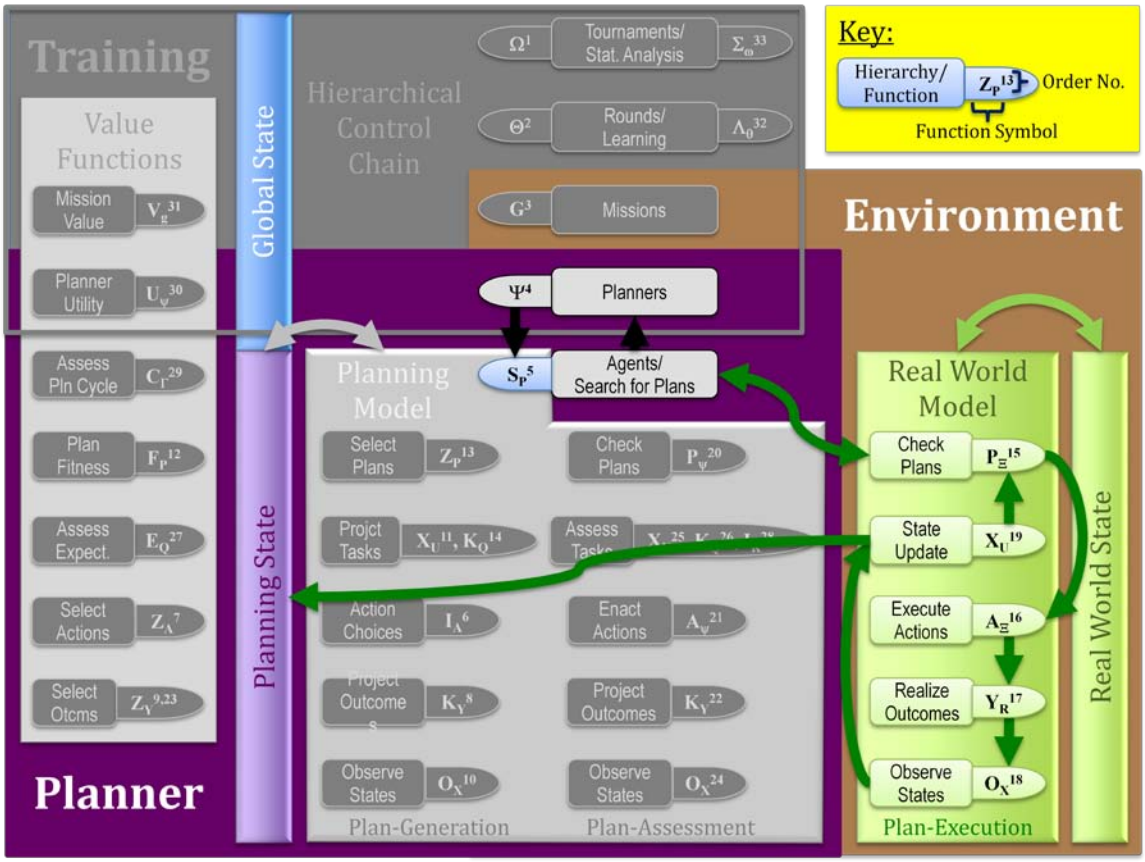


Figure 9.2. DB&T Step 2: Update Planning-Model

Step two of the DB&T, **update planning-model**, is shown in Figure 9.2. In this step, the planning state is implemented to receive new state values from observations of the real world state. These observations are channeled to the planning state through the *state update* function ($X_U^{19}$), the last required plan-execution function. Functions 5, and

15-18 are identical to those used in DB&T step one. State updates are normally completed at the beginning of the plan-assessment phase, so this implementation is used to only check that the planning state can be updated after each plan-execution cycle.

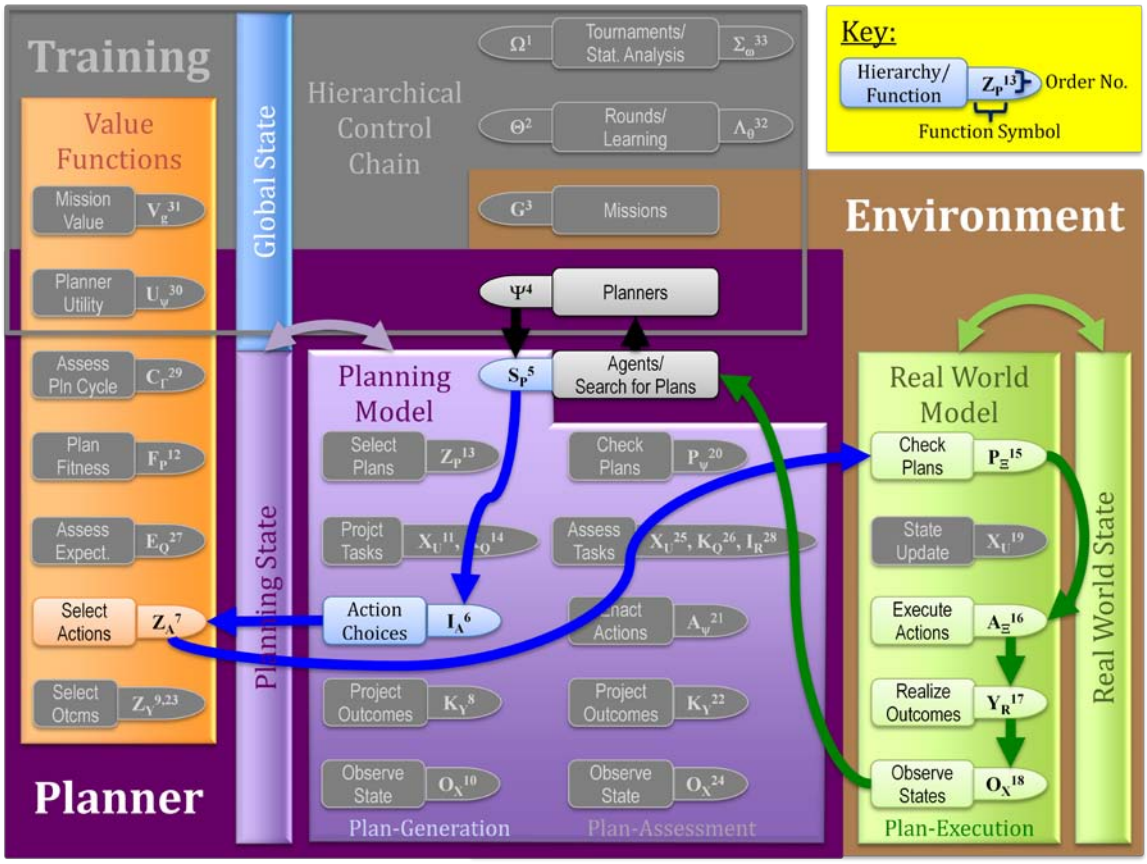


Figure 9.3. DB&T Step 3: Choose and Execute Single Actions

Step three of the DB&T, **choose and execute single actions**, is shown in Figure 9.3. This step adds onto the step one implementation by including two additional functions of plan generation: *determine action choices* ($I_A^6$), and *action selection* ($Z_A^7$). This is the first step that requires use of the planning model, which is a near duplicate of the real world model, and includes all rules of the mission or game (see Chapter 6 for more detail). The action choices are determined based on the agents under consideration and the rules of the game. The action selection function can select among these available

choices as the designer wishes (e.g., random in RISK game, weighted feature conditions in US&RO mission, etc.). The choices made are now driving the plan-execution and this process should be able to operate iteratively and indefinitely.

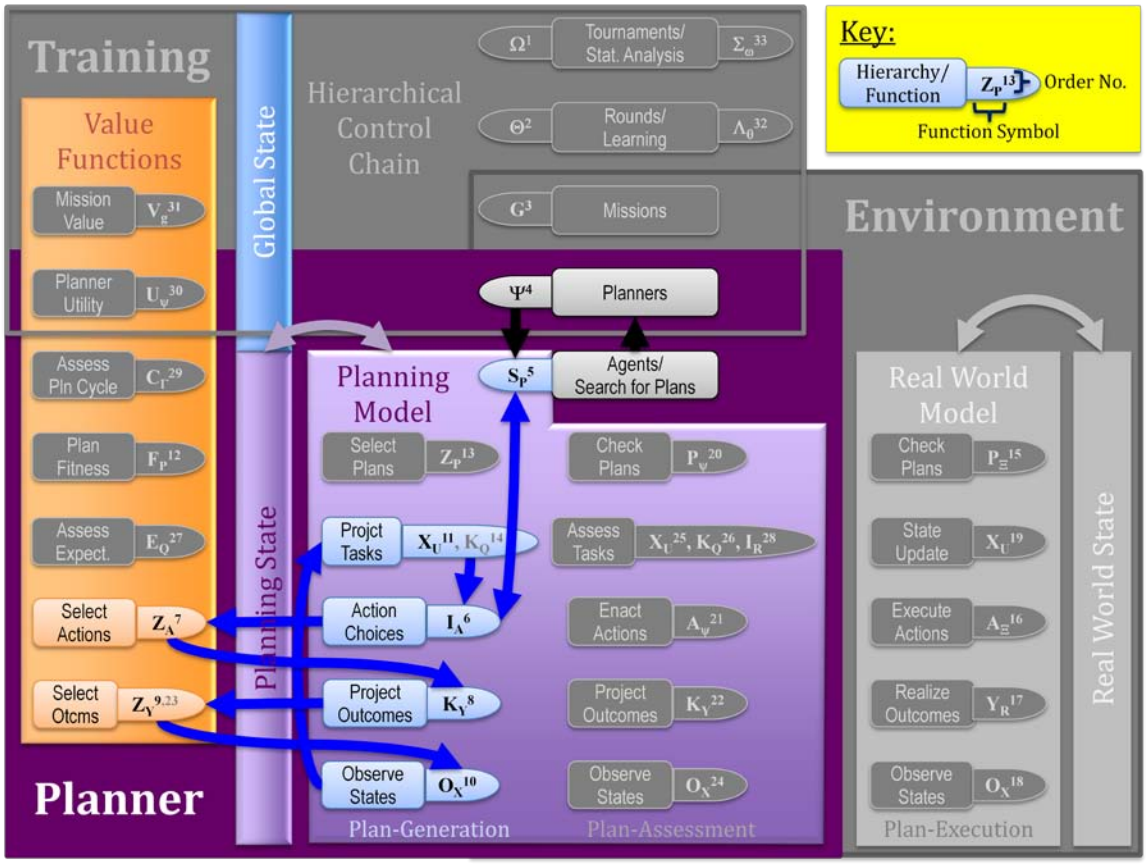


Figure 9.4. DB&T Step 4: Generate Arbitrary Multi-Action Plans

Step four of the DB&T, **generate arbitrary multi-action plans**, is shown in Figure 9.4. This step concentrates on implementing a major portion of plan-generation (functions 5-11) or seven of the first ten plan-generation functions: *search for plans* (introduced above), *determine action choices* (described above), *action selection* (described above), *project outcomes* ($K_Y^8$), *outcome selections* ($Z_Y^9$), *observe states* ($O_X^{10}$), and *state update* ($X_U^{11}$) functions. The *search for plans* functional implementation is upgraded to handling multiple plans for multiple agent groups. The

*determine action choices* and *action selection* can be the same as previously implemented, or upgraded as the application demands. Function 8, *project outcomes*, can be implemented as done in RISK game, where dynamic Bayesian networks project multiple action outcomes, or in a simpler manner, such as projecting a single outcome. Function 9, *outcome selection*, can be implemented with risk aversion as used in RISK, or in a simpler manner. Functions 10 and 11, *observe states* and *state update*, should perform similarly to those same functions used in plan-execution, shown in DB&T steps one and two, respectively. Note that this *state update* does not perform a permanent change in the planning state, and multiple planning states are stored for each contemplated action in all generated plans.

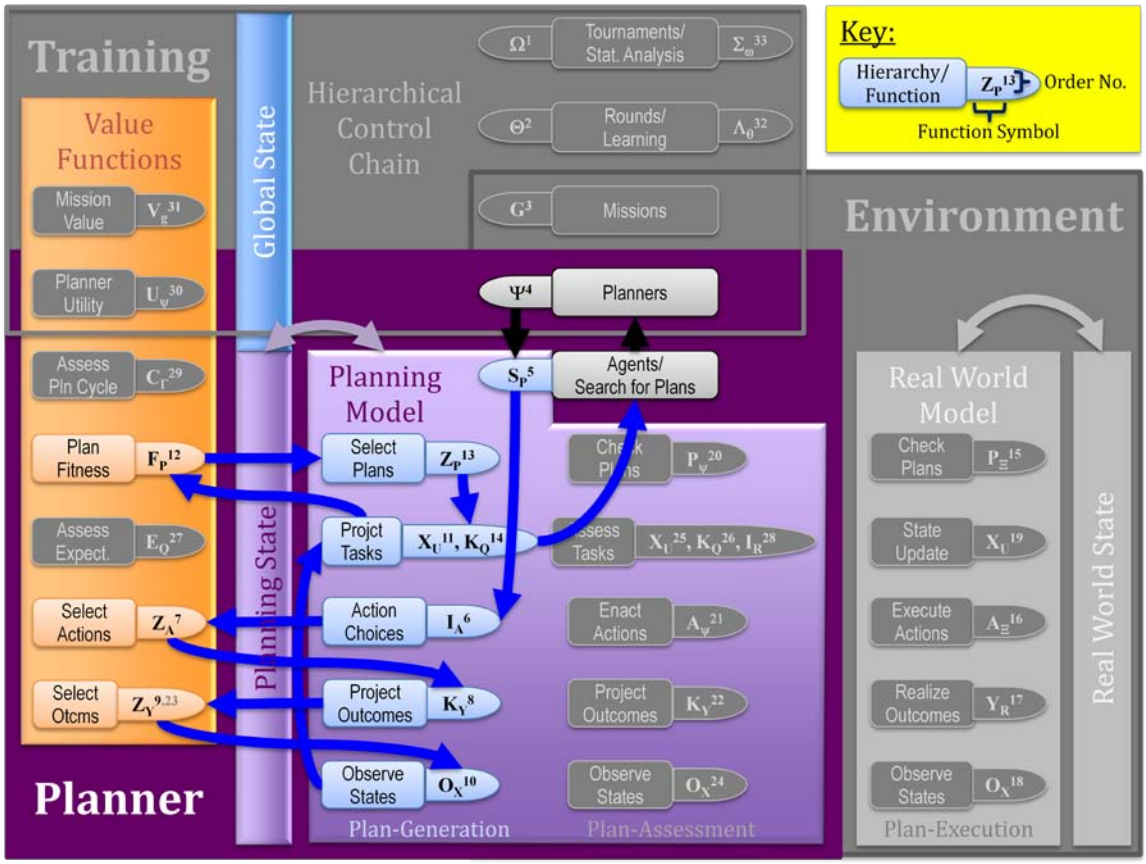


Figure 9.5. DB&T Step 5: Generate and Choose Plans

Step five of the DB&T, **generate and choose plans**, is shown in Figure 9.5. This step completes the plan-generation implementation by adding the final three functions: *plan fitness* ($F_P^{12}$), *plan selection* ($Z_P^{13}$), and *project expectations* ($K_Q^{14}$). The previous seven functions should be the same as those implemented in DB&T step five. *Plan fitness* is a function that measures the value of a plan by observing the plan end state, the probability to get to that state, and the diversity in plan selection. This *plan fitness* function can be implemented in a similar fashion to the one used in RISK, which uses all three aspects identified here, or a simpler method could be formulated just based on state. *Plan selection* is a function that determines which plans are selected, and at what point the plan should be terminated or pruned based on optimizing *plan fitness*. An evolutionary selection approach was used in RISK and can be applied here or a new method can be designed to more specifically meet application needs. The *project expectations* function is a task driven function. The designer must identify what constitutes a task, such as a conquered territory for RISK game or rescued individuals in US&RO mission. Once a task is identified, a *project expectations* function can be implemented that monitors plans to determine what tasks are projected to be completed and with what probability of success. This DB&T step of fully implementing plan-generation is at the heart of the ADP&E system, and plan-generation costs could be applied to expended agent resources, time expended to plan based on computations, and probability of successful plan completion based on projected expectations.

Step six of the DB&T procedure, **assess plans**, is shown in Figure 9.6. This step is the entirety of the plan-assessment process (nine functions) and also includes the first L&A function. More specifically, this step includes the implementation of ten new

functions: *check plans can be assessed* ($P_{\Psi}^{20}$), *enact planned actions* ($A_{\Psi}^{21}$), *project outcomes* ($K_Y^{22}$), *outcome selection* ($Z_Y^{23}$), *observe states* ($O_X^{24}$), *state update* ($X_U^{25}$), *project expectations* ($K_Q^{26}$), *assess expectations* ($E_Q^{27}$), *initiate re-planning* ($I_R^{28}$), and *planner cycle assessment* ($C_{\Gamma}^{29}$). Functions 20 and 21, *check plans can be assessed* and *enact planned actions* are almost identical to the *check plans can be executed* and *execute actions* (functions 20 and 21 match15 and 16, respectively), except that the input is from the planning model and state, instead of the real world model and state. Functions 22 to 26 ($K_Y^{22}$, $Z_Y^{23}$, $O_X^{24}$, $X_U^{25}$, $K_Q^{26}$) are identical to functions 8 to 11, and 14 ($K_Y^{8}$, $Z_Y^{9}$, $O_X^{10}$, $X_U^{11}$, $K_Q^{14}$), respectively. However, plan-assessment is simpler than in plan-generation, since plan-assessment does not assess multiple generated plans at once.

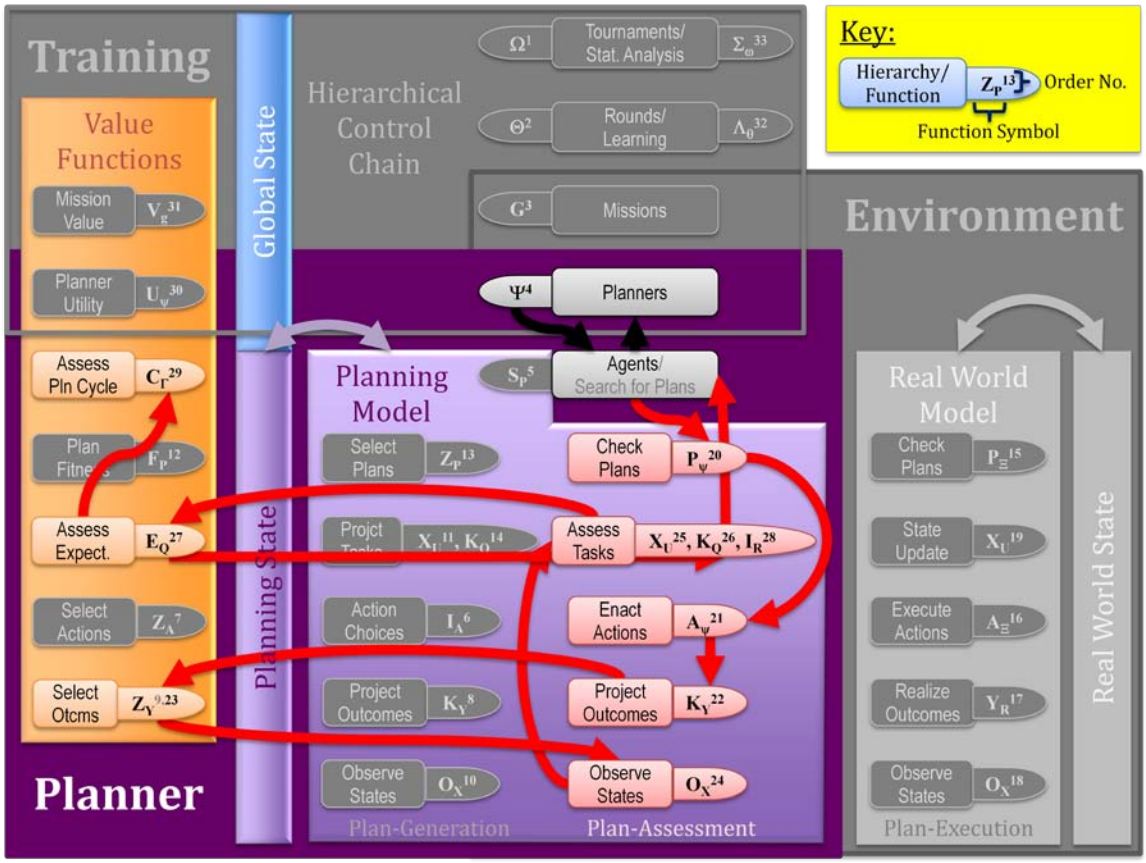


Figure 9.6. DB&T Step 6: Assess Plans

The *assess expectations* function is unique, because this function compares the expectations projected in plan-generation with those projected in plan-assessment. A substantial difference in expectations can trigger (acceptance threshold described in Chapter 6) the *initiate re-planning* function, which reverts back to the *search for plans* function (5), and generates new plans for the current agents under consideration. In any case, the *assess expectations* function activates the *planning cycle assessment* function ($\mathbf{C}_{\Gamma}^{29}$), which outputs the number of tasks executed per time ($T_P/t = \mathbf{C}_{\Gamma}(\psi, \Gamma, T_P, C_A, t)$). This function has as inputs the current planner ($\psi$), the current agents ($\Gamma$), the completed tasks ($T_P$), the action costs ($C_A$), and time ($t$) to determine the efficiency and effectiveness of the planning cycle.

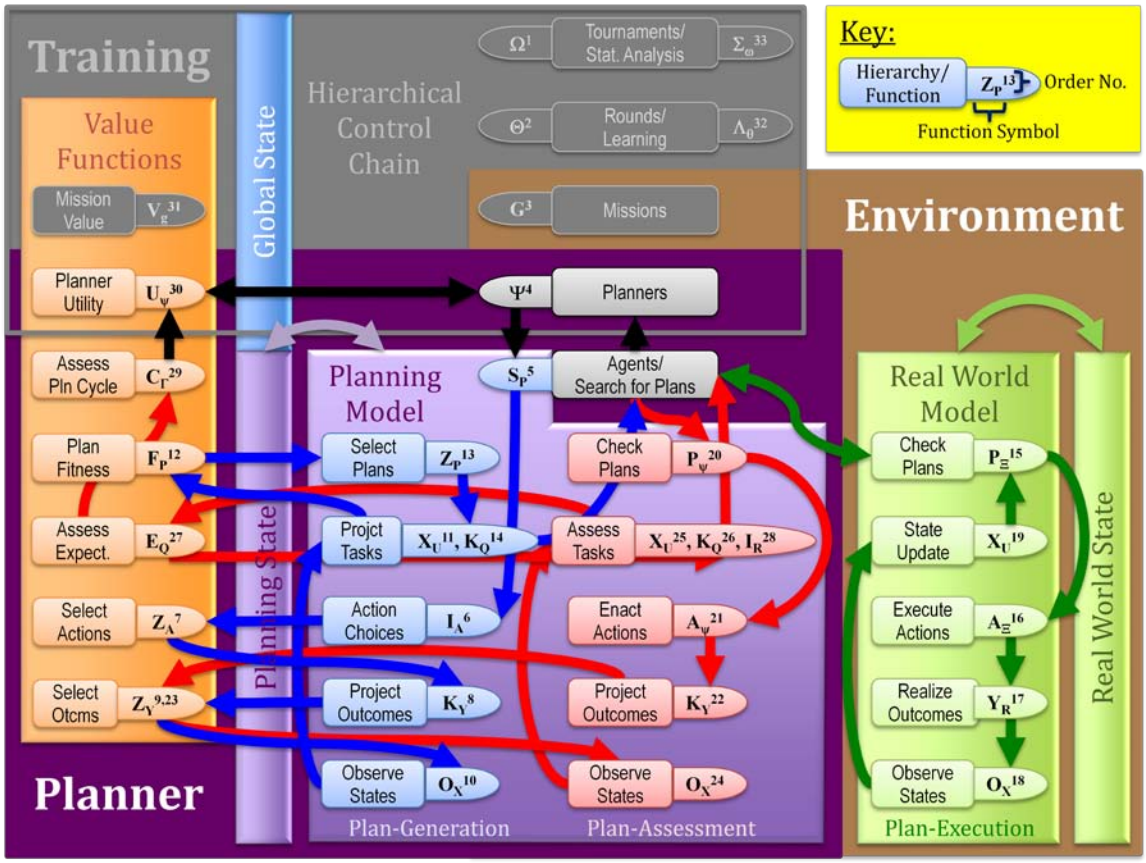


Figure 9.7. DB&T Step 7: Coordinating Full Core Planning Cycle

Step seven of the DB&T, **core planning cycle**, is shown in Figure 9.7. This step includes all three planning phases: plan-generation, plan-execution, and plan-assessment. New features include the coordination of information among the phases, such as the transition functions: 14 to 15, 19 to 20, and 28 to 5. In addition, one new L&A function is developed (*planner utility* ($\mathbf{U}_{\Psi}^{30}$)) as well as the inclusion of all parameters linked to the planner. The *planner utility* function ($U = \mathbf{U}_{\Psi}(\psi, X_{\Xi})$) outputs utility ($U$) from the inputs of a particular planner ($\psi$) based on the fully realized state ($X_{\Xi}$).

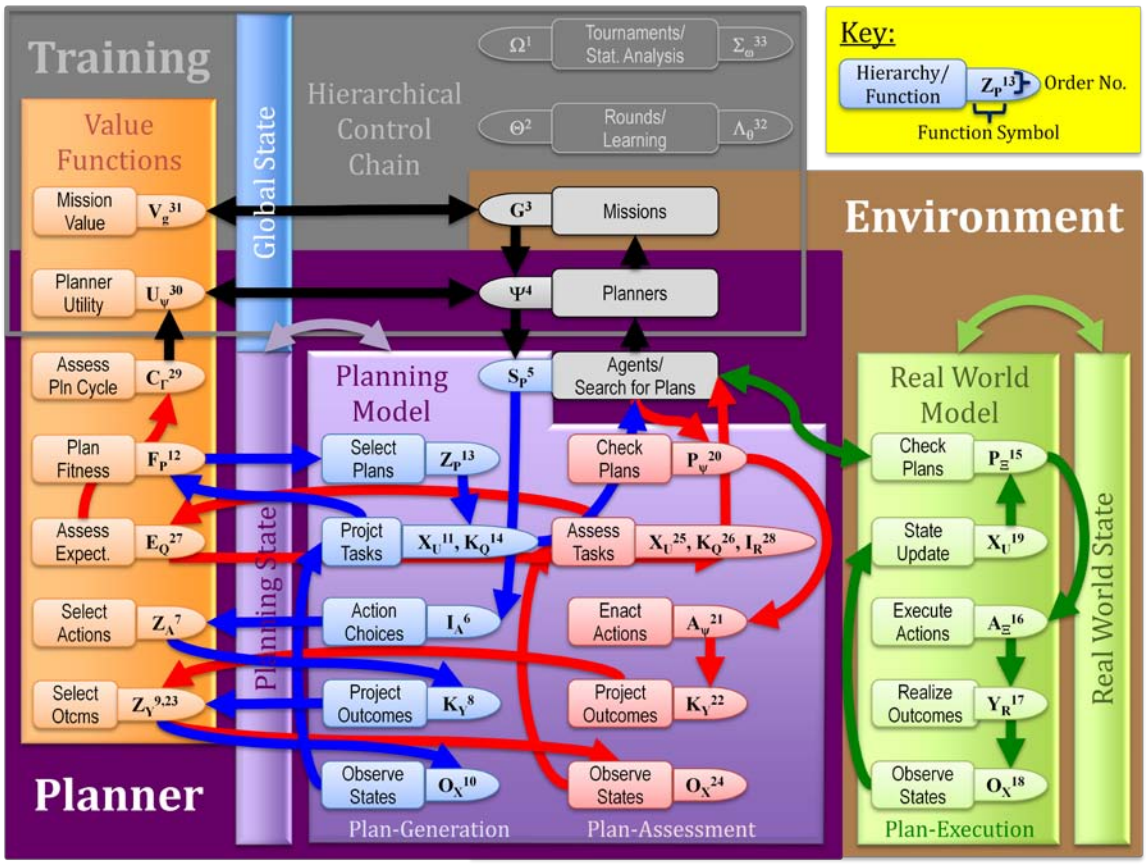


Figure 9.8. DB&T Step 8: Coordinating Multiple Planners for a Single Mission

Step eight of the DB&T, **a single mission or game**, is shown in Figure 9.8. This step combines the coordination of multiple planners by adding one HCC function

(mission) and one L&A function (mission value). The *mission function* ($\mathbf{G}^{3}$) coordinates planners either as turn-based or as simultaneous entities, and outputs the next game ($g$) and planners ($\psi$) to use, given the current round ($\theta$), games ($G$), and planners ($\Psi$) states (i.e., $\{g, \psi\} = \mathbf{G}(\theta, G, \Psi)$). The *mission value* function ($\mathbf{V}_{\mathbf{g}}^{31}$) compares the performance of all planners in a game or mission, and outputs the values for all planners ($V_{\Psi}$) given the current game ($g$), set of planners ($\Psi$) in the game, and the fully realized state ($X_{\Xi}$) after a mission or game is completed (i.e., $V_{\Psi} = \mathbf{V}_{\mathbf{g}}(g, \Psi, X_{\Xi})$).

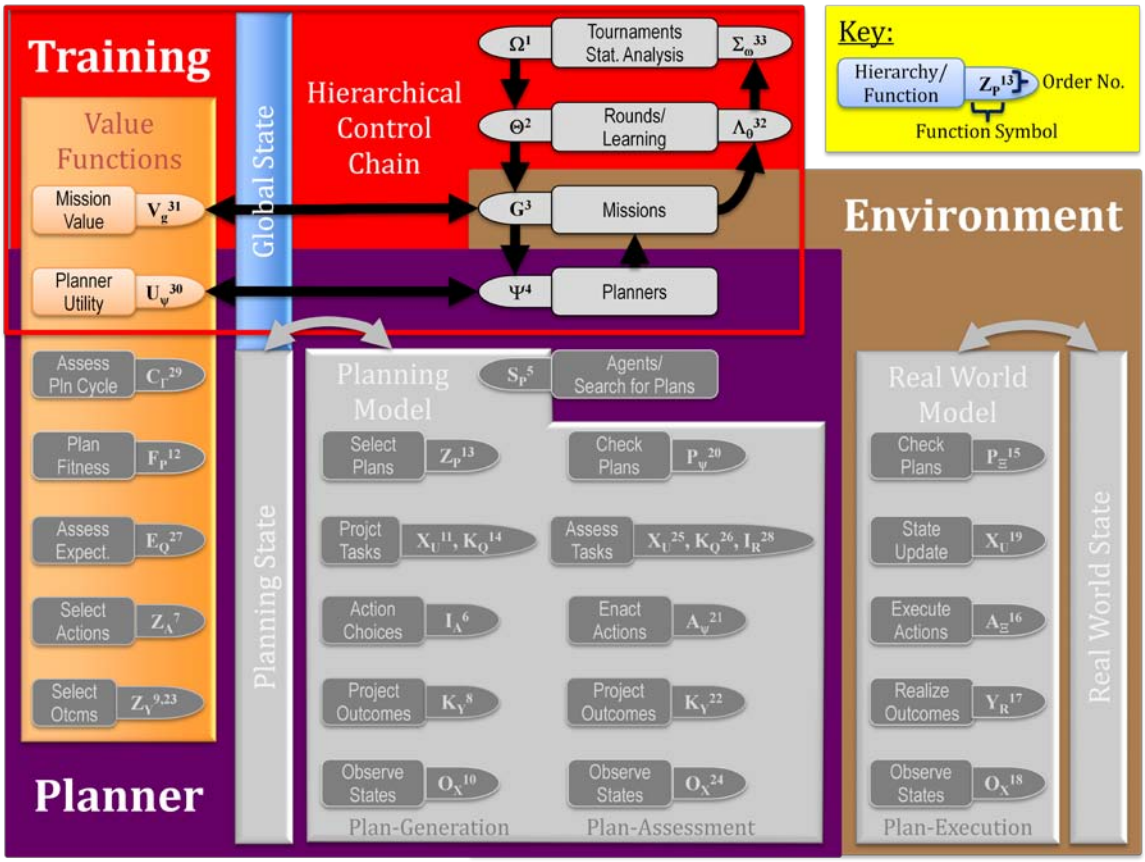


Figure 9.9. DB&T Step 9: Meta-Learning Strategy for Training Planners

The final step, nine, of the DB&T, **a complete tournament**, is shown in Figure 9.9. This step focuses on the coordination of future missions or games using the HCC,

and on the L&A of completed missions or games to train planners. The first two overall functions are the HCC functions *tournaments* ($\mathbf{\Omega}^{1}$) and *rounds* ($\mathbf{\Theta}^{2}$). The *tournaments* function outputs the next round ($\omega$) given the current rounds ($\Theta$) and tournament ($\omega$) under consideration (i.e., $\theta = \mathbf{\Omega}(\omega, \Theta)$). The *rounds* function outputs the next round ($\theta$) and game ($g$) given the current tournament ($\omega$), rounds ($\Theta$), and games ($G$) under consideration (i.e., $\{\theta, g\} = \mathbf{\Theta}(\omega, \Theta, G)$).

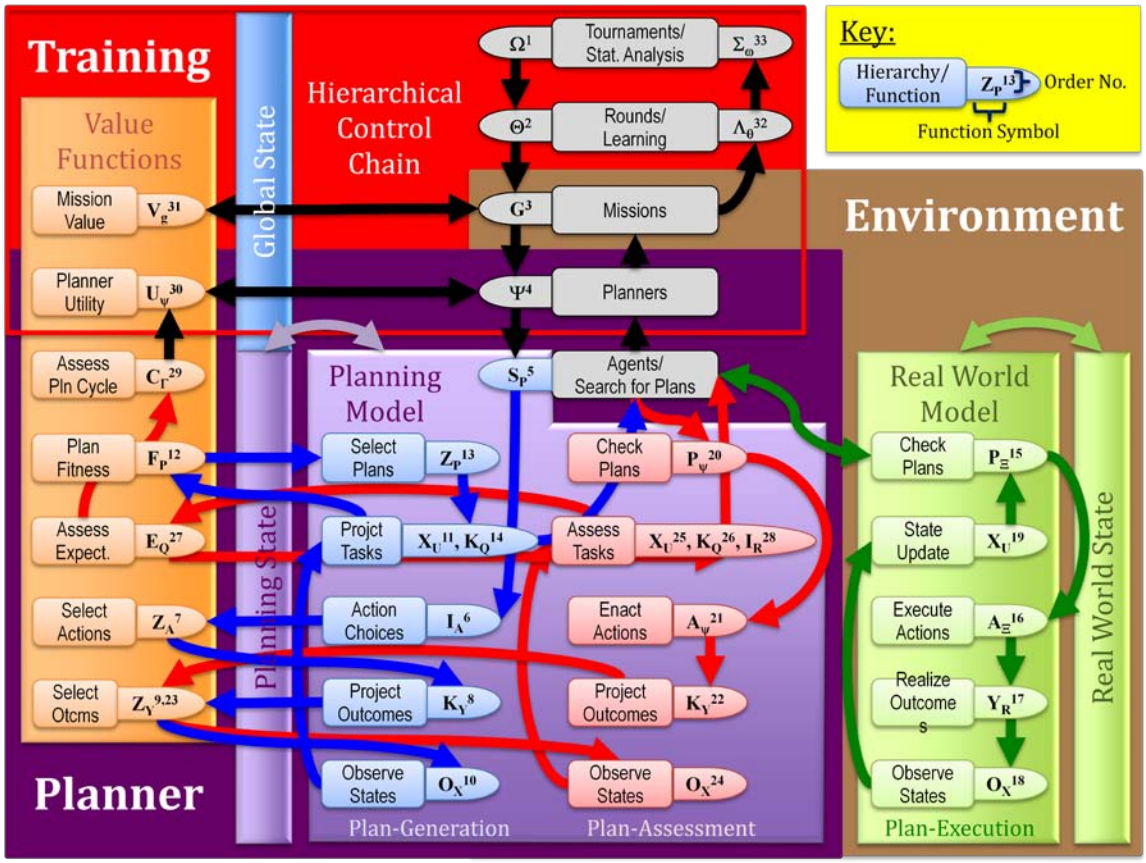


Figure 9.10. Operation of Entire ADP&E System

The last two overall functions are the L&A functions *evolutionary learning* ($\mathbf{\Lambda}_{\theta}^{32}$) and *statistical analysis* ($\mathbf{\Sigma}_{\omega}^{33}$). The *evolutionary learning* function outputs a new set of parameters ($\Pi$) for a future planner given the current round ($\theta$), the games or mission ($G$)

completed, the planners ($\Psi$) for all missions, current planners' parameters ($\Pi$), the fully realized state ($X_{\Xi}$), and the evolutionary parameters ($e_{\theta}$) used to train in the current round ($\Pi = \Lambda_{\theta}(\theta, G, \Psi, \Pi, X_{\Xi}, e_{\theta})$). The *statistical analysis* function outputs the means and standard deviations of parameters values ($\Sigma_{\omega}$) across multiple generational rounds given the current tournament ($\omega$), rounds ($\Theta$), and parameter values ($\Pi_{\Theta}$) for a set of rounds (i.e., $\Sigma_{\Pi} = \Sigma_{\omega}(\omega, \Theta, \Pi_{\Theta})$).

A final step is to test the operation of the entire ADP&E system as illustrated in Figure 9.10. This pictorial view of the architecture and process flow is complex, but if a designer follows the previously described nine-step process, this can be completed in a manageable way. Note that the lower levels of the architecture operate faster and more often than the upper level structures due to the hierarchical decomposition of the problem space.

## 9.3 Implementation Summary

In previous sections, the ADP&E system has been described with an architectural diagram, and a step-by-step implementation strategy that combines the general architecture description in Chapter 6 with the lessons learned in Chapters 7 and 8. Table 9.2 above summarizes the description of the architecture by examining all 33 functions from several aspects. The first, second, and third columns show the functional name, symbol, and order of the 33 functions, respectively. The order is not always sequential, because many aspects of planning require an iterative process until a phase is complete. The fourth column shows the next step, or where the function can transition given the circumstances. Column five shows where the function is first implemented in the nine

step DB&T procedure described in Section 9.2. Column six shows the phase associated with each function (Note that the HCC and L&A phases are functions of the overall training phase), column seven shows the corresponding hierarchical level, and column seven shows the function with its inputs and outputs.

Table 9.2. All ADP&E System Functions with Related Attributes

| Function Name | Symbol | Order No. | Next Step | DBT Step | Cycle Phase | Hierarchy Level | Function – Input/Output |
|---|---|---|---|---|---|---|---|
| Tournaments | $\Omega$ | 1 | 1,2 | 9 | HCC | Tournaments (1) | $\theta = \Omega(\omega, \Theta)$ |
| Rounds | $\Theta$ | 2 | 2,3 | 9 | HCC | Rounds (2) | $\{\theta, g\} = \Theta(\omega, \Theta, G)$ |
| Missions | $\mathbf{G}$ | 3 | 3,4 | 8 | HCC | Missions (3) | $\{g, \psi\} = \mathbf{G}(\theta, G, \Psi)$ |
| Planners | $\Psi$ | 4 | 4,5 | 1 | HCC | Planners (4) | $\{\psi, \gamma\} = \Psi(g, \Psi, \Gamma)$ |
| Search For Plans | $\mathbf{S_P}$ | 5 | 5,6 | 1 | I | Agents (5) | $P = \mathbf{S_P}(S_\psi, T_P, \tau_P, R, u_R)$ |
| Determine Action Choices | $\mathbf{I_A}$ | 6 | 5,7 | 3 | I | Actions (8) | $A = \mathbf{I_A}(\gamma, M_\psi, X_P)$ |
| Action Selection | $\mathbf{Z_A}$ | 7 | 5,8 | 3 | I | Actions (8) | $A_Z = \mathbf{Z_A}(\Gamma, A, H, R, C_A, t)$ |
| Project Outcomes | $\mathbf{K_Y}$ | 8 | 9 | 4 | I | Outcomes (9) | $Y = \mathbf{K_Y}(A_Z, \gamma, M_\psi, X_P)$ |
| Outcome Selection | $\mathbf{Z_Y}$ | 9 | 10 | 4 | I | Outcomes (9) | $Y_Z = \mathbf{Z_Y}(Y, r_Y)$ |
| Observe States | $\mathbf{O_X}$ | 10 | 11 | 1 | I | Observations (10) | $O' = \mathbf{O_X}(Y', \gamma, M_\Xi, X_\Xi)$ |
| State Update | $\mathbf{X_U}$ | 11 | 12 | 2 | I | Tasks (7) | $X_\psi = \mathbf{X_U}(\Gamma, O', \tau_U)$ |
| Plan Fitness | $\mathbf{F_P}$ | 12 | 13 | 5 | I | Plans (6) | $F = \mathbf{F_P}(X_P, J_\psi, Pr(T_P), D)$ |
| Plan Selection | $\mathbf{Z_P}$ | 13 | 14 | 5 | I | Plans (6) | $P_Z = \mathbf{Z_P}(F, P)$ |
| Project Expectations | $\mathbf{K_Q}$ | 14 | 5,15 | 5 | I | Tasks (7) | $Q = \mathbf{K_Q}(P_Z, T_P)$ |
| Check Plans Are Executable | $\mathbf{P_\Xi}$ | 15 | 16,20 | 1 | II | Plans (6) | $P' = \mathbf{P_\Xi}(P_Z, \gamma, M_\psi, X_P)$ |
| Execute Actions | $\mathbf{A_\Xi}$ | 16 | 17 | 1 | II | Actions (8) | $\{A_I, P'\} = \mathbf{A_\Xi}(P', t)$ |
| Realize Outcomes | $\mathbf{Y_R}$ | 17 | 18 | 1 | II | Outcomes (9) | $Y' = \mathbf{Y_R}(A_I, \gamma, M_\Xi, X_\Xi)$ |
| Observe States | $\mathbf{O_X}$ | 18 | 19 | 1 | II | Observations (10) | $O' = \mathbf{O_X}(Y', \gamma, M_\Xi, X_\Xi)$ |
| State Update | $\mathbf{X_U}$ | 19 | 15,20 | 2 | II | Tasks (7) | $X_\psi = \mathbf{X_U}(\Gamma, O', \tau_U)$ |
| Check Plans Can Be Assessed | $\mathbf{P_\Psi}$ | 20 | 5,21 | 6 | III | Plans (6) | $P'' = \mathbf{P_\Psi}(P', \gamma, M_\psi, X_P)$ |
| Enact Planned Actions | $\mathbf{A_\Psi}$ | 21 | 22 | 6 | III | Actions (8) | $\{A_I, P''\} = \mathbf{A_\Psi}(P'', t)$ |
| Project Outcomes | $\mathbf{K_Y}$ | 22 | 23 | 4 | III | Outcomes (9) | $Y = \mathbf{K_Y}(A_Z, \gamma, M_\psi, X_P)$ |
| Outcome Selection | $\mathbf{Z_Y}$ | 23 | 24 | 4 | III | Outcomes (9) | $Y_Z = \mathbf{Z_Y}(Y, r_Y)$ |
| Observe States | $\mathbf{O_X}$ | 24 | 25 | 1 | III | Observations (10) | $O' = \mathbf{O_X}(Y', \gamma, M_\Xi, X_\Xi)$ |
| State Update | $\mathbf{X_U}$ | 25 | 26 | 2 | III | Tasks (7) | $X_\psi = \mathbf{X_U}(\Gamma, O', \tau_U)$ |
| Project Expectations | $\mathbf{K_Q}$ | 26 | 27 | 5 | III | Tasks (7) | $Q = \mathbf{K_Q}(P_Z, T_P)$ |
| Assess Expectations | $\mathbf{E_Q}$ | 27 | 28,29 | 6 | III | Tasks (7) | $Q = \mathbf{E_Q}(Q, Q'', r_Q)$ |
| Initiate Re-Planning | $\mathbf{I_R}$ | 28 | 5 | 6 | III | Tasks (7) | $P = \mathbf{I_R}(P'', Q)$ |
| Planner Cycle Assessment | $\mathbf{C_\Gamma}$ | 29 | 5,30 | 6 | L&A | Agents (5) | $T_P/t = \mathbf{C_\Gamma}(\psi, \Gamma, T_P, C_A, t)$ |
| Planner Utility | $\mathbf{U_\psi}$ | 30 | 4,31 | 7 | L&A | Planners (4) | $U = \mathbf{U_\psi}(\psi, X_\Xi)$ |
| Planner Value | $\mathbf{V_g}$ | 31 | 3,32 | 9 | L&A | Missions (3) | $V_\Psi = \mathbf{V_g}(g, \Psi, X_\Xi)$ |
| Evolutionary Learning | $\Lambda_\theta$ | 32 | 2,33 | 9 | L&A | Rounds (2) | $\Pi = \Lambda_\theta(\theta, G, \Psi, \Pi, X_\Xi, e_\theta)$ |
| Statistical Analysis | $\Sigma_\omega$ | 33 | 1,end | 9 | L&A | Tournaments (1) | $\Sigma_\Pi = \Sigma_\omega(\omega, \Theta, \Pi_\Theta)$ |

**References:**

1. T.C. Lethbridge and R. Laganière. Object-Oriented Software Engineering: Practical Software Development using UML and Java. Second Edition McGraw Hill, 2001.

2. J. Vaccaro, C. Guest, "Learning Multiple Search, Utility and Goal Parameters for the Game RISK," *IEEE Congress on Evolutionary Computation*, Vancouver, Canada, July 2006, pp. 4351-4358.

3. J. Vaccaro, C. Guest, "Automated Dynamic Planning and Execution for a Partially Observable Game Model: Tsunami City Search and Rescue," *IEEE World Congress on Computational Intelligence (WCCI'08), in press,* Hong Kong, China, June 2008.

# Chapter 10

# Discussion and Future Work

## 10.0 Abstract

Discussion and future work is focused in four sections: (1) a technical summary of what has been accomplished, (2) conclusions drawn, (3) design improvements, and (4) new applications. The technical summary provides a brief overview of the dissertation. This overview includes: a summary of the chapters, the accomplishments, challenges uncovered, and challenges met. The conclusions include a new formulation of the planning problem, its extensibility to more domains, and a review of new directions of future research. Design improvements include five areas: hierarchy, modularity, extensibility, learning, and human/computer interface. New applications include extensions of current results as well new application domains that expand on the ADP&E framework and capabilities.

## 10.1 Technical Summary

This dissertation is subdivided into four main areas: introduction and problem definitions, background and related work, challenges and generalization insights, and a defined approach with specific applications. Chapter 1 introduces the overall challenges, Chapter 2 describes twelve specific challenges related to generating plans (Section 2.4.1), and Chapter 5 further describes challenges from three perspectives. These three challenge

areas are environmental, planner-based, and continual improvement through learning. The generalization insights to these challenges are described in Chapter 9. The defined approach and two specific applications are described in Chapter 6, 7, and 8, respectively.

There are eight accomplishments listed in Chapter 1 Section 1.3. First, POMDP extensions are described in Chapter 6 as fundamental building blocks (Section 6.1.1) in terms of a three phase planning cycle (Section 6.2). Second, a scalable planning framework is also described in Chapter 6 in terms of a ten-tier hierarchy (Section 6.1.2). Third, risk aversion and probability of plan success control is described in Chapter 7 in terms of risk-averse outcome selection (Section 7.1.7), and probability of completing interdependent battle campaigns (Section 7.1.4). Fourth, a search comparison across several techniques with results is described in Chapter 7 for the stochastic RISK game (Section 7.3.1.3). Fifth, the integration of the temporal cost of planning into the general approach is described in Chapter 6, Section 6.2.1. The temporal cost of planning implementation for the two applications is described in Section 7.3.3.3 for RISK game (50,000 action limit), and in Section 8.1.3 for US&RO mission. Sixth, the strategies and results for improving the effectiveness of the planners for both applications are shown in Sections 7.3.3 and 8.3.3. The versatility of the approach is described in Chapter 9 in terms of the two applications described in Chapters 7 and 8. Finally, a general implementation strategy for future applications is described in Chapter 6, Section 6.3 Design, Build and Test. Specific implementation details for the two specific applications are described in Section 7.2 and 8.2, and a design criteria is discussed in Chapter 9, Section 9.2, which describes the lessons learned in completing the two applications.

The challenges investigated in this dissertation are in three chapters. Chapter 1 describes the challenges involved with building an effective planning framework, and concentrates on the architecture, process flow, and continual improvement aspects of the system. Chapters 2 and 5 describe the learning, traceability and justification challenges. Chapter 2 describes a set of proposed metrics: (1) risk sensitive prediction, (2) action selection conditions, (3) expectation acceptance threshold, (4) plan selection fitness, (5) planning cycle assessment, (6) planner utility, and (7) mission value (Section 2.5). Chapter 5 describes basic machine learning challenges: exploitation vs. exploration, scalability, and credit assignment. The challenges associated with characterizing the problem, its extensibility and design criteria are described in Chapter 5 in terms of the main two essential objects: an environment, and planners.

Challenges were met in all three areas described above. For the planning framework challenge there were three major accomplishments: (1) a ten-tier hierarchy was constructed for modularity and scalability (described in Section 6.1.2), (2) a process flow was developed that directed plans through goals and available actions, and trained planners based on observations and responses (described in Section 6.1.3), and (3) a three phase planning-cycle was developed, which trades off plan-generation, -execution and -assessment in terms of planning costs, risks, and successful expectations (described in Section 6.1.4). For learning challenges, three levels of trained improvement were implemented in both applications: plan-generation search and selection (Sections 7.3.1 and 8.3.1), planning cycle assessment (Sections 7.3.2 and 8.3.2), and training planners (Sections 7.3.3 and 8.3.3). For traceability and justification challenges, all seven metrics are defined (Sections 2.5, 6.1.1, and 9.2) and learned (Sections 7.3 and 8.3). For

characterization and extensibility, classes and objects are examined and defined for two specific applications and future general applications (Section 9.1). For future design, build and test (DB&T) challenges, Section 6.3 introduces a general nine-step process for implementing an ADP&E system. Sections 7.2 and 8.2 describe this implementation process for two specific applications, RISK game and US&RO mission, respectively. Section 9.2 describes this nine-step process with all 33 functions in general terms to DB&T future applications.

## 10.2 Conclusions

This dissertation attempted solution of planning problems that are very large, partially observable, and stochastic (VLPO&S). In so doing, various aspects of planning have been uncovered and further defined to fit these problem domains. There are many planning algorithms in existence for solving all sorts of problems. These algorithms are for specific types of planning, such as discrete problems, motion planning, decision theoretic planning, etc. [1]. These algorithms form a solid foundation, but further work was needed to enable use in VLPO&S environments. This dissertation reformulates the planning problem in a broader sense as defined in Chapter 2. Three major developments under this broader planning definition have been completed. First, a hierarchical framework provides the flexibility to add or change features at any of the ten levels of planning through either human interaction or simulated training. Search is an important component in planning, and new search algorithms have been developed for large stochastic problems (Section 7.3.1) that outperform well-established methods (Section 3.2.2) [2]. Finally, the bidirectional process flow in the hierarchy allows simple observations to flow up the tiers to feed back on planners' ability, while tournaments

compare and contrast results of multiple games or missions to train planners to become more effective and efficient. Over the span of the hierarchy, there are three major learning areas: plan-generation search and selection, planning cycle assessment, and planner(s)' utility. The associated parameters in each of these three areas have been shown that they can be trained simultaneously in an autonomous process. This approach can be used to help minimize assumptions and biases made from SME by separating the effective SME from the not-so-effective SME through online training.

As documented in Chapter 9, the characterization and extensibility of our approach is still in its infancy. First, there is a need for a greater exploration of the two applications. More detailed statistics can be collected to establish greater significance in the results, and establish more general conclusions. For instance, agents in the US&RO application can be multi-threaded so a single tournament takes around twelve hours of computer time instead of a month. Thus, many more tournaments could be run in a reasonable amount of time. Second, an application should be studied that has both the stochastic elements of the RISK game and the partial observability of the US&RO (see Table 9.1 and 9.2). For instance, one could consider poker as a possible candidate, because the cards are often hidden, there are multiple players, and getting a particular card is based on chance.

There are currently several major directions planned for future research. First, many issues are still not explored. This includes pursuing an application that has both partial observability and stochastic elements, and has both cooperative and non-cooperative planners competing for dissimilar but overlapping goals. Also, giving more

distributed control to agents for local decision-making vs. the current complete centralized control given to the planners needs to be considered in future applications. A degree of authority can be given from each agent to a variety of planners, and a degree of independent authority is given to itself [3]. This is far more realistic as to how humans multi-task, when balancing work, family, friends, school, etc. This type of environment raises interesting questions about trust among planners and agents. Also, the balance of attention and focus in planning is also in question. Do the agents view attention differently than the planners? Do the agents' desires for low-hanging fruit sacrifice the planner's long-term goals? How does deception play a role in distorting other planners' plans or goals? These are just a few of the questions that can be asked if the system is expanded to these new difficult problem domains.

## 10.3 Design Improvements

Design improvements in the planning hierarchy can take several forms. First, having all ten layers of hierarchy is not necessary for all planning problems, and can be condensed for smaller problem domains to make the planner more efficient. However, ever increasing computing power and cloud computing capabilities [4] will make multi-planner applications much faster in the near future. Second, populating the hierarchy with SME should be downplayed in future versions. Providing more automation and less human interaction in the ADP&E system will eliminate bias and save on man-power in future developments. This automation can include having more seamless process flow up and down the hierarchy, where a location and mission requirements are only available, and the number of planners, agents, their types, etc. will be optimized based on the situation at hand. Finally, the environment itself can benefit from a hierarchical

representation that best maps to the ten-tier ADP&E approach. This representation can then be used to display the workings of the ADP&E system on the environment in question. Currently, the simulations are implemented without visualization in mind, and built for performance.

Design modularity can be improved several ways. First, in the implementation of the RISK game and US&RO mission simulation, there were dependencies that crossed more than two tiers in the hierarchy to save on computation time. For instance, actions, outcomes, and observations were computed together to simplify the process. This however, made it more difficult to modify features such as flood water level changes that a future developer might want to make variable. By restricting the overlap in the hierarchy as described in Chapter 6, features can be changed more easily at all levels of the hierarchy. Note that the six highest tiers were always kept separate in the two developed applications, and thus features were easily manipulated. Second, with the modularity of the system, modules can be competed separately or interdependently to compare different techniques or algorithms, as this dissertation illustrated with competing search techniques. Third, the modularity provides independence of agents, and thus they should be multi-threaded in the future to speed the simulation n-fold, where n is the number of agents.

Extensibility design improvements are briefly mentioned in the previous subsection, and are an important part of improving the ADP&E concept. Three extensibility improvements are summarized as follows. First, we can combine capabilities used in RISK game and US&RO mission application into one system. Second, we can

design distributed control agents to provide more local capabilities. Finally, we can study the effects of trust, deception, attention, and focus, and how they pertain to the type and number of planners involved, and their authority over the number and type of agents at their disposal. These problems are very difficult in their own right, but much can be learned about the interaction of multiple players in complex game environments.

Learning design can be improved in several ways. First, we can include more cycle assessment parameters in the learning process, such as communication, model update, and assessment variability with the cost to do these tasks. For example, the length of the plans in US&RO was determined to be best for one round trip by using a separate simulation to compare results. It would have been more interesting to have plan length be variable as in RISK game and learned as one of the many parameters trained during the tournament rounds. Second, we can compare a variety of machine learning algorithms in training plan-generation search and selection, planning cycle assessment, and planner(s)' utility to compare the value of using the current evolutionary approach with other approaches, such as reinforcement or Hebbian learning. Third, we can adaptively learn the rules of the game and characteristics of the environment, thereby increasing the automation of the ADP&E system by minimizing the need for SME input. We can also include more game-phase discrimination in planning, similar to what was done in the RISK game, but in more general terms, such as early plans for learning, and later plans for achieving. Finally, we can provide for a more general adaptation of both condition features and goal weights as the game progresses to include pressures from choosing both good short-term actions and long-term plans.

There are several possible human/computer interface improvements, since the currently built system can only be modified through the Matlab codebase. The RISK game does allow for human players, but the US&RO mission tool does not allow for a human player. One of the main considerations for improvement is in visualization of the planners in action at all three phases of operation: plan-generation, -execution, and -assessment. For example, in plan-generation, the number of ill-conceived plans examined before choosing a quality plan is not given, nor are dead-end areas of search, which are currently discarded. These can provide negative reinforcement in future searches. In plan-execution, plans that are in trouble can be illustrated to the human user on the fly, and resulting outcomes can provide automatic justification in verbal or graphical form. In plan-assessment, a traceable probability of plan success can be given to the human-operator on the fly, and troubled agents can be identified as alerts. Also, the ten tiers in the hierarchy can be shown in a graphical form to the user, to show what parts of the hierarchy are currently activated, such as which phase the planner is in, what agents are currently being considered for planning, etc. Given the diversity of the hierarchical tiers, these are just a few of many possible interface improvements to this system.

## 10.4 New Applications

Before any new applications are attempted, there are several ways to expand on current results. For the RISK game, one can increase the number of contemplated actions beyond the current 50,000 actions per player per game. This number is far too small a sample size based on the current results shown in Section 7.3.3. For the US&RO, water can recede from the peak flood stage, healthy individuals can wander to seek better refuge, more planners can be used to compete for use of resources, and communication

access among planners, agents, and people can be further optimized to meet these new demands. There is certainly a need for more simulations in both applications to increase the statistical significance of current and future results, and many more aspects of the ADP&E system, as described in Tables 9.1 and 9.2, can be evaluated and improved.

An important step in the near future is implementing a test application that includes both stochastic and partial observability elements. One case already mentioned is the game Poker. Many poker games, such as Texas Hold'm or Omaha, have elements of chance in receiving cards and partial observability in that each player can only see their own held cards and the shared table cards. Also, the players can learn a variety of strategies based on whether the game is limit or no-limit, and according to the number of players in a hand. Another real-world mission problem that has both uncertainty and partial observability is the control of traffic signals based on observations from road sensors. Currently, many traffic light locations are either optimized for a particular type of traffic pattern, or have a simple time limit switch that changes them deterministically. Having a shared understanding among traffic sensors and signals could save in time and fuel efficiency, and help avoid unnecessary traffic jams.

There are several ways to extend the complexity of the application domain. First, problems can be attempted that include more distributed control for agents to a variable degree (only use delegated plans from planner(s) as guidance and not used verbatim). This inclusion may require leveraging other technologies, such as swarm intelligence [5], belief-desire-intent models [6], etc., or the current planner functional framework could be applied directly to the agents, making them planners on a smaller scale. Having

individual agents with a degree of autonomy could increase the overall efficiency of a planned operation, especially when communication costs are at a premium, an agent can learn to do a localized task more optimally than directed, or the operation may be covert and communication is ill-advised. Second, future applications can require multiple planners (often many) that need to share resources, while each planner is contemplating the use of resources for different objectives. An extension to the US&RO problem is mentioned above, but there are many other problems that fit this situation. Another interesting application is use in a cyber domain, especially given the global reach of the Internet. Power grids, banking services, email services, data repositories, etc. are available through the Internet, if the correct access can be achieved. Thus, sharing the Internet with the many hundred millions of people, some who want to use the Internet as a resource, while others want to degrade this resource, or use it as a means to degrade another resource (e.g., take down a power grid). In conclusion, there are many future challenges in automated planning, and this dissertation sheds some light on these challenges. Given the complexity of VLPO&S planning problems, automated planning will have a bright future.

**References:**

1. S. M. LaValle, Planning Algorithms (Book), *Cambridge University Press*. 2006.
2. J. Vaccaro, C. Guest, "Planning an Endgame Move Set for the Game RISK: A Comparison of Search Algorithms," *IEEE Transactions on Evolutionary Computation, Vol. 9, No. 6*, December 2005, pp. 641-652.
3. J. Vaccaro, C. Guest, "Modeling Social Influence via Combined Centralized and Distributed Planning Control," MODSIM World Conference and Expo, Virginia Beach, VA, October 2009.

4. R. Buyya, C. S. Yeo, S. Venugopal. “Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities.” *Department of Computer Science and Software Engineering,* The University of Melbourne, Australia. pp. 9. July 31, 2008.

5. H. Shah-Hosseini, “The intelligent water drops algorithm: a nature-inspired swarm-based optimization algorithm,” *International Journal of Bio-Inspired Computation*, Vol. 1, Nos. 1/2, pp. 71-79, 2009.

6. M. E. Bratman. “Intention, Plans, and Practical Reason.” *CSLI Publications.* 1999.