INVESTIGATING THE COGNITIVE PROCESSING OF EXPERIENCE FOR DECISION MAKING IN ROBOTS: ACCOUNTING FOR INTERNAL STATES AND APPRAISALS

By

Stephen Michael Gordon

Dissertation

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in

Electrical Engineering

May, 2009

Nashville, TN

Approved:

Professor Kazuhiko Kawamura

Professor D. Mitchell Wilkes

Professor Gautam Biswas

Professor Nilanjan Sarkar

Professor Megan Saylor

ACKNOWLEDGEMENTS

I would like to thank everyone that helped me during my graduate studies, whether it was assistance at school, or in the many other aspects of my life. This has been a difficult, yet very rewarding, process and it would not have been possible without the help of everyone.

First of all I would like to thank my fiancé, Meghan Bates whose never ending support kept me going many times when I thought my own resolve would fail. She has helped me focus on what is important and I cannot thank her enough. I would also like to thank my brother who, though he is an English major, has somehow managed to read (through his own studies) many of the same books that were the inspiration for this work. Discussing these topics with him gave me the unique ability to have an "outsider's opinion" as my research progressed. I must also thank my parents whose constant sacrifice, support, and guidance enabled me to develop the tools necessary to be in this position.

I would like to thank all of the students (both past and present) at the Center for Intelligent Systems (CIS). Not only did their prior work, and current knowledge, provide the backdrop for this research, but their friendship made it enjoyable. In particular, Erdem Erdemir, Jonathan Hunter, Juan Rojas, Chris Costello, Palis Ratanaswasd, Joe Hall, Katherine Fleming, Huan Tan, Xi Luo, Sean Thornton, Albert Spratley, and Will Dodd.

Extra special thanks must go to Flo Wahidi, who is truly the angel of the CIS. No one works as hard as she does and has such a positive impact on the day-to-day workings of the lab. Her help, support, and seemingly endless availability of suggestions cannot be acknowledged enough.

I would like to thank my advisor Dr. Kazuhiko Kawamura (Dr. K). He challenged me to think outside of the box, and to consider problems from new and intriguing directions. Without his confidence and guidance, this research would simply not have been possible.

And, I am appreciative of my committee members who offered guidance and thoughtfulness during the writing and implementation of research. My topic involves several fields of study which has been both rewarding and challenging and I appreciate their input. I would also like to thank Dr. Carl Frankel whose help and guidance really spurred and inspired this work.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Motivation

Decision making in autonomous systems requires purposive deliberation that utilizes knowledge of the goal specific value of particular states as well as an understanding of what is relevant in the current situation. In addition, decision makers embedded in real-world environments require the ability to balance fast commitment against deliberation to ensure that system operating frequency keeps pace with that of the surrounding environment. It is therefore necessary that these systems be able to quickly identify sufficient solutions to the problem at hand, but this ability is contingent upon a number of other factors. First, these systems must be able to identify which aspects of the present situation are most relevant to the current goals. This ability enables weights to be properly assigned that allow feature extraction and concept formation. Second, such systems must be able to mine utility functions from experience in order to prioritize and rank potential responses. This involves associating the goal-relevant information with evaluation signals that indicate goal benefit/harm. Third, such systems must have an appreciation of their own ability to trade between solution quality and deliberation time, in order to appropriately portion deliberation given their own resources and the demands imposed on them by the situation. Finally, because learning is crucial for such an adaptive decision maker, error tracking must be deployed to facilitate performance monitoring as well as to foster an understanding of which knowledge systems adequately fit the current task and which require further training. It is believed that for autonomous systems (e.g., cognitive robots) to make decisions that reflect an understanding of goal relevance, goal specific situational value, and urgency pressures, these systems must mine from their own experience the knowledge that will ultimately be used to inform each of the multi-dimensional evaluation criteria just described.

In robotics, balanced decision making is critical. Robots operate in environments that are often characterized by complex stochastics and large, or continuous, state spaces. Real-time decision-making techniques (e.g., LRTA* [Korf, 1990], RTDP [Barto, et al., 1995], D* [Stentz, 1995] and D* Lite [Koenig and Likhachev, 2002]), as well as anytime algorithms [Zilberstein, 1996] [Dean, et al., 1993] [Paquet, et al., 2005], have been successfully applied to robotic

systems. However, these methods often require maintaining tabular state-value or heuristic functions (*V(s)* and *h(s)*, respectively), which can be prohibitively expensive in large state spaces. Function approximators have been used, with some success, to reduce the representational cost of maintaining such functions, but many approximators require preset, static-sized feature vectors that can omit relevant state information. Furthermore, most real-time systems do not adjust their performance on a situation-dependent basis, which prohibits learning adaptations that improve solution quality.

Rather than use static feature vectors, tabular state value and heuristic functions, or preset performance parameters, it would be preferable to have a system "bootstrap" itself through the cognitive processing of its own experience. The result of this process should be the derivation of experience- and task-based dynamic feature mappings that associate situations with task-dependent appraisals that can be used to internally signal the deliberation process and adaptively tune control parameters. Such signals should identify and include:

1. What is *relevant* in the current situation?
2. What *utility* should be attached to response options in order to achieve the current goals?
3. How *urgent* is the current situation?
4. How well does the current knowledge and the chosen response *fit* the situation?

This dissertation describes how current psychological and neuroscientific research on emotion and emotional processing can be used to inform the design of robots that, through the processing of their own experience, are capable of such balanced decision making.

The Importance of Emotion

In this research the term emotion is used in a manner similar to that as Sloman [2001a] in which emotional states are defined as those states that mediate a system's cognitive processes, or have the potential for such mediation yet are suppressed by a filter or priority mechanim. While this definition will be expanded in Chapter IV, it is important to note that, in this respect, emotion and emotion-based processes are functionally and fundamentally related to goals and the behaviors necessary to achieve those goals [Frijda, 1986]. Keeping this in mind, it becomes worthwhile to discuss the importance of emotion in robots because all robots possess goals (even if only implicitly), and many robots also possess the capacity to arbitrate amongst and perform

actions in pursuit of those goals [Fellous and Arbib, 2004]. This is not meant to imply that the idea of emotion in robots must parallel human-centric concepts of emotion (e.g., the "happy/sad" states described by Damasio [1994] and Ekman [1992]). Rather, robot emotion should be investigated from the perspective that certain biological processes are known to facilitate real-time, adaptive decision making and commitment and that many of these processes have direct correlates to robot control. Ultimately, the degree of "emotion" possessed by any robot will be, in part, defined by both the utility and reward structures implicit in that robot's design, as well as the levels of architectural control and processing required for that robot to be successful within its "ecological" niche [Sloman, et al., 2004] [Arbib and Fellous, 2004]. Due to the fundamental differences between robots and humans, and the goals each pursue, it is highly unlikely that the final emergent control states, which in humans are often classified as emotions, could be labeled in robots using the same generic terms. Therefore, while this dissertation discusses emotion, the key concepts that should be kept in mind are those related to the functional purpose of specific emotional states, the mechanisms that provide that purpose, and how each mechanism may apply to robot technology.

Area of Investigation

In biology, the emotion system evolved to enable adaptive, real-time control in complex environments [Arbib and Fellous, 2004]. This system often operates as an innate reinforcement mechanism, but also integrates aspects of cognitive decision making with low-level control [Rolls, 2004] [Damasio, 1994] [Pfister and Böhm, 2008]. The cognitive processing that subserves emotional states also enhances control by not only forcing innate responses, but by attaching utilities to actions within the planning cycle, focusing attention, signaling urgency, and measuring error [Rolls, 1999] [Zeelenberg and Pieters, 2006] [Frijdja, 1986] [Scherer, 1997]. These processes can be both automatic and controlled: automatic operations are often used to appraise relevance, urgency, or utility, while controlled operation measure error and perform *post hoc* evaluations and reflections [Baumeister, et al., 2007] [Richter-Levin, 2004]. Furthermore, some of the evaluations that underlie emotion provide a means to collapse complex potential outcomes onto a common currency scale that can be used for deliberative cognitive control (e.g., predicting and planning) [Rolls, 1999] [Slovic, et al., 2003] [Ortony and Turner, 1990]. Here, and throughout this dissertation, the term *cognitive control* refers to the type of

executive control defined by many psychologists and neuroscientists in which top-down executive processes utilize attention and working memory, planning and internal rehearsal,error correction, and novelty detection to purposefully respond to complex situations [Posner and Snyder, 1975] [Botvinick, et al., 2001]. While the influence of emotion on cognitive control, at times, may yield negative results (e.g., losing one's temper), research suggests that emotion and emotion-based processes, as a whole, are more adaptive than maladaptive [Damasio, 1994] [Bechara, et al., 1997].

This dissertation investigates how theories of emotion and, specifically, the cognitive processing and appraisals that enable emotion can be applied to a cognitive robot to improve task performance. The focus of this dissertation is *not* on the development of a new emotion model for control or to make the robot *externally* appear emotional. Innovative work in these areas can be found in [Arkin, 2004] [Breazeal, 2002] [Breazeal and Brooks, 2004] [Gockley, et al., 2006]. Rather, emotions are approached from the perspective that they provide goal-contingent and situation-based evaluations of *functional* importance to the decision-making process. This involves processing both the current situation and past experience with respect to: what is *relevant* and *urgent*, how much *utility* should be attached to specific responses, and how well current knowledge and response capacities *fit* the situation. There will be three simultaneous aspects to this approach:

1. Processing and mining experience, stored as episodic memory, for relational information that can be used to derive situation-based appraisals;
2. Representing the mined relations and appraisals for use in online decision making;
3. Integrating these appraisals within the control process.

Contribution of Work

The contribution of this work will be to investigate how theories of cognitive processing can be used to mediate decision making in order to enable appropriate online performance in complex situations. This will entail investigating flexible methods for *representing* experience, initially stored as episodic memory. Once represented, it will be important to develop relational structures that allow the current situation to be matched against experience and associations to be formed between individual experiences and the various appraisals they entail. These appraisals must be integrated into decision making in a manner that extends beyond simply inserting utility

values. It is necessary to investigate *when* and *how* parameters within the decision-making process should be dynamically tuned in order to facilitate real-time responsiveness while maintaining adequate solution quality. Finally, because this research is part of a larger study on robotic cognition, the developed system must be designed around and integrated within a complex cognitive architecture used for robot control.

Organization of Dissertation

This dissertation begins in Chapter II with a discussion on cognitive robotics, functional aspects of cognition, and cognitive architectures. The goal of Chapter II is to review the fundamental concepts of cognitive robotics and cognitive architectures and provide the necessary backdrop upon which this dissertation research will be developed. Chapter III reviews sequential decision-making processes, such as MDPs and POMDPs, the methods used to evaluate these processes, and the techniques for modifying these methods in order to obtain real-time operation. Whereas Chapter II discusses cognitive robotic control, Chapter III describes specific techniques by which deliberative, and ultimately cognitive control may be realized, while also presenting the limitations of current control methods. Chapter IV reviews psychological and neuroscientific theories on emotions, as well as theories that view emotion as being comprised of specific appraisals and evaluations based on the cognitive processing of events and concerns. Chapter IV then describes specific implementations in which theories on emotion have been used within robotic control applications.

Chapter V combines the discussions of Chapter's II, III, and IV and describes how ideas and theories from each chapter may be integrated and used to inform system design. This chapter then discusses the implemented system and relates each component back to the engineering and psychological theories used in its design. Chapter V also provides overall system layout, as well as how the system is specifically realized through the ISAC cognitive architecture [Kawamura, et al., 2008]. Chapter VI describes the grocery-bagging experiment used to test the developed system, and presents the results of both simulation and integrated hardware experiments, as well as the discussion of results. Chapter VII offers final thoughts, and providing directions for future research. Two appendices conclude this dissertation: Appendix A describes the variables and functions used to implement the designed system, while Appendix B provides some example code for using these variables and functions.

CHAPTER II

OVERVIEW OF COGNITIVE ROBOTICS

Many well-defined, single task problems have been studied by applying targeted, task-specific solution methods. In robotics, such examples include grasping [Grupen and Huber, 2005], navigation [Koku, et al., 2003], and obstacle avoidance [Arkin, 1998]. Each example falls under the general category of techniques known collectively as *artificial intelligence* (AI): a scientific discipline which provides the necessary backbone for creating intelligent systems that can act, react, and adapt in different environments. Research in AI began in the 1950's with development of computer systems and computer programs [McCarthy, 1959]. Over the years, there have been a number of different approaches to the study and creation of artificially intelligent systems. Early work focused on symbol manipulation and logical computation. Research has since expanded to include sub-symbolic connectionists systems, reactive systems, and fuzzy systems, to name a few. Recently, however, a new approach has begun to emerge that is aimed at investigating how more general, cognitive-level behavior and control may be produced in "embodied" agents, such as robots. Termed *cognitive robotics*, this approach focuses on organizing and understanding how different components can be used to create a functioning whole, while utilizing psychological and neuroscientific research on biological cognitive systems to inform system design. Cognitive robotics is *not* an alternative to AI, but an attempt at the next step towards more general intelligent systems capable of approaching human cognition.

As this researcher sees it, there are three major tenets to cognitive robotics. These tenets reflect themes that recur frequently in the literature on artificial cognition and cognitive robotics, but are by no means intended to be an exhaustive list. The first tenet of cognitive robotics is to study *how* the tools developed for targeted AI systems can be integrated and organized in order to create more adaptive, general-purpose systems. Organization is key in cognitive robotics. The second tenet is the notion that in order to create artificial cognitive systems, these organized components should reflect those identified by psychological and neuroscientific research as necessary (and at times sufficient) for cognitive ability. The third tenet is that a truly cognitive system must be situated in the real-world, and thus be required to cope with dynamic, unstable,

and noisy environments in which appropriate responses must be chosen using limited resources in a time critical fashion.

One popular method for specifying the organizational scheme for a cognitive robot is through the use of a *cognitive architecture*, which diagrams how each cognitive component integrates and interacts to produce behavior. Yet cognitive architectures need not always be used for robot control; many architectures have been developed as models for investigating human cognition, or to implement cognitive abilities on non-robotic system [Anderson, 1983] [Newell, 1990] [Sloman, 2001a]. Furthermore, not all cognitive robotic applications employ full cognitive architectures; many approaches focus on only a subset of cognitive abilities, and thus limit their research to less expansive architectural designs [Shanahan, 2006] [Krichmar and Edelman, 2005] [Beer, 2000].

Components of Cognitive Systems

There are a variety of mechanisms believed to be important for true cognitive functionality. A few of these include: attention, working memory, long-term memory, and executive control. While this is not an exhaustive list, it does highlight many of those components commonly found in the different approaches to artificial cognition. This section provides a brief discussion of such mechanisms.

Short-Term Memory, Attention, and Working Memory

In order to utilize sensory information, an organism must have a means of actively maintaining that information for some duration of time once the information has been detected. The initial buffer that maintains perceptual information in biological cognitive systems is commonly referred to as *sensory memory*. All sensor inputs pass through sensory memory; thus, this buffer is believed to be of unlimited capacity (practically speaking) [Sperling, 1960]. However, retention periods for sensory memory are extremely short, and information not immediately attended to is "dumped" and forgotten [Purdy and Olmstead, 1984].

Information retained from sensory memory is passed on to *short-term memory*, a limited capacity, short-term store. Though it may vary across individuals, the capacity of short-term memory is believed to be somewhere in the range of seven to nine chunks of information [Miller, 1956]. Due to this limited capacity, short-term memory is able to retain information for much

longer periods of time than sensory memory. Research suggests that the retention duration for short-term memory is in the range of 20-30 seconds, comparably much longer than the 100-300 milliseconds duration of sensory memory [Purdy and Olmstead, 1984].

Closely related to the concept of short-term memory is the notion of *working memory*. Both short-term and working memory systems retain and manipulate a finite amount of information for short periods of time [Gathercole, 1999]; however, working memory is believed to be specifically involved in the active maintenance and cognitive manipulation of task-relevant information [Baddeley and Hitch, 1974]. One of the primary roles of working memory is to maintain information relevant to the current task, so that it can be directly accessed by the organism's other cognitive processes. There are two primary paths by which information may enter working memory: 1) external sensory information via the sensory- and short-term memory pathways and 2) highly activated or recalled information via long-term memory [Baddeley, 2000]. Each pathway into working memory is modulated by attention, a mechanism that permits only situation- or task-relevant information to enter. Functionally, working memory is situated at the three-way intersection of short- and long-term memory and the higher-order cognitive processes. Thus, working memory is in a unique position to act as the staging ground where various informational chunks become bound together to affect task performance [Baddeley, 2000].

Procedural, Episodic, and Semantic Long-Term Memory

Figure 1 provides a simple, functional illustration of the flow of information through the various memory systems. Most of the information that enters this process is eventually forgotten. However, over time and through attention, rehearsal, and repetition, information in short-term and working memory can become consolidated in *long-term memory*, where it may be retained indefinitely. Like sensory memory, long-term memory is a buffer with unlimited capacity, again, practically speaking [Landauer, 1986]. But unlike sensory memory, long-term memory is capable of retaining information for extremely long periods of time, possibly for the remainder of the organism's life. Due to its limitless capacity as well as the indefinitely long retention periods, a critical and practical issue in forming, retaining, and retrieving long-term memories is structure and organization [Norman, 2002].

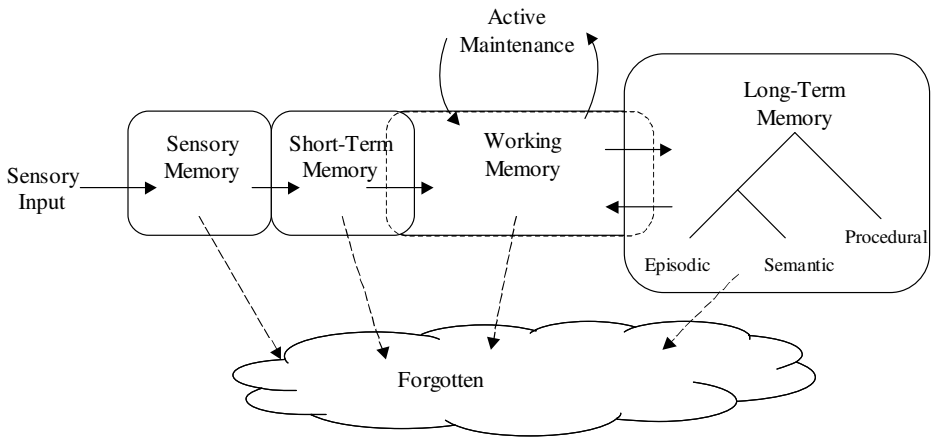


Figure 1. Flow of Information to the Various Memory Systems

Functionally, long-term memory can be divided into two main groups: *procedural* and *declarative*. Procedural long-term memory stores implicit knowledge related to the performance of skills, behaviors, or tasks. Examples include: "how to ride a bike" or "how to write your name". Research has shown, however, that this knowledge is not limited to motorized skills, but also extends to such abilities as "reading text in reverse" and "solving puzzles" [Baddeley, 1998]. Procedural memory enables organisms to perform seemingly complex tasks without having to exert equal amounts of conscious, cognitive control. Acquisition of such ability requires that information be learned through constant practice and repetition until performance becomes "automatic". Procedural memory is part of the adapative mechanisms that enable organisms to function in dynamic and often dangerous environments without the constant need for conscious, deliberative control [Logan, 1988] [Schneider, 1999].

Whereas, procedural memory stores implicit knowledge related to performance, declarative memory stores explicit knowledge for use in planning, rehearsal, and deliberation. There are two main sub-systems in declarative memory: *episodic* and *semantic* [Baddeley, et al., 2002]. The distinction between these two sub-systems has been suggested to be analogous to the difference between "remembering" and "knowing" [Tulving, 2002], where episodic memory stores *remembered* information, and semantic memory stores *known* information. According to Tulving [2002] and Gardiner [2001] remembered information is that which can be recalled within the context in which it was originally stored. Examples may include: "your wedding day", "the day you graduated from school", or "what you did this morning". Therefore, episodic

memories, or episodes, are those events that can be recalled in combination with the various contextual details specifically related to the experienced event. Likewise, knowing that "Paris is the capital of France" or that "the sum of the square roots of any two sides of an isosceles triangle is equal to the square root of the remaining side" [Baum, 1900] are examples of semantic memory, where such information is simply often known, or believed, and the context in which it was originally stored cannot be recalled. Thus semantic memory retains facts and beliefs learned or mined from a lifetime of experience, yet not specifically associated with any single experience.

Episodic memory retains contextualized, subjective episodes from the rememberer's past [Tulving, 1983]. This system maintains the "what", "when", and "where" of an event [Clayton and Dickinson, 1998] [Nyberg, et al., 1996]. Therefore, information stored in episodic memory must be *experienced* [Nuxoll and Laird, 2004]. A unique property of episodic memory is that it enables an organism to mentally "travel back in time" [Tulving, 2002] and consciously re-experience previous events. Such re-experiencing allows organisms to not only re-live the sequences of events that occurred, but also to "re-feel" the emotions and other internal states originally felt during those events [Tulving, 2002]. Research suggests, however, that much of the re-experienced internal information is considerably dampened during recall [Loewenstein, 1996].

Finally, individual episodes within episodic memory have been compared to the idea of storing reels of footage within the brain, in which different frames can be stored at different levels of acuity [Tulving, 1983]. Subjectively significant features are typically stored with high acuity, while other less significant, background features are stored with less acuity, or not at all. Therefore, it is believed that episodic memory interacts closely with goal setting and evaluation processes in order to capture the significant details of an event [Burgess, et al., 2002] [Aggleton and Pearce, 2002]. As previously mentioned, working memory is also believed to interact with goal setting and deliberative cognitive processes and research suggests that the two systems (episodic and working memory) interact with each other and are highly related [Baddeley, 2000]. Specific application research has modeled this relation by using the information stored within working memory as the sole source of information for forming episodes [Nuxoll and Laird, 2004].

Executive Processes

Processes with executive functionality are necessary for goal-directed, purposive behavior. These high-level abilities perform many of the complex aspects of cognitive control, such as *deliberation*, *planning*, *outcome monitoring*, and *anticipation*. In addition, executive processes play a critical role in influencing the behavior of other components, such as working memory and episodic memory [Baddeley and Hitch, 1974] [Schneider , 1999] [Baddeley, 2000]. However, because executive processes encompass such a wide range of abilities and interact with various other components these processes are difficult to study alone, and are often studied in relation to the component processes with which they are believed to interact.

Typical applications tend to generalize executive functionality as planners and decision makers that operate on abstract symbols and concepts [Anderson, 1983] [Newell, 1990]. This parallels much of the early work in AI, in which symbol manipulation and symbolic decision making were highly popular and well studied. But there have been other conceptions of executive systems as well. Baddeley and Hitch [1974] proposed a *central executive* that interacts with working memory, providing resource management and exerting executive control over sub-components. Baddeley [2000] argued that this conception of a central executive must *not* be confused with notions of a control homunculus, or "magic box" within the human brain. Instead, the central executive should be viewed as a general learning function that is trained to perform as a meta-manager over sub-component functioning. Other conceptions of an integrated central executive and working memory system also exist [Schneider, 1999].

From an architectural point-of-view, Sloman [2001a] and Ortony, et al., [2004] both place executive functioning at the top of the information flow and control hierarchy. This is comparable to the ideas of Baddely and Hitch [1974] and Schneider [1999], but at a much more general level. The work of Sloman [2001a] and Ortony, et al., [2004] notes that it is difficult to conceptualize executive processing as a single component, and therefore the gaps in executive models must often be filled in as needed when top-down control is required. Such a view is fundamentally different than the bottom-up control afforded by reactive and hard-wired components.

Cognitive Architectures

Early cognitive architectures were designed with the specification that a working model of human cognition should be developed independently of the underlying hardware on which the architecture was implemented [Anderson, 1983] [Newell, 1990]. Therefore, while human cognition was implemented through the use of expansive neural networks, artificial models were created using various computing systems that utilized different hardware structures and software algorithms. Such architectures compartmentalized components, which enabled tests to be performed on specific functions and system performance to be analyzed under a variety of conditions.

Symbol Processing Systems

Two of the most well-known cognitive architectures were developed based on the *physical symbol-system hypothesis* [Newell and Simon, 1963]. This hypothesis proposed that the ability to perform symbolic computation and manipulation was both necessary and sufficient for the creation of artificial cognition. These two architectures are the ACT family of architectures [Anderson, 1976] [Anderson, 1983], of which ACT-R [Anderson and Lebiere, 1998] is the most recent, and the Soar family of architectures [Newell, 1990] [Lehman, et al., 2006]. Each approach is distinct with respect to many of the implementation details, but at a more abstract, functional level both architectures are laid-out using very similar components: long-term and working memory, as well as perceptual and actuation systems (Figure 2).

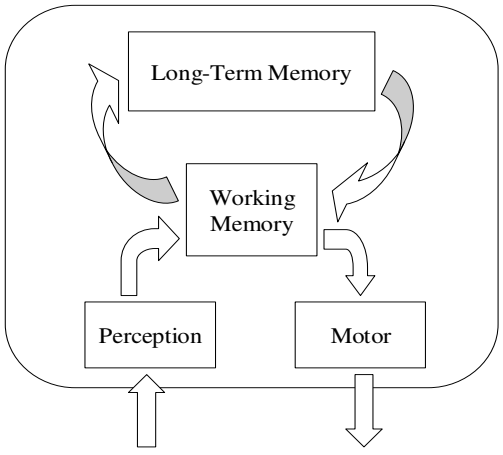


Figure 2. General Layout of Early Production Systems (i.e., ACT-R and Soar [Anderson and Lebiere, 1998] [Newell, 1990])

Both ACT-R and Soar are members of a class of architectures known collectively as production systems, in which procedural *if-then* rules are used to operate on symbolic data. Each system perceives the environment and stores the perceptual information in a global working memory buffer. The contents of working memory are used to activate information in long-term memory, which may then be recalled and placed in working memory. The procedural rules operate on the contents of working memory, potentially retrieving new information when required. Over time, information in working memory also triggers actions, which modify the environment and thus the process continues.

At a certain level of abstraction both ACT-R and Soar operate in a similar fashion. Below this level of abstraction, however, the details of each architecture are quite different. ACT-R divides long-term memory into procedural and declarative systems, in which procedural memory retains production rules while declarative memory retains semantic information "chunks" [Anderson and Lebiere, 1998]. Within ACT-R, declarative information is recalled through the use of a spreading activation network in which each information chunk has an associated activation value and is connected by weights to all other chunks. If two chunks are strongly associated, for example "water quenches thirst" and "cups hold water", then the weight between these chunks should be high and the presence of one chunk in working memory should cause the activation and retrieval of the second chunk. Once chunks are retrieved they can be used as conditionals to trigger production rules or to retrieve more chunks. Production rules either cause actions to be taken on the environment or chunks to be created/deleted. Learning in ACT-R is accomplished by modifying the association strengths (i.e., weights) between different chunks, as well as by updating the cost and reward values associated with each production rule.

Unlike ACT-R, the original Soar system did not differentiate between types of long-term memory; however, more recent work has divided Soar's long-term memory into procedural, episodic, and semantic sub-systems [Lehman, et al., 2006]. Soar is a goal-based system in which all tasks are formulated as goals that must be met. When Soar is faced with a task, it creates a *problem space* for representing the potential states that may arise during task execution. The problem space describes the effects of performing different operations and actions in the current state. During deliberation, Soar repeatedly fires production rules using the information in working memory, and then selects states to pursue using specialized operators. The behavior of the Soar system can be visualized as movement through the problem space.

When an impasse is reached, Soar creates a sub-goal to resolve the impasse and operation iteratively continues within the problem space for the new sub-goal. Once the sub-goal is achieved, Soar creates a new production rule to summarize the solution process, and thus learning is performed. Within the Soar architecture this learning process is known as *chunking*, but is conceptually distinct from the chunk representations used in ACT-R.

Soar and ACT-R are some of the most well-known and well studied rule-based cognitive architectures but there are others, such as EPIC [Keiras and Meyer, 1997] and Prodigy [Carbonell, et al., 1990]. EPIC is another production rule system that utilizes a global working memory. However, in place of the basic perceptual and actuation systems that ACT-R and Soar use, EPIC has a rich set of perceptual-motor peripherals that provide input/output capabilities and dictate various physical constraints and limitations with which the cognitive system must cope. EPIC can be viewed as a model of cognition in which performance trade-offs must be made when multiple tasks are required. Prodigy operates much like Soar in that it is primarily devoted to the single task of navigating a complex problem-space [Carbonell, et al., 1990]. Two types of rules are used by Prodigy: domain rules which model action conditions and effects, and control rules which dictate architecture performance, such as the selection or rejection of states. Prodigy also deploys explanation-based learning in order to re-use past experience for the current problem. Within the Prodigy system, this functionality is achieved through case-based techniques that attempt to solve current problems using previous problems as examples, or analogies [Veloso and Carbonell, 1993].

Connectionist Systems

The architectures just described were largely based on the physical symbol system hypothesis and were therefore designed as symbolic production systems. This is due, in no small part, to the fact that much of early AI was devoted to the study of symbolic approaches for intelligence. However, symbol manipulation is only one piece of the puzzle. Another equally important piece is the ability to use and manipulate sub-symbolic information. Sub-symbolic approaches generally use highly interconnected networks that pass numeric data in an attempt to re-create some of the vast complexity of the human neural system. Unfortunately, it is extremely difficult to create full cognitive architectures using purely connectionist systems and, therefore, these systems have been much less studied than the symbolic approaches. One reason for this

may be that complex, interconnected networks provide little *discernable* insight into how their performance may be understood from a psychological standpoint.

Additionally, from a more pragmatic engineering point-of-view, full connectionist systems do not model cognition well because of the large amounts of training examples needed to learn basic concepts. And once a concept has been learned, the underlying network often becomes brittle and will not learn new concepts without forgetting the old. There has been, however, some very interesting research in this area [Thrun and Pratt, 1998].

Brain-based devices (BBDs) [Krichmar and Edelman, 2005] are computational approaches to cognition that attempt to simulate the functionality of the nervous systems found in biological agents. BBDs possess neural dynamics and selection principles that allow them to appropriately adapt to their environment in a manner similar to the dynamic systems approaches investigated by Beer [1995] [2000]. Because the architectural design of a BBD is based on current neuroscientific understanding of the brain, these systems offer a unique ability to compare the results of simulated neural activity to collected experimental data [Krichmar and Edelman, 2005].

Architecturally BBDs are pure connectionist systems in which functions have, generally, not been compartmentalized. In other words, there are no separate components designed *a priori* to store memories, plan, or perform actions. However, there are different sub-collections of artificial neurons that are intended to model particular regions of the brain and may thus become associated with the activities believed to occur in those regions. The connections between subsystems are loosely structured and are allowed to organize themselves over time. Such self-organization occurs through active learning in the environment while the BBD is engaged in behaviors intended for the completion of a task. As the system learns, the connection weights between different neurons are updated via Hebbian learning [Hebb, 1949], a process that strengthens connections between simultaneously active neurons and weakens connections between inactive sets. In addition, a value system is used to modulate active connections when salient sensory events occur.

Recent Hybrid Approaches

Symbol manipulation and sub-symbolic computation are both useful techniques for the study of artificial cognition. Therefore, recent approaches to architecture design have begun to

study how both techniques can be combined to provide better cognitive models and more adaptive systems. Such architectures are known as *hybrid* architectures and have been used to model the various levels of control believed to be available in humans and other mammals.

Based on the notion of controlled and automatic processing, CAP2 [Schneider and Chein, 2003] is a hybrid architecture that models two levels of control. The first level is composed of traditional sub-symbolic (e.g., neural networks) functions. This level requires large amounts of consistent training in order to be effective, but once trained it operates automatically reacting quickly to stimuli and behaving like a general auto-associative mechanism [Schneider and Chen, 2003]. The second level is the controlled level that performs deliberation and planning. This level is composed of connected sets of sequential processes that are designed to be functionally similar to the production-rule processing systems of Soar and ACT-R. In CAP2, the controlled processes are used in novel situations in which automatic responses have not yet been learned and, thus, problems must be solved without prior training examples.

A second hybrid cognitive architecture is the RCS (Real-time Control System) approach of Albus and Barbera, [2004]. Early research using this architecture focused on low-level, real-time control systems [Barbera, et al., 1979], but has evolved to include production rule systems and declarative knowledge structures [Albus, 2002]. Unlike some of the other cognitive architectures that were first proposed as models of cognition and then later used for robot control, RCS was specifically developed for robotic platforms. All symbols within the world model of the RCS system are grounded to signals, objects, and states arising in the physical environment. RCS is organized hierarchically using various functional processing nodes at each level. Each node consists of five basic elements: behavior generation, world modeling, sensory processing, value judgment, and knowledge database. Such node design enables RCS to distribute control while maintaining a strict control hierarchy; both are features required in a cognitive architecture that marries low-level control operations to high-level cognitive functions [Albus and Barbera, 2004].

The Comprehensive Human Intelligence and Performance (CHIP) architecture [Shrobe, et al., 2006] is a multi-level hybrid architecture that provides a good illustration of a three-layer design that incorporates reactive, deliberative, and reflective control. At each successively higher layer, the control processes simultaneously become slower and yet capable of handling more complex problems. In addition to the three layers of control, CHIP also uses the sequential *sense-*

plan-act (SPA) architectural schema common to robotic control systems. The CHIP architecture can be represented generally using a 3x3 grid as shown in Figure 3.

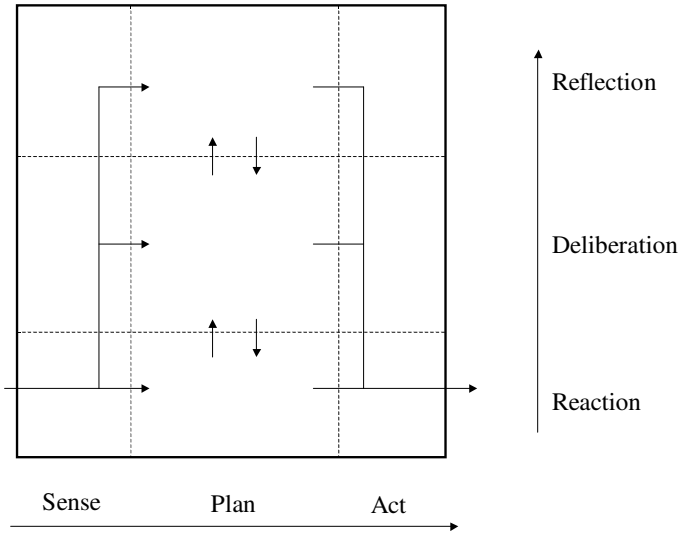


Figure 3. Basic Model of 3x3 Architecture

CHIP is one of many architectures that trifurcate control in this way [Sloman, 2001a] [Kawamura, et al., 2008] [Ortony, et al., 2004]. In each of these architectures, the lowest level of control performs reactive or reflexive responses (e.g., an emergency stop before a collision). The next level of control varies slightly from architecture to architecture but is generally responsible for performing simple deliberation or routine skills. This primarily involves control just beyond that afforded by simple reactions (e.g., following a path or grasping a familiar object) [Ortony, et al., 2004]. The final layer performs the most complex form of control, which is often some form of complex deliberation, reflection, or meta-management. This level of control is typically capable of generating and analyzing plans, arbitrating among abstract response options, biasing parameters in the lower control levels, and performing *post hoc* evaluations.

CHAPTER III

OVERVIEW OF SEQUENTIAL ROBOTIC DECISION-MAKING PROCESSES

As mentioned at the beginning of Chapter II, one of the fundamental aspects of cognitive robotics is organization. Cognitive architectures are organized layouts for robotic control that specify how components should interact at the macro-level. Each of the cognitive architectures described in the previous chapter required at least one component for deliberation and planning. This chapter will begin by describing two popular models used within AI for representing complex, sequential decision processes: the Markov Decision Process and Partially Observable Markov Decision Processes. Methods for using these models will be discussed as will methods that enable real-time performance for each technique by focusing the search through the decision state space or allowing anytime interrupts. Finally, methods for approximating solutions in large, or continuous, state spaces will be described.

Throughout this chapter it is important to note that each solution method involves tradeoffs: standard approaches require large amounts of memory for maintaining tabular state-value functions, function approximators discretize and reduce the size of the state space but may possibly ignore relevant information, and real-time search and anytime algorithms require many performance parameters to be preset as well as, typically, requiring additional and costly heuristic functions.

Markov Decision Processes

One of the most common techniques for representing sequential decisions in a stochastic domain is through the mathematical framework known as a Markov Decision Process (MDP). The standard representation for a MDP is the tuple $\{S, A, T, R\}$. Here, S is represents the finite set of discrete states that an agent can occupy and A represents the set of all actions available to that agent. The effects of taking action $a \in A$ in state $s \in S$ are modeled using the state transition function, T: $S \times A \rightarrow S$. The transition function maps state-action pairs onto successor states with a given probability. Typically this function is written as $T(s, a, s')$ and returns the probability of occupying state s' after performing action a in state s. A fundamental assumption of this type of transition function is that all transitions obey the Markov property: the effects of taking an action

depend only on the current state, and not on the entire history of prior states. Finally, R is a reward function that maps states to a specific numeric reward, $R: S \rightarrow \mathbb{R}$.

For a given MDP, a policy π is known as a solution to that MDP and determines which action should be taken in each possible state. Thus, the output of $\pi(s)$ is an action, $a \in A$, to be performed in state s. Given a MDP and π, it is possible to determine the expected value for reaching any state within the S. This value, denoted $V(s)$, is the expected cumulative reward if policy π is followed indefinitely. $V(s)$ can be found by solving the recursive Bellman equation given in Equation (1). The parameter γ is a discount factor used to focus $V(s)$ on more immediate returns.

$$V^{\pi}(s) = R(s) + \gamma \cdot \sum_{s' \in S} T(s, \pi(s), s') \cdot V^{\pi}(s') \tag{1}$$

For every MDP there is at least one policy, π^*, that maximizes $V(s)$ such that $V^{\pi^*}(s_j) \geq V^{\pi_i}(s_j)$, $\forall\, i$, $\forall\, s_j \in S$. Such a policy is called the *optimal policy* and is the "best" that an agent can expect to do given its reward function. Equations (2) and (3) can be used to determine π^*, where at each step actions are chosen to maximize $V(s)$.

$$V(s) = R(s) + \max_{a \in A} \left(\gamma \cdot \sum_{s' \in S} T(s, a, s') \cdot V(s') \right) \tag{2}$$

$$\pi^*(s) = \arg\max_{a \in A} \left(\sum_{s' \in S} T(s, a, s') \cdot V(s) \right) \tag{3}$$

Methods to Solve Markov Decision Processes

There are two popular, fundamental techniques for solving MDPs: *value iteration* and *policy iteration* [Russell and Norvig, 2003]. Value iteration uses a method known as *dynamic programming* [Bellman, 1957] to recursively solve Equation (2). The algorithm begins by initializing $V_0(s)$, $\forall s \in S$, to arbitrary initial values. At each subsequent step, updated values $V_i(s)$ are determined from $V_{i-1}(s)$, as shown in Equation (4). Value iteration repeats until the difference

$\sum_{s \in S} \left\| V_i(s) - V_{i-1}(s) \right\|$ falls below a given threshold. Equation (3) can then be used to return the optimal policy.

$$V_i(s) = R(s) + \max_{a \in A} \left(\gamma \cdot \sum_{s' \in S} T(s, a, s') \cdot V_{i-1}(s') \right) \tag{4}$$

Like value iteration, policy iteration is an iterative recursive algorithm. Policy iteration begins with an arbitrary initial policy π_0, but then calculates $V^{\pi_0}(s)$, $\forall s \in S$. The resulting value function is used to calculate a new policy π_1, and this process continues until $V^{\pi_{i-1}}(s) = V^{\pi_i}(s)$, $\forall s \in S$. At this point, no further improvements are possible and the final π_{i+1} is optimal.

Partially Observable Markov Decision Processes

Markov Decision Processes are simplified decision models for stochastic domains. In a MDP only the effects of performing an action are uncertain. However, in many domains it is not only actions that are uncertain but also observations. This is especially true for robotic applications. Sensor error, incomplete domain knowledge, and simple deficits in processing ability can impart a great deal of added uncertainty to the decision-making process. Because the agent cannot know for certain which state it is in, it must maintain the belief that it is in a particular state. The current belief state is updated by determining the likelihood of an actual state given all of the information that has been observed. To incorporate observational knowledge the traditional MDP framework is extended to: {S, A, T, R, Ω, O}. This representation, known as a Partially Observable Markov Decision Process (POMDP) [Sondik, 1971] includes the standard MDP model as well as a set of possible observations, Ω, and the probability model, O, which maps states and actions to observations, O: $S \times A \rightarrow \Omega$. The techniques for solving POMDPs are mathematically analogous to those for solving MDPs; however, the additional complexity from including probabilistic observations and belief states typically exacerbates the computational capabilities of most systems. Solving POMDPs is PSPACE-hard [Papdimitriou and Tsisiklis, 1987], and therefore POMDPs are usually applied only to small environments.

Real-Time Search

The bulk of current decision-making research in both artificial intelligence and robotics focuses on the identification of optimal solutions. In other words, given a domain specified as a MDP the goal is to determine π^*. To do this, dynamic programming techniques such as *value iteration* and *policy iteration* are available. When using these techniques agents typically maintain a tabular list of possible states along with associated values, or actions. Decision making requires that the agent merely consult these lists and select the best option. Unfortunately, such a process often requires a considerable amount of computational effort; and thus the derivation of most optimal policies is performed offline. But if the environmental stochastics change, then policies computed offline can quickly become sub-optimal leading to disastrous consequences. Furthermore, as domain complexity increases, the size of the associated state space tends to increase exponentially. This is known as the "curse of dimensionality" [Bellman, 1957] and places considerable stress on an agent's ability to maintain and reference an optimal policy. While exponential increases in state space size are never desirable, in robotic applications such increases can be especially problematic due to the need to operate in the real world in real-time. Therefore, many robotic applications employ real-time algorithms that only partially search the state space before committing to an action [Korf, 1990] [Paquet, et al., 2005] [Koenig and Likhachev, 2006].

Real-time search techniques are techniques that enable an agent to operate efficiently when time-critical decisions are necessary. Many of these techniques either perform a limited amount of local search before selecting an action or attempt to focus the search on the most relevant states [Geffner and Bonet, 1998]. For example, consider the state-action space shown in Figure 4. Using traditional methods an agent, such as a robot, would have to compute an expected value, $V(s)$, for every state before choosing which action to perform. Such a process could require several minutes or hours to perform. During this time, the environment may change requiring either a complete re-computation of a new policy or the continued use of a sub-optimal policy. However, an agent using real-time search techniques might only focus on the local state-space (see subset of Figure 4), and compute a partial policy that can be applied immediately. Such an agent is better suited to cope adaptively with a changing environment.

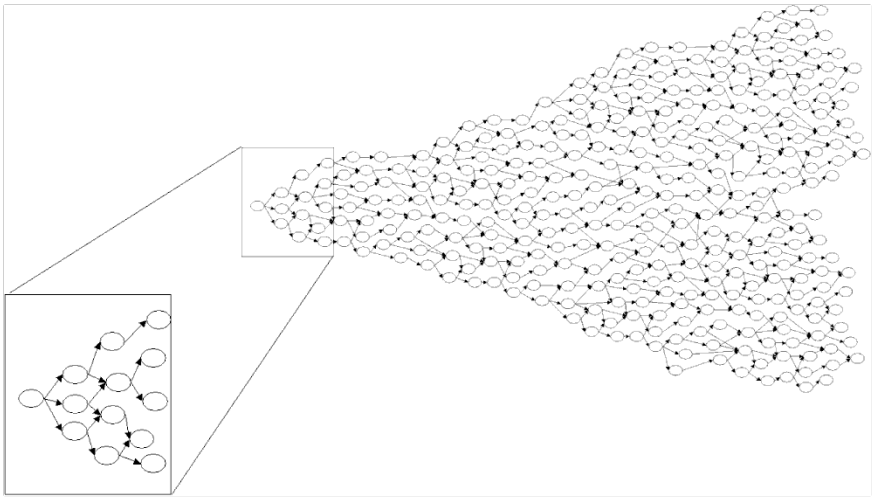


Figure 4. State Space, *S*, with Local Search Window

A variety of techniques have been proposed to perform real-time search. Each of these techniques rely on embedded domain knowledge, such as heuristic evaluation functions, and many still require an explicit listing of state values (i.e., *V(s)*). For example, one of the simpler real-time search techniques requires a preset heuristic function, *h(s)*, which it uses to construct the value function *Q(a, s)* [Watkins, 1989]. Unlike *V(s)*, *Q(a, s)* explicitly maintains the expected value for performing *a* in *s*, which combines the cost of performing an action, *c(a, s)*, and a heuristic estimate *h(s)*. To perform real-time search, actions are chosen to maximize Equation (5) [Geffner and Bonet, 1998].

$$Q(a, s) = c(a, s) + h(s) \tag{5}$$

While computationally simple, this technique does not assure that an agent will eventually reach its goal. Without performing updates and looking just one step ahead, an agent may easily become trapped within a local loop. A modification to this approach, proposed in Korf [1990], allows an agent to avoid such situations by updating the heuristic estimates after each action. This method, known as Learning Real Time A* (LRTA*), uses the heuristic function to provide initial estimates for $V_0(s)$, but at each time-step, *t*, $V_t(s)$ is updated using the known cost value and the previous state value, as shown in Equation (6).

$$V_t(s) = c(a, s) + V_{t-1}(s_a), \tag{6}$$

where

$V_0(s) = h(s),$ (7)

$s_a = Successor(a, s)$ (8)

In LRTA* actions are still chosen by maximizing the right-hand side of Equation (6), but performing updates on $V_t(s)$ simultaneously enables an agent to avoid local loops and performance to converge to optimal as the amount of experience tends to infinity [Korf, 1990]. To determine s_a, LRTA* uses a predefined successor function, as shown in Equation (8). This approach, however, does not incorporate the fact that the effects of actions are often probabilistic, and thus LRTA* ignores the state transition function developed for MDPs.

A method known as Real-Time Dynamic Programming (RTDP) [Barto, et al, 1995] extends LRTA* by incorporating *expected* state values into Equation (6). RTDP is better equipped to operate in domains modeled with traditional MDPs while still maintaining the real-time search and optimal convergence properties of LRTA* [Barto, et al., 1995]. The modified equation for RTDP is shown in Equation (9). As with LRTA*, initial estimates of $V_0(s)$ are provided by the preset heuristic function $h(s)$.

$$V_t(s) = c(a,s) + \gamma \cdot \sum_{s' \in S} T(s,a,s') \cdot V_{t-1}(s') \quad (9)$$

The Focused Dynamic A* algorithm, known as D*, is another real-time search technique that utilizes heuristic knowledge to determine optimal paths through the state space [Stentz, 1995]. D* is designed to cope with environments in which transition costs change over time. The initial policy determined by D* is developed offline, under the assumption of a static environment, but as repairs to this policy are needed (due to the failure of the static environment assumption), D* quickly performs local "patches" by focusing on only the most relevant states. This is achieved by sweeping backwards from the goal state to the current state, updating just those states affected by the new information. Due to its efficiency D* has been used on a variety of robotic platforms [Matthies, et al., 2000] [Thayer, et al., 2000].

An algorithmically similar approach to Stentz's D* has been developed by Koenig and Likhachev [2002]. Their approach, known as D* Lite, performs an incremental search and is based on their notion of lifelong planning, in which cost estimates are carried forward from one

search to the next [Koenig and Likhachev, 2001]. Like D*, D* Lite works backwards from the goal state to the current state and performs updates when the transition cost between states has changed. More recently, Koenig and Likhachev have proposed another variation on real-time search known as Real-Time Adaptive A* (RTAA*) [Koenig and Likhachev, 2006]. RTAA* is an *anytime* algorithm that performs a depth-limited local search, similar to that depicted in Figure 4. During look-ahead, heuristic updates are performed using knowledge of the accumulated transitions costs for particular states. The update procedure is shown in Equation (10). Here $g(s)$ represents the accumulated cost for reaching state s, and s' is the next state to be expanded [Koenig and Likhachev, 2006].

$$h(s) = g(s') + h(s') - g(s) \tag{10}$$

In addition to being applicable to MDPs, real-time search techniques can also be applied to POMDPs. A method known as Simple Online Value Iteration (SOVI) is an online search technique developed by Shani, et al., [2005]. SOVI is an extension of the Heuristic Search Value Iteration (HSVI) algorithm of Smith and Simmons [2004], which is an offline technique for finding approximate solutions to POMDPs with large state spaces. Both HSVI and SOVI maintain upper and lower bounds for the value estimates $V(s)$. In each approach, the upper bound is used to direct exploration while the lower bound is used to form the current policy. As experience is acquired, both methods update these bounds to improve performance; however, SOVI improves over HSVI in that the latter technique uses computationally expensive methods (such as linear programming) to update these bounds while the former technique relies on computationally simpler methods. SOVI updates the upper bound on $V(s)$ by directly using the heuristic function. Updates for the lower bound are determined using depth-limited prioritized sweeping, a learning method described in Sutton and Barto [2000]. SOVI further decreases computation time by only focusing on belief states that have been encountered in the current episode, rather than applying updates to all possible belief states [Shani, et al., 2005].

A similar real-time search technique for POMDPs is the Real-Time Belief Space Search (RTBSS) method developed by Paquet, et al., [2005]. This is another depth-limited search technique that relies on heuristic knowledge to direct the search process. In addition to guiding the search order, RTBSS also uses its heuristic knowledge to prune undesirable branches from

the state space. Performing only a local search as well as pruning branches enables RTBSS to function in highly complex domains while still performing online, deliberative decision making. But unlike many of the methods discussed thus far, RTBSS does *not* improve its value estimates over time. This makes the algorithm heavily dependent on the quality of its task-specific heuristic function, but also affords RTBSS total flexibility when deployed in previously unseen environments [Paquet, et al. 2005].

Anytime Algorithms

Anytime algorithms are another approach to real-time decision making that share much in common with the real-time search techniques just described. An algorithm is said to have the *anytime* property when it exhibits a trade-off between the quality of a solution and the time required to produce that solution [Zilberstein, 1996]. In addition, in any anytime algorithm solution quality should be a monotonically increasing function of both time and the quality of the input [Haddawy, 1996] [Dean and Boddy, 1988] [Horvitz, 1987]. In other words, as computation time increases, the quality of the solution must not decrease. Furthermore, many anytime algorithms allow computation time to be specified as a parameter or provided as an interrupt signal. Interruptible anytime algorithms are those that can be stopped at any point and return a solution, while contract algorithms must know the time allocated for the current decision in advance [Zilberstein and Russell, 1995].

Naturally, there are various metrics that can be used to measure the quality of a solution produced by an anytime algorithm. Three such metrics discussed by Zilberstein and Russell [1995] are: *certainty*, *accuracy*, and *specificity*. Certainty is used to indicate the degree of fit, or belief, that a particular solution is correct. Anytime algorithms that use this metric should provide solutions that are more certain to be correct when given more time to find those solutions. Accuracy is used to indicate the difference between the current "approximate" solution and the optimal one. As computation time tends towards infinity the difference between the current solution and the optimal one should converge to zero. Finally, specificity is used to indicate the level of detail present in the solution. With more time, such algorithms should return solutions in which finer levels of detail have been filled in [Zilberstein and Russell, 1995].

Research in Haddawy [1996] describes an anytime algorithm that performs a "rational refinement" of its policy over time using the specificity metric. The system constructs a policy at

the most abstract level and at each subsequent iteration the algorithm replaces a portion of the abstract plan with the next lowest level of abstract information [Haddawy, 1996]. Similarly, Horsch and Poole [1998] construct an anytime algorithm that continually refines the policy, represented as a decision tree, through the incremental inclusion of more information. The approach of Horsch and Poole, however, relies on a heuristic function to avoid considering all possible extensions to the current decision tree.

The RTBSS algorithm mentioned earlier [Paquet, et al., 2005] is another anytime algorithm that uses a depth-limited search to focus computation on reachable states. In order to maintain the monotonically increasing aspect of an anytime algorithm, RTBSS relies heavily on the preprogrammed heuristic function, $h(s)$. A similar depth-limited search is described by Dearden and Boutilier [1994] in which planning and action execution are interleaved in order to limit future searches to only those states that actually occurred as a result of actions. Like RTBSS, the work of Dearden and Boutilier [1994] assumes the presence of a heuristic function that provides value estimates for states and enables pruning.

Dean, et al., [1993] developed an anytime algorithm that, in some ways, is a composite of the approaches of Dearden and Boutilier [1994] and Horsch and Poole [1998]. The approach taken by Dean, et al., [1993] is to recursively define an envelope of states ε, determine an optimal policy for the states within ε, and then to add the fringe states (or states that can be immediately reached from any state within the envelope) to ε. After each expansion, a new optimal policy is determined using the previous optimal policy and the new states.

As the environments in which robots are deployed become increasingly more complex, the need for online decision-making methods that can operate under time constraints becomes crucial. Real-time search techniques and anytime algorithms have proven to be efficient tools for performing this type of time-critical decision making. These methods have been developed for both standard MDPs as well as the computationally more complex POMDP framework. However, with each of these techniques the quality of the results is critically dependent upon the quality of the heuristic knowledge, and the burden is often on the programmer to provide appropriate and sufficient heuristic functions. Furthermore, many of these techniques require that a tabular listing of states be maintained that can later be referenced for necessary value function information. As environmental complexity continues to increase, accessing such tabular listings

will also prove to be infeasible and intractable. It is necessary that more general abstraction and function approximation methods be developed.

Learning Functional Models

The methods such as value iteration and policy iteration discussed at the beginning of this chapter assumed that the value function *V(s)* was known. Some of the later techniques, such as LRTA* and RTDP, assumed that the system possessed an initial guess for *V(s)* but then attempted to learn, during online task execution, a better approximation for this function. These systems used the observed rewards to update the estimates of *V(s)* for the current state and occasionally for a finite number of previous states: $\forall s' \in S \mid T(s', a, s) \neq 0$. In general, approaches that use acquired reward observations to update the parameters of the system's value estimates are known as *reinforcement learning systems*.

In reinforcement learning the desire is to learn the value of occupying particular states. Once this value is known, any of the previously described methods can be used to select the action that leads to the state with the highest *expected* value. In order to learn the value function, states are repeatedly sampled from the environment, rewards are received, and the value function is updated. Such direct sampling methods are collectively known as *Monte Carlo* methods, and are typically used to update the value estimates for the sampled states only [Sutton and Barto, 2000]. On the other end of the spectrum are the dynamic programming techniques that attempt to recursively update the value estimates using the Bellman equation (Equation 1).

Monte Carlo methods are powerful because they enable online learning in situations in which it is not feasible to update recursively the value estimate for *all* states. Dynamic programming methods are powerful because they make more efficient use of the available data by "backing up" the new value estimate of state *s* to all other states that have a non-zero probability of reaching *s*. The recursive updates performed in dynamic programming are based on the intuition that if state *s* is known to be "good", then states that reach *s* with high probability should also be deemed as "good". However, it is computationally very expensive to perform an entire dynamic programming "back up" whenever *V(s)* changes for a single state. Fortunately, a method known as temporal difference (TD) learning has been devised that combines the positive features of both Monte Carlo and dynamic programming techniques [Sutton, 1988] [Sutton and Barto, 2000].

To better understand how TD learning works, one must note that in the Bellman equation (Equation 1) the value of each state is recursively linked to the value of each of its possible successor states. Therefore, when a reward is received in a one state, a TD error signal can be calculated as shown in Equation (11), that reflects the difference between where the agent is and where the expects to be going. Because at the time of update the actual successor state is typically known the term $\{V(s')\}$ has been used to replace $\{\sum_{s' \in S} T(s,\pi(s),s')V(s')\}$.

$$\delta = R(s) + \gamma V(s') - V(s) \tag{11}$$

$$\Delta V(s) = \alpha\delta \tag{12}$$

In this way, the reward received from a sampled state is used to increment recursively the value estimate for all states in the state space. The unique feature of TD learning, however, is that it no longer requires the computationally expensive full recursion through the state space. Rather, TD learning makes judicious use of the online samples and updates the value estimates for only those states that have been visited recently. The amount of each update is proportional to the temporal proximity and transition probability between the sampled state and the current state. In TD learning, temporal proximity is determined through the use of functions known as eligibility traces that store the impact a previous state has had on reaching the current state. More recently visited states have stronger eligibility traces and receive greater updates. An example eligibility trace is shown in Equation (13), and the augmented update rule is shown in Equation (14). In Equation (13), λ is a decay factor on the eligibility trace, s_t is the current state being sampled and γ is the decay parameter from Equation (1).

$$e_t(s') = \begin{cases} \gamma\lambda e_{t-1}(s') & if \quad s' \neq s_t \\ \gamma\lambda e_{t-1}(s') + 1 & if \quad s' \neq s_t \end{cases}, \forall s \in S \tag{13}$$

$$\Delta V(s_t) = \alpha\delta e_t(s_t) \tag{14}$$

Function Approximators

Learning methods such as TD learning enable value estimates and heuristic functions to be learned using direct sampling methods without the computational overload that comes from

using the full recursive, dynamic programming methods. However, the value functions and heuristic functions discussed up to this point still require tabular listings for all possible states, which is computationally infeasible in complex environments. Therefore, it is often desirable to use approximation methods to compress large state spaces to a more compact form. Function approximators are techniques used to represent, in finite space, state spaces that have continuous or infinite domains [Szepesvári and Smart, 2004] [Sutton and Barto, 2000].

Function approximators map high-dimensional state spaces into lower-dimensions and attempt to learn value estimates over the low dimensional space. A simple example of a function approximator is the mapping that divides a continuous 2D navigable state space into a discrete grid for mobile robot navigation. Once the state space has been discretized, the robot can attempt to learn $V(\tilde{s}), \forall \tilde{s} \in \tilde{S}$, where $\tilde{S}$ is the new approximate state space. Therefore, the robot does not have to learn value estimates associated with every possible location in the continuous and infinite state space. Instead, the robot learns value estimates for the new state space, $\tilde{S}$, with the hope that $\tilde{S}$ provides an adequate approximation of *S*.

There are a variety of methods to implement a high-to-low dimensional mapping. In addition to grids, state features can be histogrammed using pre-defined bins or tile codings can be used that create multiple overlapping grids [Sutton and Barto, 2000]. Clustering techniques and nearest neighbor methods can be used to create symbolic classes and assign states to the nearest class. Interpolation methods can also be used, potentially in conjunction with nearest neighbor approaches, to create continuous but lower-dimensional representations. Finally, regression and neural network approaches can be used; often in conjunction with other mappings that create numeric feature vectors from the current state. Methods such as neural networks generalize within the value function, while other methods, such as tile coding and linear interpolation, generalize within the state representation [Sutton and Barto, 2000].

It is known that many reinforcement learning methods that utilize tabular listings for *V(s)* will eventually converge to the correct values given sufficient experience [Sutton and Barto, 2000]. However, it is an open question whether many of same techniques retain their convergence properties when used in conjunction with function approximators [Sutton, 1999]. Several recent studies have investigated the convergence properties of function approximation methods, proving convergence for various types [Perkins and Precup, 2003] [Melo, et al., 2008] [Szepesvári and Smart, 2004] [Gordon, 1995].

Gordon [1995] provided convergence proofs for a class of function approximators known as *contraction mappings*. A contraction mapping is a function *f* defined over a state space *S* that maps points in *S* onto *S*, while simultaneously contracting the distance between any two mapped points by no less than a constant factor α. Such functions have fixed points within *S* to which the recursive sequence *f(s)*, *f(f(s))*, *f(f(f(s)))*, …, will eventually converge at a rate of at least α. Gordon [1995] proved that when $V(\tilde{s})$ is represented using such a contracting function approximator, value iteration converges to within a finite bound of the optimal value function, *V*(s)*. In addition, both Gordon [1995] and Szepesvári and Smart [2004] noted that convergence holds for function approximators that are also merely non-expansion operators, provided that they meet further constraints, such as being *interpolatible* [Szepesvári and Smart, 2004]. Examples of contraction mappings are nearest neighbor techniques, linear interpolations, and state aggregation methods such as grids and tile codings [Sutton and Barto, 2000].

Research in Melo, et al., [2008] investigated the convergence of Q-learning methods [Watkins, 1989] when used with function approximation. Q-learning is a reinforcement learning method that learns values, known as Q-values, for state-action sets. The work of Geffner and Bonet [1998], described earlier, used Q-learning to perform real-time search. Melo, et al., [2008] show that under certain conditions linear approximation techniques will converge to an optimal Q-value function given enough experience. Of the conditions they investigate, a particularly interesting one is that when the discount factor, γ, on future rewards is approximately one, the learned policy is not likely to generalize well to nearby policies. However, when γ is much less than one, safe generalization is possible. Therefore, the degree to which the current reward depends on future states impacts generalization and convergence. Similar results have been obtained in Perkins and Precup [2003].

There are other types of function approximation methods besides contraction mappings and linear approximators. Neural network techniques have been used [Tesauro, 1990] as well as Bayes classifiers [Haykin, 2008]. However, these methods are capable of exaggerating small differences between points in the state space and convergence has not been proven. Still, these methods remain extremely important as they often deliver acceptable results in practice.

CHAPTER IV

OVERVIEW OF PSYCHOLOGICAL EMOTION RESEARCH, EMOTION PROCESSES, AND ROBOT EMOTION SYSTEMS

When the words "emotion" and "robot" appear in the same sentence, it seems that the most common reaction is to fancifully imagine robots that feel "happy" or "sad", or at least provide the outward appearance of such feelings. As such, the approach to robot emotions that has been taken by many researchers has been to design and implement artificial emotions loosely based on the theoretical concepts of emotional states in humans and to utilize these states for control. This is a very top-down approach that mirrors, in many ways, the top-down methodology taken by early AI researchers when they began designing intelligent systems: start with the meta-level symbols and states; outline the specific signals that should trigger those states; then determine what changes those states should enact upon the world. In this way, control is achieved through the use of complex finite state machines (FSMs) in which certain states have been couched in psychological emotion terms.

As mentioned in Chapter I, this dissertation is going to take a more bottom-up approach and explore the fundamental components and appraisal processes that enable agents, within the context of their current goals, to learn (from experience) relations and associations that can later be deployed to inform and improve performance. Such processes must go beyond the simple assignment of utilities or the pre-defined switching between control states, and should integrate into all aspects of control and deliberation. The approach described here is developed using psychological and neuroscientific research that points to the existence of multiple, fundamental mechanisms that collectively underlie the high-level control states that have been conceptualized using FSMs. These include, but are not limited to, the appraisal processes mentioned earlier: *relevance*, *utility, urgency*, and *fit*. It is believed that the cognitive processing necessary to derive such appraisals both subserves cognitive control and provides the low-level mechanisms upon which higher-order states, and eventually those demarcated as emotions (artificial or otherwise), may be derived.

Whereas the previous two chapters discussed the need for architectural approaches to cognitive control and the current methods by which robotic systems may perform sequential, real-time decision making, and function approximation, this chapter investigates how emotions,

or more appropriately cognitively processed appraisals and low-level evaluation signals, provide the information needed for adaptive, intelligent, and real-time control. The discussion begins with a brief overview of the concepts surrounding basic emotions before describing how emotions may fit into an architectural view of intelligence. Next, some of the underlying components and functions that contribute to emotional states are described. Appraisal processes will then be discussed from the point-of-view that they are fundamental aspects of both the adaptive purpose and construction of emotion. Throughout this discussion the concepts of *relevance*, *utility*, *urgency*, and *fit* will be discussed as they relate to particular theories. Finally, this chapter concludes with describing current implementations and models of robot emotions.

Basic Emotions

Research into the roles that emotion plays in human cognition has led to numerous theories postulating and questioning the importance and rational basis of emotional processing. Indeed, any researcher attempting to wade through the quagmire of terms, frameworks, and architectures is likely to get quickly bogged down by the numerous and varied details of specific approaches. This is due, in part, to the fact that as of yet there has been no consensus on what is actually meant by the terms "emotion" and "affect" [Pfister and Böhm, 2008] [Picard, et al., 2004] [Barrett, 2006] [Kleinginna and Kleinginna, 1981] [Sloman, 2001b] [DeLancey, 2002]. Fortunately, such confusion does not reside at all levels of emotion research. Taking a step back, one can identify that a large portion of the literature regarding the influence of emotion on cognition is, in fact, moving in the same direction. There is considerable consistency across the multitude of theoretical frameworks with respect to many of the general theories proposed for emotion. For example, many theories consider that emotion operates in a manner that is functionally similar to that of utility within the decision sciences. Additionally, while it has long been accepted that emotion influences behavior and decision making, the notion that such influences are, on average, more beneficial than detrimental is becoming more prevalent [Damasio, 1994] [Bechara, et al., 1997] [Rolls, 1999].

Much of the traditional emotion research over the past few decades has focused on well-known concepts such as the basic emotions suggested by Ekman [1992] [1999]. In his research, Ekman investigated the universality of specific emotions as they are manifested externally in an attempt to fortify Darwin's [1872] claim that certain emotional responses are innate and have

been selectively chosen by evolution for their adaptive value. Ekman's [1992] basic emotions include "happiness", "sadness", "anger", "fear", "disgust", and "surprise". The labeling of these emotions as basic, however, raises the interesting question of what constitutes a *basic* emotion? Ortony and Turner [1990] argue that for an emotion to be considered basic that emotion must be present in all humans, homologous in animals, and selectively chosen by evolution. Panksepp [1998] [2000] provides a similar argument proposing that certain emotions are basic and that these emotions are identifiable from their homologous and neurobiological origins in all mammalian species. The notions of homology, natural selection, and neurobiological origins seem to validate Ekman's identification of a set of basic emotions; however, Panksepp [1998] suggests a different set of basic emotions: "seeking", "rage", "joy", "distress", "care", "lust", and "play", and then extends these to specific behavioral purposes.

Damasio [1994] [1999], whose research in many ways resembles that of William James [1884], investigates the connection between somatic states, emotion, and rational decision making, and also utilizes the concept of basic emotions. He uses the same list as Ekman and believes that these emotions are biologically determined and, for the most part, automatic. However, in addition to this set of basic emotions, which he terms *primary*, Damasio [1994] goes on to identify a set of secondary emotions that are more socially-derived (e.g., "embarrassment" and "jealousy"). While Damasio investigates how the notions of primary and secondary emotions are tied to somatic states and conscious feelings, Sloman [2001b] investigates the architectural basis for emotions from an information processing point-of-view. As such, Sloman [2001a] [2001b] proposes three categories of emotions, coinciding with three distinct layers of control: reaction, deliberation, and meta-management. Primary emotions, such as those referenced by Damasio [1994] and Ekman [1992], reside at the reactive layer of information processing. Secondary emotions are formed within the deliberative layer, and tertiary emotions exist at the reflective, or meta-management, layer. According to Sloman [2001b], these tertiary emotions are perturbances that involve a loss of meta-control, in which it becomes impossible for the individual to focus on anything else except for the emotion-eliciting event. Examples include "infatuation" or extreme "embarrassment". Other researchers have also proposed emotion classifications based on either architectural position or cognitive function. A notable example of this is the OCC model [Ortony, et al., 1988], in which emotions are separated based on the situation that elicits them and the effect that they have on cognition. Ortony, et al., [1988] believe

that emotions constitute valenced reactions to perceptions of the environment, and that these reactions can be broken into three distinct classes. These classes include reactions concerned with the consequences of events, the actions of agents, and the aspects of objects.

Yet even though multiple lists of basic emotions can be found within the literature, there are still the open questions of what *is* an emotion and what criteria should be used when classifying emotions? DeLancey [2002] argued that there was probably no scientifically appropriate class of things referred to by the general term *emotion*. This view is somewhat reflected by Sloman, et al., [2004], who suggests that emotion is a cluster concept and that while it may be possible to identify clear instances of states in which emotion is, or is not, present, demarcating beyond this requires specific understanding of the types of information processing being deployed. He offers a general description of an emotional state as an

> "…episodic or dispositional state in which a part of it… has detected something which is either actually interrupting, preventing, disturbing, or modulating one or more processes which were initiated or would have been initiated independently of this detection, or disposed to interrupt, prevent, disturb, etc. such processes but currently suppressed by a filter or priority mechanism" [Sloman, et al., 2004, p. 230].

This provides a definition of emotion from a functional standpoint, but does not help to differentiate groups of emotions, or identify basic emotions. In order to group emotions, a similarity/difference criterion is required. Sloman [2001b] proposes differentiation based on levels of information processing. Ortony and Turner [1990], Panksepp [1998], and Griffiths [1997] [2004] propose that similarity should be based on homologous and neurobiological measures. Both views are tractable and approachable; however, from an engineering point-of-view having more details is always preferable. Frijda [1986] [1995] proposes, like Sloman, that emotions provide specific information, but continues by suggesting that rather than attempt to extract functional information from the abstract concepts, it may be more appropriate to study the constituent processes that form those concepts. Therefore, rather than focusing on high-level emotional states (e.g., "happy", "sad", etc.) it is more appropriate to focus on the goals and concerns of the individual as well as the types of information-laden signals and appraisals that would be adaptively useful to an individual with such goals and concerns. If one adopts this methodology, the idea of emotion or emotion-based processes in robots is not fanciful, but instead quite practical.

Emotion Processing: Architectural Views and Processing Models

Cognitive architectures have been traditionally deployed for tasks that require symbolic and deliberative decision making. This can be seen in early work with the Soar and ACT-R models discussed in Chapter II. More recently, researchers have begun to investigate architectures that extend beyond simple procedural reasoning. This has included integrating components for reaction and reflection with the standard systems used for deliberation. One example is the CHIP architecture, also described in Chapter II [Shrobe, et al., 2006]. Furthermore, as researchers begin to understand more about how emotion integrates with cognition, and those researchers branch from simple theories of basic emotions to functional theories of causation and influence, it becomes increasingly important to understand how and where emotion fits into architectural designs and approaches to cognition. One approach specifically tailored to answering such questions is the CogAff schema of architectures developed by Sloman [2001a].

Levels of Information Processing

Sloman [2001a] approaches cognitive architecture design from an information processing point-of-view, and defines CogAff as a collection of information processing mechanisms that are intended to enable agents to meet their needs in complex, dynamic environments. In this manner, CogAff (Figure 5) defines a high-level ontology for architectural components and connections. This architecture is loosely divided into the same type of 3x3 grid, proposed for CHIP. Horizontally the schema is organized according to the processes of *sense-plan-act*, while vertically it is divided into *reactive*, *deliberative*, and *meta-management* layers. A global alarm system is used to detect situations, or events, that demand urgent attention. As discussed earlier, each level of processing is capable of different types of emotions.

At the reactive level events trigger highly automatic and innate appraisals that are used in the representation of simple, basic emotions. This includes appraisals for fight/flight and pleasure/displeasure. It is important to note that in Sloman's [2001a] architecture, the emotions that exist at a particular level are not restricted to that level, but rather are determined based upon the information processing available at that level. Therefore, in Sloman's theory, the most basic emotions are those that are derived from the most basic forms of reactive information processing, and these typically coincide with the basic emotions proposed by other researchers (e.g.,

Damasio [1994] and Ekman [1992]). Moving up the hierarchy, the evaluations that exist at the deliberative level result from the ability to compare distinct options (in order to choose among them) and to make basic predictions about the future. As such, the processing available at this level is implicated in the formation of more complex emotions, such as the secondary emotions proposed by Damasio [1994]. The information processing that occurs at the meta-management level results in complex evaluations that involve predictions about the future, reflections about the past, and reasoning about goal-relevant consequences of actions. Since this layer of control resides at the top of the hierarchy, intense emotions, which result in the loss of control at this level, may have disastrous consequences at the lower-levels as well. Finally, all of the processes at each of the three levels are capable of triggering the *alarm mechanism*, which in CogAff signals when rapid redeployment of functional resources is necessary.

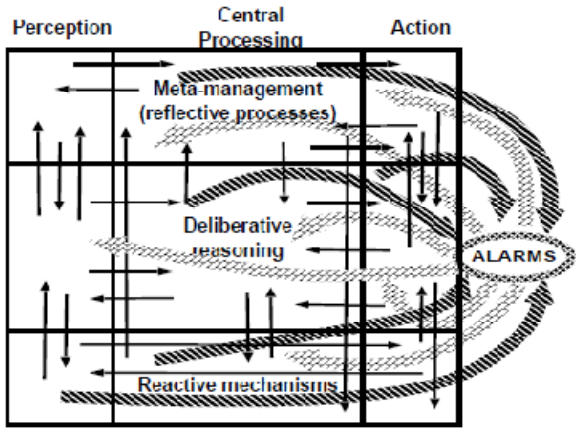


Figure 5. CogAff Architectural Schema [Sloman, 2001a]

Like Sloman, Ortony, et al., [2004] also proposed a three-level architecture in which different types of evaluations and emotions arise at each of the different levels (Figure 6). In their model the first level is a reactive level and is proposed as the origin of basic evaluations that determine approach and avoidance tendencies as well as interrupt the higher-levels of control. These interrupts are analogous to the alarms in the CogAff architectural schema [Sloman, et al., 2004]. However, Ortony, et al., propose that the signals available at this level are only simple forms of affect, referred to as *proto-affect*, which is a two-dimensional signal that assigns values and valence to stimuli, but is not sufficient for full emotional states. The second level is responsible for routine control over the execution of well-learned behaviors. This level provides

appraisals related to present as well as possible future conditions, and these appraisals are believed to be the basis for primitive emotions. Therefore, for Ortony, et al., [2004], emotional states are only realized at the routine level (or above), in which evaluations include basic predictions about the future.

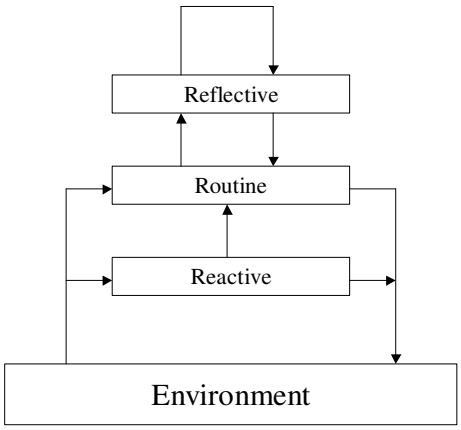


Figure 6. Information Processing Model by Ortony, et al., [2004]

Both the reactive and routine levels are modeled as having direct connections to sensory input and motor output. The reflective level, however, neither receives direct sensory input nor provides direct motor output. This level of processing is responsible for providing biases to each of the lower levels, and performs symbolic computations concerning past, present, and possible future events. Ortony, et al., [2004] propose that the processing involved at the reflective level is relatively slow and most likely implicated in feedback control and analysis, instead of direct causation of behavior. This is consistent with the feedback control theory proposed by Baumeister, et al., [2007] and discussed in the next section.

Processing Models

Frijda and Swagerman [1987] propose an architectural design for creating a computer-based emotion system. This approach is based on the theory that emotions are composed of numerous components, the most important being the various cognitive processes for appraising emotion-eliciting events [Frijda, 1986]. Within their model, once an event has been captured and encoded, relevance appraisals are performed that evaluate the event with respect to the system's goals. Next, diagnostic information is extracted that indicates whether the event should be

approached or avoided, as well as how much uncertainty should be attached to that event. An evaluator then determines whether the event requires an urgent response, and establishes a control precedence signal that assigns priority to the various behavioral responses. Based on these appraisals, actions are proposed and strategies are adjusted as necessary before the actual execution of response behaviors is initiated. Throughout this process, a regulation system is used to evaluate system state and on-going needs. A variation of the model initially developed by Frijda and Swagerman [1987] is provided in Figure 7.

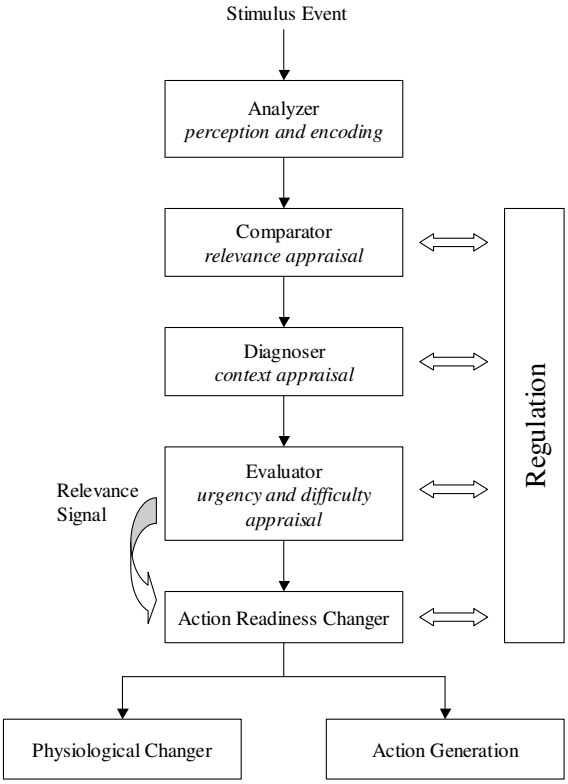


Figure 7. Flow Chart of Emotion System [Frijda and Moffat, 1994]

Scherer [1981] proposes a *component process model* of emotion in which emotions are considered to be episodic responses to events of relevance. Scherer identifies five systems involved in the emotion process. The first system is similar to those discussed by Sloman [2001a], Ortony, et al., [2004], and Frijda and Swagerman [1987]. This system performs information processing on events and evaluates those events through the use of learned associations (i.e., memory) and forecasting abilities (i.e., prediction). Once the evaluations have

been determined, a supporting system modifies the individual's internal state, a leading system plans and selects actions, an acting system performs behavior, and a monitoring system processes feedback. Though it is intended to be a theory of emotional processes, Scherer's [1981] component process model has many of the elements of a simple control architecture (i.e., evaluation, planning, execution, and monitoring).

Of the five systems identified in Scherer's model, the most relevant for this discussion is the information processing system. In order to perform its evaluations, this system utilizes a number of appraisals, which Scherer [1981] [1997] refers to as *stimulus evaluation checks* (SEC), each of which can be performed at the various levels of processing identified by Sloman [2001a] and Ortony, et al., [2004]. In Scherer's theory, there are numerous potential SECs, however, the primary checks are those for *novelty*, *pleasantness*, *goal-conduciveness*, *coping potential*, and *compatibility with standards* [Scherer, 1997].

Affect as an Information Carrying Component

The high-level approach to designing artificial emotion that was described at the beginning of this chapter assumes that events trigger emotional states that then impact control. This idea is conceptually represented in Figure 8. Yet as the discussion has thus far indicated, multiple levels of processing are often present for complex control and each level is capable of its own distinct responses. In addition, there are a variety of appraisal-based cognitive processes that facilitate the development of different emotional states. From an engineering perspective, the interplay between the multiple levels of processing and each of the various appraisals may be conceptualized better by mentally rotating Frijda's model (Figure 7) counterclockwise 90^{o} and then overlaying it onto the CogAff schema (Figure 5).

Since this dissertation is interested in the bottom-up approach to emotion-based control, it is necessary to understand the types of low-level signals (and functions) that both enable the different appraisals and eventually form critical components for those states labeled as "emotion". According to Frijda [1995], a key low-level component for appraisals (and thus emotions) is the notion of affect: a signal that carries aspects of positive/negative evaluation.

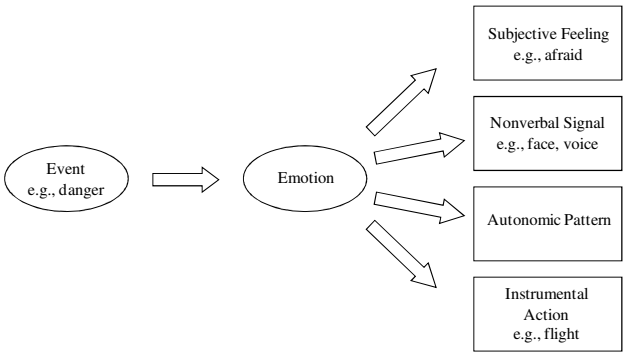


Figure 8. Traditional View of Emotion as a Response to an Event [Russell, 2003]

The theory of affect as a core mechanism in the construction of emotion has been heavily investigated by Russell [2003], who, like Frijda [1986], suggests that a number of processes both precede and constitute (or rather become categorized as) "prototypical" emotions. Figure 9 presents Russell's model, which has been slightly adapted for this discussion. Conceptually, Russell's [2003] model is not unlike that proposed by Frijda and Swagerman [1987]; both models propose that foundational components are responsible for those influences on cognition typically attributed to "emotion". Russell's model, however, has a much higher degree of parallelism, and Russell downplays the role of appraisal processes in favor of focusing more on the notion of *core affect*. Interestingly though, Russell notes that the combination of core affect with the other low-level components still results in the formation of basic appraisals, such as attention-guiding relevance and basic utility.

For Russell, core affect represents a "non-reflective feeling that is an integral blend of hedonic and arousal values" [Russell, 2003, p. 147]. In this model, core affect is a two-dimensional, object free, and ever-present internal signal. From an information processing point-of-view, core affect might be found in both the reactive and deliberative layers. This has been discussed by Ortony, et al., [2004], who use the terms *proto-affect* and *affect* to describe the signals that arise at these levels of processing. As shown in Figure 10 the two dimensions comprising Russell's [2003] notion of core affect are those spanned by {pleasure, displeasure} and {activation, deactivation}. Similar theories can also be found in the work of Winkielman and Trujillo [2007], and Larsen and Diener [1992]. Typically, in these theories one dimension is reserved for valence, or a measure of "good/bad", while the other is reserved for "intensity/arousal".

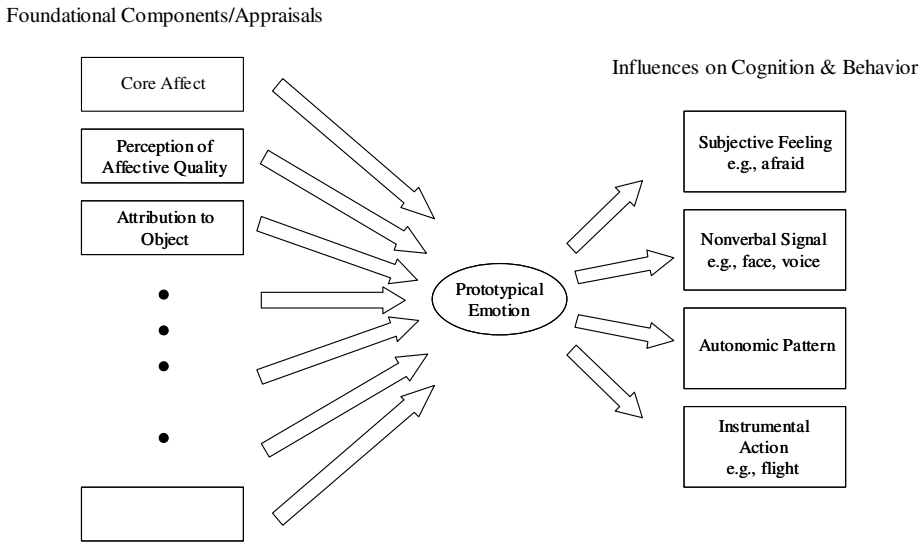


Figure 9. Model of Emotion Arising as Patterns of Core Mechanisms (adapted from [Russell, 2003])

Russell [2003] proposes that when arousal is high, core affect is within the foreground of cognition. He notes, however, that though it may be consciously accessible, core affect is always just beyond the realm of conscious control. This idea is similar to Damasio's [1994] conception of *feeling*, but does not require the same visceral invocations. Furthermore, because it is a simple two-dimensional signal that indicates valence and arousal, core affect is also believed to be the mechanism that provides the common currency discussed by Cabanac [1992], Peters [2006], and to a certain extent Rolls [1999]. For control purposes, compensation of undesired core affect is achieved through the use of predictions of *future* affective states to guide decision making.

Whereas in Russell's theory core affect is object free, *affective quality* is linked to an event and represents the capacity of that event to produce changes in core affect. Therefore, while core affect is localized within the organism, affective quality is localized within the event. The combination of these signals results in *attributed affect*, a mechanism that serves two purposes in Russell's theory: to guide attention to a specific object/stimulus, and to provide direct access to affective quality. Thus core affect provides utility while attributed affect indicates relevance and enables focusing. Finally, it is worth noting that in Russell's theory prototypical emotions, or the basic emotions of Ekman [1992], Damasio [1994], and Panksepp [1998], arise as the core mechanisms develop and then succumb to categorization and mental representation.

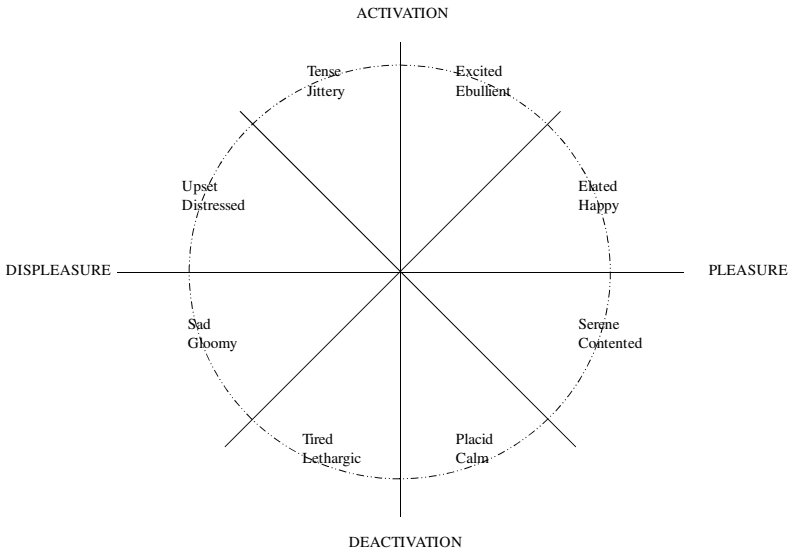


Figure 10. Core Affect as Two-dimensional Signal [Russell, 2003]

Research by Baumeister, et al., [2007] builds on Russell's theory, but views emotion as a dual process responsible for feedback control with both controlled and automatic aspects. The automatic portion of the process is provided by *automatic affect*, a concept similar to core affect. The controlled process is the result of the formation of high-level emotional states (Russell's prototypical emotions) and is much slower to develop than the relatively processes of automatic affect. Baumeister, et al., [2007] suggest that the primary role of full-fledged emotions is not to cause behavior directly, per se, but rather to stimulate *post hoc* cognitive processing in order to learn new associations between affect, stimulus, and behavior sets. A modification of Figure 9 that includes the ideas of dual processing and feedback control is shown in Figure 11.

Automatic affect is an extension of core affect in which further cognitive processing has been added to help differentiate the varying affective responses. Automatic affect guides decision making by providing quick evaluations that allow potential responses to be structured according to the individual's goals and approach/avoidance tendencies. These evaluations also enable the predictions of possible future emotional states that further aid decision making by providing weight for each of the different responses [Baumeister, et al., 2007].

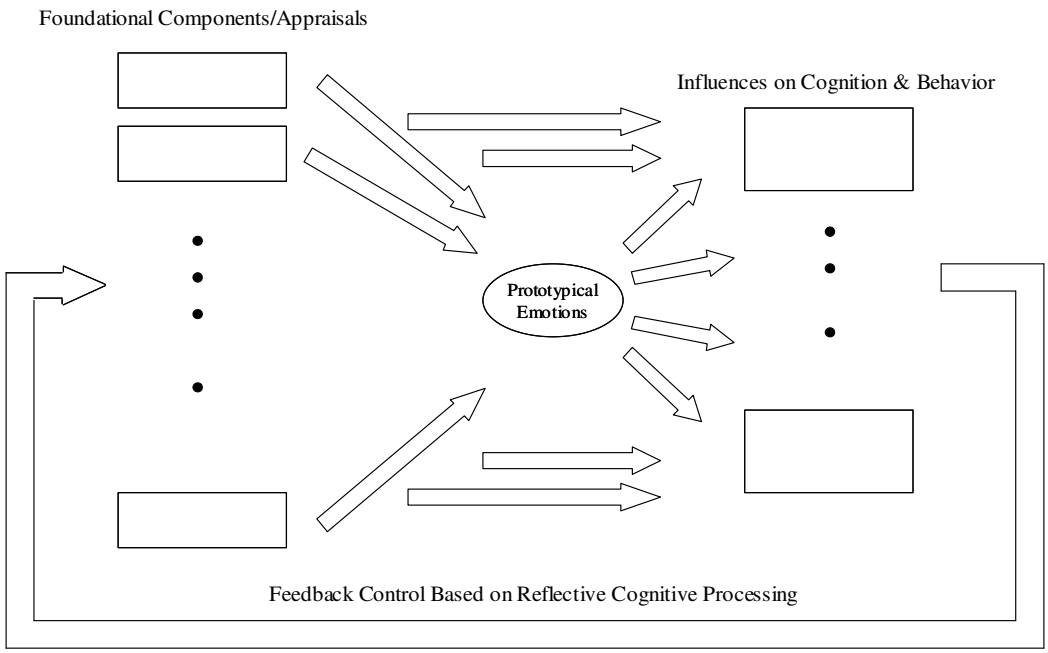


Figure 11. Dual View of Emotion-Based Control Using Automatic and Controlled Processes

Once the decision has been made and the outcome has been observed, the slower controlled process (i.e., full-fledged emotion) will have developed and can then serve its purpose of guiding performance evaluation and stimulating cognitive processing. This completes the feedback loop, altering the evolution of and associations for future appraisals. A similar notion of control has also been developed in work by Carver and Scheier [1998], in which the feedback signal is analogous to the error signals of modern control theory. Interestingly, Baumeister, et al. [2007] note that their feedback model is, essentially, a reinforcement learning model. They contend, however, that in humans the addition of further cognitive skills buffers and improves upon the basic abilities of reinforcement learning. In robotics and machine learning, such buffering could be achieved through techniques such as *explanation-based learning* [Mitchell, 1997].

Appraisals and Functions

The philosopher Jeremy Bentham [1789] proposed that human decision making was a form of hedonic calculus. For years this view was discounted, with researchers focusing on "cold" cognitive processes, rather than "hot" emotions [Slovic, et al., 2003]. However, the tide has turned and it is now largely accepted that emotion plays an integral role in cognition and

decision making. The discussion to this point has described what types of information processing are necessary for the development of general-level appraisals and emotional processes. The notion of two-dimensional, affect-based signals indicating valence and arousal has been described, as well as how such signals may contribute to the construction of emotion. The current section will describe goal-relevant appraisals as they are implicated in the construction of emotion. The intention is to begin functionally describing emotion from a level below that of the abstract states commonly referred to as emotional, but above that of the valenced signals, such as core affect. The following discussion will focus on the concepts of *relevance*, *utility*, *urgency*, and *fit*.

Zeelenberg and Pieters [2006] argue that emotion is a powerful force in decision making, and that emotion has evolved precisely for the production of behavior. They argue that the utility of specific emotions, and the appraisals that underlie them, is found in their ability to affect future behavior, and that each appraisal provides a particular function with its own adaptive value. The primary role of emotion, as they see it, is to serve as motivator. This form of appraisal is rooted in experience and extracts information related to "how the individual is doing" (with respect to the current goals) and "what should be done next" [Zeelenberg and Pieters, 2006]. It is important therefore, that the individual have the ability to determine how well its current response *fits* the situation, or whether a new strategy/behavior is required. In addition, Scherer [1997] suggests that measures of fit enable the individual to determine to what extent certain events are under control and Frijda [1986] proposes that a primary function of appraisals is to indicate the degree of fit between responses and events. If a particular response results in the individual moving closer to its goal, then the degree of fit should likely be very high. Interpreting such evaluations as error signals yields a view similar to the feedback control theories described by Carver and Scheier [1998] and Baumeister, et al., [2007].

A motivation-based appraisal has also been proposed by Peters [2006]; however, Peters' motivation function is more akin to an *urgency* indicator, which is derived from low-level affect, and influences the speed at which information is processed. From a functional perspective, Peters' motivation appraisal is critical for situated agents; it signals when to start and when to stop, and ensures that system needs are met in a timely and appropriate fashion. A similar function, *speed*, has been described in research by Pfister and Böhm [2008]. Frijda [1986] also proposes that appraisals for urgency establish changes in control precedence and action

readiness, which have a direct impact on control and behavior, and Scherer [1997] suggests that such measures of urgency are informed by checks of goal-significance and coping potential. Such low-level urgency-based appraisals are compatible with the process of automatic affect proposed by Baumeister, et al., [2007], and are also reflected in separate research by Bechara, et al., [1997] and Loewenstein [1996]; in both theories appraisal-induced visceral states guide and, occasionally, force behavior. Panksepp's [1998] evolutionarily primitive affect programs, which include aspects of the proto-affect signals proposed by Ortony, et al., [2004], and the interrupts proposed by Sloman [2001a] are also compatible with this notion of urgency-based appraisal.

Frijda [1986], Scherer [1997], Peters [2006], and Pfister and Böhm [2008] all propose that *relevance* detection is a critical function of emotion. In Peters' theory, low-level affective signals are used to identify pertinent events (i.e., stimuli) that should occupy an individual's focus of attention and influence decision making. Scherer [1997] includes relevance within the goal-significance check and uses this signal to mediate the influence from other SECs. Pfister and Böhm [2008] propose that certain emotional states have the power to grab one's attention and focus it intently on specific causes, events, or possible outcomes. Pfister and Böhm [2008] also extend the relevance function and argue that the processes for identifying relevance may actually construe the current situation to match the current emotional state. In other words, such processes are not only reactive but often proactive as well.

It has been proposed that utility-based signals enable problems to be identified and behavioral responses to be prioritized [Zeelenberg and Pieters, 2006]. Research by Slovic, et al., [2003] proposes that low-level affect acts as a heuristic signal within the decision-making process. With respect to the individual's goals, affect aids comparison between different response options and assigns weights for indicating relative importance. There have been several theories that specifically propose utility-based appraisals for "emotional" decision making [Tversky and Kahneman, 1986] [Mellers, et al., 1999] [Mellers, 2000]. However, unlike standard utility values, which provide specific monetary-type gains, affect-based utilities measure anticipated pleasure and pain as well as other such hedonic factors. Affect-based utility indicates preference. In addition, it has been proposed by Kahneman and Tversky [1979] that the mapping function for hedonic value is non-linear, concave for gains, convex for losses, and has a steeper slope for losses than for gains. Yet, Cacioppo and Bernston [1999] have noted that the best representation for signals such as core affect, proto-affect, or automatic affect, may be to define

separate functions over both gains and losses. Their research indicates a slight bias towards positive affect at the zero point, but that negative affect has a steeper slope. Cacioppo and Bernston [1999] suggest that this produces exploratory behavior in affectively neutral situations, while enabling stronger reactions to negative versus positive stimuli, which is most likely an evolutionary advantage that encourages *safe* exploration. Figure 12 presents the functions proposed by Kahneman and Tversky [1979] and Cacioppo and Bernston [1999].

The theory that affect provides an information-laden signal for use as utility in the appraisal process has been proposed by Schwarz and Clore [1988], Peters [2006], Slovic, et al., [2003], Pfister and Böhm [2008], and Frijda [1995]. In particular, Schwarz and Clore [1988] propose the *affect-as-information-mechanism* (AIM) and contend that affect provides information related to how an individual "feels" about a situation and that this information guides decision making. Predictions and projections of future feelings are then used to adjust responses toward, or away from, particular situations. This type of processing resides at the deliberative and routine levels of information processing [Sloman, et al., 2004] [Ortony, et al., 2004] and thus it is conceivable that utility-based appraisals are employed during intense deliberation, even when the individual has access to specific monetary-type utility information [Slovic, et al., 2003].

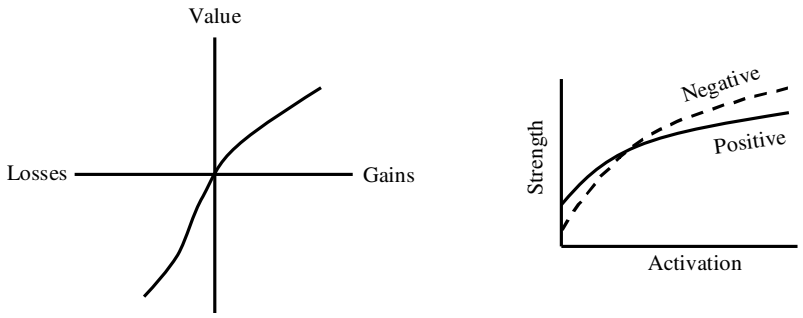


Figure 12. Proposed Representations of Affect-Based Utility Functions by (a) Kahneman and Tversky [1979] and (b) Cacioppo and Bernston [1999]

In order for affect to be useful as a utility signal, it is necessary to collapse numerous, goal-relevant evaluations onto a common scale, or currency [Cabanac, 1992]. By providing a common currency, affective signals are able to alleviate much of the need for costly and tedious logical evaluations. This same functionality has been noted by Peters [2006], but can also be

found in the neurobiological research of Montague and Berns [2002] and Rolls [2004]. Interestingly, the role of "affect as information" appears to be commonly associated with the idea of involuntary feelings, while the notion of "affect as common currency" seems to treat affect as a high-level signal that enables conscious comparisons [Peters, 2006] [Slovic, et al., 2003] [Bechara, et al., 1997] [Cabanac, 1992].

Appraisals that utilize predictions of future states are based upon the individual's prior experience and thoughts relevant to the current situation (which are, in turn, influenced by other appraisals). However, the individual's current mood as well as other situation-independent, incidental appraisals may also affect these predictions [Peters, 2006]. The latter provide examples of how noise in the appraisal process may lead to inappropriate, or maladaptive, behavior. Similar theories of how experiences are used to develop predictions of future emotional states have been invoked by Baumeister, et al., [2007] and Damasio [1994]; Baumeister, et al., through their feedback control theory and Damasio through his somatic marker hypothesis.

The appraisals that have been discussed here are viewed as part of the bottom-up process necessary for the development of those states often referred to as emotional. In addition, these appraisals are critical for appropriate, adaptive, and timely decision making. *Relevance* detection enables the individual to identify which goals, events, and stimuli should be focused on and considered. *Utility* enables construction of the decision-space and prioritization of the various response options. Navigation of this decision-space is then achieved by incorporating predictions of future hedonic and utility-based signals (e.g., goal-relevant utility). *Urgency* appraisals, based on the individual's current goals and needs, impose time constraints on the decision process by influencing the manner in which the decision-space is searched, as well as specifically signaling events that demand either urgent attention or immediate action. Finally, appraisals for *fit* enable error-tracking and help to reprioritize the decision-space when the chosen strategy (response) is underperforming. Signals for facilitate *post hoc* cognitive processing that ultimately improves the quality of future appraisals and thus performance.

Robot Emotions

There are two general approaches to designing artificial emotions, or emotion-based processes, for robots. While there can be (and often is) considerable overlap between these

approaches, each approach serves as a guide for classifying artificial emotion research. The first approach is from a functional perspective, in which it is desired to have control signals, and/or states, that improve task performance along some criteria. In this case, the control signals (i.e., situation-based appraisals and low-level affect-based evaluations) are entirely internalized and are used to improve such measures as autonomy, performance, and adaptability. This is the type of approach that will be taken in this dissertation, and this section will highlight similar approaches.

The second approach is aimed at enabling better human-robot interaction by providing robots with the means to model emotions and, in some cases, display emotional states. While social interaction is not a concern of the current investigation (and thus these approaches will not be further discussed), interesting and intriguing research has been conducted in this area. For example, research by Breazeal [2002] and Gockley, et al., [2006] investigates how artificial systems equipped with the ability to maintain internal states and display emotion (e.g., "smiling") can improve human-robot interaction. In addition, notable research by Picard [1997], Picard, et al., [2001], Dautenhahn [2002], and Liu, et al., [2007], investigates methods by which robots and artificial systems can understand human emotion (and behavior) in order to adapt themselves to human emotional states and needs.

Control System Approaches

Ahn and Picard [2005] implement a computational framework for "affective-cognitive learning" and decision making. They define an affective agent as an agent that "has both cognition and emotion and its motivation can be modeled by reward and that 'internal reward from cognition and emotion' and 'external reward from the external world' can explain motivation in its learning and decision making" [Ahn and Picard, 2005, p. 2]. Similar to the work of Shanahan [2006] described later, their system uses an internal simulation loop that models expectation and evaluates internal reward. However, the system designed by Ahn and Picard [2005] implements emotion solely as an internal reward signal, or utility value. Thus emotion only alters behavior selection and does *not* alter the deliberative approach behind such selection. The emotional states used by Ahn and Picard [2005] are given labels, such as "feeling good" or "feeling bag", and utilize an internal bias so that the robot prefers those states in which "feeling good" is more likely. Through the use of *affective-cognitive decision* blocks, the robot monitors

the external environment and performs internal simulation to identify actions that lead to desirable emotional states.

Gadanho and Hallam [1998] design an emotion-based control system using four of the basic emotions proposed by Damasio [1994] and Ekman [1992]: "happy", "sad", "fear", and "anger". Each emotional state is assigned an intensity value and the dominant emotion is determined by the state with the highest associated intensity value, provided that value is above a pre-defined threshold. The intensity values were derived by monitoring perceptual signals (both internal and external), such as "battery power", "collision (bump) detection", and "amount of current activity". The relation between intensity values and emotional states was pre-defined through specific rule sets that connected intensity to artificial hormones that combined in various ways to create emotional states. The approach described in Gadanho and Hallam [1998] uses the derived emotional state to provide reward signals for use in reinforcement learning.

Like Gadanho and Hallam [1998], both Canamero [1997] and Cos-Aguilera, et al., [2005] develop systems that monitor internal signals related to basic "bodily" needs. Such homeostatic systems are used as control structures to keep "physiological" variables within a certain range. The motivation signals arise from the deprivation of those internal variables, and emotions are treated as control states that indicate the need for compensation. For example, Cos-Aguilera, et al., [2005] implement a homeostatic system which feeds a set of internal drives that, upon compensation, provide reward signals that create associations between the deprived internal state and the behaviors used to compensate that state. The underlying control system is based on Brooks' [1986] *subsumption architecture*; however, the behavioral responses are based on the current associations rather than a preset behavioral hierarchy. The homeostatic variables used by Cos-Aguilera, et al. [2003] are "nutrition" (battery power), "stamina" (activity), and "restlessness" (lack of exploration). The drives are based on these variables and represent desires such as the "need to find food", the "need to rest", or the "need to explore".

Canamero [1997] explores how low-level signals that measure internal variables in simulated creatures can be used to create emotion-based control states. These creatures are placed in a simulated world and are given goals related to surviving in that world. Along with each internal variable there is an associated *motivation* signal that indicates which goal should be pursued. Examples of the internal variables include "aggression", "cold", and "fatigue". There are six defined emotional states, each of which can be triggered by a specific type of stimulus.

Each emotion is modeled as a signal that influences one or more motivations by increasing or decreasing motivational intensity. In this way, the motivation signals are used as utility signals for deciding which goal to pursue, while emotion acts as a reward signal for each of the various goals. The events that trigger each emotion are preset by class type. For example, "achievement of a goal" causes "happiness".

Another low-level approach to robot emotion has been investigated by Moshkina and Arkin [2003]. Their model combines *traits*, *attitudes*, *moods*, and *emotions* (TAME) as distinct components in a behavior-based system [Arkin, 2004]. Arkin, who contends that emotions, from the perspective roboticists, consist of a subset of motivations that can be used to dynamically modulate ongoing behavior [Arkin and Vachtsevanos, 1990], proposes that the components of the TAME model should modify behavioral parameters to induce selection of certain behaviors. Traits and attitudes determine the basic dispositions of the robot; traits are innate, task-specific responses, while attitudes are learned responses based on consistent and persistent training. Attitudes are used to signal approach/avoidance tendencies and to prune the decision-space. Moods are stimulus-independent summaries of previous affective reactions over a temporally extended window. Emotion is a specific reaction to an event of relevance to the current goals. The specific emotions in the model of Moshkina and Arkin [2003] are pre-defined and are used to facilitate quick responses to salient events, as well as provide goal-based evaluation signals (i.e., utility). While the TAME model is more comprehensive than the models described previously (i.e., it incorporates decision-space pruning and urgency evaluations), it still relies heavily on preset emotional states that arise from stimuli in a pre-determined fashion, and utilizes specifically defined behavioral influences.

Architectural Approaches

While the architectures of Sloman [2001a] and Ortony, et al., [2004] are chiefly theoretical designs that are intended to provide guidelines for laying out specific architectures, the approaches by Shanahan [2006], Gadanho [2003], and McCauley and Franklin [1998] are designed to incorporate emotion and emotion-based processes in realized systems that utilize a cognitive architecture. Shanahan's [2006] architecture implements a dual-loop control process composed of a higher-order executive loop embedded within a first-order reactive loop. The reactive loop determines responses to environmental stimuli as those stimuli are detected in real-

time. In parallel, the higher-order loop operates on the same input, but rehearses potential decisions and, when necessary, vetoes the first-order reactions. During the rehearsal process, affect-based salience, or utility, is associated with potential responses and provides interrupt signals to the first-order system. One intriguing note about Shanahan's system, however, is that the affect-based interrupts proceed in a "top-to-bottom" fashion (higher-order loop to first-order loop), which suggests that the higher-order loop operates at a higher frequency than the first-order loop. This is counterintuitive to much of the emotion and architectural control discussions up to this point. The systems of Sloman [2001a] and Ortony, et al., [2004] primarily model interrupts proceeding in a "bottom-to-top" fashion and feedback control operating "top-to-bottom".

Gadanho [2003] extends the type of emotion system proposed by Gadanho and Hallam [1998] into a full cognitive architecture. Within the architecture, a homeostatic system monitors the internal state and signals when one or more physiological variables deviate from the (pre-defined) acceptable range. If deviation occurs, the system pursues the goal of returning the deviated variable to its acceptable range. Deviation facilitates negative reward, while compensation facilitates positive reward, as well as association between that reward and the reward-producing behaviors. Within the system proposed by Gadanho [2003], utility is the primary appraisal process; however, events interpreted as harmful for goal satisfaction can trigger basic interrupts. The interrupts are used to re-adjust the focus of attention towards a different goal and to reset the decision-cycle. While this may be construed as a form of basic relevance detection, this process only enables goal switching, and does not otherwise modify the focus of attention or the interpretation and contribution of specific utility values. Finally, the calculation of utility from the homeostatic variables is performed using externally, pre-defined rule sets.

Another architecture for emotion was developed by McCauley and Franklin [1998] and was implemented on a system called "Conscious Mattie" (CMattie) [Franklin, 1997]. This architecture is loosely based on the Pandemonium Theory of Selfridge [1959]. In their system, McCauley and Franklin [1998] use a variety of interconnected "codelets" which respond to the perceived activity of all other codelets by becoming more or less activated. The entire collection of codelets represents a loose connectionist network in which the activation of one codelet may influence the activation of others. Furthermore, connections between codelets can be created or

destroyed, strengthened or weakened. Emotions are modeled using a set of emotion codelets, each of which updates the gain value of a vector of four real numbers that represent four of the basic emotions proposed by Damasio [1994] and Ekman [1992]. Thus, each codelet has a short pre-defined rule that determines which stimuli initially trigger that codelet.

In order for this system to function properly, CMattie must be actively situated in an environment and able to detect the changes produced by its actions. Using its emotions, CMattie learns to recognize and pursue pleasurable states while simultaneously recognizing and avoiding non-pleasurable states. This is performed using an internal learning mechanism designed to create associations between internal states and the system's goals and drives. The learning mechanism, called unsupervised internal reinforcement [McCauley and Franklin, 1998], differs from classic reinforcement learning by utilizing a reinforcement signal that is generated internally, in response to action-related environmental changes. Using an internally generated signal to provide reward is an emotion-centric concept, and is reflected in much of the research concerning emotion and robots [Canamero, 1997] [Gadanho and Hallam, 1998] [Gadanho, 2003].

Each of the approaches just described highlights a trend in robotic emotion research: the desire to use emotion as a utility signal based, typically, on human-centric goals (e.g., survivability) and to pre-define emotional states (e.g., "happy", "sad", "restless") and then define the triggering events for those states. While both the architectural approaches and the control system approaches consider emotion to be primarily a utility signal, the architectural approaches do branch slightly into the concepts of relevance and urgency, as these concepts are more of a concern at the architectural level of control. Unfortunately, the literature is sparse on approaches that implement other types of appraisals (rather than just utility) as constituent parts of emotional states. In addition, most approaches seem to be more concerned with how specifically pre-defined emotional states can improve survival-based autonomy rather than what methods may improve task- and niche-specific autonomy.

CHAPTER V

RESEARCH METHODOLOGY

Motivation and Focus

In order to create robotic systems that can cope with the challenges presented by complex, dynamic environments it is necessary to take an integrated approach that combines high-level architectural design with real-time decision-making capabilities. Chapter II discussed the need for cognitive architectures in order to realize complex, cognitive control. Chapter III described decision-making approaches for robotic systems and different techniques that enable those approaches to operate in real-time; either through the use of real-time search methods or anytime algorithms. Chapter III also highlighted the fact that many systems rely on preset functions (e.g., heuristic functions [Geffner and Bonet, 1998] [Barto, et al., 1995]), parameters (e.g., depth and thresholds [Paquet, et al., 2005] [Dean, et al., 1993]), and feature vector representations [Sutton and Barto, 2000] in order to perform real-time decision making. Chapter IV described emotion research as well as some fundamental appraisals and signals that are believed to underlie and precede various emotional states. The appraisals that were focused on were those of *relevance*, *utility*, *urgency*, and *fit*.

Each of the aforementioned appraisals serves a basic function, but within engineering research, utility seems to receive the most attention. This may be due to the fact that utility enables the comparison, prioritization, and selection of responses. However, for the design of a cognitive robot embedded in the real world, utility may be necessary but it is not sufficient. In addition to utility, it is important that a cognitive robot be able to filter the vast number of perceivable aspects of a situation to identify *only* those aspects that are the most relevant to the current goal. Furthermore, because the external environment often imposes time constraints on decision and action, robots must be able to appraise the current situation with respect to the time required, and allowed, for deliberation. This includes the ability to adaptively tune the decision-making process and, as necessary, interrupt ongoing deliberation. Finally, a cognitive robot composed of such abilities should possess the capacity to assess and evaluate how well its current knowledge structures fit the situations to which they are being deployed. This type of self-evaluation is useful when identifying those structures that require further training.

This dissertation investigates how the offline cognitive processing of experience can be used to develop and train the cognitive processes that ultimately appraise *relevance*, *utility*, *urgency*, and *fit*. These appraisals are based on psychological theories of emotion and emotional states, and each appraisal must integrate with the decision-making process, while the decision-process as a whole must be integrated into a larger cognitive architecture. This chapter describes the approach taken for the design and implementation of this system, but first it is necessary to describe what is meant by the phrase "cognitive processing of experience" and to outline how such processing may be accomplished.

Cognitive Processing of Experience and Episodic Memory

Much of the discussion up to this point has focused on the type of appraisals that may be useful for cognitive control. While the investigation of these appraisal constitutes a large portion of this research, both theoretically and empirically, it is necessary to not only understand what appraisals are necessary and how they can be used to impact control, but also to understand *how* such appraisals may be learned through experience, specifically the cognitive processing of experience.

Chapter II described the different types of long-term memory systems that are believed to exist. This included discussions of procedural, episodic, and semantic long-term memory. Of these systems, the most crucial for the current research is *episodic memory*, the memory system devoted to the retention and retrieval of an individual's unique subjective experiences that, when necessary, can be "re-lived" from the individual's own auto-centric point-of-view [Tulving, 1983]. Tulving [1983] [2002] refers to this ability as *autonoetic consciousness* and argues that this type of memory is unique to humans. Other researchers, such as Clayton et al. [2002] and Morris [2002], however, loosen Tulving's restrictions on episodic memory and argue for the existence of "episodic-like" systems in animals. Their arguments are based on research that indicates certain animals are able to process situations with respect to remembered contextual cues. In other words, these animals appear to have the ability to retrieve previously encountered information related to "what", "when", and "where", and to use this information to impact future behavior.

Research has shown that food storing birds have the ability to retain not only the specific locations in which food had been previously stored, but also to appreciate the type of food stored

as well as the duration of storage [Clayton, et al., 2002]. In addition, rats have shown the ability to process visual scenes with respect to spatial context and research suggests that this processing exhibits its own episodic-like characteristics [Morris, et al., 1990] [Aggleton and Pearce, 2002]. Furthermore, this processing appears to be mediated by the rat's current goals and needs, a notion that agrees with human experiments and indicates that the focus of attention and emotional state may be used both in the creation of episodic memories and in their recall [Kapur, 1999]. This somewhat reflect back to Tulving's proposal for autonoetic consciousness, whereby an individual is able to consciously re-experience a previous event and re-live many of the subsequent details (i.e., thoughts and feelings) of that event. Yet, as noted in Chapter II research suggests that such recall and re-living is only a dampened form of the actual experience [Loewenstein, 1996]. It stands to reason, therefore, that if an individual's current emotional state mediates which aspects of a situation become encoded into episodic memory, and that subsequent recollection of that episode informs future deliberation through the re-experiencing of those previous states, then the process of forming episode and learning situation-based appraisals is best intertwined.

Within this dissertation, a basic episodic memory system will be used to provide the experiential database from which the cognitive processing of experience will train each appraisal. In this context, the phrase "cognitive processing of experience" is used to denote that the specific algorithms that are applied are both inspired and based on theories of cognitive processing in humans and animals, in particular the myriad processes required to form, abstract, associate, and retrieve episodic information that is either stored or indexed by complex relational patterns within the brain. This point will be further addressed throughout this chapter, specifically in the section Relational Mapping.

During training, the basic episodic memory system will be used to create an auto-associative network in which individual episodes have been generalized, abstracted, and linked with learned appraisals. Partial pattern matching will then be used to retrieve this information. Finally, the retrieved information will be used to impact deliberation.

ISAC Cognitive Architecture

The architecture used in this research is the ISAC cognitive architecture [Kawamura, et al., 2008] shown in Figure 13. This architecture has three distinct control loops similar to those

described by Shrobe et al. [2006], Sloman [2001], and Ortony, et al. [2004]. These loops provide *reactive*, *routine*, and *deliberative* control. In addition, there are multiple memory systems, such as *short-term*, *long-term*, and *working memory*. Of these systems, long-term memory is further subdivided into the three categories: *procedural*, *episodic*, and *semantic*. Finally, there is a complex, higher-order Executive Control Agent that (among other things) assigns goals, generates plans, and selects responses.

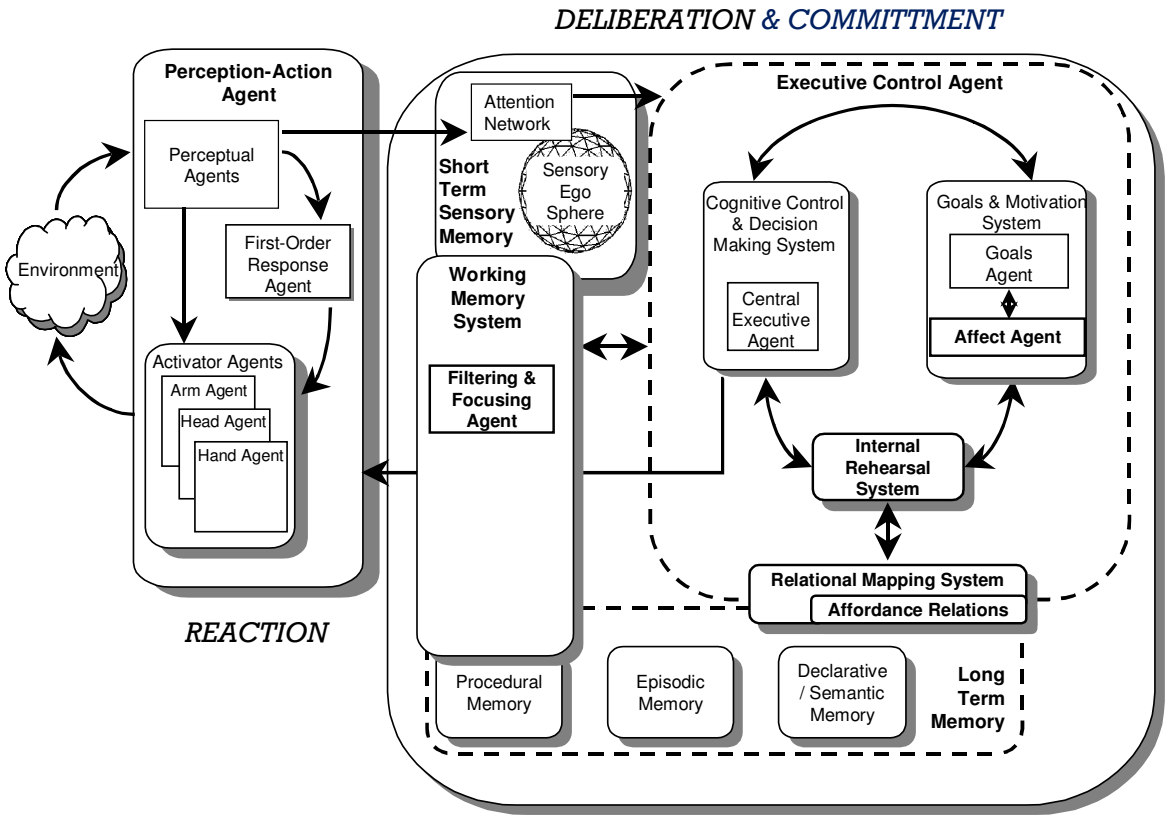


Figure 13. The ISAC Cognitive Architecture

Component Descriptions

The *Sensory EgoSphere* (SES) [Peters, et al., 2001] is a short-term memory system that integrates multi-modal sensory information. The underlying data structure is designed as a complex geodesic dome consisting of a set of sparsely interconnected vertices. Within the ISAC architecture, 1962 are used, however, this number is merely a parameter of the system. The structure of the SES enables the use of spreading activation networks [Pinker and Mehler, 1988] that perform spatio-termporal coincidence detection and mediate the salience of each percept.

Salience values are used as attentional markers, but also facilitate perceptual binding [Peters, et al., 2001] [Kawamura, et al., 2008].

The *First-Order Response Agent* (FRA) [Ratanaswasd, 2007] initiates routine behaviors without reliance on more complex, high-level control structures. Routine responses are either preset or learned over time. Currently, routine responses are represented as stored percept/behavior combinations.

The *Working Memory System* (WMS) [Gordon and Hall, 2006] implements working memory in the ISAC architecture. The WMS is designed to integrate and filter perceptual and procedural information using knowledge of the current goals and situation-based appraisals. Additionally, the WMS provides the gateway through which the high-level cognitive processes interact with the low-level perception and action processes. The WMS is currently implemented through the use of a Working Memory toolkit (WMtk) written in ANSI C++ and using a multi-layer neural network function approximator [Phillips and Noelle, 2005].

The *Long-Term Memory* (LTM) system is composed of three distinct parts: procedural, semantic, and episodic. Procedural LTM retains information related to the performance of behaviors while semantic LTM retains facts and beliefs about the world. For example, percept attributes (described in the next section) are stored in semantic LTM, and behavioral control laws are stored in procedural LTM. Episodic LTM retains linked episodes consisting of state-action-outcome sets. States (introduced in the next section and described in Chapter VI) are based on the information stored in the WMS, actions represent the selected and performed behaviors, and outcomes are the sensed perceptual events, along with appraisals, that result from those behaviors.

The *Relational Mapping System* (RMS) maintains abstracted representations of states and state features and associates these representations with statistically determined evaluation information (e.g., utility). The contents of the individual relational maps within the Relational Mapping System are mined from experience (i.e., episodes) and each map enables auto-associative access and retrieval.

The *Affect Agent* uses knowledge of the current goals to interpret the evaluations retrieved from the RMS and adjusts the system's decision-making strategy accordingly. Adjustment includes adaptively tuning parameters in the cognitive cycle, and interrupting this cycle when necessary.

The *Goals Agent* assigns goals to the system. Goals determine the current task and evaluations. Goal setting is critical for both the cognitive control components and the low-level perception/action components.

The *Central Executive Agent* (CEA) [Ratanaswasd, et al., 2006] initiates cognitive control, plans, and selects responses by utilizing knowledge of the current state (from WMS) and the current goals (from the Goals Agent). The CEA searches the decision-space and prioritizes response options. The current "best" plan is maintained within the WMS, so that it may be rapidly deployed if needed. During planning, interrupt signals are used to halt the planning process and trigger the activation of the best plan. Interrupts are provided by the Affect Agent, or by the low-level perception systems.

The *Internal Rehearsal System* (IRS) [Hall, 2007] [Erdemir, et al., 2008] internally simulates actions, predicts outcomes, and retrieves evaluations. The IRS uses knowledge of the current goals to retrieve evaluative information from the Relational Maps. In addition, offline simulation by the IRS is critical in the development of the Relational Maps.

Flow of Information through the ISAC Architecture

In the ISAC architecture an array of *Perceptual Agents* are used to detect external stimuli. These agents operate in parallel and independently perceive and process information. There are no type restrictions on perceptual agents. As perceptual information is detected that information is sent to each of the three separate control loops (reactive, routine, and deliberative). This involves passing information directly to the *Activator Agents* where reactive responses may be triggered, to the First-Order Response Agent where routine responses may be triggered, or to the SES where that information may eventually be used for deliberation.

As information is presented to the SES, the Attention Network employs spreading activation networks to bind coincidental percepts and flag highly salient percepts. At each node of the SES, activation values from neighboring nodes are discounted in proportion to their distance from the original node and summed. The resulting sums indicate salience, which in turn effects both retention time and the likelihood of that percept being further passed to the WMS.

The flow of information into the Working Memory System (WMS) comes from three directions: perceptual information is passed in from the SES, behavioral information is passed in from long-term memory, and evaluative and response information is passed in by the Executive

Control Agent (ECA). The flow of information out of the WMS may also take three paths: behavioral information is passed to the Activator Agents, situation information (percepts and behaviors) is passed to the ECA, and situation and evaluative information is passed to long-term memory (for storage). Behavioral information is used to trigger actions, the effects of which are detected by the Perceptual Agents. Situation and evaluation information is retained in long-term memory until it is required for offline learning, or recalled by the WMS. Situation information also initiates (or interrupts) the internal, cognitive, decision cycle. Finally, the information that is passed to the ECA is used to initiate or interrupt the deliberation process.

Information flows through the ECA in a continuous loop (Figure 13). Deliberation is initiated in the CEA using incoming information from the WMS. The IRS simulates actions and retrieves evaluations. The Goals Agent sets goals that effect the interpretation of evaluations, and the Affect Agent interprets each evaluation and adjusts the decision-making parameters accordingly. Finally, the response information is sent to the WMS, where it is either added to the currently forming plan or used to trigger actions directly.

Flow of Control through the ISAC Architecture

As mentioned earlier, there are three levels of control in the ISAC architecture, *reactive*, *routine*, and *deliberative*. Interaction between these levels is bi-directional: top-to-bottom and bottom-to-top. The flow of control is mediated by the WMS, as this system provides the primary interaction between the low-level Perception-Action Agent and the high-level ECA. When control flows from bottom-to-top, it is through interrupt signals that are based on reactions to certain stimuli. These reactions may either be preset or learned responses. The interrupt signals are passed as saliency markers through the Attention Network and WMS. When control proceeds from top-to-bottom, it is through override signals. These signals interrupt and override all currently activated routine responses and allow deliberative control to proceed.

Learning Processes within the ISAC Architecture

Learning in the ISAC architecture is accomplished in several ways. First, stored episodes are mined for relational information that can be used to evaluate situations. This involves creating representations of the current situation that can be used to access the Relational Maps. This is primarily an unsupervised task. Second, the IRS performs internal rehearsal to simulate

experience and create associations between input states and expected outcomes. Internal rehearsal, or "mental simulation" is performed offline, and requires that representative states be appropriately sampled from LTM.

In addition to the learning enabled by the LTM data structures, the WMS uses TD learning (Chapter III) to identify the correct percepts and actions that should be attended to, given the current goals. This is accomplished through the use of the Working Memory toolkit (WMtk) [Phillips and Noelle, 2005]. The inputs to the WMtk are numeric feature vectors that represent the current state and the list of possible information chunks (e.g., percepts and behaviors) to which attention can be given. The output is a filtered list of these chunks. The goal of learning in the WMS is to improve decision making by reducing the number of possibilities that must be considered.

Outline of the Approach

This dissertation investigates how the offline cognitive processing of experience can facilitate the derivation of useful online cognitive processes that appraise the current situation with respect to *relevance*, *utility*, *urgency*, and *fit*. Each appraisal is based on psychological emotion research and must integrate with the decision-making process, while the decision-process as a whole must integrate into the ISAC cognitive architecture. Each appraisal is developed and trained using the system's own unique experience, and this experience is stored as episodes in the system's episodic memory. The remaining sections of this chapter describe the approach taken for the design and implementation of the necessary control system, the algorithms used to learn each appraisal, as well as the integration/instantiation of this control system within the ISAC architecture.

The approach taken in this research is to design the deliberation process as a sequential decision maker that recursively searches the decision-space to determine appropriate responses. The appraisal for *relevance* will be charged with charged with creating the internal representations, while the appraisals for *utility* and *urgency* will be functions of these learned representations. The appraisal for *fit* will be performed post hoc using the observed outcomes of each behavior. The learning process, as a whole, employs a layered, incremental approach in which knowledge discovered at the earlier layers is used as a baseline for learning at the later

layers. During the planning and search process each successive state will be used to create representations and trigger appraisals that direct the next step of the search.

Many specific learning algorithms are discussed in the following pages. These algorithms support two broad types of learning. The first type of learning is aimed at acquiring the necessary *domain knowledge* to successfully perform the given task, and involves developing appreciations of whether certain situations should be approached or avoided. These appreciations are based on an understanding of the important, perceivable aspects of each situation. This domain knowledge is primarily contained in the appraisals *relevance* and *utility*.

Whereas the domain knowledge enables successful completion of the task, *performance-based* knowledge enables the robot to relate its current decision-making capabilities to different situations in order to adjust aspects of its deliberation to better match its abilities and the demands imposed by those situations. This is important because robots are required to function as embedded agents in real-world environments, and those environments often impose strict time and resource constraints. Therefore, the second type of learning involves acquiring performance-based knowledge and developing both an understanding of the robot's capacity to perform the task, and how well the robot's knowledge fits the task. In this research, the primary goal of acquiring performance-based knowledge is to decrease deliberation time in those situations in which the robot is "confident" in its abilities, and to enable fast commitment when immediate response is demanded. In addition, performance-based knowledge is also used to enable basic identification of those (learned) components that are underperforming and require further training. Performance-based knowledge is primarily contained in the appraisals *urgency* and *fit*.

Figure 14 presents the general system layout. The steps involved in processing incoming stimuli and planning responses are described as follows:

1. Use detected perceptual information to create the basic state representation, s_i. This includes current stimuli as well as any reward information related to the previous state, s_{i-1}.
2. Process s_i, with respect to the current goals, and create a set of feature vectors $\boldsymbol{f}_i^v$ that capture and represent goal-relevant information from s_i.
3. In parallel:
    a. Use $\boldsymbol{f}_i^v$ to access relational maps that retain the various utility evaluations $\boldsymbol{u}_i$ for s_i.

b. Use $\boldsymbol{f}_i^v$ to determine the time afforded for deliberation, or the urgency imposed by s_i.

c. Process all rewards to determine if a trained component incorrectly predicted an evaluation in s_{i-1}.

4. In parallel:
   a. Use $\boldsymbol{u}_i$ to prioritize the potential responses to s_i.
   b. Use the appraisals of time afforded to adjust the search parameters *depth* and *breadth* to ensure timely, yet accurate, responses.
   c. Interrupt, if needed, the deliberation process.
5. Select the best action/response, and either:
   a. Continue planning by passing s_{i+1} to Step (2).
   b. Perform the selected action.

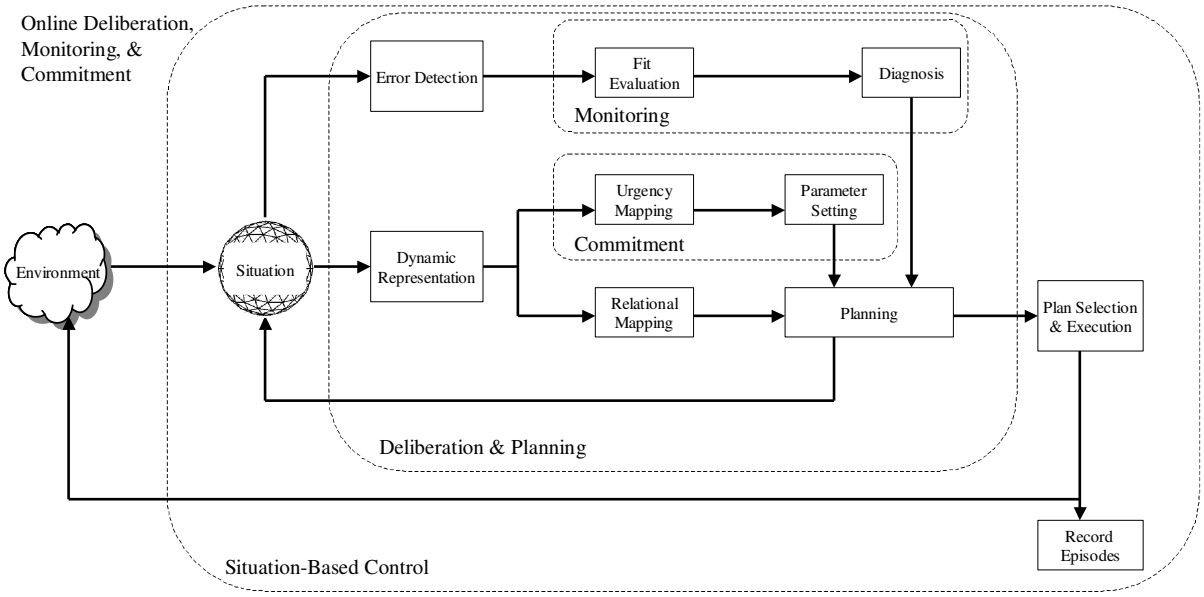


Figure 14. Block Diagram for the Implemented Control System

In the following sections, the primary components of the *Deliberation & Planning* block, as they pertain to the current research, are described and the flow of information through this control system is given. The inputs and outputs of each block are described, along with the

functional purpose of each block. In addition, the techniques used to achieve that purpose, as well as experimental validation of those techniques, are presented.

Before describing the flow of information through the system, or the specific techniques used, however, it is necessary to introduce the basic state representation used for this work as well as some sample data useful for performing preliminary evaluations of each component. While a full description of the variables, functions, and signals allowable for each state is deferred to Chapter VI, the basic state representation can be described using the following three types of information:

1. *Percepts* – The representation of percepts includes the location and known attributes for each percept. An example is shown in Table 1.
2. *1st Order Logic Elements* – Information about percepts, their locations and attributes, is used to create a set of 1st order logic symbols and predicates, such as *Reachable(*$Percept_1$*)* or *InBag(*Bag_0*,* $Percept_1$*)*.
3. *Evaluation Signals* – As the state is processed, various signals are added to the state representation. There are two types of evaluation signals. External signals indicate rewards, while internal signals indicate appraisals.
    a. Rewards – External evaluations of the previous state
    b. Appraisals – Internal evaluations of the current state (i.e., *relevance*, *utility*, *urgency*, and *fit*)

Within each state, goals are assumed to be implicit. This assumption, however, can easily be relaxed by including additional structures within the state representation. The episodic memory system that is used throughout this dissertation can be conceptualized as a set of state/action sequences that end at a final state in which the (implicit) goal has been accomplished or no further corrective action can be taken.

Table 1. Sample Percept Representation

Percept	Location	Attributes								
id	*(x, y, z)*	*name*	*color*	*weight*	*size*	*firmness*	*temperature*	*price*	*type*	*healthy*

Throughout the remaining sections of this chapter, it is assumed that the robot has sufficient initial knowledge, and the basic abilities required, to correctly associate attributes with

percepts and formulate logic expressions. It is also assumed that the robot receives external evaluations (i.e., rewards) either during task performance or immediately after task completion. This assumption is necessary because these rewards provide the foundation for learning the different appraisals that, ultimately, guide decision making.

The complete task used to evaluate the implemented system is described in detail in Chapter VI, however, it is necessary to introduce some sample data that will be used to evaluate each system component. The sample data consists of 10 groceries, each with a corresponding set of attributes (as shown in Table 1). Table 2 lists these groceries and their attributes. The grocery data was collected from a local grocery store. Average measurements were used to assign values to the attributes *color*, *firmness*, and *healthy*. Attribute *type* indicates location on the standard food pyramid [Food Pyramid - MIT]: T_1 = "Grains", T_2 = "Vegetables", T_3 = "Fruits", T_4 = "Dairy", T_5 = "Meats & Proteins", T_6 = "Fats & Sweets", T_7 = "Not a food item".

Table 2. Sample Groceries

name	color	weight (oz)	size (in^3)	firmness	temp.(F^o)	price ($)	type	healthy
soda	RED	70	294	hard	75	1.50	T_6	NO
chicken	PINK	24	108	hard	32	3.05	T_5	YES
milk	WHITE	68.8	160	hard	45	1.67	T_4	YES
cereal	BLUE	17	227.5	hard	75	3.15	T_1	NO
potatoes	BROWN	80	504	hard	75	3.99	T_1	YES
strawberries	MAGENTA	16	140	soft	55	2.30	T_3	YES
bread	BROWN	16	325	soft	75	0.89	T_1	YES
rotisserie	BROWN	32	160	hard	150	7.99	T_5	YES
eggs	YELLOW	24	144	soft	45	1.59	T_5	YES
hot_soup	WHITE	12	48	hard	175	3.50	T_2	YES

Dynamic Situation Representation

As perceptual information is sensed that information must be transformed into a representation suitable for the later cognitive processes. In AI and machine learning, such representations are known as *feature vectors* and capture the important state information in a form that can be used by various algorithms. Feature vectors often require preset mappings for individual states, however as previously described, this research requires that the robot learn which aspects of the situation are the most *relevant* to the current goal, and then use that information to create its own feature vectors that reflect this knowledge. This provides more adaptability than using preset mappings, and increases the flexibility of the robot to adjust itself

more appropriately to different goals. Thus, cognitively processing experience to create flexible feature mappings is the first step in the appraisal process.

In order to determine which percepts are relevant to the current goals, the robot must have basic knowledge of the physical attributes for each percept. Given this knowledge, the goal of the system is to determine statistically which attributes are relevant for the current task, and which can be ignored. While it is trivially accepted that humans are capable of abstracting information and creating categories of information to which attention should be given, animal research indicates that many animals, particularly vertebrates, are also capable of creating object categories that are mediated, to some extent, by the creature's current goals [Morris, 2002]. Research suggests that in humans this type of processing is performed by the prefrontal cortex, a primary site that is believed to be responsible for working memory and attentional focusing [Gazzaniga, et al., 2002]. As discussed in Chapter II, working memory is responsible for focusing an individual's attention on only the most task-relevant information. Various computational methods have been employed by researchers in an attempt to emulate this ability in artificial systems. In particular, the production rule system ACT-R models this functionality through the use of a complex associative network in which *chunks* of information are interconnected by weights that mediate association and retrieval [Anderson, 1983] [Anderson and Lebiere, 1998].

Further research by Phillips and Noelle [2005] implements a feedforward neural network in which specific chunks of information are selectively attended to and used to complete an orienting task. Based on task performance, a TD learning algorithm is used to modify the network weights. This type of training continues until the system learns which information should be attended to in each of the different situations that may be encountered. It is interesting to note that the network designed by Phillips and Noelle [2005] learns "what to focus on" and not "what to do". Therefore, though network training is based on task performance, the network itself cannot affect task performance except by altering what information is presented to the planning system.

While the model of Phillips and Noelle [2005] reflects aspects of the neurological functioning of working memory, their approach, as well as the approach taken in ACT-R, assumes that the network is provided *a priori* with a set of information chunks and that the only task for the network is to choose from these chunks. Yet, it is difficult to make this assumption in

complex environments, and neither system addresses *how* these chunks, as well as the specific representations for these chunks, may be acquired by the system rather than preset by the designer (i.e., engineer). What is needed is the ability to both determine what perceptual information should constitute a chunk as well as how to re-use chunks when similar perceptual information is present. To this end, the current research focuses on statistically mining a set of attribute weights from experience and applying these weights to concept formation via unsupervised clustering.

Input/Output

The input to this component is the current state representation, s_i. The output is a set of symbolic feature vectors, $\boldsymbol{f}_i^v$, that capture relevant information from s_i. Each $f_{ij}^v \in \boldsymbol{f}_i^v$ is a variably sized representation of different elements from s_i, that have been extracted from the 1st order logic and have had the specific percepts replaced with abstracted perceptual symbols. The structure of each f_{ij}^v assumes that the sequential order of information is important. Therefore, $\boldsymbol{f}_i^v$ represents goal-relevant abstractions of specific state information as it sequentially appears to the system. The output feature vectors are appended onto the current state representation as part of the evaluation signals.

Implementation: Weight Learning

In traditional clustering it is often known which attributes are the most important for categorization. In order to derive appraisals for relevance, however, this dissertation makes no such assumption. Thus it is necessary to learn which attributes are the most relevant for the development of goal-oriented clusters, and to what extent those attributes should contribute to cluster formation. To learn this information, the system employs a weight-learning algorithm that looks for statistical patterns in individual grocery bags and uses the evaluations for those bags to increment weights associated with specific attributes. Only grocery bags that contain more than one grocery are used during training. For each bag B_k, the probability of each attribute $P(A_i = V_{ij} \mid B_k)$ is determined and the largest value $max_{Bk} = \max\{ P(A_i = V_{ij} \mid B_k) \} \; \forall j$, is used to update that attribute's weight value, as shown in Equation (15). The value E_c is the evaluation for the cth constraint and can have value [-1, 1], α_ω is the learning rate, and $P(A_i = V_{il})$ is the unconditional probability over all known groceries of the attribute-value pair with highest bag probability,

max_{Bk}, where l = argmax { $P(A_i = V_{ij} \mid B_k)$ } $\forall j$. The term $1.0 - P(A_i = V_{il})$ is used to discount the importance given to highly probable attribute-value pairs.

$$\omega_i = \omega_i + \alpha_\omega * (1 - P(A_i = V_{il})) * max_{Bk} * \{ (1.0 + E_c)/2.0 - \omega_i \} \qquad (15)$$

A more specific form of this update rule is shown in Figure 15. This modified rule only drives individual weights closer to the desired values when the corresponding attributes appear with some probability "greater than chance". If the probability of a specific attribute is below this value, then the associated weight is driven away from the desired value, E_c. In Figure 15, two constants, β_p and β_n, are used to determine the threshold for each update. When $\beta_p = \beta_n = 0.0$, the *if-then* rule shown in Figure 15 is equivalent to the general form shown in Equation (15). Once the final weights have been obtained, they are normalized on the scale [0, 1] by dividing each weight by the maximum value over all weights. This is for presentation and comparison purposes only and does not affect the performance of the clustering algorithm described in the next section, as it is merely multiplication by a constant scalar.

IF

$\{ (E_c > 0)$ AND $max_{Bk} > \beta_p / N) \}$

OR IF

$\{ (E_c < 0)$ AND $max_{Bk} > \beta_n / N) \}$

THEN

$\omega_i = \omega_i + \alpha_w * (1.0 - P(A_i = V_{il})) * max_{Bk} * \{ (1.0 + E_c)/2.0 - \omega_i)$

ELSE

$\omega_i = \omega_i + \alpha_w * (1.0 - P(A_i = V_{il})) * max_{Bk} * \{ (1.0 - E_c)/2.0 - \omega_i)$

Figure 15. *If-Then* Version of Equation (15)

Validation and Evaluation: Weight Learning

To evaluate the weight-learning algorithm, the 10 sample groceries were first split into three clusters using only the *temperature* attribute. These clusters are shown in Table 3 and were derived using a standard *k*-means algorithm. Next, 30 random sets of groceries were generated and each set was evaluated using a preset rule that gave reward of –1.0 for all sets in which more

than one grocery class was present and +1.0 for sets in which only one grocery class was present. For this test, the size of each set was constrained to the interval [2, 4]. The 30 sets, along with evaluations, are shown in Table 4.

Table 3. Classification Scheme Used to Reward Sets

Class	Grocery
C_1	chicken, milk, strawberries, eggs
C_2	soda, cereal, potatoes, bread
C_3	rotisserie, hot_soup

The learned weights are shown in Table 5. The *temperature* attribute is correctly predicted as the most important for classification. Other attributes, such as *color*, *size*, *firmness*, *price*, and *healthy* also receive non-zero weight; indicating that the particular data (Table 4) contained additional "hidden" patterns that affected weight assignment. Because the current system is designed to learn pattern information from its own experience, it is important that the weight-learning algorithm identify all possible patterns in the input data. This need validates the current weight-learning algorithm as a suitable approach to identifying basic relevance markers, i.e., attributes.

Table 4. 30 Randomly Generated Sets of Groceries with Evaluations

#	set	evaluation	#	set	evaluation
1	hot_soup cereal	-1	16	hot_soup soda eggs	-1
2	bread strawberries bread	-1	17	rotisserie eggs rotisserie cereal	-1
3	potatoes rotisserie	-1	18	strawberries milk soda eggs	-1
4	potatoes bread	1	19	chicken bread bread chicken	-1
5	bread potatoes soda	1	20	rotisserie potatoes cereal eggs	-1
6	soda strawberries eggs rotisserie	-1	21	chicken bread bread hot_soup	-1
7	eggs eggs milk	1	22	milk bread	-1
8	hot_soup potatoes	-1	23	potatoes potatoes cereal	1
9	strawberries rotisserie	-1	24	potatoes cereal bread potatoes	1
10	hot_soup milk milk bread	-1	25	chicken potatoes eggs	-1
11	eggs strawberries hot_soup	-1	26	eggs rotisserie eggs	-1
12	hot_soup chicken eggs chicken	-1	27	milk rotisserie strawberries	-1
13	eggs eggs cereal	-1	28	strawberries hot_soup hot_soup cereal	-1
14	eggs strawberries	1	29	chicken soda hot_soup	-1
15	soda cereal eggs	-1	30	rotisserie strawberries cereal milk	-1

Table 5. Weight Values

name	color	weight	size	firmness	temp	price	type	healthy
0.0	0.27	0.0	0.19	0.43	1.0	0.45	0.0	0.24

The weight learning algorithm treats individual attributes as though the patterns formed by the attributes are linearly separable, and thus patterns in which it is necessary to create an intermediate attribute space cannot be detected using the current approach. These include any patterns, such as the complex XOR pattern, in which multiple attributes can be considered to combine to form new "meta-level" attributes that predict pattern classification. It is argued, though, that this limitation does *not* affect performance of the system as a whole, because many standard clustering algorithms (such as those used in this dissertation) require an initial fixed attribute space, prohibiting the dynamic addition of new attributes based on various combinations/permutations of the original attributes.

Further validation of the weight-learning algorithm is performed by repeating the previously described test using multiple attributes (*price* and *healthy*) to partition the data. The clusters were created using the approach described in the next subsection, and Table 6 provides the three clusters obtained. This time, 20 random sets were generated and evaluated using the same evaluation rule (i.e., "do not mix grocery types"). Table 7 presents the results from two separate trials (each with 20 different random sets). While in each case the learned weights are only partially consistent with each other, the weights still identify significant attributes. This is shown by using both learned weight sets to re-cluster the data; using the same approach that created the initial clusters (Table 6). The new clusters are shown in Table 8. Even though the weights vary between trials, and do not identify only the attributes *price* and *healthy*, both final partitions are identical and only differ from the original by switching two groceries (shown in bold). This is a good example of the importance and appropriateness of experienced-based learning.

Table 6. Classification Scheme Used to Reward Sets

Class	Grocery
C_1	soda, cereal
C_2	**milk**, eggs, bread
C_3	chicken, hot_soup, potatoes
	strawberries, rotisserie

Table 7. Weight Values

	name	color	weight	size	firmness	temp	price	type	healthy
Trial 1	0.0	0.37	0.0	0.22	0.61	0.15	0.71	0.0	1.0
Trial 2	0.0	0.39	0.35	0.60	0.97	0.57	0.31	0.75	1.0

Table 8. Classification Using Learned Weights (Both Trials)

Class	Grocery
C_1	soda, cereal
C_2	**strawberries**, eggs, bread
C_3	chicken, hot_soup, potatoes **milk**, rotisserie

Implementation: Conceptual Clustering

Once the attributes have been given symbolic labels and the appropriate weights have been determined, percepts are partitioned using a variation of the conceptual clustering algorithm COBWEB [Fisher, 1987]. COBWEB creates a hierarchical class partition, in which individual clusters are created to simultaneously maximize attribute predictability per cluster, while minimizing the total number of clusters. The output of the COBWEB algorithm is a hierarchical concept tree, such as the one shown in Figure 16, in which more general concepts are located higher in the tree.

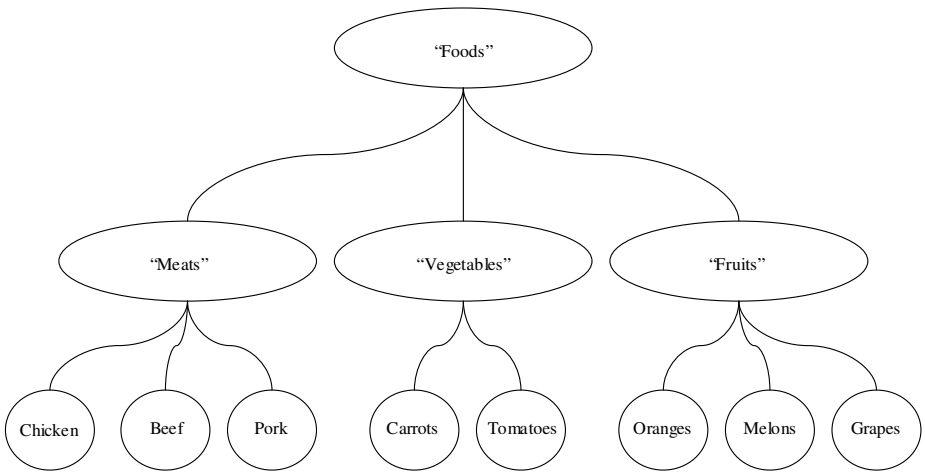


Figure 16. Example Concept Hierarchy for "Foods"

The COBWEB algorithm begins with a single root node at the top of the tree. As new observations are incrementally added to the tree, these observations are filtered down from the root node to the leaf nodes by determining which nodes best predict the attributes for the new observation. At each node, the basic COBWEB algorithm considers four different operations: place the observation in the best child node (if there are child nodes below the current node), create a new child node, split the current node in two, or merge the two best nodes. To determine which operation to perform, each operation is assigned a utility value, known as *Category Utility* (CU_k) [Gluck and Corter, 1985], that indicates the "usefulness" of performing that operation at node k. The standard equation for calculating CU_k is shown in Equation (16).

$$CU_k = P(C_k)\left(\sum_i \sum_j P(A_i = V_{ij} \mid C_k)^2 - \sum_i \sum_j P(A_i = V_{ij})^2\right) \tag{16}$$

Here, $P(C_k)$ is the probability that an observation is a member of class C_k, $P(A_i = V_{ij})$ is the probability that attribute A_i has value V_{ij} over the set of observations, and $P(A_i = V_{ij} \mid C_k)$ is the conditional probability for that attribute-value pair over just the observations in class C_k. When the COBWEB algorithm considers each operation, it calculates that CU_k as if that operation had been performed, and then selects the operation that results in the highest CU_k.

The modification of the COBWEB algorithm for this dissertation involves modifying the CU_k equation to include the specific attribute weights previously determined. This ensures that the concepts created by COBWEB reflect goal-relevance. The modified CU_k equation is shown in Equation (17), where ω_i is the learned weight for attribute A_i.

$$CU_k = P(C_k)\left(\sum_i \omega_i \sum_j P(A_i = V_{ij} \mid C_k)^2 - \sum_i \omega_i \sum_j P(A_i = V_{ij})^2\right) \tag{17}$$

Because the COBWEB algorithm creates a full concept hierarchy, which begins at a root node (representing *all* concepts) and eventually branches into leaf nodes (representing the most *specific* concepts) it is important to have a means of pruning the tree to identify those concepts that are the most significant. One technique to do this is to use the CU_k value at each level of the hierarchy (descending from root to leaves) and prune branches below a certain threshold.

However, Biswas et al., [1995] note that as one descends from the root node to the leaf nodes, the values of CU_k tend to initially increase before eventually decreasing towards the leaves. Therefore, Biswas, et al., [1995] suggests that rather than using a preset threshold, branches should be pruned at the level in which CU_k begins to decrease. This is the method used to prune branches and create the final concepts in this research.

Validation and Evaluation: Conceptual Clustering

Using the weight values shown in Table 9, the modified COBWEB algorithm was used to cluster the 10 sample groceries. The resulting partitions are shown in Figure 5.8. Beside each concept node, the corresponding CU_k value is displayed. Using the pruning technique adopted from Biswas, et al. [1995], the pruned concept tree is used to give the final concepts shown in Table 10. However, here the final concepts have been formed by pruning the partition tree at one step beyond the maximum CU_k values. This is only done to better illustrate concept formation. Ordinarily, pruning on this tree would result in the three categories $\{C_0, C_1, C_2\}$.

Table 9. Weight Values

name	color	weight	size	firmness	temp	price	type	healthy
0.0	0.0	0.0	0.0	0.5	1.0	0.0	0.0	0.25

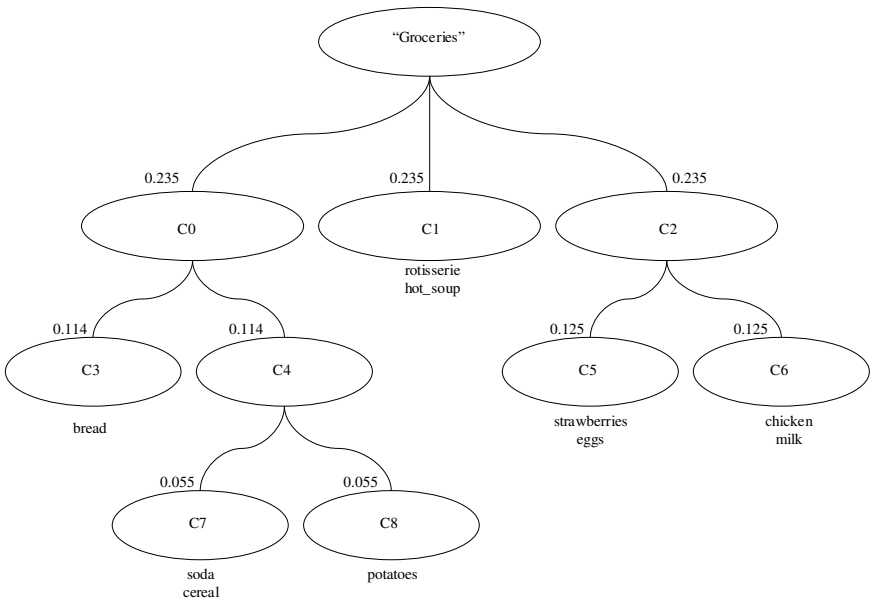


Figure 17. Resulting Partition for Sample Groceries Using Weight Values (Table 9)

Table 10. Final Classification Scheme Using Pruned Tree

Class	Grocery
C_3	bread
C_4	soda, cereal, potatoes
C_1	rotisserie, hot_soup
C_5	strawberries, eggs
C_6	chicken, milk

The COBWEB algorithm provides a simple and intuitive method for classifying groceries. Furthermore, given an understanding of which attributes are most relevant to the current task the modified *CU* equation enables the identification of more specific, goal-relevant classes. The final partitions are based on probabilistic measures, and can be used to both predict unknown attributes or to classify previously unseen groceries. A final test of the effectiveness of this technique is shown by using the pruned partition tree to classify a previously unseen grocery, *oranges*. The new grocery and the resulting classification are shown in Table 11.

Table 11. Weight Values

Grocery									Class
name	*color*	*weight*	*size*	*firmness*	*temp*	*price*	*type*	*healthy*	
oranges	ORANGE	64	504	hard	75	4.99	C3	YES	C_4

Due to the structure of the CU_k function, the COBWEB algorithm has a slight bias towards large classes (i.e., $P(C_k)$) [Fisher, 1987], and thus tends to create fewer concepts that incorporate more training instances at the expense of attribute predictability (i.e., $P(A_i = V_{ij} \mid C_k)$). The final categories are still useful concepts, but often occur at lower points in the partition tree. In this case, the upper portion of the partition tree is reserved for very general concepts that are relatively few in number. However, this bias is acceptable when the goal is to quickly identify general classifications that predict task performance. The ultimate goal in this dissertation, with respect to the concept identification, is not to identify the best natural kinds for perceptual categorization, but to identify rough partitions that are statistically significant with respect to the current goal. Finally, because COBWEB is an incremental algorithm that performs a local hill-climbing search [Fisher, 1987], the final partitions are also sensitive to the initial order of the input. To alleviate this sensitivity, the Anchored Dissimilarity Order (ADO)

algorithm [Biswas, et al., 1995], is used to pre-order groceries before classification. The ADO algorithm selects the next instance for classification by maximizing distance between that instance and the previous *N* instances. In the ADO algorithm, the Hamming distance is used as the distance metric.

Relational Mapping

Once feature vectors have been created from the current state, these vectors must be used to retrieve the utility evaluations that guide the deliberation process. These evaluations are determined by matching each feature vector to internally generated and trained maps where each point on the map is associated with a different evaluation. The relational maps described here are implemented as a set of self-organized neural networks that average and retain the individual feature vectors that have been acquired through experience.

The rational for using the self-organized map technique [Kohonen, 1988] [Kohonen and Somervuo, 1998] is based on the need to combine individual symbolic relations (i.e., feature vectors) into a map structure that facilitates organization and association, while linking the generalized relations to specific appraisals. Additionally, such an auto-associative relational technique is also supported by the psychology and neuroscience literature. Research indicates that encoding experience, typically human episodic memory, is based on the ability to auto-associate different representations of salient, relevant features in the environment as well as the structural relations composed of those features [Burgess, et al., 2002] [Aggleton and Pearce, 2002]. This research indicates that the hippocampus is a critical structure for memory formation and retrieval, and that this structure is highly implicated in performing relational pattern matching [Nadel and Moscovitch, 1997] [Morris, et al., 1990].

Neural research on rats suggests that the ability to perform allocentric spatial processing is strongly tied to the ability to deploy prior experience for future tasks (often maze navigation, etc.) [Morris, et al., 1990] [Moser and Moser, 1998]. Such processing combines incoming perceptual information into spatial arrays (i.e., feature vectors) that reflect the important arrangements and patterns of information, as it exists in the environment. In humans, however, this representational ability is believed to extend beyond spatial patterns and include abstracted temporal, and sequential, patterns of stimuli [Aggleton and Pearce, 2002]. These representations act as “event codes”, or indexing schemes, to retrieve specific memories or activate specific

appraisals [Burgess, et al., 2002] [Marr, 1971]. Therefore, as the current situation is unfolding, relational representations are formed that both facilitate the encoding of that situation in long-term memory as well as activate previously encoded representations, which can ultimately provide access to the various neocortical sites that have been implicated in the storage of long-term, episodic memory [Burgess, et al., 2002].

Machine intelligence research that focuses on relational structures and self-organizing maps (SOMs) has been conducted by Provost, et al., [2006], who use self-organizing distinctive state abstractions (SODA) to learn high-level perceptual features that define distinctive states. High-level actions that traverse between distinct states are learned, and then policies are generated using the distinctive states and high-level actions. In addition, research by Martinez, et al., [1990], Sehad and Touzet [1994], and Smith [2002] also use SOMs to abstract state representations and learn general policies. Smith [2002] uses SOMs to provide discretized states and actions, which are then used to implement a standard Q-learning algorithm. One SOM is used to discretize the continuous input space, while a second SOM is used to form associations and representations of actions. The Q-values for all state-action pairs are updated when reward is received for individual sets. The amount of update is determined by the two neighborhood functions $\theta_s(\ )$, and $\theta_a(\ )$, for the state and action SOMS, respectively. Sehad and Touzet [1994] also use a SOM to implement reinforcement learning, however, their approach involves utilizing the SOM to optimally organize standard lookup tables. Sehad and Touzet [1994] only use numeric weight vectors, defined *a priori*, and store utility evaluations as a member of these vectors.

There has been research by other groups related to the concept of creating relational maps to abstract and represent experience [Kawewong, et al., 2008] [Sudo, et al., 2007]. One such example uses Self-Organizing Incremental Neural Networks (SOINN) [Shen and Hasegawa, 2005] to create basic common patterns and then, hierarchically, form associations between patterns in order to create more abstract pattern representations. Research by Strosslin, et al., [2005] uses recurrent networks to represent navigation information related to location and action. These networks are trained from experience using Hebbian learning [Hebb, 1949]. A method described by Kuipers, et al., [2004] uses the well-known, statistical-based SLAM technique [Thrun, et al., 2005] to derive local maps, while simultaneously developing hierarchical and topological representations between local map features and actions. However, each of these

techniques is primarily focused on localization and navigation of the external environment. The self-organizing, relational maps described here, focus on abstract goal-relevant situation representations in order to provide evaluations that ultimately guide search through the decision-space.

Input/Output

The inputs to the relational maps are the symbolic feature vectors, $\boldsymbol{f}_i^v$, associated with s_i. The outputs are a vector of utility signals, $\boldsymbol{u}_i$, and a vector of confidence values, $\boldsymbol{\chi}_i$, that indicate whether s_i should be approached/avoided, and how well $\boldsymbol{f}_i^v$ matches the trained map, respectively. The vectors $\boldsymbol{u}_i$ and $\boldsymbol{\chi}_i$ are appended onto the current state representation as part of the evaluation signals.

Implementation: Self-Organizing Maps

The importance of the relational map is that it generalizes and forms associations between different episodes (as well as the structural components within those episodes), maintains basic evaluative information, and if needed enables a constant-sized access cost to the tabular episodic memories. The specific technique used to implement these relational maps is the self-organizing map described by [Kohonen, 1988] and [Kohonen and Somervuo, 1998].

A Self Organizing Map (SOM) is a multidimensional neural network that uses unsupervised learning to generate generalized and associative representations of the input space [Kohonen, 1988]. SOMs are composed of an interconnected set of vertices, v_i, and each vertex has an associated weight vector $\boldsymbol{w}_i$ that represents a complete instance, i.e. feature vector. During training, the weight vectors are collectively modified by individual training instances, and over time regions of the map self-organize into basic patterns that reflect the trends in the training data.

To train a SOM, a distance function is used to match each training instance $\boldsymbol{x}_i$ to the "nearest" vertex. Once v_i has been determined all weight vectors in the map are updated using the update rule shown in Equation (18).

$$\boldsymbol{w}_j = \boldsymbol{w}_j + \theta(v_i, v_j, t)* \alpha(t)*(\boldsymbol{x}_i - \boldsymbol{w}_j) \qquad (18)$$

Here, $\theta(v_i, v_j, t)$ is a neighborhood function that determines the amount of update performed at node v_j based on the distance between nodes v_j and v_i (note: $\theta(v_i, v_j, t) = 1.0 \; \forall i = j$). The function $\alpha(t)$ is the learning rate, and both $\alpha(t)$ and $\theta(v_i, v_j, t)$ are designed to decay over time; a measure used to ensure convergence.

Though frequently used for numeric data, SOMs are not restricted to such domains. The key to training a SOM is to have: 1) an appropriate distance function that is defined over the range and types of inputs, and 2) an update function that can modify the desired representation of $\boldsymbol{w}_i$. Research in Kohonen and Somervuo [1998] proposed the use of SOMs for symbol strings and detailed a method for averaging string representations (i.e., symbolic weight vectors). Their method used the Levenshtein (or edit) distance [Levenshtein, 1966] to determine the minimum number of insertions, deletions, or substitutions required to transform one string into another. Dynamic programming was used to find compute an "average" string using all nearest neighbor training instances defined by a discretized neighborhood function, $\theta_d(v_i, v_j, t)$.

Self-organizing symbolic feature vectors is only one of the critical aspects of the required relational maps. It is also necessary to associate evaluative information (utility appraisals), with each individual relational instance, v_i. Therefore, the method for training symbolic SOMs proposed by Kohonen and Somervuo [1998] has been extended to include additional numeric dimensions. The technique involves overlaying two SOMs (one symbolic and one numeric), but treating them (and training them) as a whole. A cross-sectional example is shown in Figure 18. Here, the numeric weight vector has two dimensions.

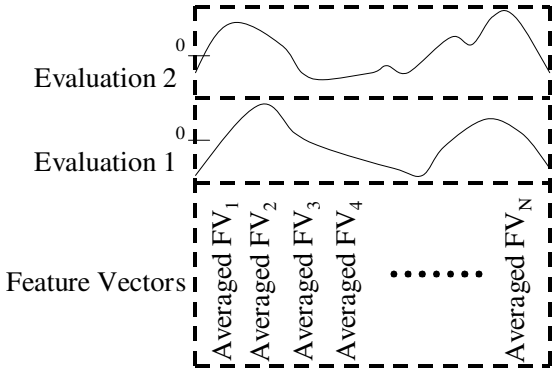


Figure 18. Combined Symbolic and Numeric SOM

Training the hybrid SOM is accomplished by concatenating the feature vectors (from adjacent vertices) in order to compute an aggregate distance function to identify the nearest vertices for each training example. Once the appropriate vertices are found, the individual weight vectors are modified based on their respective counterparts in the current training instance. This allows both the symbolic map and the numeric map to be trained simultaneously. The aggregate distance function is shown in Equation (19), where *Euclidean( )* indicates the standard Euclidian distance. The vector $\boldsymbol{w}_i$ is now the concatenated weight vector with components $\boldsymbol{w}^s_i$ and $\boldsymbol{w}^n_i$, the symbolic and numeric portions, respectively. Likewise, $\boldsymbol{x}_i$ represents the individual training instances with symbolic and numeric portions, $\boldsymbol{x}^s_i$ and $\boldsymbol{x}^n_i$. The values γ_s and γ_n are additional weights that allow preferential status to be assigned to either one of the individual distance functions. Finally, retrieval is based on matching an input vector to one, or both, of the maps. This is achieved by setting γ_s and γ_n appropriately (e.g., $\gamma_n = 0$ to match only using the symbolic map).

$$Dist(\boldsymbol{w}_i, \boldsymbol{x}_i) = \gamma_s * EditDist(\boldsymbol{w}^s_i, \boldsymbol{x}^s_i) + \gamma_n * Euclidean(\boldsymbol{w}^n_i, \boldsymbol{x}^n_i) \qquad (19)$$

The trained SOM is used to provide the appraisal vector $\boldsymbol{u}_i$ by matching the feature vectors $\boldsymbol{f}^v_i$ to the symbolic SOM, and setting $\boldsymbol{u}_i$ equal to the retrieved numeric vector $\boldsymbol{w}^n_i$. However, because the *EditDist( )* function only returns discrete values it is often the case that multiple vertices are "similarly different". For example, the symbol string "car" is equidistance from both "cat" and "bar". Depending on the situation, either match may be acceptable but to eliminate random behavior and to ensure that the best matches are found, a distance matrix is calculated using the distance values for each $v_j \in V$ and then that matrix is smoothed by averaging across the N vertices nearest each v_i. The distance matrix not only indicates those *regions* of the map that best match the input string, but also which specific vertices within those regions are closest to the input string. The distance to the nearest vertex, using the smoothed distance matrix, is returned along with the numeric evaluations stored at that vertex. The distance values for each component of $\boldsymbol{u}_i$ are used to create the confidence vector $\boldsymbol{\chi}_i$ as shown in Equation (20), where d is the individual distance measure and D is the maximum allowable distance (i.e., the size of the largest stored feature vector).

$$\chi_{ik} = 1 - \frac{d}{D} \tag{20}$$

Validation and Evaluation: Self-Organizing Maps

To evaluate the SOM method for creating relational maps, five sample strings were generated and assigned numeric values (i.e., evaluations). Training was performed using five variations of each string in which 25% random noise had been injected (i.e., randomly replacing a symbol within the string, or replacing the numeric value of that string 25% of the time). The five sample strings, evaluations, and randomly generated strings are shown in Table 12. These data were used to train a SOM of size 15 x 15. The trained symbolic SOM is shown in Figure 19 and the numeric SOM is shown in Figure 20. In Figure 20, learned numeric values are indicated by height. The lower right corner of the map (corresponding to the "neutral" region in Figure 19) is associated with evaluations of 0.0, the upper right corner (the "worst" region) is associated with the most negative evaluations, and the upper middle (the "best" region) is associated with the most positive evaluations. The close proximity of the "best" and "worst" regions is based on the syntactic similarity between the underlying symbol strings. Finally, the trained SOM is used to retrieve an evaluation for the string "blets". The computed distance matrix is shown in Figure 21, and the averaged evaluation from the best matching nodes is 1.99944.

Table 12. Five Symbol/Evaluation Sets with 25% Random Noise

"worst"	"bad"	"neutral"	"better"	"best"
worsj -2	bad -1	neutall 0	bettei 1	best 2
worst -2	pay -1	ndutral 0	bntter 1	bjst 2
aorsr -2	bmd -2	neutbal 0	betcer 1	best 2
worsd -2	baq -2	seunral 0	bevler 1	bkzt 2
worst -2	bad -1	nputoah 0	bmtdhr 1	besj 2

bmtdhr	bmtdhr	bmtdhr	bmtdhr	best	best	best	best	best	best	best	worst	worst	worst	worst
bmtdhr	bmtdhr	bmtdhr	bmtdhr	best	best	best	best	best	best	best	worst	worst	worst	worst
bmtdhr	bmtdhr	bmtdhr	bmtdhr	best	best	best	best	best	best	best	worst	worst	worst	worst
bmd	bmd	bmd	bmtdhr	best	best	best	best	best	best	best	worst	worst	worst	worst
bmd	bmd	bmd	bmd	better	better	better	better	better	better	better	worst	worst	worst	worst
bmd	bmd	bmd	bmd	better	better	better	better	better	better	aorsr	aorsr	worst	worst	worst
bmd	bmd	bmd	bmd	better	better	better	better	better	better	aorsr	aorsr	aorsr	aorsr	aorsr
baq	baq	baq	bmd	better	better	better	better	better	better	aorsr	aorsr	aorsr	aorsr	aorsr
baq	baq	baq	pay	pay	better	better	better	better	better	aorsr	aorsr	aorsr	aorsr	aorsr
baq	baq	baq	pay	pay	pay	better	better	better	better	aorsr	aorsr	aorsr	aorsr	aorsr
baq	baq	baq	pay	pay	pay	pay	better	better	better	better	aorsral	ndutral	ndutral	ndutral
pay	pay	pay	pay	pay	pay	pay	seunral	seunral	seunral	neutbal	neutbal	ndutral	ndutral	ndutral
bad	bad	pay	pay	pay	nputoah	neutaal	seunral	seunral	neutaal	neutaal	neutbal	ndutral	ndutral	ndutral
bad	bad	bad	nputoah	nputoah	nputoah	nputoah	nputoah	seunral	seunral	neutaal	neutaal	neutbal	ndutral	ndutral
bad	bad	bad	nputoah	nputoah	nputoah	nputoah	nputoah	seunral	seunral	neutaal	neutaal	neutbal	ndutral	ndutral

Figure 19. Self-Organized Map of Symbol Strings

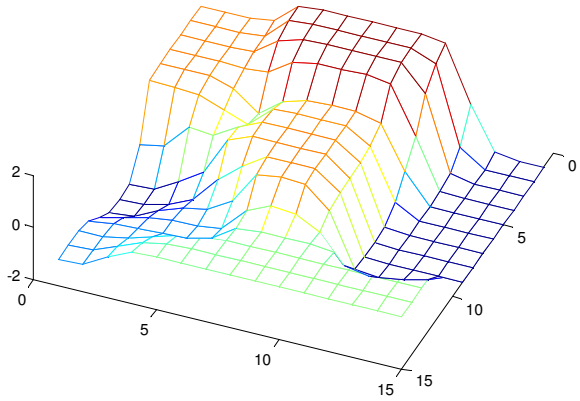


Figure 20. Self-Organized Map of Size 15x15 for Numeric Evaluations

3.20	3.20	3.20	2.83	2.47	2.10	2.10	2.10	2.10	2.10	2.77	3.43	4.10	4.10	4.10
3.20	3.20	3.20	2.83	2.47	2.10	2.10	2.10	2.10	2.10	2.77	3.43	4.10	4.10	4.10
3.47	3.47	3.38	2.92	2.47	2.10	2.10	2.10	2.10	2.10	2.77	3.43	4.10	4.10	4.10
3.73	3.73	3.64	3.11	2.58	2.13	2.13	2.13	2.13	2.13	2.79	3.44	4.10	4.10	4.10
4.00	4.00	3.91	3.30	2.69	2.17	2.17	2.17	2.17	2.38	3.02	3.67	4.10	4.10	4.10
4.00	4.00	4.00	3.40	2.80	2.20	2.20	2.20	2.20	2.62	3.26	3.89	4.10	4.10	4.10
4.00	4.00	4.00	3.40	2.80	2.20	2.20	2.20	2.20	2.83	3.47	4.10	4.10	4.10	4.10
4.00	4.00	4.11	3.82	3.22	2.51	2.20	2.20	2.20	2.83	3.47	4.10	4.10	4.10	4.10
4.00	4.00	4.22	4.24	3.96	3.13	2.51	2.20	2.20	2.83	3.47	4.10	4.10	4.10	4.10
4.00	4.00	4.33	4.67	4.69	4.07	3.13	2.51	2.20	2.62	3.28	3.92	4.14	4.13	4.13
4.33	4.33	4.56	4.78	5.00	4.69	3.99	3.29	2.90	3.00	3.33	3.76	4.08	4.17	4.17
4.33	4.44	4.67	4.89	4.91	4.72	4.26	3.88	3.49	3.27	3.28	3.59	4.01	4.20	4.20
4.33	4.44	4.58	4.71	4.64	4.46	4.30	4.24	4.18	3.86	3.52	3.50	3.80	4.10	4.20
4.00	4.11	4.27	4.42	4.38	4.19	4.11	4.14	4.17	3.86	3.52	3.40	3.70	4.00	4.20
4.00	4.00	4.07	4.13	4.20	4.20	4.20	4.23	4.27	3.97	3.63	3.30	3.60	3.90	4.20

Figure 21. Computed Distance Matrix Using Input String “blets”

Finally, this SOM technique is also used to represent broader situations as they form during the episode. Specifically, this includes a representation of the combined number and types of grocery bags present. However, in this research, rather than force the system to learn goal-relevant attributes that define useful bag types, a single attribute is used to create these types: the number of groceries per bag. This choice is also based on the fact that there is a much smaller set of consistent attributes for grocery bags than for groceries. Using this attribute bag types are created, which are used to develop feature vector representations that incorporate more generalized knowledge of the current situation.

So far, the methods to cognitively process and appraise relevance (fuzzy clustering and feature vector extraction) and utility (SOMs) have been described. Each method relies on a corpus of experience, in this case episodic memory, and learns the appropriate knowledge through offline processing. By themselves, these appraisals are sufficient to enable improved task performance with respect to the task-specific constraints. However, in addition to performing the task correctly, robots must also perform it in a timely manner (i.e., before conditions in the environment change) and, when failures occur, be able to identify which knowledge systems contributed to that failure. Therefore, the next two sections describe components that focus on evaluating system performance to improve deliberation time and enable fast commitment, as well as to provide a basic error signal that indicates which components are not performing correctly and thus which knowledge should not be trusted because further training is required.

Urgency

In this research, appraisals for urgency determine the amount of time allowed for deliberation as well as whether or not the current deliberation process should be interrupted. Interrupt signals are generated in response to actual and expected external conditions. Urgency appraisals made before deliberation begins are used to adaptively preset the decision-making parameters *depth* and *breadth*, and are inspired from the notion of contract and anytime algorithms described in Chapter III. Later appraisals (i.e., interrupts) are based on innate responses to external events, are used to halt deliberation in favor of rapid resource deployment, and are inspired by Sloman's [2001] alarm mechanisms. The cognitive processes that enable these urgency appraisals are trained using offline simulation and rehearsal in order to form

relations between input states, deliberation time, search depth and breadth, and expected solution quality. These processes operate on previously acquired experience to create new internally-generated experience that can then be matched to future situations.

Internal rehearsal is a mental process that occurs in humans and enables people to simulate and practice specific behaviors without the need for physical action [Hesslow, 2002]. This type of mental simulation proceeds "as if" the person was actually performing the behavior and is a critical method by which humans learn. This ability has been accounted for in artificial systems with the work of Jirenhed, et al., [2001] and Erdemir, et al., [2008], as well as the architectural research of Shanahan [2006]. In the work of Jirenhed, et al., [2001] and Erdemir, et al., [2008] a robot uses an "internal world" to rehearse actions and to investigate the consequences of action. Within this internal world, the robot may either possess a model of the physical environment *a priori*, or be required to learn this model or features of the model through training. The latter is the case in the research conducted by Erdemir, et al. [2008], in which the robot is required to develop its own understanding of the physical world as well as its ability within the world; where the robot's ability is dependent on the robot's *unknown* physical morphology.

Research by Shanahan [2006] takes an architectural approach to internal rehearsal and models a dual loop process in which routine behaviors are constantly produced in response to situations, and these behaviors proceed unabated unless interrupted by higher-order cognitive processes. These higher-order cognitive processes run in parallel to the routine behaviors, but perform mental simulation of each routine behavioral response before that response is executed [Shanahan, 2006]. This, however, assumes that the higher-order processes operate at a *higher* frequency than the routine behaviors; an assumption that provides a great deal of difficulty to system developers when applied to physical systems (i.e., robots), a point that has been noted in research by Hall [2007].

Due to the limitation of the approaches described by Hall [2007], and later Ratanaswasd [2008] (in which rehearsal must be performed after the system receives a command but before it executes an action) in this dissertation the rehearsal process is designed as an offline processing tool. This is similar to the approach used by Erdemir, et al., [2008] where rehearsal is performed offline and used to develop relational knowledge that can later be deployed online in a time-efficient manner. First, the robot is allowed to sample its past experience in order to develop a

basic state transition model. This model is used when the robot needs to predict specific changes in the external environment. Second, using basic knowledge about the expected effects of each action the robot mentally simulates its performance on the task, and then self-evaluates its performance on the internally-generated experience using the most recently learned appraisals. The self-evaluation enables the robot, over time, to develop a "sense" of how its performance should be expected to vary by both situation and deliberation time. While it should be acknowledged that the learned relevance and utility appraisals may not always reflect the true appropriateness of the robot's actions, it is argued that regardless of accuracy these mechanisms are those that will be used at the next decision-cycle and thus internal rehearsal can, at the very least, is a form of offline pre-processing in which the robot practices explores its ability to appreciate how much computational "effort" should later be applied during actual tasks.

Furthermore, this type of internal rehearsal has a bias towards optimistic evaluations of system performance and ability. This is due to the fact that the relevance and utility appraisals are trained using the same experience that is later used for rehearsal (i.e., the system is trained and tested on the same set). While this is a fundamental bias inherent in internal rehearsal, it is also an acceptable bias for this research because, in the worst case, such a bias can only cause errors on the task, which then provide new experience to learn from and thus the system's niche is ultimately expanded.

Input/Output

The inputs required for the urgency appraisals are the current state s_i and the feature vectors $\boldsymbol{f}_i^v$. From these inputs, the system estimates the allowable deliberation time t_i^a, the expected solution quality q_i^e, the expected deliberation time t_i^e, and the search parameters necessary to achieve these estimates. If deliberation has already begun, the interrupt signal i_i is generated as needed. Each of the urgency appraisals is appended onto the current state representation within the evaluation signals.

Implementation: Bayesian Networks

Two methods are used to appraise urgency. The first method employs a Bayesian network to predict the amount of time until a significant change in s_i occurs. Significant changes are defined as those changes that remove action possibilities or reduce actions likelihoods. The

specific significant changes used in this research are described in Chapter VI. To identify significant changes, the Bayesian networks determine the probability distributions over all possible future states $s_i' \in S$, given s_i.

Bayesian networks enable probabilistic reasoning through the application of Bayes' Rule. These networks use probabilistic models to relate observable "evidence" variables to unobservable "hidden" variables, and to describe the likelihood of state transitions given estimations of the previous states. The probabilistic models are often either preset or learned from observations. There are two basic models used by Bayesian networks. One model retains the transition probabilities between hidden variables in different states (typically successive states). The other model retains the probabilities associated with observing each evidence variable, given knowledge of the hidden variables.

To make predictions with Bayesian networks, it is only necessary to know the transition model between states. In this research, state transitions are assumed to be 1^{st} order Markov processes (Chapter III), and thus the current state distributions are only dependent on the previous states and not the entire history of states. The training data is comprised of states in which all variables are known and have been sampled from the environment at a rate of 1 Hz. The rational for this sampling rate is to alleviate the computational cost associated with more precise predictions by limiting prediction precision to a range more appropriate for the physical system used in this research. Furthermore, in order for the continuous environment to be representable within the Bayesian network, the environment must be discretized, which also reduces the need for more precise measurements. Finally, this sampling rate enables the time measurements to be made based on the number of prediction steps until a minimum threshold is reached on state likelihood.

Validation and Evaluation: Bayesian Networks

To evaluate this component, groceries were placed at random on a simulated, moving conveyor belt. The continuous state space (i.e., the two dimensional locations on the conveyor belt) was discretized into 16 bins. Two additional bins were also added to represent the position of groceries that were no longer on the conveyor belt. The bin labels are shown in Figure 22. The speed of the conveyor was set to 0.033 m/s. Groceries were placed on the conveyor, sample data

was generated, and the transition model was trained. Figure 23 presents the trained transition model.

For this evaluation, significant changes were defined to be "groceries falling off the end of the conveyor". Given that a grocery was initially placed at the beginning of the belt, Figure 24 shows the probability estimates for that grocery's *predicted* position at times t = {5, 20, 40, 46} seconds. At time, t = 46 seconds the grocery falls off of the conveyor. At this time, the Bayesian network predicts that the grocery is in bin 9 (not on the conveyor) with probability 0.71.

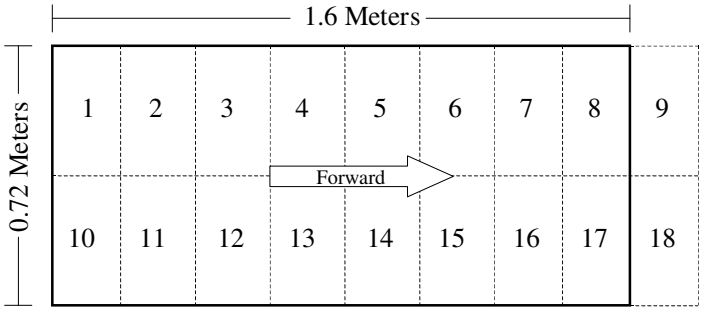


Figure 22. Dimensions and Bin Distribution for Conveyor Belt

Bin_{i-1} \ Bin_i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	**0.800**	**0.200**	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	0.000	**0.804**	**0.196**	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
3	0.000	0.000	**0.824**	**0.176**	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4	0.000	0.000	0.000	**0.820**	**0.180**	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
5	0.000	0.000	0.000	0.000	**0.800**	**0.200**	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
6	0.000	0.000	0.000	0.000	0.000	**0.800**	**0.200**	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
7	0.000	0.000	0.000	0.000	0.000	0.000	**0.800**	**0.200**	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**0.800**	**0.200**	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**1.000**	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
10	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**0.800**	**0.200**	0.000	0.000	0.000	0.000	0.000	0.000	0.000
11	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**0.810**	**0.190**	0.000	0.000	0.000	0.000	0.000	0.000
12	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**0.836**	**0.164**	0.000	0.000	0.000	0.000	0.000
13	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**0.800**	**0.200**	0.000	0.000	0.000	0.000
14	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**0.800**	**0.200**	0.000	0.000	0.000
15	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**0.800**	**0.200**	0.000	0.000
16	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**0.800**	**0.200**	0.000
17	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**0.800**	**0.200**
18	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**1.000**

Figure 23. Learned Transition Model, i.e., $\boldsymbol{P(Bin_i \mid Bin_{i-1})}$

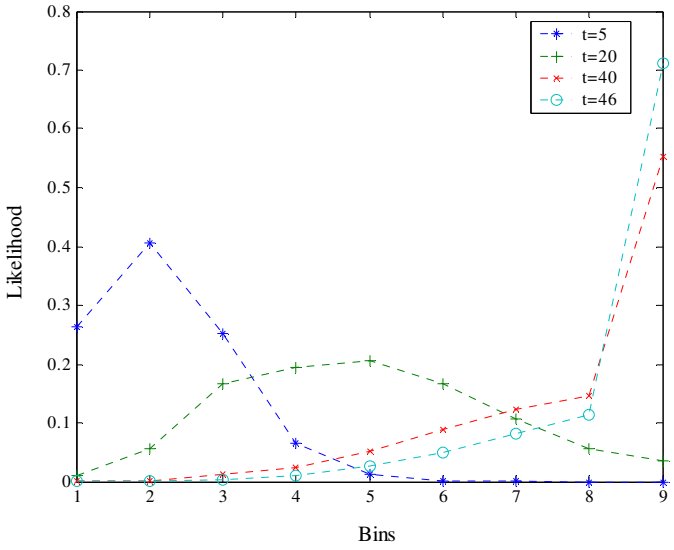


Figure 24. Grocery Distribution at t = {5, 20, 40, 46} Seconds

Implementation: Performance Profiles

The second technique used to appraise urgency requires that the system learn performance profiles [Zilberstein, 1996] to represent relationships between input state, specific decision-making parameters, deliberation time, and solution quality. The performance profiles are generated through offline, internal rehearsal, and require that the system possess some level of domain knowledge (i.e., *relevance* and *utility*). As the domain knowledge improves, the performance profiles become more useful in reducing deliberation time while preserving solution quality. The learned profiles enable the system to estimate how "good" it can expect to do, given the situation. The profiles also enable estimates of deliberation time, given both the current situation and different decision-making parameters. The format used to store performance profiles is shown in Table 13.

Table 13. Storing Performance Profiles

Feature Vectors ($\boldsymbol{f}_i^v$)	Depth (d)	Breadth (b)	Deliberation Time (t_i^e)	Solution Quality (q_i)
$\boldsymbol{f}_0^v$	d_0	b_0	t_0^e	q_0
$\boldsymbol{f}_1^v$	d_1	b_1	t_1^e	q_1
.	.	.	.	.
.	.	.	.	.
$\boldsymbol{f}_n^v$	d_n	b_n	t_n^e	q_n

Once the system has learned to appraise relevance and utility, episodes and states are randomly selected and rehearsed (offline) using the learned knowledge. In other words, the system re-evaluates its past experience, in light of the new knowledge, and forms new plans "as if" the previous situations were actually occurring (again). During the rehearsal process, the system uses the stored state information to create new feature vectors (using relevance appraisals), selectively explores different search parameters *d* and *b*, prioritizes responses (using utility appraisals), and selects actions. For each situation and pair of search parameters, the system records the time required for deliberation as well as the solution quality obtained during goal accomplishment, and uses this information to estimate the expected solution quality and deliberation time for future situations. The solution quality is taken as the total sum of rewards and cost from the current state to the goal state. The result is internally-generated experience, that can be stored and that enables extraction of the performance profiles.

During online task performance each set of feature vectors is matched to the data stored in the performance profiles, and the *N* best matches are returned. The matches are first pruned by removing all instances in which the expected deliberation time exceeds the allowable deliberation time: $t_i^e > t_i^a$. Next, the "best" match is selected from the remaining set by maximizing Equation (21). In this equation, the parameter ξ is used because q_i does not grow at the same rate as t_i^a. Empirical tests suggested a value of $\xi = 2.0$. Finally, the best match is used to set the search parameters *d* and *b*.

$$\frac{q_i^{\xi}}{t_i^a} \tag{21}$$

Validation and Evaluation: Performance Profiles

Proper evaluation of this approach requires that all of the implemented components have been integrated, trained, and used to internally rehearse past experience. Therefore, initial validation of this component is based on the knowledge that performance profiles are known to capture the type of performance-based information required by this research [Zilberstein, 1996] [Zilberstein and Russell, 1995]. Furthermore, accessing each training instance is performed using the feature vector matching techniques that have already been discussed in the section Relational Mapping. Filtering the remaining instances (Equation 21), is a simple, greedy search that

accounts for all three important measures: solution quality, deliberation time, and similarity between the current situation and past experience. Final validation of this technique is deferred until Chapter VI.

Fit

The system developed in this dissertation has multiple, integrated components that each require individual training. The components are connected in an incremental fashion, and the learning required to train each component is critically dependent on the output of the previous (trained) component. Therefore, if any component is improperly trained, system performance, as a whole, will suffer. The final appraisal implemented in this research, focuses on using feedback signals, coupled with the expected performance measures provided by $\boldsymbol{u}_i$ and $\boldsymbol{\chi}_i$, to identify those components that may be underperforming and require further training. This appraisal is based on the notion of a degree-of-*fit* between the system's current knowledge and the situations and tasks that are encountered. This appraisal is not a primary focus of the current research, and it is not within the scope of this research to investigate how the system should best use this knowledge to decrease errors, however, the appraisal for fit is included for completeness with the respect to the discussions of Chapter IV, and to open the investigation on whether such self-evaluative information can be extracted using knowledge of the other appraisals processes. In other words, can the system develop basic appraisals *about* its appraisals? Ultimately this knowledge would be most useful for designers and engineers when evaluating the performance of a system composed of the types of emotion-based processes implemented in this work.

Input/Output

The inputs required to determine *fit* evaluations are the current state s_i and the feature vectors $\boldsymbol{f}_i^v$. In addition, the expected utility appraisals and confidence values from the previous state are required, $\boldsymbol{u}_{i-1}$ and $\boldsymbol{\chi}_{i-1}$. Using these inputs, a vector of ongoing *fit* appraisals, $\boldsymbol{\phi}_i$, is updated and appended onto the current state representation.

Implementation

The method to evaluate fit requires that the system receive feedback during task performance. It is preferable that feedback be received immediately after the execution of an

action, and in many systems this is the case. However, in this dissertation it is *not* necessary to make this assumption. *Fit* can be evaluated whenever feedback (reward) is received, but it is necessary that each feedback signal be attributed to a specific portion of the environment. In other words, the system must know what aspect of the current situation corresponds to each feedback signal. As external evaluations are received, a feedback matrix ϑ is created. The dimensions of ϑ are n x m, where n is the number of constraints/dimensions to evaluate and m is the number of components that have been evaluated. This notation differs slightly from that used up to this point; thus far, the appraisals $\boldsymbol{u}_i$ and $\boldsymbol{\chi}_i$ have been described as vectors of length n because it has been assumed that between each pair of states only one action has been performed, and therefore appraisals are only needed that refer to the state component affected by that action. For fit appraisals, this notation is extended because evaluations need not be received immediately after an action has been performed it is necessary to incorporate the appraisals for all portions of the current state. The matrices U and X are used instead of the vectors $\boldsymbol{u}$ and $\boldsymbol{\chi}$, and both matrices have the same dimension as ϑ.

Using the matrices U and ϑ an error matrix E is calculated that reflects pure difference between the learned utility appraisals and the external evaluations. The calculation of E is shown in Equation (22). In this equation the confidence matrix X is not used to determine appraisal error, because the confidence values are used in the later calculations for the individual fit appraisals.

The fit vector $\boldsymbol{\phi} = [\phi_1^c \ \phi_1^m \ \phi_1^p \ \phi_2^c \ \phi_2^m \ \phi_2^p \ \ldots \ \phi_n^c \ \phi_n^m \ \phi_n^p]^T$ is calculated using the matrix E and X. Each component of $\boldsymbol{\phi}$ corresponds to a different component/appraisal combination to which blame/credit can be potentially applied. Each ϕ^c corresponds to a fit appraisal of the goal-relevant classifications, each ϕ^m corresponds to a fit appraisal of the trained relational maps, and each ϕ^m corresponds to a fit appraisal for the planning algorithm (described in the next section). These values are trained using a specific update rule described by Equations (23-25), respectively, where each rule is repeated for all values of k.

$$E = \vartheta - U \tag{22}$$

$$\phi_j^c = \phi_j^c + \alpha * X_{jk} * (F^c(E_{jk}) - \phi_j^c) \tag{23}$$

$$\phi_j^c = \phi_j^c + \alpha * (1.0 - X_{jk}) * (1.0 - \phi_j^m) * F^m(E_{jk}) + \alpha * X_{jk} * (0.0 - \phi_j^m) * (1.0 – F^m(E_{jk})) \quad (24)$$

$$\phi_j^p = \phi_j^p + \alpha * X_{jk} * (F^p(E_{jk}) - \phi_j^p) \quad (25)$$

In each equation, the term $F(E_{jk})$ is used to provide the target value for the weight update rule. The maximum value of any E_{jk} is confined to the range [-2, 2], because each utility appraisal can have a value between [-1, 1]. The function $F(E_{jk})$ maps this continuous value to either 0/1. If the target is 0, then the specific component either performed correctly or should not be blamed for the error. If the target is 1, then that component performed incorrectly and should share a portion of the blame for each mistake.

The degree to which each component should receive credit for an error (or be complimented for a success) is determined primarily by the confidence value X_{jk}. When X_{jk} is high, then the relational map is confident in its appraisal of the current situation. If an error occurred, it is (most likely) due to either the goal-relevant feature vectors misrepresenting the current situation, or the planning algorithm choosing a path that should not have been chosen. If the absolute value of E_{jk} is greater than 1, then the expected utility evaluation and the final external evaluation were of opposite sign, and the error is most likely due to the goal-relevant feature vectors misclassifying the current situation. For example, a dangerous situation has been represented, through clustering and data compression, as similar to a previous good situation, and thus the system has failed to separate these situations. If, however, the absolute value of E_{jk} is less than 1, then the system knew the correct answer (i.e., the final evaluation and the internal evaluation shared the same sign), but pursued an incorrect course of action anyways. Reasons why this may occur are discussed in Chapter VI.

The above explanations cover Equations (23) and (25). Explanation of Equation (24) is provided by noting that when confidence is low, errors are most likely due to insufficient experience contained in the relational map or incorrect generalization within the relational map. The degree with which the map should then be blamed for any errors is proportional to the size of those errors. If confidence is high, however, then the relational map should receive credit for any success in proportion the “lack of errors” present in the utility appraisals. Finally, in each of

the equations just described, the value of α is used to reduce the credit/blame assignment based on the amount of risk taken by the system. The rational is that in risky situations, the system is less likely to know the correct response and thus credit/blame should be reduced accordingly.

Validation and Evaluation

The process of developing fit appraisals requires an integrated system that performs actions and receives feedback/reward for those actions, while simultaneously forming expectations of its own performance. Thus initial validation focuses on the equations for determining the individual fit appraisals. The general form for the update equations is derived from the well-known regression techniques used to learn basic patterns [Mitchell, 1997] [Witten, 2000]. However, these equations have been slightly modified to reflect the specifics of the current system. The standard learning rate has been replaced by "risk" factor that reduces the amount of update in risky situations. In addition, the confidence X_{jk} is used to mediate which component receives credit/blame, while the error E_{jk} is used to determine whether credit or blame is appropriate and how much should be given. This is intuitively based on knowledge of the underlying system and the fact that the value X_{jk} reflects a measure of fit between the situation and the current knowledge structures, while the value E_{jk} dictates whether the fit was good or bad. Experimental validation of this component is deferred until Chapter VI.

Planning

The planning algorithm performs a depth-first search through the current decision space. The retrieved evaluations from the relational maps are used to order potential responses from best to worst. The breadth search parameter is used to keep the *b* best responses, and to prune the rest. At each planning step the best response is chosen, the state resulting from application of that response is expanded and then used as the input for the next decision-cycle. Once the maximum search depth has been exceeded, or there are no more states to expand, the algorithm returns to the previous depth and expands the next best state. The current best plan is maintained in the form of a policy over the current search window. If an interrupt signal is generated, or planning must be stopped, the best plan is returned and used to execute actions.

Input/Output

The input to the planning algorithm is the current state s_i, coupled with utility appraisals $\boldsymbol{u}_i$, confidence measures $\boldsymbol{\chi}_i$, search depth d, search breadth b, and any interrupt signals i_i. The output of the planning algorithm is the current best, local policy π_L, and the next state s_i' to be expanded. Here s_i' is used to denote successive states in the planning process, while s_{i+1} is reserved for the state that results from the chosen action.

Implementation

The planning algorithm implements a depth-first search through the decision space, S. At each step of the search, the state with the highest priority is selected and expanded. Expanded states are passed back to the beginning of the decision-cycle, where they are assigned feature vectors and appraised with respect to the current goals. The appraisals are then used to determine the new state's priority, which ultimately determines when (and if) that state is expanded. The depth and breadth parameters (d and b) are used to constrain the search to a local window around s_i. If the search depth exceeds d, or there are no further states to expand, the algorithm backs up to the previous depth in which states still remain to be expanded. During the search process, only the b best states are considered for expansion, the remaining states are pruned from the search. As states are expanded, the evaluations of those states are used to update a local policy π_L, defined over S. If an interrupt signal i is generated, planning is stopped and π_L is used to determine the response. The current planning algorithm is not a true *anytime* algorithm due to the fact that it is only capable of a fixed, one-step lookahead, and thus solution quality may not be a strict, montonically increasing function of deliberation time. However, as experience increases and the trained components (i.e., goal-relevant classifications and relational maps) improve, it is expected that the planning algorithm should begin to approximate this standard anytime property. Pseudo-code for the recursive portion of the *Plan* component is shown in Figure 25.

```
Plan(s_i)

    A_i = PossibleActions(s_i)
    S' = PossibleSuccessors(s_i, A_i)

    if(current_depth > d  or  Empty(S')  or  Interrupt( ) )
        return 0
    else
        For each s_j ∈ S'
                {u_i, χ_i} = AppraiseUtility(s_j)
                ρ_i        = SetPriority(u_i, χ_i)
        Order(S')
        Prune(S', b)
        current_max = -θ
        For each s_j ∈ S'
                ρ_j = ρ_j + Factors(a_j) * Plan(s_j)
                if(ρ_j > current_max)
                        UpdatePolicy(π_L, s_i, a_j)
                        current_max = ρ_j
        return current_max
```

Figure 25. Pseudo-code for the Recursive *Plan( )* Algorithm

At each iteration, the current state that should be evaluated and expanded is passed into the *Plan( )* algorithm. The various appraisals are treated as variables that can be requested and filled in as necessary. The *relevance* and *utility* appraisals, which are contained in the current feature vectors and the vectors $\boldsymbol{u}_j$ and $\boldsymbol{\chi}_j$, are locally requested for each state as it is encountered during the planning process. The *urgency* appraisals $\{d, b, i_i\}$, however, are considered functions of the external state and not the internal search process, and therefore these inputs are generated by the external state and are not products of the search through *S*.

The first step in the *Plan( )* algorithm is to determine the set of possible actions A_i that can be performed in response to s_i. This is done using the *PossibleActions( )* function. Next, A_i is used by the function *SuccessorStates( )* to determine the set of possible successor states *S'* that would result from applying each action $a_j \in A_i$ to s_i. The algorithm then determines whether it should backup to the previous planning step, or continue forward. Three different checks are made when determining whether or not to proceed. The algorithm checks whether the maximum depth *d* has been exceeded. If the current depth is greater than *d*, the search is backed up to the previous search step. The algorithm checks the set of successor states, *S'*. If *S'* is empty, then

there are no actions that can be performed and planning is not required in this step. Finally, the algorithm checks for any pending interrupts to the deliberation process. This last check is performed using the *Interrupt( )* function.

The *Interrupt( )* function polls the cognitive processes that appraise urgency and check for significant changes in s_i. If a significant change has occurred the interrupt signal i_i is generated and deliberation is stopped so that an action may be performed. If the system selects to continue deliberation, then the algorithm assigns priority values, ρ_j, to each of the possible successor states within *S'*. This requires appraising each $s_j \in S'$ with respect to the current goals. The values $\boldsymbol{u}_j$ and $\boldsymbol{\chi}_j$ are determined using the trained relational maps. The priority values then determine when, and if, s_j should be expanded. To determine ρ_j a multi-dimensional reward signal is calculated as shown in Equation (26).

$$\boldsymbol{eval}_j = \boldsymbol{\chi}_j * \boldsymbol{u}_j \tag{26}$$

The priority for each state $s_j \in S'$ is determined by taking the Euclidean distance between $\boldsymbol{eval}_j$ and the preset vector, $\boldsymbol{\psi}$, that represents the highest values obtainable for each $eval_{jk}$. The calculation of ρ_j is shown in Equation (27).

$$\rho_j = \left(\sum_k (eval_{jk} - \psi_{jk})^2 \right)^{\frac{1}{2}} \tag{27}$$

Once each s_j has been given a priority value, the *Order( )* function arranges *S'* from highest priority to lowest priority, to ensure that the highest priority states are expanded first. The states with the lowest priority are pruned using the *Prune( )* function. As each remaining s_j is searched, the priority value is modified by adding the weighted sum of the best ρ_j' reachable from s_j. During the search, if a successor state with higher priority is found, the function *RevisePlan( )* is used to update the policy π_L.

Validation and Evaluation

Experimental validation of the planning component is deferred until Chapter VI, due to the fact that this component requires an integrated system in order to be tested. However, it is still appropriate to discuss the rational for this component's design. As it has been discussed throughout this dissertation, robotic decision making must be sensitive to real-world constraints, such as deliberation time and interruption. In addition, rarely are complex environments encountered in which optimization along a single dimension is sufficient for optimal performance. Therefore, the planning algorithm used by robot operating in a complex environment must be capable of exhibiting anytime-like properties, even when knowledge of the world is insufficient and incomplete.

The implemented planning algorithm must also be designed to appropriately utilize the various appraisals provided to it, which may extend beyond utility measures, towards adaptive parameter tuning, interrupts, and relevance detection. In the current planning algorithm, *relevance* is captured through the creation of goal-relevant feature vectors that facilitate access and retrieval from stored relational maps. *Utility* is used to arbitrate between response options. In this approach, the learned utility appraisals enable intelligent decision making, however, these appraisals are modulated not by the likelihood of occurrence, but by their current goal significance and the degree to which they match (i.e., confidently believed to applicable in) the current situation. *Urgency* appraisals enable the system balance fast commitment against deliberation, as well as to interrupt deliberation when necessary. The planning algorithm can be adaptively tuned using the parameters depth and breadth, which facilitates deliberation that respects the potential time-critical nature of different tasks. In addition, because the planning algorithm maintains the current best plan, in the form of a local policy, deliberation can be interrupted at any time; a property that facilitates real-time reaction. Finally, though *fit* is not explicitly incorporated into the planning process, maintaining a local policy is useful when retrospectively evaluating performance. It is argued here, and shown through comprehensive results in Chapter VI, that the implemented planning algorithm utilizes each appraisal to adaptively improve performance and generate more cognitive behavior.

System Integration

A lot of material (components, equations, and algorithms) has been presented in this chapter. This is necessary to truly investigate the types of emotion-based, cognitive processing that might be applicable to robots. The appraisals that facilitate the development of emotional states are not standalone evaluations, but rather a collection of cognitive processes that *together* enable appropriate, adaptive behavior. Urgency appraisals can only mediate deliberation if the system has basic knowledge of its own performance abilities, and such knowledge is intricately related to situation-based expectations and predictions of utility. Interpreting utility requires that relevant features of the situation be detected. Therefore in new situations, an adaptable system must (at minimum) be able to appraise what is relevant in that situation, determine the utility of individual responses, and be able to adjust its own deliberation time as dictated by its goals and the situation. With this collection of appraisals, comes the need to evaluate the amount of fit between each learned component and the new situation. Though it is beyond the scope of this work, this final appraisal could be used to allow more global adjustments to be made that impact the nature of the individual relevance, utility, and urgency appraisals. In this research, fit is only used to identify "problematic" components.

The behavior produced by a complex system with the ability to appraise relevance, utility, urgency, and fit is rarely optimal, but it should also be equally rarely disastrous (see Chapter IV). Instead, the behavior induced by the concerted influence of emotion-based appraisals is often simply sufficient. Proper investigation of how these appraisals may be derived and utilized to impact behavior requires an understanding of each appraisal's architectural role within the cognitive control process. It is argued that the control system that has been described throughout this chapter is ideally suited for integration within a complex, cognitive architecture, such as the ISAC architecture. The control system requires a buffer for perceptual and state information. This is closely related to the ideas of short-term and working memory in the ISAC architecture. In addition, the creation of goal-relevant dynamic representations (i.e., feature vectors) is fundamentally linked to relevance appraisals and can be compared to some of the functions assigned to ISAC's Working Memory System. In this context, working memory must interact closely with the current goals to appropriately filter and process incoming stimuli. The control system utilizes utility to prioritize responses, make predictions, and recursively generate plans. Higher-order control systems (e.g., those for *urgency* and *fit*) also mediate the planning

process. In the ISAC architecture, this functionality is captured in the Executive Control Agent, where planning is performed by the CEA, predictions and evaluations are performed by the IRS (interacting with the Relational Maps), mediation of the decision cycle is accomplished by the Affect Agent, and error-tracking is performed by the combination of the Goals Agent, WMS, and IRS. A graphical representation of how the current control system integrates with the ISAC architecture is shown in Figures 26.

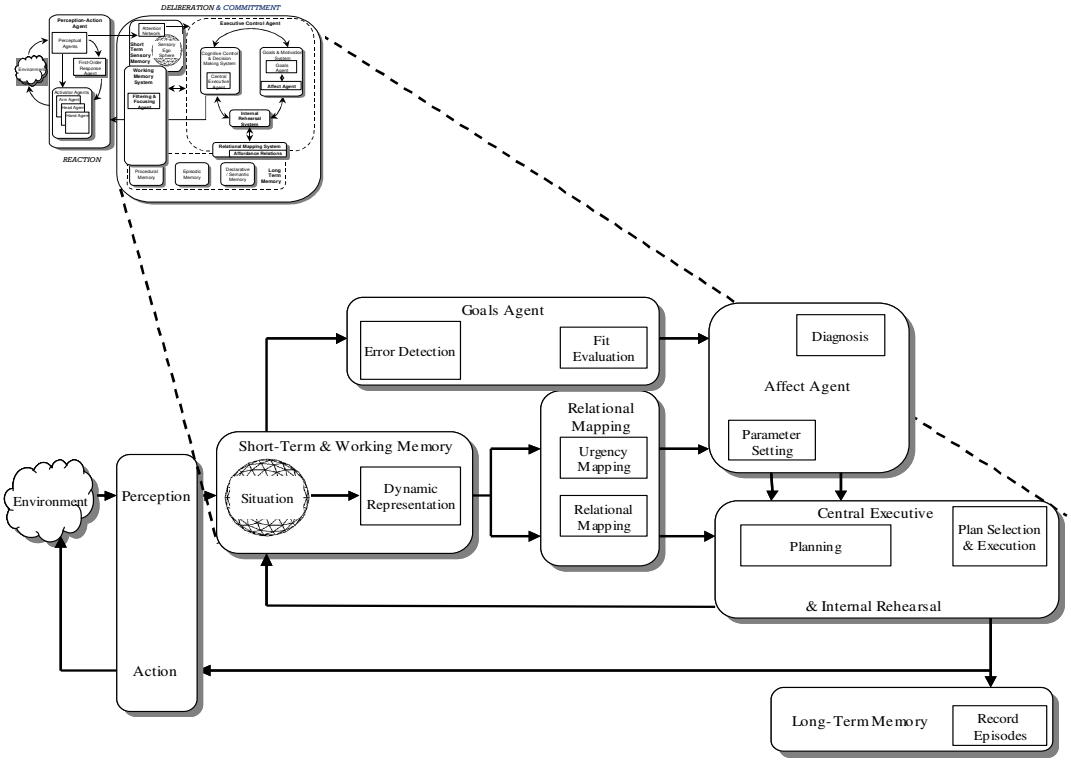


Figure 26. Revised View of Implemented Control System

CHAPTER VI

EXPERIMENTAL DESIGN, RESULTS, AND DISCUSSION

Experimental Hypothesis and Assumptions

The implemented control system is designed to use multiple levels of cognitive processing to derive (from experience) appraisals that usefully and adaptively guide decision making. To test and evaluate this system, experiments are needed that require the system to identify what is *relevant* in the current situation, assign *utility* as appropriate, manage deliberation time when *urgent* action is required, and evaluate the level of *fit* between the current knowledge structures and the situation. In addition, the designed experiments should require the satisfaction of multiple, simultaneous concerns, in order to stress the system's ability to use its appraisals to perform balanced decision making. Finally, the experiment must enable specific evaluation of how well the system:

1. Learns domain knowledge and is able to deploy that knowledge for improved task performance.
2. Learns goal-relevant classifications that can be used to create useful feature vectors that capture significant aspects of the situation.
3. Improves on the task with experience derived from both random exploration and planned exploitation.
4. Uses the learned domain knowledge to develop appreciations of its own performance ability, including deliberation time and solution quality.
5. Deploys the acquired performance-based knowledge to reduce deliberation time without sacrificing solution quality.
6. Identifies components that are not performing well, and require further training.

During each experiment, it will be necessary for the system to learn to correctly appraise each situation along multiple dimensions in order to appropriately balance behavior selection. No single constraint will be given *a priori* the status "more important", therefore while the system attempts to balance behavior selection, there will be no preset constraints on how such balance is achieved. This should cause the system to select which constraints it deems to be more

important, using some criteria. Furthermore, because many of the algorithms used to implement the system are unsupervised, system performance may exhibit oscillatory behavior while simultaneously observing an underlying trend, e.g., improvement or non-improvement. This should cause the standard performance curves to look oscillatory and, at times, erratic. In addition, due to the fact that behaviors are preferentially ordered using a one-step lookahead, as the accuracy of the relational maps increases the system's initial reactions will become more accurate. This increase in accuracy should result in the system requiring less time to obtain an appropriate solution, which will improve performance when appraising urgency and mediating the deliberation cycle.

Finally, because the system is, by design, an experiential learner, aspects of the behavior may demonstrate sensitivity to the type of experience encountered as well as the underlying distributions from which that experience is drawn. In other words, increasing experience may cause the system to become temporarily trapped in a local niche. Within the niche, performance should be very accurate, but near the fringes of the niche performance should decrease. The location of this niche determines, in part, the type of new experiences acquired by the system (i.e., it will begin to take different paths through the decision space). The new experience should expand the niche as well as, possibly, shift the location of the niche. Therefore, the system should improve performance while simultaneously choosing more difficult situations. Each of the issues just listed are hypotheses on system performance, and are more formally listed as follows:

1. The system should learn to balance multiple constraints, and in so doing, to select the most important constraints as a function of the statistical processing of its own experience.
2. The unsupervised learning performed by the system should cause performance to oscillate, but this oscillation should be centered about a specific trend (i.e., improving performance).
3. With increased training, the system's initial appraisals of a situation will prove to be more correct, and thus it will be easier to mediate decision making with respect to deliberation time, which will improve performance on urgency appraisals.

4. The learned appraisals will be sensitive to the experience used to train the system, and as performance improves the system will (through its improved decision making) encounter more difficult situations, which will act to counterbalance the improved performance.

Overview of Grocery Bagging: Experimental Layout

The experiment used to test the implemented control system is designed based on the "everyday" task of bagging groceries. In this *grocery-bagging* experiment, the robot is situated at the end of a conveyor belt and must successfully bag all groceries that appear on the belt. To the left and right (and within the robot's workspace) are tables upon which boxes (grocery bags) are placed. As groceries are deposited in front of the robot, the robot must place each grocery in a bag, and in so doing, observe certain constraints. The experimental layout is shown in Figure 27.

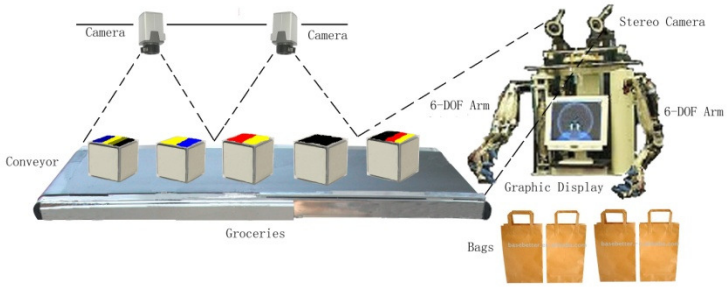


Figure 27. Experimental Layout for Grocery Bagging (Developed by Huan Tan)

While bagging groceries, three constraints must be observed to ensure success. These constraints are:

1. *Do not destroy any groceries* (e.g., "crush bread with potatoes", "break eggs with milk", "mix hot and cold items").
2. *Do not overload a bag* (e.g., "20 lbs of groceries in a single bag").
3. *Do not use too many bags* (e.g., "10 groceries and 10 bags").

An external critic determines whether or not the robot has successfully adhered to these constraints, and provides reward based on this determination. The specific evaluation functions used within the critic are listed in section Overview and Description of Simulation Experiments.

Hardware

The hardware platform used for this experiment is the ISAC humanoid robot, shown in Figure 28 [Kawamura, et al., 2008]. ISAC has two 6 Degree-of-Freedom (DOF) arms, powered by pneumatic air muscles. Proprioceptive sensors attached to each arm joint enable feedback control of ISAC's arms. Attached to each arm is a two-fingered, "pincher" gripper that enables ISAC to grasp simple objects. ISAC has two microphones for sound detection, and a stereovision head for visual localization and tracking. There are no proprioceptive sensors attached to the stereovision head; rather four stepper motors are used to enable open-loop pan/tilt control for each camera.

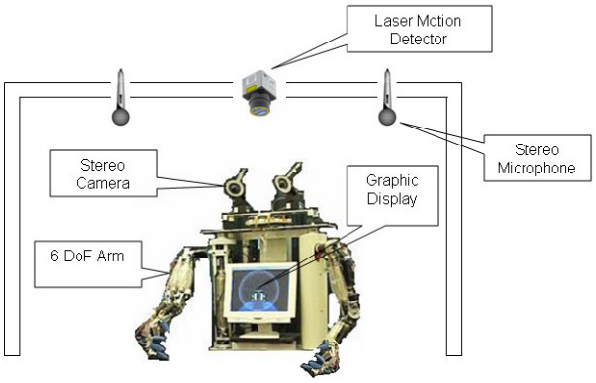


Figure 28. ISAC Humanoid Robot [Kawamura, et al., 2008]

For the grocery-bagging experiment, two stationary, additional cameras are used to increase the range of ISAC's visual tracking system, and to "free up" the stereovision system for more specific tracking (e.g., tracking the current grocery to be grasped). These cameras are mounted above the conveyor belt and provide estimates of each grocery's position on the belt. Both of the additional cameras are generic, USB webcams.

The conveyor belt used for these experiments is a modified treadmill in which the control console has been replaced with a computer control program connected via a standard 25-pin parallel port. Using the computer controller, ISAC can start/stop the conveyor as needed, however, for these experiments ISAC is not able to further vary the speed of the conveyor. A collection bin is located at the end of the conveyor belt to collect groceries that have not yet been bagged. Finally, the entire conveyor-bin-camera system is raised off the ground, in order for ISAC to be able to reach groceries on the belt. The conveyor-bin-camera system is shown in Figure 29.

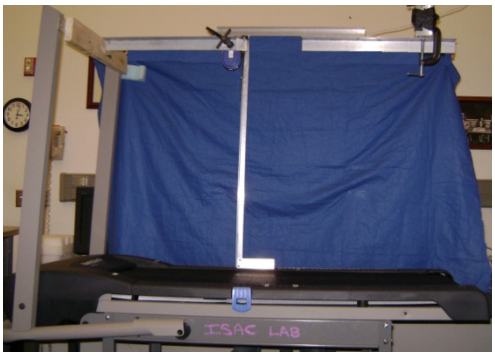


Figure 29. Conveyor-Bin-Camera System

Groceries

As described in Chapter V, each grocery has nine different attributes, which are either symbolic or numeric. A total of 50 groceries are used for this experiment, and are listed in Table 14, however, during experimentation the complete set of groceries will be divided into different subsets for training and testing purposes. These subsets are presented, when appropriate, later in this chapter.

Testing and evaluating cognitive robotic systems on a task such as grocery bagging is difficult because the robot's morphology and physical constraints often limit the possible scenarios that can be used for experimentation. Thus simulation analysis is critical for system evaluation, but ultimately final testing must also involve the type of real-world scenarios that the robot may potentially encounter. This, however, includes the type of scaffolded simple problems used in much of modern robotics research. To this end, the groceries used for this dissertation are brightly colored objects that have been designed to be perceived easily and to afford grasping as defined by ISAC's physical attributes. In addition, while the colors listed with the attributes for each grocery are based on measurements of real groceries, the colors used for grocery detection have been chosen to make detection easier and, therefore, do not necessarily correspond to the original attributes. Loosening this restriction alleviates pressure on the Perceptual Agents by allowing them to associate single colors with objects, and not requiring them to simultaneously track multiple objects of the same color (e.g., *soda*, *granola*, and *frozen_pizza*). A small handle is also attached to each grocery to improve its graspability.

Table 14. Groceries and Attributes

name	color	weight (oz)	size (in^3)	firmness	temp.(F^o)	price ($)	type	healthy
granola	RED	10	48	hard	75	2.29	T_1	YES
tissue	WHITE	8	252	soft	75	1.45	T_7	NA
soda	RED	64	294	hard	75	1.50	T_6	NO
ice_cream	BLUE	22.56	70	hard	32	2.99	T_4	YES
frozen_pizza	RED	10.5	81	hard	32	1.25	T_1	NO
chicken	PINK	24	108	hard	32	3.05	T_5	YES
milk	WHITE	68.8	160	hard	45	1.67	T_4	YES
cereal	BLUE	17	227.5	hard	75	3.15	T_1	NO
oranges	ORANGE	64	504	hard	75	4.29	T_3	YES
potatoes	BROWN	80	504	hard	75	3.99	T_2	YES
strawberries	MAGENTA	16	140	soft	55	2.30	T_3	YES
bread	BROWN	16	325	soft	75	0.89	T_1	YES
rotisserie	BROWN	32	160	hard	150	7.99	T_5	YES
eggs	YELLOW	24	144	soft	45	1.59	T_5	YES
hot_soup	WHITE	12	48	hard	175	3.50	T_2	YES
chips	YELLOW	12	378	soft	75	2.68	T_6	NO
yogurt	BLUE	24	75	hard	45	1.59	T_4	YES
cookie_dough	YELLOW	16	31.5	soft	75	2.50	T_6	NO
frozen_fruit	BLUE	48	180	hard	32	7.00	T_3	YES
waffles	YELLOW	12	121.5	hard	32	2.79	T_1	YES
frozen_vegetables	WHITE	16	96	hard	32	1.19	T_2	YES
cheese	PURPLE	8	45	soft	45	2.49	T_4	YES
ketchup	RED	32	84	hard	75	1.58	T_2	YES
popcorn	BLUE	20	108	hard	75	2.79	T_1	NO
hot_chocolate	BLUE	2.25	35	hard	75	1.00	T_6	YES
trash_bags	GREEN	48	256	hard	75	10.99	T_7	NA
ziploc_bags	BLUE	10	99	hard	75	2.99	T_7	NA
marshmallows	WHITE	16	110	soft	75	2.00	T_6	NO
fruit_juice	MAGENTA	64	198	hard	75	2.18	T_3	NO
coffee	RED	23	180	hard	75	8.41	T_7	YES
tuna	BLUE	20	85.75	hard	75	7.39	T_5	YES
cheezits	RED	11.5	162	hard	75	2.00	T_1	NO
ritz_chips	GREEN	80	189	hard	75	3.00	T_1	NO
vegetable_soup	BROWN	18.8	31.25	hard	75	2.50	T_2	YES
choc_cookies	BROWN	16	135	hard	75	2.99	T_6	NO
peanut_butter	BROWN	28	45	hard	75	2.50	T_6	YES
macNcheese	BLUE	7	39.38	hard	75	0.75	T_1	NO
vegetable_oil	YELLOW	48	132	hard	75	2.69	T_6	NO
spaghetti	BROWN	12	22	hard	75	1.19	T_1	YES
rice	WHITE	32	90	hard	75	1.99	T_1	YES
green_beans	GREEN	14.5	25	hard	75	0.50	T_2	YES
bagels	BROWN	20	72	soft	75	1.99	T_1	YES
ribs	BLACK	20	336	hard	150	10.99	T_5	NO
tomatoes	RED	20	90	soft	75	2.75	T_3	YES
lettuce	GREEN	16	156	hard	60	2.49	T_2	YES
cucumbers	GREEN	28	72	hard	50	2.40	T_2	YES
bananas	YELLOW	32	216	soft	75	4.00	T_3	YES
cake	BROWN	44	549	soft	75	6.99	T_6	NO
ground_beef	RED	36.8	120	hard	45	7.57	T_5	YES
teriyaki_bowl	BLACK	17	105.88	hard	150	5.29	T_5	NO

Software

Multiple software components have been developed for this research. In addition to the implemented control system (Chapter V), components have been written to track groceries, control the conveyor belt, assign grocery labels to percepts, and evaluate task performance. The vision software used by ISAC's Perceptual Agents to identify and track groceries is based on basic computer vision algorithms implemented with the OpenCV software library [OpenCV]. As groceries are placed on the conveyor belt, a command interface is used to dynamically assign grocery labels to the new perceivable objects (i.e., percepts). This enables both groceries and colors to be re-used if necessary. Using this interface, grocery sequences can be either preset, or entered at runtime.

Another software component specifically developed for this research, yet independent of the implemented control system is the external critic that evaluates system performance. This critic monitors the external state and uses preset evaluation rules to provide feedback to ISAC. The specific preset rules are discussed in detail in section Overview and Description of Simulation Experiments. Monitoring the external state is performed through interaction with the Sensory EgoSphere, as well as monitoring the commands given to the Activator Agents. Information from the SES informs the critic of the location of specific groceries, while information from the Activator Agents signals what actions are being performed and, thus, what postconditions (see section Behavioral Repertoire) to expect.

Finally, a simulation environment has been developed to enable rapid evaluation and testing of the implemented system, as well as to provide the means to perform multiple, repeatable experiments in a manner that would simply not be possible with the physical system. The simulation environment is written in ANSI C++ and uses an existing model of ISAC [Ratanawasd, 2007] coupled with a model of the conveyor belt to simulate the grocery-bagging environment. Graphics in the simulation are performed using OpenGL. Experimental trials are performed by either preloading a stored state, preloading a stored episode, or selecting groceries from a menu. The simulation environment can be run in two different display modes: fast or slow. These modes, however, only determine whether (or not) to display the graphical interface. When the simulation operates in *slow* mode, graphics are used to display system behavior, but this mode is primarily for display purposes only, or if a user wants to interactively select groceries. Graphics are not used when the simulation operates in *fast* mode. This mode is

primarily for comprehensive evaluation and testing purposes. In fast mode, states or episodes must be preloaded. In both modes, ISAC's deliberation processes operate at the same speed, however, in the *fast* mode actions, and their effects, occur immediately. The graphical display (slow mode) is shown in Figure 30.

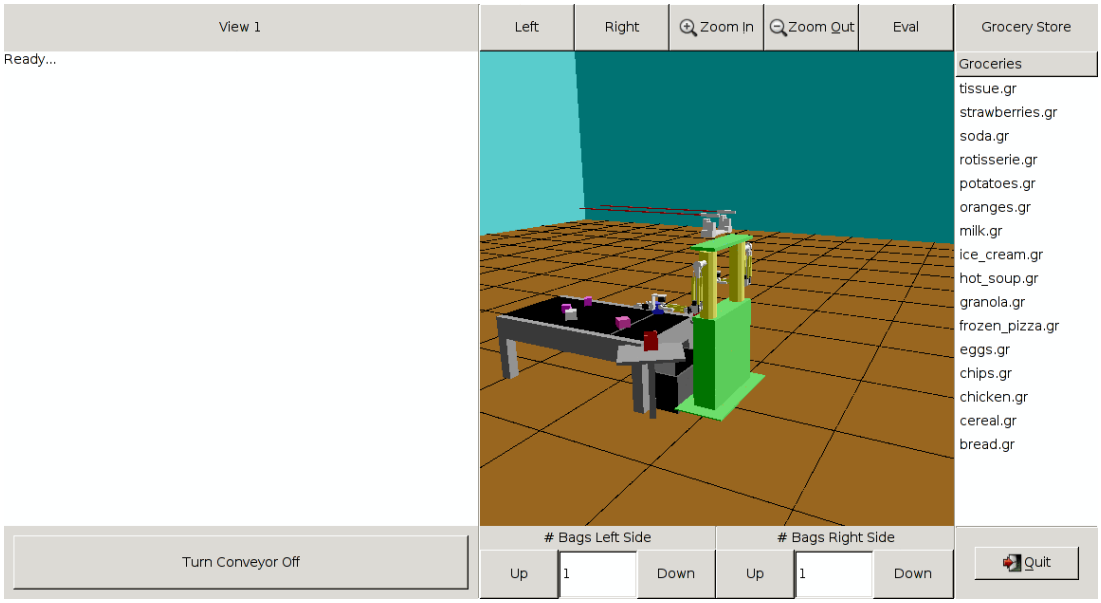


Figure 30. GUI for Grocery Bagging Simulation

State Representation

For this research, the state representation consists of three basic types of information: *Percepts*, *1st Order Logic Elements*, and *Evaluation Signals*. Goals are not explicitly represented within the state formulation, but rather are implicitly represented within the deliberation process as "attractor" states in the decision space, *S*. In this research, attractor states are those states in which no further actions can be performed. Excluding goals from the state representation enables more appropriate system evaluation and testing by eliminating the possibility of dynamic goal switching during task performance. It should be noted, however, that within the ISAC architecture goal switching (i.e., switching from "bagging groceries" to another task such as "greeting people") can be allowed through the use of the Goals Agent (Chapter V). In this case, the state representation would need to be extended to incorporate the current goal as well as the specific evaluation dimensions for that goal.

The allowable percepts for this experiment are groceries and grocery bags. The groceries are listed in Table 14, and are visually detected using color identification. Grocery bags are not visually tracked, but rather preset locations are input to the system and used to identify specific bags. Individual grocery bags are differentiated by name (e.g., bag_1, bag_2, etc.) and, for this research, it is assumed that grocery bags remain stationary and that groceries and grocery bags are the only percepts that need to be tracked. Other percepts, such as people (face detection), are ignored to ease deliberation requirements and enable better comparison across trials. However, this assumption can be loosened if further logic elements are included that represent the additional perceptual information for the planning components.

The allowable 1st order logic elements are based on the implicit goal of bagging groceries, and are presented in Table 15. These elements are used for planning and are determined from the current percepts as well as the previous state and knowledge of the last action performed.

Table 15. Allowable 1st Order Logic Elements

Element	Description
InRightHand(g_i)	Grocery g_i is in ISAC's right hand
InLeftHand(g_i)	Grocery g_i is in ISAC's left hand
BagEmptyRight(b_i)	Bag b_i is empty on ISAC's right side
BagEmptyLeft(b_i)	Bag b_i is empty on ISAC's left side
InBag(g_i,b_j)	Grocery g_i is in bag b_j
ReachableRight(o_i)	Object (grocery or bag) o_i is reachable by ISAC's right hand
ReachableLeft(o_i)	Object (grocery or bag) o_i is reachable by ISAC's left hand
OnConveyor(g_i)	Grocery g_i is on the conveyor belt
ConveyorTurnedOn	*true* if the conveyor belt is on, else *false*
BagsEmptyRight	*true* if at least one bag is empty on ISAC's right side, else *false*
BagsEmptyLeft	*true* if at least one bag is empty on ISAC's left side, else *false*

The final components of the current state representation are the *Evaluation Signals* that maintain information about the internal appraisals and external rewards. The external rewards are provided by the critic and are based on the actions taken in previous states. The internal appraisals are those described thoroughly in Chapter V. Each evaluation signal is listed in Table 16, along with a brief description.

Table 16. Evaluation Signals

Signal	Significance
$\boldsymbol{\omega}$	Learned weights for each attribute (relevance)
$\boldsymbol{u}_i$	Utility appraisals
$\boldsymbol{\chi}_i$	Confidence in each utility appraisal
t_i^a	Allowable deliberation time
t_i^e	Expected deliberation time
q_i^e	Expected solution quality
i_i	Interrupt signal
ϑ	Feedback values (matrix)
$\boldsymbol{\phi}_i$	Fit appraisals

Behavioral Repertoire

For this experiment ISAC requires a complete set of behaviors designed for bagging groceries. These behaviors are given to ISAC as initial knowledge. With each behavior, an associated set of preconditions determines whether that behavior is allowable in the current state. In this context, "allowable" behaviors are those can be considered and attempted by ISAC, regardless of outcome. For example, the behavior *BagGrocery(bag_i, $grocery_j$)* is allowable if $grocery_i$ is on the conveyor belt and is *Reachable( )*. An object is "reachable" if that object is within ISAC's workspace. There are also postconditions associated with each behavior; however, postconditions are only used for planning, internal rehearsal, and simulation, and are not used to modify the representation of the current external state. The behaviors, preconditions, and postconditions are listed in Table 17.

Control and execution of each behavior is performed locally by specific Activator Agents. In particular, the behaviors *BagGroceryLeft(b_i,g_j)* and *BagGroceryRight(b_i,g_j)* are hierarchically composed of the low-level control behaviors *ReachToObject(g_j)*, *GraspObject(g_j)*, *ReachToBag(b_i)*, *ReleaseObject(g_j)*. However in this dissertation, these behaviors have been abstracted out, to speed decision making. If this were not the case, ISAC would need to learn the necessary behavioral combinations required for bagging groceries, before learning the appraisals that are the focus of this research.

Table 17. Behavior List with Pre- and Postconditions

Behavior	Preconditions/Postconditions
TrackGrocery(g_i)	Pre: *OnConveyor(g_i) = true*
RequestNewBagLeft( )	Pre: *EmptyBagsLeft = false* Post: *EmptyBagsLeft = true* *ReachableLeft(b_{new}) = true*
RequestNewBagRight( )	Pre: *EmptyBagsRight = false* Post: *EmptyBagsRight = true* *ReachableRight(b_{new}) = true*
BagGroceryRight(b_i, g_j)	Pre: *OnConveyor(g_j) = true* *ReachableRight(g_j) = true* *ReachableRight(b_i) = true* Post: *OnConveyor(g_j) = false* *ReachableRight(g_j) = false* *InBag(b_i, g_j) = true*
BagGroceryLeft(b_i, g_j)	Pre: *OnConveyor(g_j) = true* *ReachableLeft(g_j) = true* *ReachableLeft(b_i) = true* Post: *OnConveyor(g_j) = false* *ReachableLeft(g_j) = false* *InBag(b_i, g_j) = true*
Wait(g_i)	Pre: *ConveyorTurnedOn = true* *OnConveyor(g_i) = true* Post: *ReachableLeft(g_i) = true* OR *ReachableRight(g_i) = true*
StopConveyor( )	Pre: *ConveyorTurnedOn = true* Post: *ConveyorTurnedOn = false*
StartConveyor( )	Pre: *ConveyorTurnedOn = false* Post: *ConveyorTurnedOn = true*

Overview and Description of Simulation Experiments

The simulation environment was used to perform comprehensive evaluation and testing of the implemented control system. The simulation is designed to mirror the actual grocery-bagging layout, minus the noise and non-determinism commonly associated with real-world robotic experiments, typically sensor and actuator error. As previously discussed, the simulation enables recorded states and episodes to be re-used, which allows experiment repeatability – a feature not found in most physical robotic experiments. The results presented in the following sections have been obtained using the simulation environment.

Experiment Design

Several simulation experiments have been designed to evaluate and test the implemented system. These experiments evaluate how well the system learns the appropriate domain knowledge for the grocery-bagging task, how well the system learns to appraise urgency and adjust its deliberation, and whether levels of fit can be determined for each component. In these experiments, the grocery list in Table 14 was divided into two sets *GrocerySet-A* and *GrocerySet-B*, as shown in Tables 18 and 19. The system was only allowed to train using experience generated from *GrocerySet-A*, but the system was evaluated and tested using situations generated from both grocery sets.

The first two experiments were designed to evaluate how well the system learns the appropriate domain knowledge (i.e., relevance and utility appraisals) to perform the grocery-bagging task. During these experiments the system cognitively processed experience in order to extract knowledge related to forming goal-relevant grocery clusters as well as utility evaluations. At various points during the training process system performance was evaluated, however, slightly different evaluations were performed based on the manner in which experience had been acquired.

In the first experiment, experiences (i.e., episodes) were randomly selected from a database that had been generated previously. To test performance, a set of 30 sample bags, 12 sample states, and 15 full episodes were then generated using groceries from both *GrocerySet-A* and *GrocerySet-B*. The 30 test bags were generated randomly and were presented to the system one at a time. The 12 test states were composed of finalized sets of grocery bags. During evaluation and testing the system was required to appraise each bag or state using its current knowledge. These appraisals were then compared to the known values returned by the external critic, in order to measure the difference between the learned and correct values.

Table 18. *GrocerySet-A*

name	color	weight (oz)	size (in^3)	firmness	temp.(F^o)	price ($)	type	healthy
granola	RED	10	48	hard	75	2.29	T_1	YES
tissue	WHITE	8	252	soft	75	1.45	T_7	NA
soda	RED	64	294	hard	75	1.50	T_6	NO
ice_cream	BLUE	22.56	70	hard	32	2.99	T_4	YES
frozen_pizza	RED	10.5	81	hard	32	1.25	T_1	NO
chicken	PINK	24	108	hard	32	3.05	T_5	YES
milk	WHITE	68.8	160	hard	45	1.67	T_4	YES
cereal	BLUE	17	227.5	hard	75	3.15	T_1	NO
oranges	ORANGE	64	504	hard	75	4.29	T_3	YES
potatoes	BROWN	80	504	hard	75	3.99	T_2	YES
strawberries	MAGENTA	16	140	soft	55	2.30	T_3	YES
bread	BROWN	16	325	soft	75	0.89	T_1	YES
rotisserie	BROWN	32	160	hard	150	7.99	T_5	YES
eggs	YELLOW	24	144	soft	45	1.59	T_5	YES
hot_soup	WHITE	12	48	hard	175	3.50	T_2	YES
chips	YELLOW	12	378	soft	75	2.68	T_6	NO
yogurt	BLUE	24	75	hard	45	1.59	T_4	YES
frozen_fruit	BLUE	48	180	hard	32	7.00	T_3	YES
ziploc_bags	BLUE	10	99	hard	75	2.99	T_7	NA
spaghetti	BROWN	12	22	hard	75	1.19	T_1	YES
green_beans	GREEN	14.5	25	hard	75	0.50	T_2	YES
cucumbers	GREEN	28	72	hard	50	2.40	T_2	YES
teriyaki_bowl	BLACK	17	105.88	hard	150	5.29	T_5	NO
fruit_juice	MAGENTA	64	198	hard	75	2.18	T_3	NO
tuna	BLUE	20	85.75	hard	75	7.39	T_5	YES

Table 19. *GrocerySet-B*

name	color	weight (oz)	size (in^3)	firmness	temp.(F^o)	price ($)	type	healthy
cookie_dough	YELLOW	16	31.5	soft	75	2.50	T_6	NO
waffles	YELLOW	12	121.5	hard	32	2.79	T_1	YES
frozen_vegetables	WHITE	16	96	hard	32	1.19	T_2	YES
cheese	PURPLE	8	45	soft	45	2.49	T_4	YES
ketchup	RED	32	84	hard	75	1.58	T_2	YES
popcorn	BLUE	20	108	hard	75	2.79	T_1	NO
hot_chocolate	BLUE	2.25	35	hard	75	1.00	T_6	YES
trash_bags	GREEN	48	256	hard	75	10.99	T_7	NA
marshmallows	WHITE	16	110	soft	75	2.00	T_6	NO
coffee	RED	23	180	hard	75	8.41	T_7	YES
cheezits	RED	11.5	162	hard	75	2.00	T_1	NO
ritz_chips	GREEN	80	189	hard	75	3.00	T_1	NO
vegetable_soup	BROWN	18.8	31.25	hard	75	2.50	T_2	YES
choc_cookies	BROWN	16	135	hard	75	2.99	T_6	NO
peanut_butter	BROWN	28	45	hard	75	2.50	T_6	YES
macNcheese	BLUE	7	39.38	hard	75	0.75	T_1	NO
vegetable_oil	YELLOW	48	132	hard	75	2.69	T_6	NO
rice	WHITE	32	90	hard	75	1.99	T_1	YES
bagels	BROWN	20	72	soft	75	1.99	T_1	YES
ribs	BLACK	20	336	hard	150	10.99	T_5	NO
tomatoes	RED	20	90	soft	75	2.75	T_3	YES
lettuce	GREEN	16	156	hard	60	2.49	T_2	YES
bananas	YELLOW	32	216	soft	75	4.00	T_3	YES
cake	BROWN	44	549	soft	75	6.99	T_6	NO
ground_beef	RED	36.8	120	hard	45	7.57	T_5	YES

Unlike the test bags and states, the 15 test episodes were generated by initializing the decision-space using random grocery selection, and repeatedly allowing the system to form plans and perform actions. The plan-act process continued until all of the groceries in the initial state had been bagged. In these 15 test episodes there was a combined total of 119 groceries, or approximately 8 groceries per episode. The rational for this number was based on empirical tests that indicated this number was large enough to stress the system, but small enough to enable timely evaluation. For example, for a state with N groceries and only *one* bag there are $N!$ solutions to the grocery-bagging problem. Thus, 8 groceries yields approximately 40,000 solutions. If intermediate states, such as those resulting from a *Wait( )* action, are included the number of states searched for a solution could increase drastically. Since the system had not yet learned to appraise urgency and modify its deliberation, the necessarily deeper search through such a large state space would require prohibitively expensive deliberation costs. In the interest of time and solvability, each test state was limited to only a few groceries. The actual number of groceries per episode was selected from a uniform distribution over the range [4, 12].

In the second experiment, the system was trained and evaluated concurrently using its own acquired experience. This involved sequentially generating random grocery-bagging episodes, allowing the system to bag groceries using its current knowledge, providing evaluation feedback, and re-training once every M episodes. Individual episodes were generated by selecting a small set of groceries (at random), and then placing those groceries on the conveyor belt. As those groceries were bagged, additional groceries were selected (also at random) and placed on the conveyor belt. Therefore, at each step of the planning process, the system was unaware of how many and what type of groceries may appear at the next time step. In between training epochs the system was exposed to $M = 10$ episodes generated using only *GrocerySet-A* and 10 episodes using only *GrocerySet-B*. After each set of 20 episodes, the system was re-trained by incorporating the first 10 episodes (*GrocerySet-A*) into long-term memory. Evaluation was then continued using the next 20 episodes (i.e., the episodes which the system has not yet been exposed).

Once the system has demonstrated the ability to learn the required domain knowledge, additional experiments were designed to evaluate how well the system learns to appraise urgency and to adjust deliberation accordingly. In addition, this experiment is also used to evaluate the system's ability to appraise fit. During this experiment only the non-fixed, high action cost

condition is used, because it is necessary that the system employ all of its appraisals and because the high action cost should force the system to make more errors, which creates a more conducive environment for appraising fit.

During each experiment, the size of the relational map for constraints (1) and (2) was set to 40 x 40 and the size of the relational map for constraint (3) was set to 25 x 25. The difference in size of these two maps is based on the fact that much more training experience was generated for the first map. During the experiments in which the system was required to deliberate without the use of urgency appraisals to adjust the search parameters, a search depth of 3.0 and a breadth of 1.0 were used. These values were empirically determined during initial tests to adequately enable the system to evaluate a range of possible responses, while simultaneously allowing experiments to be performed timely and efficiently.

Performance Measures

Throughout each of the simulated experiments, the same critic was used to evaluate performance. This critic used preset rules to determine:

1. Number of constraint (1) violations
2. Number of constraint (2) violations
3. Whether constraint (3) has been violated
4. Deliberation time per decision epoch

Constraints (1) and (2) were evaluated on a *per bag basis*, while constraint (3) was evaluated on a *per episode basis*. Pseudo-code for the two preset rules that evaluate constraints (1) and (2) is shown in Figure 31. These rules were implemented as *if-then* checks that were based solely on the contents of each individual bag. These rules returned the value –1 if the constraint was violated, otherwise the value +1 was returned. The rule for constraint (3), however, was based on the number of groceries and the number of bags in a single episode, and returned one of a range of values. The equation for constraint (3) is given in Equation (28), where *Num_groceries* is the total number of groceries in the current episode, *Num_bags* is the total number of bags used in that episode, and κ_1 and κ_2 are constants that define the range of values that can be returned. Throughout these experiments both κ_1 and κ_2 were set to 1.5. The rational for this selection is

that empirical tests showed that such a setting kept the constraint (3) evaluations near the –1/+1 range. Finally, the critic was also used to record the deliberation time during task performance.

```
Crusher = Lightweight = Hot = Cold = false
Large_count = Medium_count = Small_count = Total_weight = 0.0

For all groceries_i in bag_j
        If(firmness_i = hard and weight_i > 20.0)
                Crusher=true
        If(firmness_i = soft and weight_i < 30.0)
                Lightweight=true
        If(temp_i > 100.0)
                Hot=true
        If(temp_i < 50.0)
                Cold=true

        If(size_i > 400.0)
                Large_count = Large_count+1
        Else If(size_i > 200.0)
                Medium_count = Medium_count+1
        Else
                Small_count = Small_count+1

        Total_weight = Total_weight + weight_i

Total_count = 2.5*Large_count + 1.5*Medium_count + 0.9 * Small_count

If(Crusher & Lightweight)
        Constraint(1)_j = -1
Else If(Hot & Cold)
        Constraint(1)_j = -1
Else
        Constraint(1)_j = +1

If(Total_weight > 165)
        Constraint(2)_j = -1
Else If(Total_count > 5)
        Constraint(2)_j = -1
Else
        Constraint(2)_j = +1
```

Figure 31. Preset Rules for Constraints (1) and (2)

$$\text{Constraint}(3) = \frac{\kappa_1 * (Num\_groceries - Num\_bags)}{Num\_groceries} - \kappa_2 \qquad (28)$$

During evaluation of system performance, six different values were recorded that relate to the system's application of its learned domain knowledge. These six values are:

1. E_1 – Difference between the external evaluation of constraint (1) and the internal appraisal for constraint (1), summed over all bags and normalized by the maximum possible error.

$$E_1 = \frac{\sum_{k=0}^{NumBags} (ExtEval_{1k} - \chi_{1k} * u_{1k})}{2 * NumBags} \quad (29)$$

2. E_2 – Difference between the external evaluation of constraint (2) and the internal appraisal for constraint (2), summed over all bags and normalized by the maximum possible error.

$$E_2 = \frac{\sum_{k=0}^{NumBags} (ExtEval_{2k} - \chi_{2k} * u_{2k})}{2 * NumBags} \quad (30)$$

3. E_3 – Difference between the external evaluation of constraint (3) and the internal appraisal for constraint (3), summed over all 15 test states and normalized by the maximum possible error.

$$E_3 = \frac{\sum_{k=0}^{NumStates} (ExtEval_{3k} - \chi_{3k} * u_{3k})}{2 * (\kappa_1 - \kappa_2)} \quad (31)$$

4. E_4 – Number of errors on constraint (1), summed over all test states per trial.
5. E_5 – Number of errors on constraint (2), summed over all test states per trial.
6. E_6 – Number of bags used, summed over all test states per trial.

In addition to these recorded values, the learned attribute weights were also recorded in order to analyze how the learned concepts changed over time. The final grocery classifications were recorded to evaluate whether or not useful grocery classifications had been formed, and the trained relational maps were recorded to analyze the types of relations that had been learned. Finally, deliberation time, search depth and breadth, and the number of states expanded was recorded for use in the experiments that evaluated urgency.

Experiment 1: Domain Knowledge Using Random Experience

Experiment Description

For this experiment 100 episodes were generated at random. The system was repeatedly trained using increasing amounts of this experience, and performance was evaluated with respect to the appraisals *relevance* and *utility*. Each training episode was generated selecting up to 20 groceries at random, and then allowing ISAC to bag those groceries without any prior knowledge of the correct appraisal values (i.e., random selection). Due to the fact that behavior selection was essentially random, a large search depth was not used and, therefore, these episodes could be generated quickly. All of the groceries used to generate this corpus of experience were selected from *GrocerySet-A*. The critic provided feedback at the end of each episode for each of the final bags (constraints 1 and 2), as well as the entire episode (constraint 3). After each training step, performance was evaluated using the combination of 30 test bags, 12 test states, and 15 test episodes. Four different conditions were used for this experiment and five trials were performed for each condition. The results presented in the following subsection are the averages over these five trials. The four experimental conditions are listed as follows:

1. Fixed weight set and high action cost
2. Non-fixed weight set and high action cost
3. Fixed weight set and low action cost
4. Non-fixed weight set and low action cost

The rational for having high and low action cost conditions was that high action costs should force the system to explore more single bag options, or "place more groceries in a bag". Increasing the number of groceries per bag should increase the rate of learning by providing more diverse experiences, but may also interfere with final task performance by outweighing uncertain appraisals. The high action cost value was set to –0.85 and, therefore, in order to prefer using a new bag for a new grocery, rather than placing the new grocery in an existing bag, the system must be highly confident that the latter action will violate one of the constraints. From Chapter V, Equation (26), shows that possible states are evaluated and preferentially ordered using the vector $\boldsymbol{eval}_j = \boldsymbol{\chi}_j * \boldsymbol{u}_j$, where confidence vector $\boldsymbol{\chi}_j$ is composed of elements bounded by the interval [0, 1] and the utility vector $\boldsymbol{u}_j$ is composed of elements bounded by the interval [-1, 1]. Thus, only highly confident, correct evaluations will "overrule" the preference for not

getting a new bag (additional action). Low action costs, however, enable the system to make decisions based solely on its learned appraisals and not on innate cost aversion. The low action cost value was set to –0.05, so that the action of requesting new bags would only be preferred when considering situations that could *possibly* have negative value. In the low action cost condition, confidence values were only used to order the desirability of potential responses.

The rational for implementing the fixed weight conditions was to isolate the process of learning the relational maps and ease the demand placed on the system's learning capabilities. In the non-fixed conditions, the system was required to simultaneously learn the weight values useful for grouping groceries into useful classes, while then using those classes to generate the relational maps. As the grocery classifications changed, so did the learned relational maps. While the non-fixed condition better reflects real world situations in which relevance and utility information must be learned concurrently, it makes it difficult to assess aspects of the learning process. Because in many robotic applications some domain knowledge is present, the fixed weight conditions were used to provide a comparison with the more noisy non-fixed conditions. The fixed weight values are given in Table 20, and the resulting classification scheme for *GrocerySet-A* is given in Table 21. The fixed weights were chosen in such a way that the resulting classification scheme would separate groceries into groups that do not violate constraint (1). In other words, if the system were to perform the grocery-bagging task using only the rule "do not mix types of groceries", constraint (1) would not be violated.

Table 20. Weight Values for Fixed Condition

name	color	weight	size	firmness	temp	price	type	healthy
0.0	0.0	0.75	0.75	0.75	1.0	0.0	0.0	0.0

Table 21. Final Clusters Using Pruned Tree and Fixed Weights

Class	Grocery
C_0	ice_cream, frozen_pizza, yogurt, cucumbers, chicken
C_1	granola, ziploc_bags, tuna spaghetti, green_beans
C_2	rotisserie
C_3	hot_soup, teriyaki_bowl
C_4	milk, frozen_fruit
C_5	oranges, potatoes
C_6	soda, cereal, fruit_juice
C_7	tissue, bread, chips
C_8	eggs, strawberries

Furthermore, the purpose of the relevance appraisals is to extract important features from the environment in a manner that enables dimensional reduction and goal-based abstraction of the current situation. For reference throughout these experiments the grocery clusters that result from assigning uniform weights to each attribute are listed in Table 22. This table indicates that without the ability (either innate or learned) to filter state information and focus on the most goal-relevant features, virtually no reduction is performed.

Table 22. Final Clusters Using Pruned Tree and Uniform Weights

Class	Grocery
C_0	granola
C_1	chips
C_2	rotisserie
C_3	tissue
C_4	oranges, potatoes
C_5	frozen_pizza
C_6	teriyaki_bowl
C_7	eggs
C_8	frozen_fruit
C_9	soda
C_{10}	bread
C_{11}	hot_soup
C_{12}	ziploc_bags
C_{13}	strawberries
C_{14}	fruit_juice
C_{15}	milk
C_{16}	tuna
C_{17}	cereal
C_{18}	green_beans
C_{19}	chicken
C_{20}	spaghetti
C_{21}	ice_cream, yogurt
C_{22}	cucumbers

Results and Discussion

Figures 32-34 present the results for each performance metric for the fixed, high cost condition averaged over all five trials. In each figure, the first column indicates system performance based on random selection. Figure 32 shows that with as little as 30 episodes of training, the system's ability to make correct utility appraisals on each constraint improves approximately 15%. However, after this initial improvement the performance levels off. This will be discussed at the end of this Experiment 1 section, but for now it is only important to note the both the initial improvement and subsequent leveling off.

Figure 33 presents the number of errors on constraints (1) and (2) during the 15 test episodes as the system is trained with increasing amounts of random experience. This figure shows that the total number of errors decreases throughout training. Much of this improvement can be explained by improvement on constraint (1), while the initial improvement on constraint (2) is followed by a trend of much slower improvement. This dramatic decrease in constraint (2) errors is explained by noting, from Figure 34, that the number of bags also increases dramatically at this stage of learning. Therefore, the system quickly identifies that the solution "throw everything in one/two bags" is not correct. As the number of bags increases, the ratio of *groceries : bags* decreases, which intuitively decreases the likelihood of constraint (2) errors.

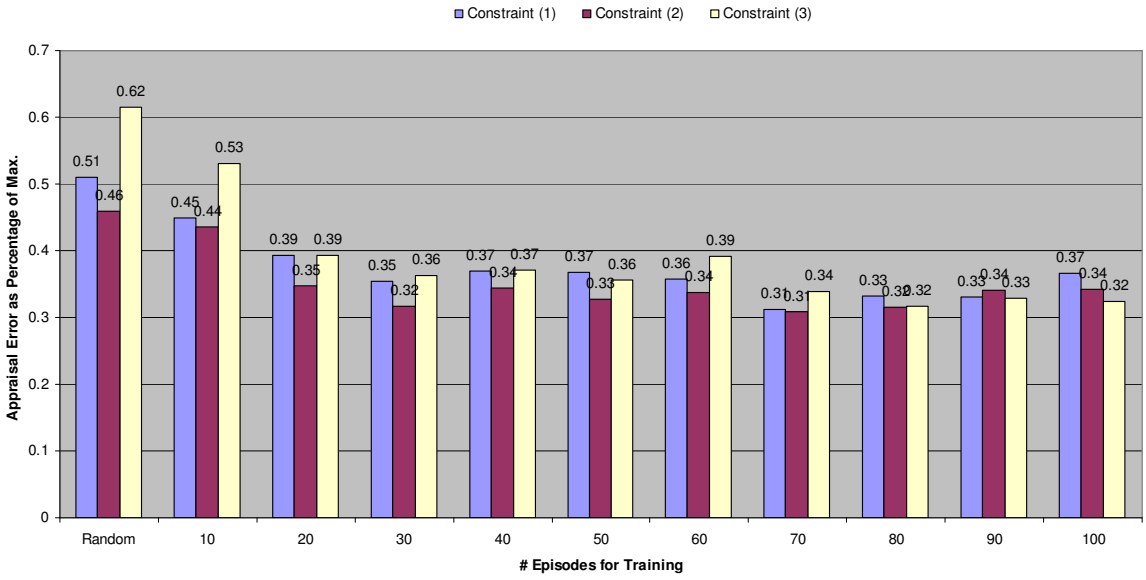


Figure 32. Appraisal Errors with Increased Training for the Fixed Weight, High Cost Condition and Random Experience

These results indicate that the system first learns to reduce the number of groceries per bag and that as a result the number of errors per trial decreases dramatically. Once this knowledge has been learned, the system then exhibits slower improvement on constraints (1) and (2), however, during this improvement the ratio of *groceries : bags* remains roughly constant. In other words, the additional improvement does not require further isolating groceries (e.g., one grocery per bag).

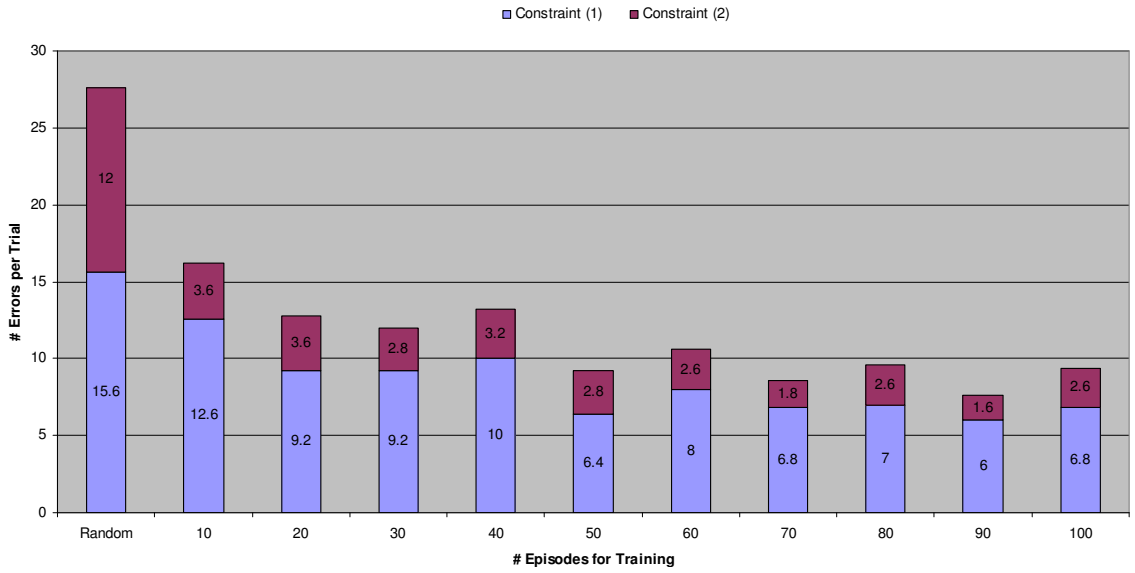


Figure 33. Total Errors Per Trial on Constraints (1) and (2) for the Fixed Weight, High Cost Condition and Random Experience

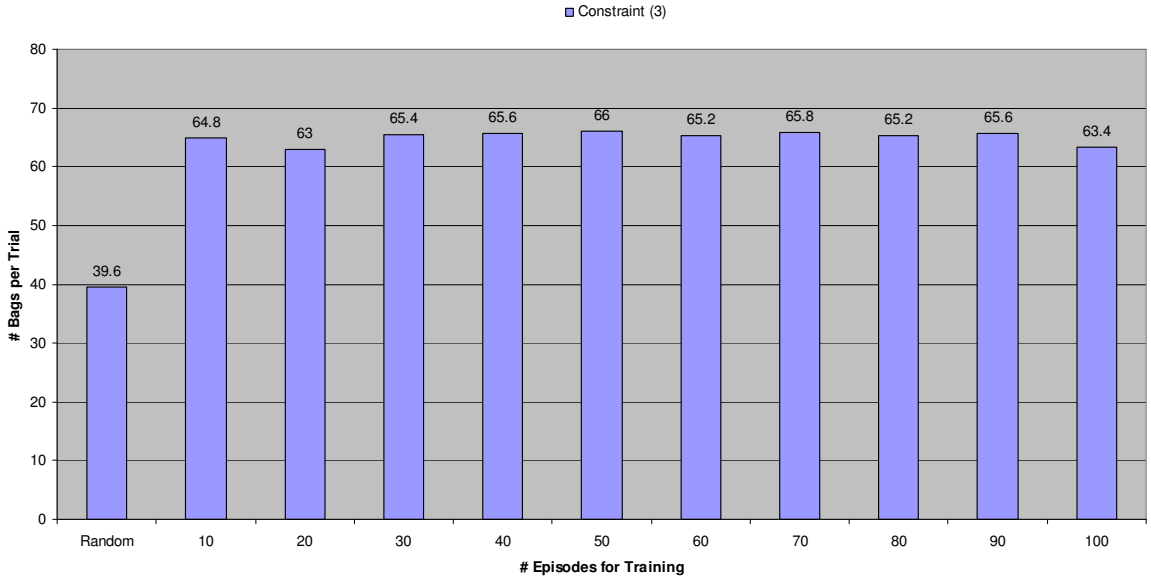


Figure 34. Number of Bags Used Per Trial for the Fixed Weight, High Cost Condition and Random Experience

As mentioned previously, the same 15 test states were used for each trial and there was a combined 119 groceries over all test states. Therefore, while the number of groceries per bag starts around 3.0, the final ratio hovers around 1.8. While this may seem very low, there are multiple points to consider when analyzing this performance.

1. For 15 tests states and a total of 119 groceries there was an average of ~8 groceries per trial. As described previously, this number was kept low to enable timely deliberation and experimentation.
2. The distribution from which the groceries were selected (i.e., *GrocerySet-A*) was designed to provide the system with ample opportunities to learn, and thus the number of potentially dangerous combinations was increased over that which may occur in a typical grocery store.
3. The simulation environment was designed to mimic the real ISAC robot, and therefore groceries appearing on one side of the conveyor belt (e.g., ISAC's left or right side) must be placed in bags on that side. In other words, while a human grocery-bagger may choose to reach across their body to grab the "eggs" on the left in order to place them in a bag on the right, ISAC is incapable of such motions, and can only consider placing the "eggs" in a bag on the left.

The result is that during this experiment (and those to follow), ISAC was presented with grocery sets in which there is a larger probability of error than in normal grocery-bagging situations, the type of deliberation demanded during this experiment required that a low number of groceries be presented to ISAC, and ISAC's physical resources limited the grocery combinations that were possible. Thus a ratio of two groceries per bag is not as low as it seems, and the fact that this ratio remains constantly above 1.0, while improvement is made on the other constraints indicates that the system is learning to bag groceries *within its grocery-bagging environment*.

However, while the system shows the ability to learn the improvement is gradual and the question may be asked whether the system's performance is merely reflecting the fixed grocery classification scheme with which it has been provided. To investigate this point, it is necessary to analyze the relational maps that were used to appraise situations. If, in fact, the system only learned to apply the grocery classification scheme when bagging groceries (i.e., only mix groceries of the same type), then the relational maps would need to contain a high percentage of uniform vectors that evaluate close to +1.0, (e.g., $C_0C_0C_0$ vs. $C_1C_3C_2$).

The final relational maps, after training on all 100 episodes, were analyzed for each trial. On average there were approximately 641 nodes in each relational map that contained a vector

with more than one element and also evaluated to ~+1.0. The averaged percentage of uniform vectors in these nodes, across all five experimental trials, was only 10.06%. Further analysis revealed that nearly 90% of these vectors had a maximum frequency of reoccurrence of any single element in that vector less than 0.75, 86% of the vectors had a maximum frequency less than 0.67, and 70% of the vectors had a maximum frequency less than 0.5. Therefore, roughly 2/3 of the positive (+1.0) vectors in the trained relational map contained enough diversity that no single element of that vector recurred with frequency greater than 50%. Examples of such vectors include C_0C_1, $C_1C_3C_2$, and $C_5C_2C_4C_5$. Further analysis revealed that, on average, there were 799 nodes in each relational map that contained a vector with more than one element and also evaluated to ~ –1.0. Of these vectors, approximately 89% had a maximum frequency of less than 0.5. Given the fact that the system was trained using randomly generated experience, it is expected that the generalized feature vectors would have a low probability of uniformity, and this analysis indicates that the there was a high percentage of diverse experience that would not lead the system towards any one response, or towards merely not mixing grocery types.

Figures 35-37 show the results for each performance metric for the non-fixed, high cost condition averaged over all five trials. It is important to note, at this point, that the results presented for the 30 test bags and 12 test states, while unique to each test, are not influenced by the action cost. Rather, these results are influenced only by the learned weights, grocery classifications, and trained relational map. These results are presented for the 30 test bags and 12 test states to enable better understanding of system performance during the episodic tests.

Even though the system in this condition was required to simultaneously learn both the weights necessary to form grocery classifications as well as the relational maps to evaluate bags and episodes, Figure 35 indicate that appraisal accuracy improved for both bags and episodes at a rate similar to that for the fixed condition. Figure 36 shows that the system is also able to improve performance with respect to constraints (1) and (2), however, this improvement was more gradual. This is explained, in part, by noting the more gradual increase in the number of bags used, as shown in Figure 37.

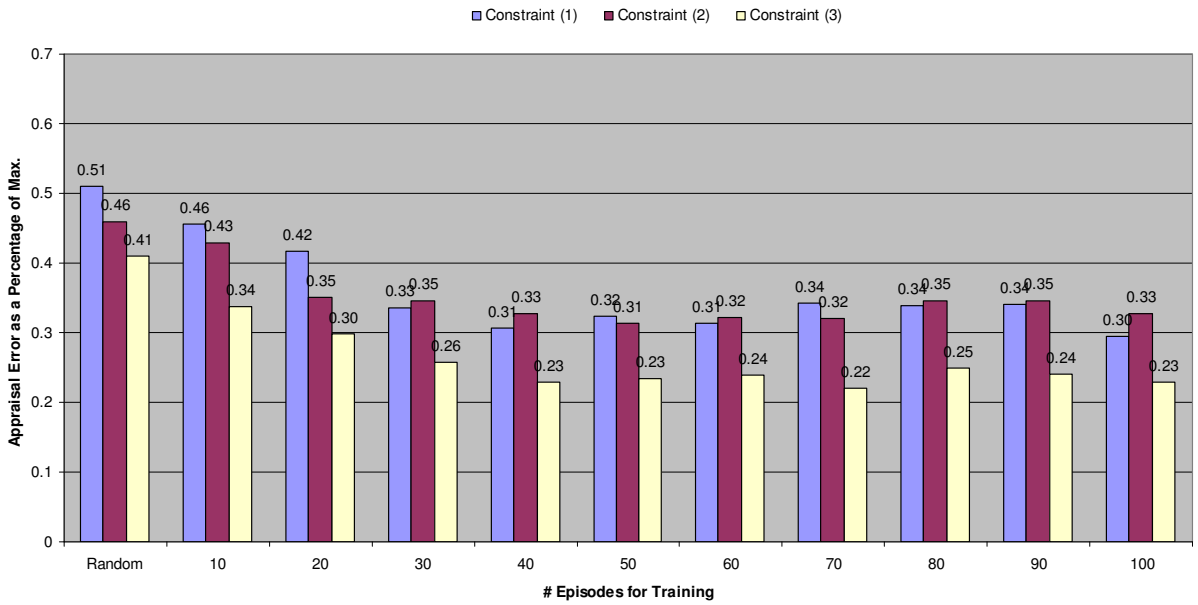


Figure 35. Appraisal Errors with Increased Training for the Non-Fixed Weight, High Cost Condition and Random Experience

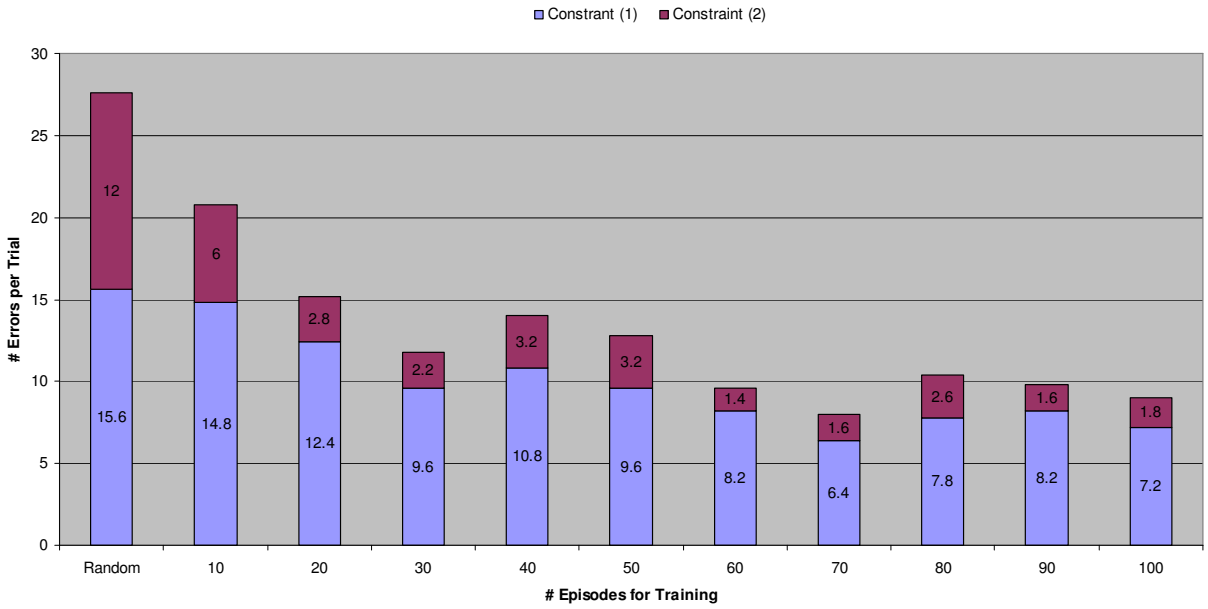


Figure 36. Total Errors Per Trial on Constraints (1) and (2) for the Non-Fixed Weight, High Cost Condition and Random Experience

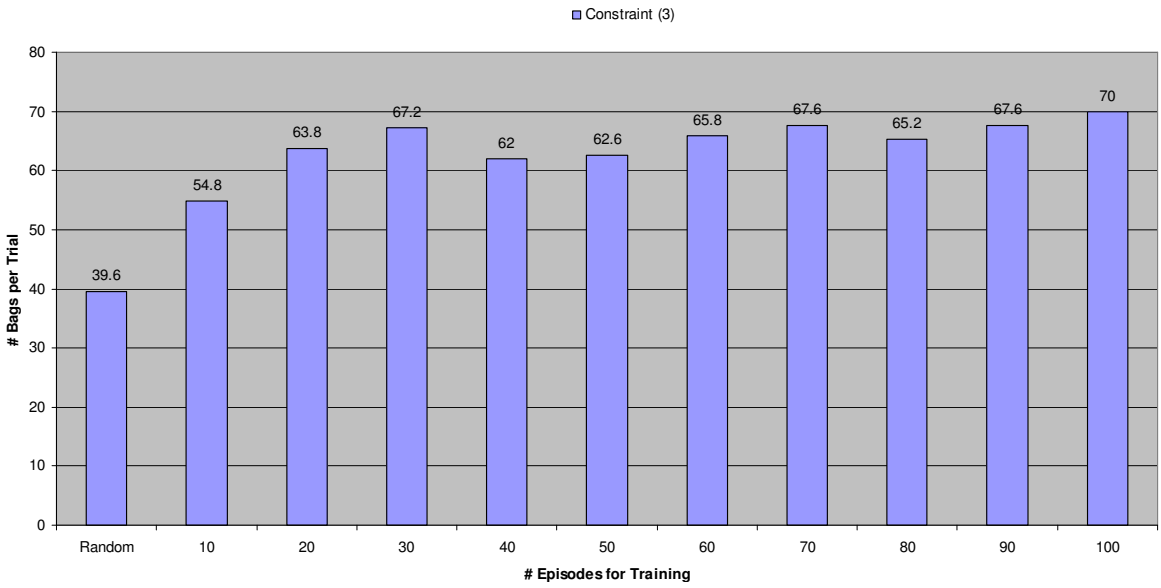


Figure 37. Number of Bags Used Per Trial for the Non-Fixed Weight, High Cost Condition and Random Experience

To better understand what is happening in this non-fixed weight condition, it is necessary to analyze how the weights change as function of the number of training episodes. However, as discussed in Chapter V, the learned weights may differ drastically even while the learned concepts are similar. Thus, the weights presented here are *not* averaged results, but instead those obtained during the first round of experiments. Figure 38 shows the value for each weight as training increases. As shown in the figure, while the weight associated with the attribute *name* immediately drops to 0.0, and the weight associated with the attribute *firmness* constantly stays at 1.0, the remaining weights gradually, and consistently, change. The weights for attributes *weight*, *price*, and *healthy* (3, 7, and 9) show some leveling off, but weights for the attributes *size*, *temperature*, and *type* (4, 6, and 8) do not reach a steady state within the training window. To understand which attributes are consistently considered "more relevant", Table 23 lists the weight value for each attribute averaged over all training epochs.

Table 23. Learned Attribute Weights

Attribute	Weight
firmness	1.0000
temperature	0.7487
weight	0.7309
healthy	0.5992
price	0.5851
size	0.5039
type	0.3726
color	0.2439
name	0.0909

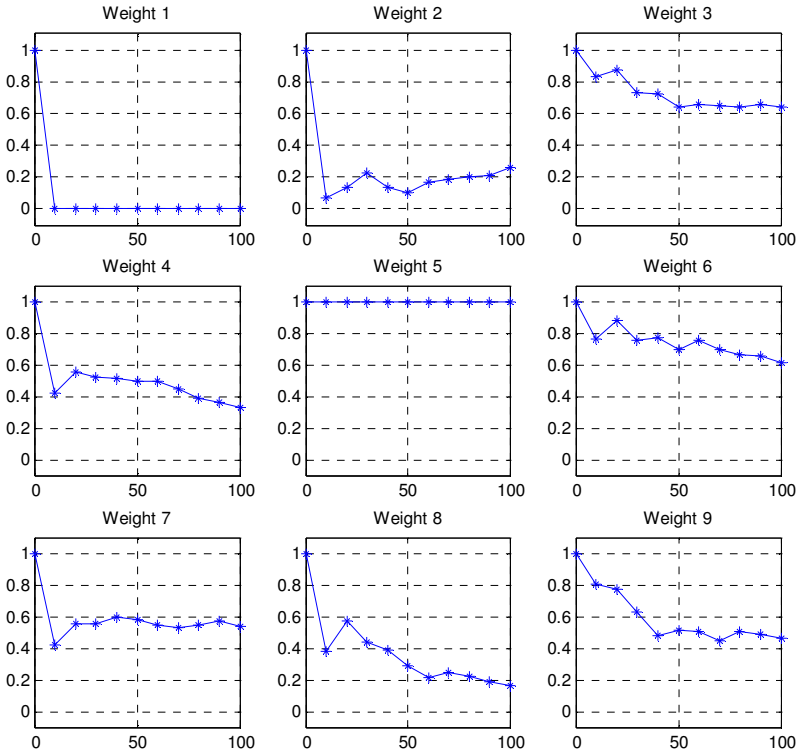


Figure 38. Learned Weights with Increased Training for the Non-Fixed, High Cost Condition and Random Experience

As the weights are continually modified the developed grocery classifications may also change, which slows down the learning process and can potentially insert a small amount of error into task performance. The reason for this error can be understood by reviewing Table 24, which provides the learned grocery clusters after training on all 100 episodes. While, for the most part, these clusters are similar to those presented in Table 21 for fixed weights, class C_1 *does* mix hot

and cold items, which would cause a violation of constraint (1) unless the system specifically learned not to place more than one C_1 grocery into a bag.

Further analysis of the learned clusters investigates the consistency of these clusters across all five trials for the four training iterations consisting of {70, 80, 90, 100} episodes. These training epochs were chosen because at this point in the training the amount of experience should be enough to prevent large fluctuations in the individual weights. During these iteration, the system only identifies the cluster scheme shown in Table 24, 65% (13/20) of the time. Of the remaining iterations, five involve breaking class C_1 into additional clusters and two involve creating multiple, seemingly spurious additional clusters. Three of the five times that C_1 is split, the result is a clustering in which constraint (1) would never be broken within a single class. However, the increased number of grocery clusters also makes it more difficult for the relational maps to generalize from its limited experience, which may explain why the system resorts to using more bags.

Table 24. Final Learned Partition Using 100 Episodes

Class	Grocery
C_0	ice_cream, chicken, yogurt, cucumbers
C_1	granola, ziploc_bags, tuna frozen_pizza, cereal, spaghetti, green_beans teriyaki_bowl, hot_soup
C_2	rotisserie
C_3	tuna
C_4	milk, frozen_fruit
C_5	oranges, potatoes
C_6	soda, fruit_juice
C_7	tissue, bread
C_8	chips
C_9	eggs
C_{10}	strawberries

Figures 39-41 show the averaged results for the fixed, low cost condition over all five trials for each performance metric. Unlike the high cost conditions, the results presented in Figures 39 and 41 are intriguing in that they more accurately demonstrate what the system is truly learning. With only 30 episodes of experience, the system makes as few mistakes as it has at any point in either of the previous two conditions. Simultaneously, though, the system also

uses more bags than it in any of the previous conditions. In other words, with little training experience the system immediately becomes pessimistic and avoids risk (i.e., multiple groceries in single bags). As the amount of training experience increases, Figure 41 demonstrates that there is a "slight" reduction in the number of bags used, but that overall this number remains constant. At the same time, the system continues to improve on constraints (1) and (2). After 100 episodes of training, the system settles on more bags but fewer constraint (1) and (2) violations.

Whereas in previous conditions the system never had a combined total of constraint (1) and (2) violations less than 7.6 per 15 test episodes (6.0 and 1.6, respectively), in this trial the system gets as low as 4.2 (3.6 and 0.6, respectively). Because action cost does *not* affect training or learning from random experience, it is concluded that the increased number of errors and decreased number of bags observed in the earlier conditions are, at least somewhat, a product of the high action cost, which may filter out the low confidence appraisals that would otherwise have been correct. In other words, the system "knew better, but did not trust itself".

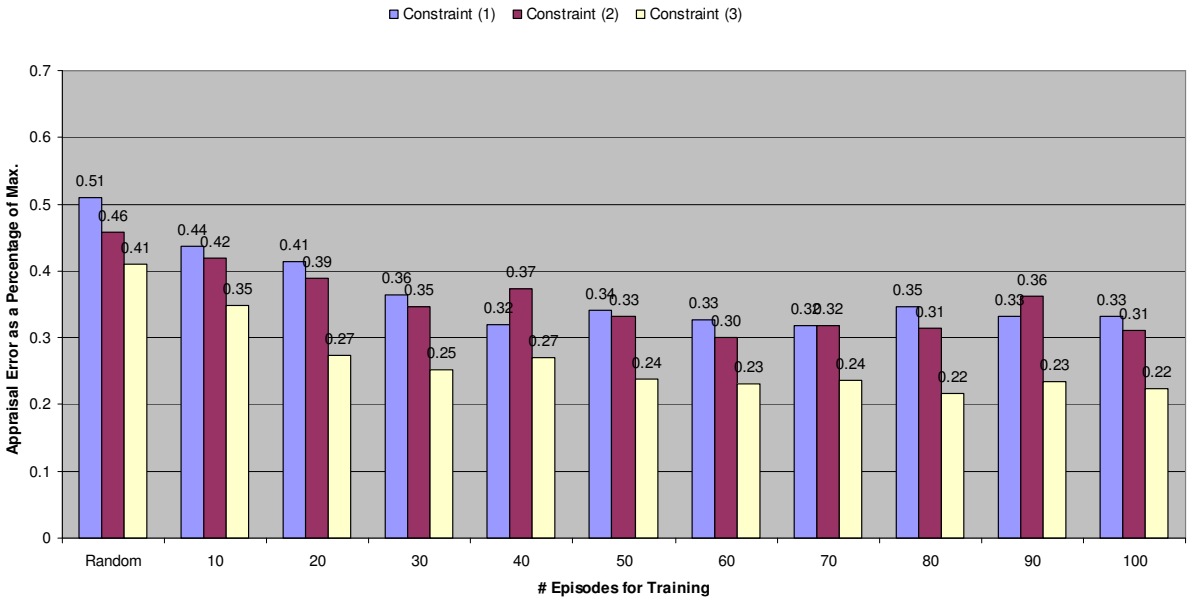


Figure 39. Appraisal Errors with Increased Training for the Fixed Weight, Low Cost Condition and Random Experience

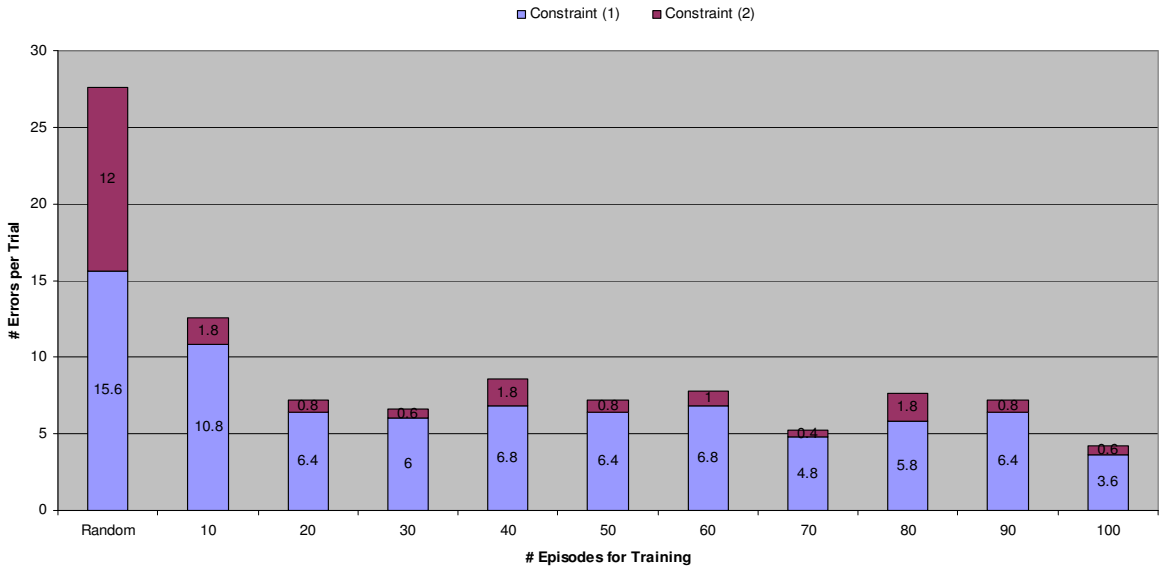


Figure 40. Total Errors Per Trial on Constraints (1) and (2) for the Fixed Weight, Low Cost Condition and Random Experience

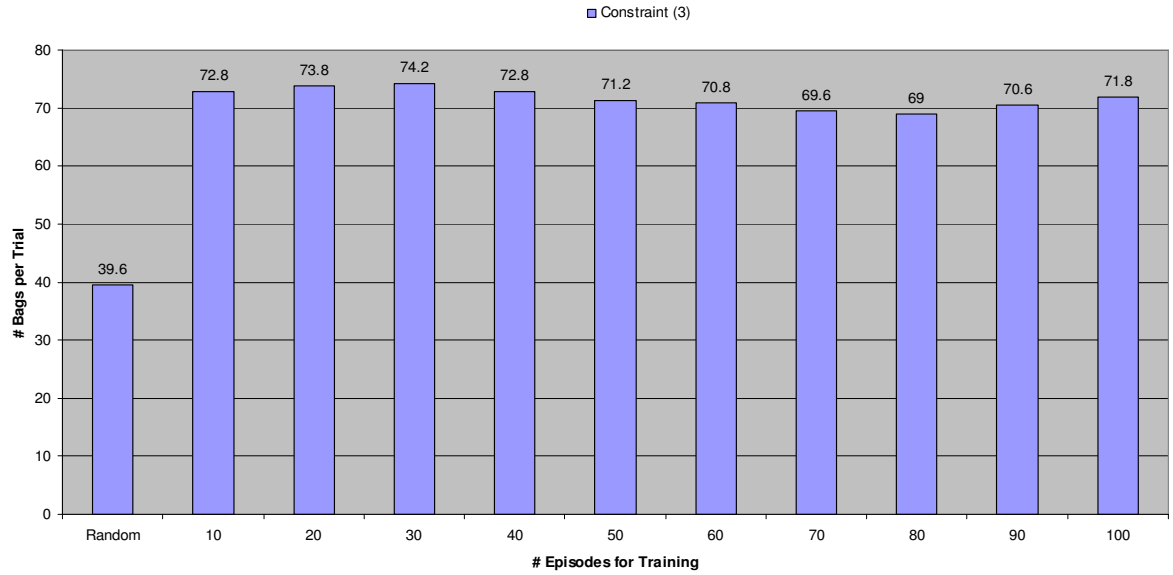


Figure 41. Number of Bags Used Per Trial for the Fixed Weight, Low Cost Condition and Random Experience

Figures 42-44 show the results for each performance metric for the non-fixed, low cost condition averaged over all five trials. Again, because the system is simultaneously charged with learning grocery classifications as well as the relational maps that provide utility appraisals, the results presented in Figures 43 and 44 are more erratic than those presented in Figures 40 and 41. In addition, as in the previous low cost condition, the system demonstrates immediate learning

on constraints (1) and (2), followed by a period of very slow learning. This trend is also present in Figure 44, which shows that the system initially uses a lot of bags to improve performance, but slowly begins to decrease the number of bags while maintaining performance on the constraints (1) and (2). While the steady decrease in the number of bags is not dramatic, it is noticeable; after initially jumping to an average of 77.6 bags per 15 test episodes, the system eventually gets this count as low as 69.4.

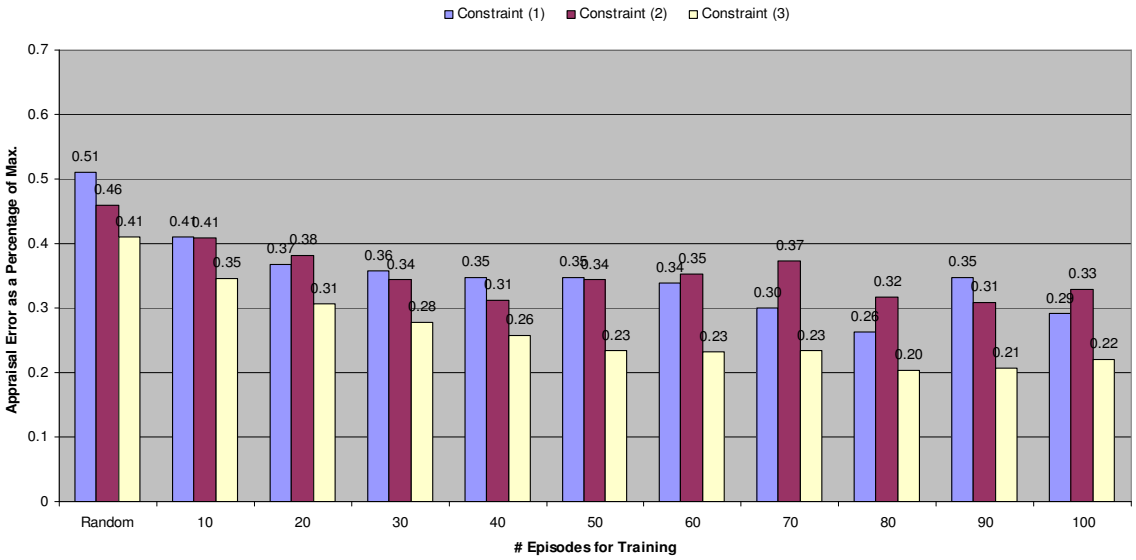


Figure 42. Appraisal Errors with Increased Training for the Non-Fixed Weight, Low Cost Condition and Random Experience

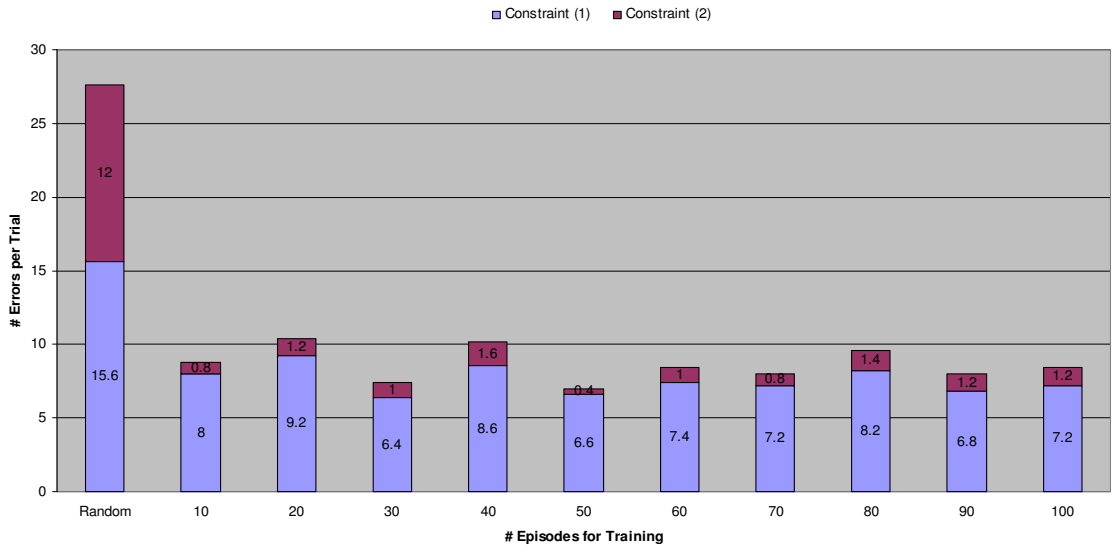


Figure 43. Total Errors Per Trial on Constraints (1) and (2) for the Non-Fixed Weight, Low Cost Condition and Random Experience

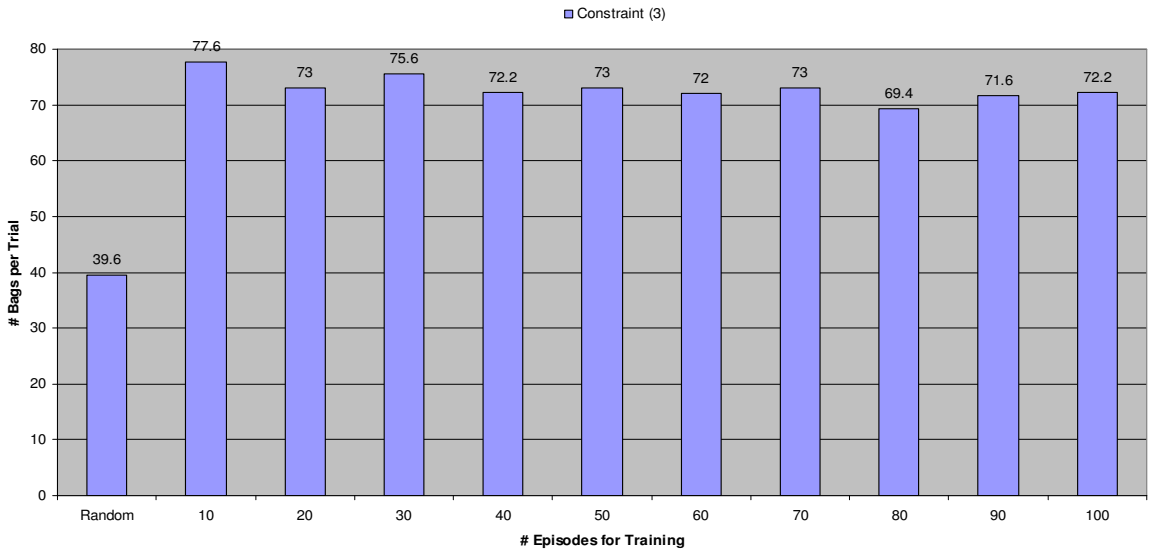


Figure 44. Number of Bags Used Per Trial for Non-Fixed Weight, Low Cost Condition and Random Experience

Figure 45 shows how the weights change during the first trial. As with Figure 38, these are not averaged values. Again, the weights associated with the *name* and *firmness* attributes immediately reach their final state, however, many of the several of the remaining weights still exhibit small fluctuations, even at 90 episodes of training. While this fluctuations may influence concept formation, after 50 episodes of training none of the weights fluctuate by an amount greater than 20% of their value at the 50 episode mark. Table 25 presents the learned attributes weights averaged over all five experimental trials, and Table 26 presents the final grocery clusters. The clusters presented in Table 26 are the same as the one presented in Table 24, and further analysis reveals that for all five trials and the four training iterations consisting of {70, 80, 90, 100} episodes, these clusters were identified 75% (15/20) of the time. Of the remaining five iterations, 60% (3/5) divided class C_1 into multiple categories while 40% (2/5) divided C_4.

Table 25. Learned Attribute Weights

Attribute	Weight
firmness	1.0000
temperature	0.7380
weight	0.7032
price	0.5801
healthy	0.4868
size	0.4160
type	0.3340
color	0.2658
name	0.0909

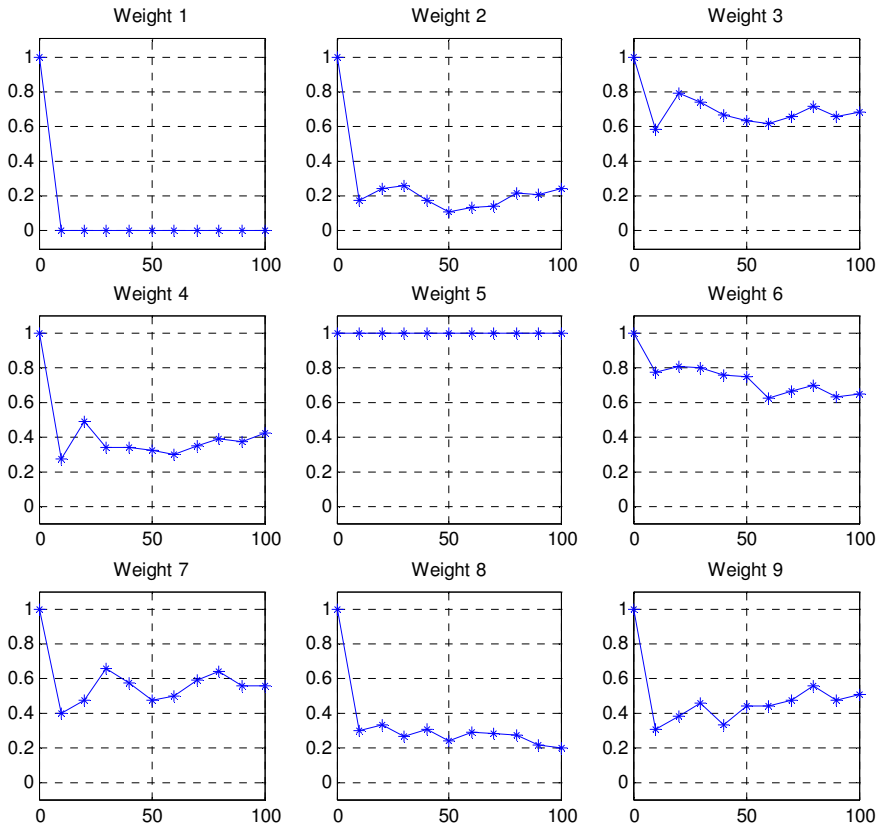


Figure 45. Learned Weights with Increased Training for the Non-Fixed, Low Cost Condition and Random Experience

Table 26. Final Learned Partition Using 100 Episodes

Class	Grocery
C_0	ice_cream, chicken, yogurt, cucumbers
C_1	granola, ziploc_bags, tuna frozen_pizza, cereal, spaghetti, green_beans teriyaki_bowl, hot_soup
C_2	rotisserie
C_3	tuna
C_4	milk, frozen_fruit
C_5	oranges, potatoes
C_6	soda, fruit_juice
C_7	tissue, bread
C_8	chips
C_9	eggs
C_{10}	strawberries

There is one final point that should be discussed for the results just presented; for each condition the system demonstrates that it is able to learn, but its learning quickly levels out while there are still many errors in performance. This is especially evident in the performance plots for all three constraints (i.e., the tests on 30 bags and 12 states). While it is expected that performance will eventually level off, the results presented thus far level off very quickly even though there is still much room for improvement. One reason for this may be the "disconnect" between the random examples used to train the system and the tests used to evaluate the system. Essentially, when the system learns it extracts weights based on statistical patterns of "good" and "bad" grocery bags, while simultaneously generalizing these bags for storage in the relational maps. However, in this experiment all of this generalized experience was initially generated at random. Once the system begins to learn and performs purposive, deliberate actions it becomes increasingly unlikely that random experience will provide much useful guidance. This is due to the fact that the system is no longer performing randomly, and should not expect to encounter random bags in the future.

The next experiment is designed to investigate system performance using non-random experience. Initial states are still generated at random, but once completed, each episodes is iteratively added to the system's growing corpus of experience (long-term memory), and is included in the next training phase. Therefore, the system will "learn as it goes". This allows the system to develop knowledge structures that are uniquely tailored to the types of situations that it expects to encounter.

The next experiment will continue to investigate how performance varies with high and low action costs, as well as with fixed and non-fixed weights. Analyses will be performed on how the weights changes, what final concepts are formed, and whether the weights and concepts ever reach a final steady state. The system will only be trained on episodes generated using *GrocerySet-A*, but will be simultaneously tested using episodes generated from both grocery sets. Finally, some concluding thoughts will be offered with respect to the system's ability to cognitively process past experience in order to learn domain knowledge, or the appraisals for *relevance* and *utility*.

Experiment 2: Domain Knowledge Using Self-Guided Experience

Experiment Description

Unlike the previous experiment in which training was performed using episodes generated randomly, this experiment allowed the system to have some control over the generation of its own training experience. Groceries were still selected at random, but after the system used its current knowledge to complete each grocery-bagging task that experience was incorporated in the next round of training.

After each training iteration, the system was allowed to perform the grocery-bagging task for 10 episodes that had been initialized using *GrocerySet-A*, and 10 episodes that had been initialized using *GrocerySet-B*. For each episode the same six values (E1-E6) were recorded as before, however, in this experiment all of the tests were performed during the acquisition of new experience and the system was only required to appraise those bags that were encountered during task performance. Each episode was generated at random by selecting up to six groceries and placing them on the conveyor belt. As the system performed the grocery-bagging task, new groceries were selected and also placed on the conveyor belt. This process continued until a predetermined number of groceries had been selected and bagged. The number of groceries was predetermined by uniformly selecting from the range [5, 20]. A higher number of groceries could be used for this experiment because, at any one time, only a maximum of six groceries were present on the conveyor belt, which restricted the computation cost associated with lengthy deliberation. The steps for this experiment are listed as follows:

1. Train the system using the current experience in long-term memory.
2. Select the number of groceries N uniformly from the range [5, 20].
3. Set the total number of groceries bagged, $total\_count = 0$.
4. Select $0 < M \leq \min(6, N - total\_count)$ using a uniform distribution.
5. Select M groceries from *GrocerySet-A* or *GrocerySet-B* using a uniform distribution and then place those groceries on the conveyor.
6. Allow the system to bag each of the M groceries.
7. Set $total\_count = total\_count + M$.
8. If $total\_count < N$ return to Step (4), else go to Step (9).
9. Provide external feedback for the final situation.

10. Measure the error between the internal appraisals and the external feedback and record the number of constraint violations.
11. If *GrocerySet-A* was used, then add the new episode to long-term memory.
12. Set *num_episode* = *num_episode* + 1.0.
13. If *num_episode* is a factor of 10, add new episodes to long-term memory and go to Step (1), else go to Step (3)

Results and Discussion for the Fixed Weight and High Action Cost Condition

The same four conditions were used for this experiment as were used for Experiment 1. However, rather than running multiple trials and then averaging the results, the system was run through 700 episodes for each of the four conditions. A square filter was then used to smooth the data in order to analyze the underlying trends. Throughout this experiment the filter width was set to three training iterations (30 episodes).

Figure 46 presents the results for the fixed, high cost experimental condition, when tested using *GrocerySet-A*. Because Figure 46 presents the results for both training and testing with the same grocery set, these results provide a good indication of what the system has actually learned. Later in this subsection, the results from testing on *GrocerySet-B* are presented in order to investigate one aspect of how the system may generalize to new experience.

Figure 46(a) shows that the number of constraint (1) errors per bag decreases with training. This decrease is achieved while the system maintains a consistent ratio of *groceries per bag*, which is shown in Figure 46(c). In addition, the number of constraint (2) errors per bag (Figure 46b) does not increase, but rather remains constant after a very slight initial decrease. Figures 46(d), (e), and (f) show that with increased training the system's utility appraisals become more accurate. This is similar to the fixed, high cost condition for Experiment 1 with the exceptions that in this experiment the system 1) does not level out as quickly, and 2) exhibits more oscillation along each performance metric. These oscillations should not be interpreted indications that training with self-guided experience is less effective and that the system learns less. On the contrary, in the fixed, high cost condition of Experiment 1 the best ratio of constraint (1) errors to bags is 0.092, which is achieved at 60 episodes of training. Analysis of Figure 46(a) reveals that performance in Experiment 2 approaches the 0.05 range at multiple points during training. Furthermore, in Experiment 1 the best ratio of constraint (2) errors to bags is 0.024 (also

occurring at the 60 episode mark), and analysis of Figure 46(b) reveals that in Experiment 2 performance oscillates around the 0.025 range.

While in this experiment the system appears to learn more, the performance is also less consistent. This is evident from the presence of several large oscillations observed in Figures 46(a) and (d), as well as smaller oscillations in each of the remaining figures. Some of this oscillatory behavior can be explained by the large amount of unsupervised learning performed by the system, primarily in the generalization that takes place at the symbolic level of the trained SOMs. However, further explanation may be provided by the fact that the system's experience is constantly updating its current knowledge, and thus shaping the new experience that it acquires. In other words, unlike Experiment 1 in which the system had no control over its own training, now the training is intricately related to its current knowledge. The result is that the system may mistakenly pursue a sub-optimal path for several training iterations before the new experience is able to "outweigh" the old experience that caused pursuit of the sub-optimal path.

In addition, many of the larger oscillations also seem to co-occur. For example, the large oscillation at the 400 episode mark in Figure 46(d) is mirrored by an oscillation at the same location in Figure 46(a), which affirms the conclusion that appraisal accuracy does affect task performance. In addition, smaller oscillations occurring just before the 100 episode mark in both Figures 46(d) and (e), and are mirrored in the performance plots for both constraints (1) and (2), as well as the ratio of groceries to bags.

To further analyze system performance, Figure 47 presents additional breakdowns of system performance. This includes recording the number of bags per episode with two, three, and four groceries, as well as the error rate on constraints (1) and (2) for each bag type. Figure 47(a) presents the rate at which bags with two, three, or four groceries are generated per episode. This figure shows that as training increases, the number of bags per episode in which two groceries are placed steadily increases, while the number of bags with three groceries initially increases before finally decreasing. The point at which this decrease begins is, intriguingly, at the same point in which a large "jump" occurs in number of constraint (1) errors. Thus the system is sensitive to its own experience, and can become "pessimistic" in response to continued punishment.

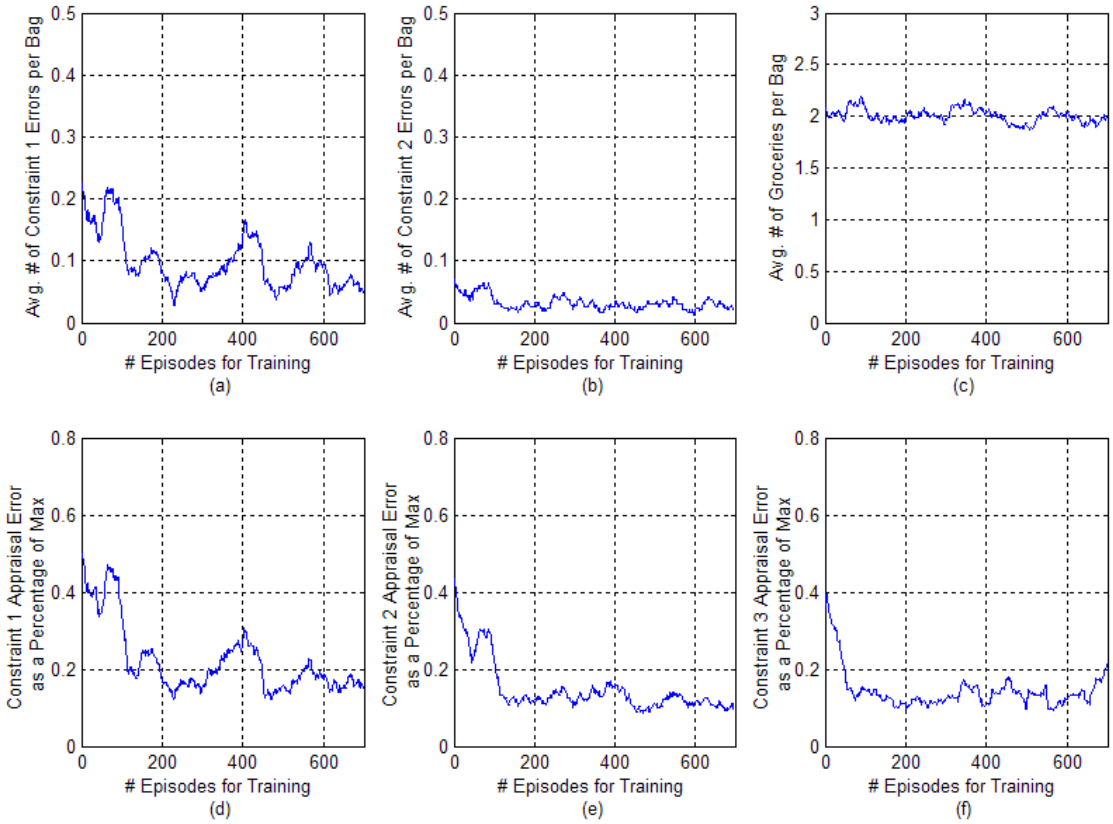


Figure 46. Evaluation Graphs for Episodes Generated with *GrocerySet-A* for the Fixed Weight, High Cost Condition and Self-Guided Experience

Figure 47(a) shows that the number of bags per episode with four groceries remains roughly constant (with a slight decrease) during training, while Figure 47 as a whole shows that the rate of occurrence for each bag type decreases as the number of groceries in that bag increases. However, Figure 47(b) shows that the rate of occurrence is also inversely related to the rate of constraint (1) violations per bag, and Figure 47(c) shows the same relationship with the rate of constraint (2) violations per bag. Because the error rate for each bag type decreases with training, it can be concluded that the system is learning how to combine groceries in order to better complete the task, but because certain bag types are more risky than others, these bags cause more errors and are thus explored less. In an experience-based learner such as this, less exploration means less opportunity to learn, and thus less learning. This is supported by the results in Figure 47.

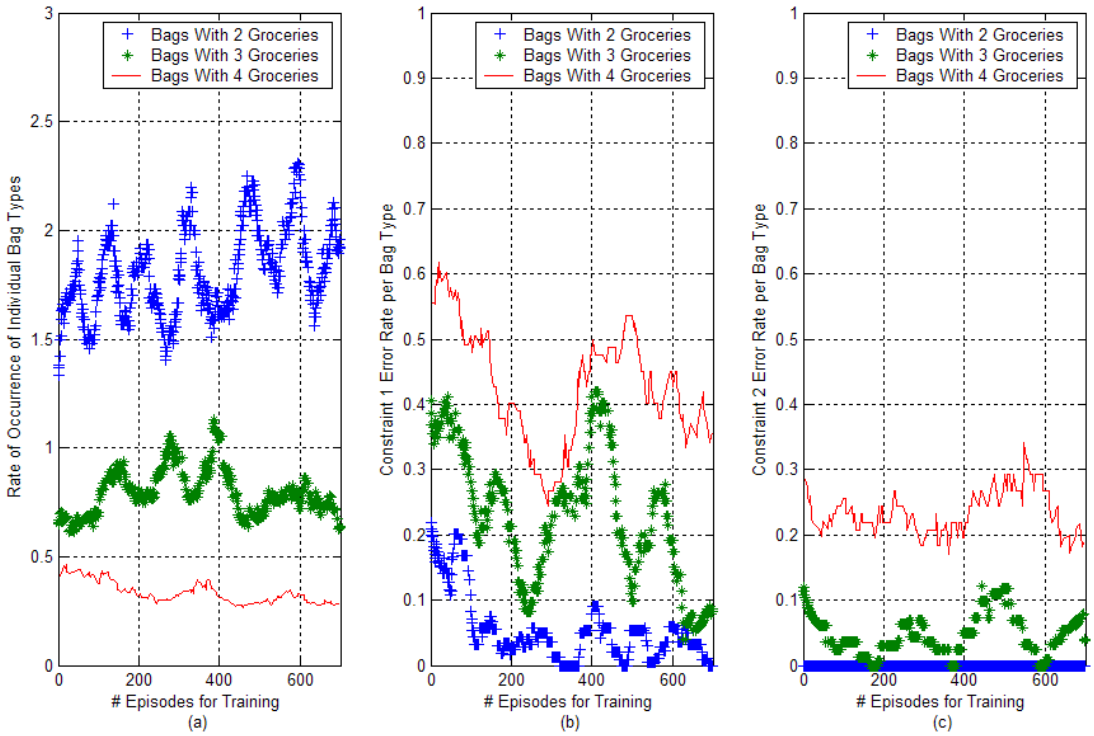


Figure 47. Breakdown of Error Rate and Rate of Occurrence for Bags with Different Amounts of Groceries for the Fixed Weight, High Cost Condition and Self-Guided Experience

Figure 48 presents the results for the episodes generated using *GrocerySet-B*. The system has not been trained on any of these groceries, but rather must place these groceries in the correct grocery cluster using the fixed set of weights and the developed grocery partition. Using this information the system must then associate the resulting feature vectors with the appropriate sections of the trained relational map. Unlike the results presented for *GrocerySet-A*, Figure 48(a) shows that the number of constraint (1) violations per bag does *not* decrease with increased training, while Figure 48(b) shows that the number of constraint (2) violations only slightly decreases with training, however, this decrease is most likely explained by a subsequent slight decrease in the average number of groceries per bag (Figure 48c). Figures 48(d) and (e) suggests that this lack of improvement is most likely due to incorrect appraisals for constraint (1) and (2). While the appraisal error does decrease with training, the amount of decrease is dramatically slower than with *GrocerySet-A* and the lowest appraisal error rate obtained for *GrocerySet-B* is only slightly better than the worst obtained for *GrocerySet-A*.

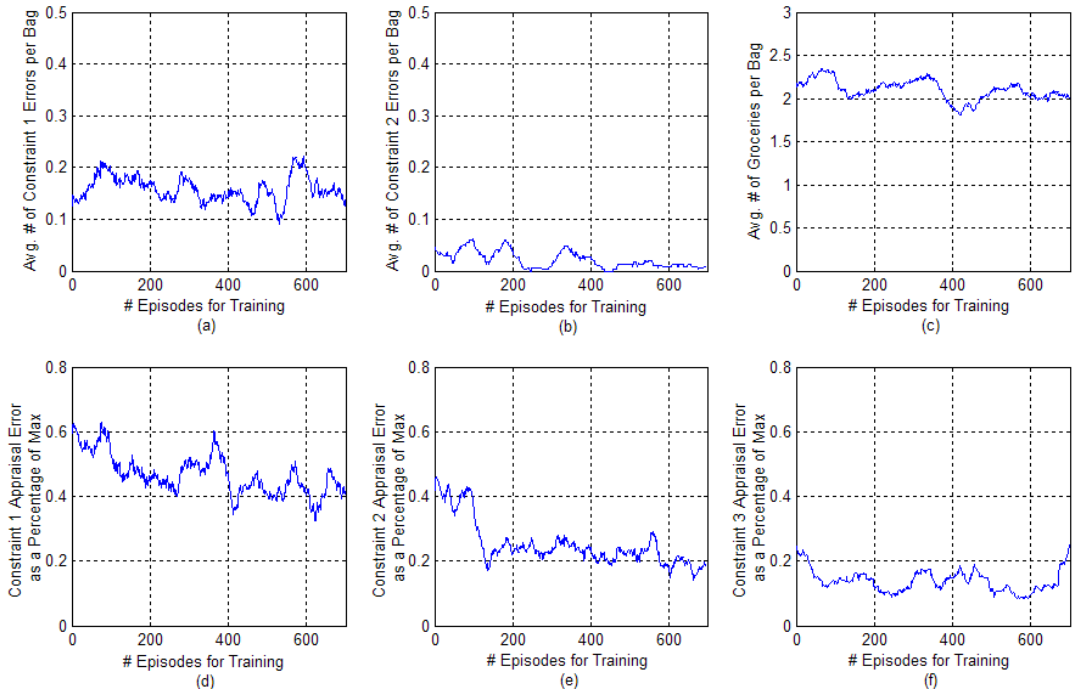


Figure 48. Evaluation Graphs for Episodes Generated with *GrocerySet-B* for the Fixed Weight, High Cost Condition and Self-Guided Experience

One highly possible explanation for this behavior is that the fixed weights do not allow the system to separate groceries in the manner that it would "prefer", based on its own experience, and thus the fixed weight set may not be best when extrapolating to new experience. In other words, the system may better off creating its own grocery clusters. To examine this possibility Tables 27 and 28 present the grocery clusters for both grocery sets. For each grocery set these clusters represent the same concepts, with the only difference being the specific groceries assigned to each cluster.

Table 27. Final Partition Using *GrocerySet-A*

Class	Grocery
C_0	ice_cream, frozen_pizza, yogurt, cucumbers, chicken
C_1	granola, ziploc_bags, tuna spaghetti, green_beans
C_2	rotisserie
C_3	hot_soup, teriyaki_bowl
C_4	milk, frozen_fruit
C_5	oranges, potatoes
C_6	soda, cereal, fruit_juice
C_7	tissue, bread, chips
C_8	eggs, strawberries

Table 28. Final Partition Using *GrocerySet-B*

Class	Grocery
C_0	waffles, frozen_vegetables, ground_beef
C_1	ketchup, popcorn, rice, hot_chocolate, macNcheese vegetable_soup, choc_cookies peanut_butter, vegetable_oil
C_2	ribs
C_3	EMPTY
C_4	EMPTY
C_5	EMPTY
C_6	trash_bags, coffee, lettuce cheezits, ritz_chips
C_7	cookie_dough, marshmallows bagels, tomatoes, bananas, cake
C_8	cheese

Inspection of these partitions reveals that the distribution of groceries across clusters is much less uniform with *GrocerySet-B* than with *GrocerySet-A*. In particular, there are no groceries from *GrocerySet-B* assigned to the either of the classes $\{C_3, C_4, C_5\}$. Intuitively, this should not matter so long as the final grocery clusters obey the same learned relationships as the corresponding clusters for *GrocerySet-A*. For example, if the system has learned that members of class C_0 can always be grouped with other members of class C_0 and that members of C_8 must be kept separate from members of C_5 for *GrocerySet-A*, then these relationships must also be true for *GrocerySet-B*.

The first test performed to determine if the relationships between clusters are different was to test the intra-cluster grouping in each cluster: could members of each cluster be mixed? For this test, nine bags were created for each grocery set using the exact contents in each of the nine clusters, i.e., bag_0 for *GrocerySet-B* contained the groceries *waffles*, *frozen_vegetables*, *ground_beef*. For both grocery sets, none of the bags violated constraint (1), and therefore this constraint was preserved for intra-cluster grouping.

In the second test, the relationship between two of the nine clusters was analyzed. This test was designed to investigate inter-cluster grouping: if members of one cluster could be grouped with members of another cluster for *GrocerySet-A*, does the same relationship hold for *GrocerySet-B*? From empirical observation the grocery classes C_1 and C_7 were chosen based on the large amounts of groceries in each class, and from the fact that certain groceries in each class

were known *not* to mix well. This test involved creating all sets of bags in which exactly one member of class C_1 and one member of class C_7 were present and asking the external critic to evaluate each bag. For *GrocerySet-A* there were 15 such bags, of which none caused a constraint (1) violation. For *GrocerySet-B*, however, there were 54 such bags, of which 16 (29.6%) caused a constraint (1) violation. Therefore, using the fixed weights designed for *GrocerySet-A*, the system learned relationships that proved contradictory when applied to a fundamentally different grocery set, *GrocerySet-B*.

Results and Discussion for the Non-Fixed Weight and High Action Cost Condition

Figure 49 provides the results for the non-fixed, high action cost condition using *GrocerySet-A*. As with the fixed, high action cost condition the system exhibits consistency in the number of groceries per bag while decreasing the number of errors per bag. Like the previous condition, Figure 49(b) shows that system performance on constraint (2) remains roughly constant with a very low error rate, and that while there is no noticeable decrease there is also no increase in the error rate per bag. In addition, Figure 49(a) shows that the error rate on constraint (1) decreases, but at a slower rate than that observed for the fixed cost condition. This slower rate of improvement is explained by Figure 49(d), which shows the system has more difficulty appraising this constraint.

Figure 50 presents the rate of occurrence and error rate for bags with two, three, and four groceries. As with the Figure 49, the trends shown in Figure 50 are similar to those observed in the fixed condition: the system steadily selects more two-grocery bags and remains roughly constant on its selection of the other two bags, while choosing the less risky three-grocery bag type more often. The order of occurrence (i.e., which bag types occur more frequently) and the order of error rate (i.e., which bag types are the most dangerous) are the same for both the fixed and non-fixed conditions, however, the error rate is noticeably higher in the non-fixed versus the fixed condition.

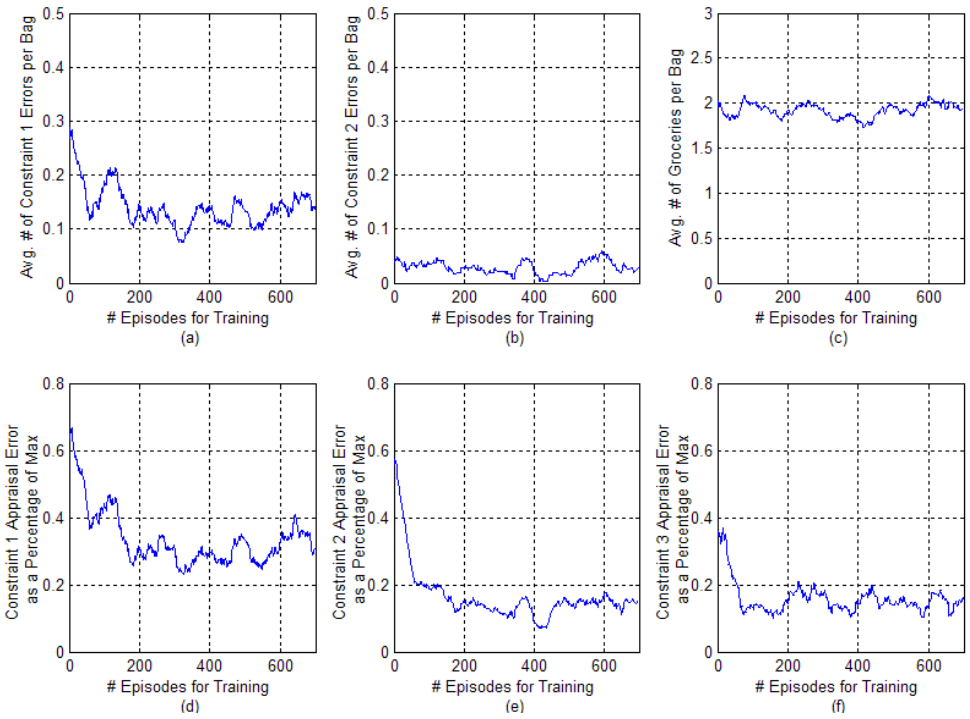


Figure 49. Evaluation Graphs for Episodes Generated with *GrocerySet-A* for the Non-Fixed, High Cost Condition and Self-Guided Experience

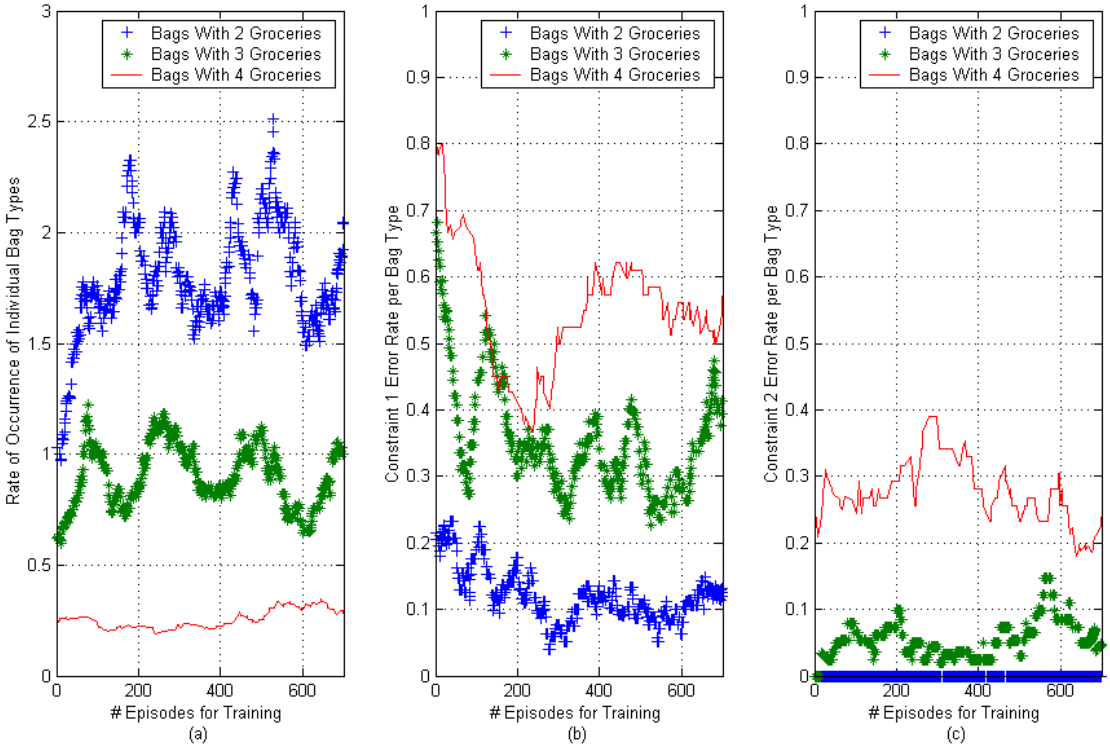


Figure 50. Breakdown of Error Rate and Rate of Occurrence for Bags with Different Amounts of Groceries for the Non-Fixed Weight, High Cost Condition and Self-Guided Experience

Some of the increases in error rates for this condition may be explained by noting that the system is required to simultaneously learn relevance appraisals and utility appraisals. Unlike the previous condition, in this condition the system was required to learn the weights for each attribute and to use these weights to develop useful grocery partitions. Figure 51 plots the individual weight values as a function of the number of episodes used for training. As in Experiment 1, the weight associated with the grocery's *name* immediately drops to 0.0, while the weight associated with *firmness* of the grocery stays at 1.0. Figure 52 shows how the percent difference between each weight value and its mean changes with increased training. This figure indicates that while the individual weight values do oscillate, with increased training these oscillations generally tend to reduce in magnitude. Interestingly, the weights that exhibit the most oscillation are those associated with the attributes *color*, *price*, and *type*, which are three attributes that are not required for grocery bagging. It appears that while these attributes are believed to be the least relevant (except for name), the system has a very difficult time pushing these weights to zero. This is explained by referring to Chapter V, and the discussion on how spurious patterns that are not relevant to the current goal may be hidden in the experience used to train the system.

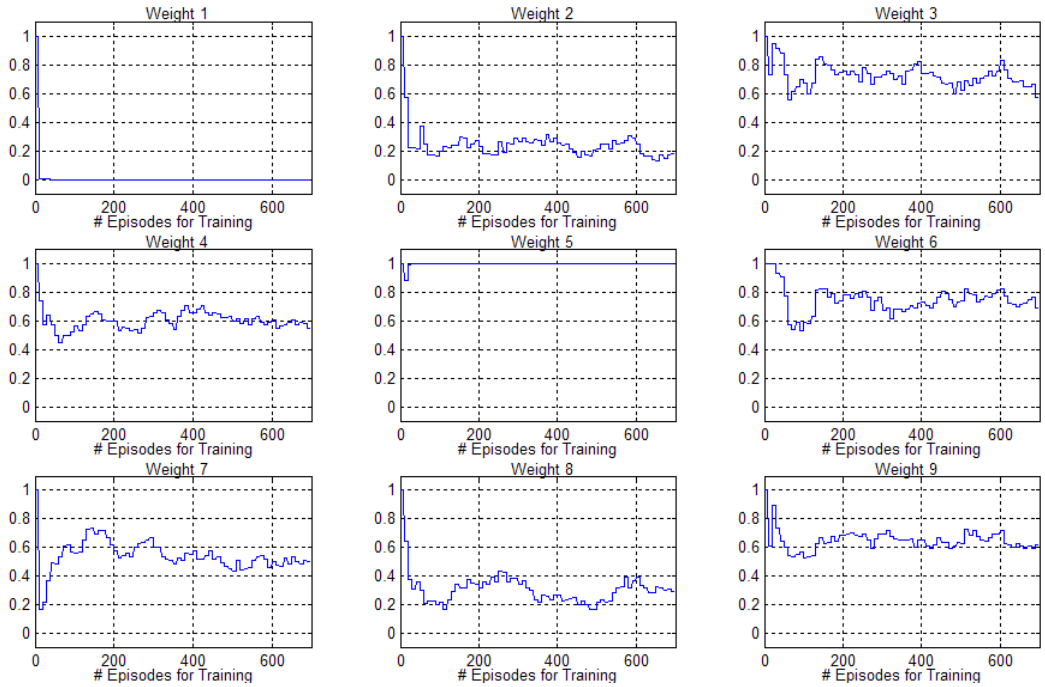


Figure 51. Learned Weights with Increased Training for the Non-Fixed, High Cost Condition and Self-Guided Experience

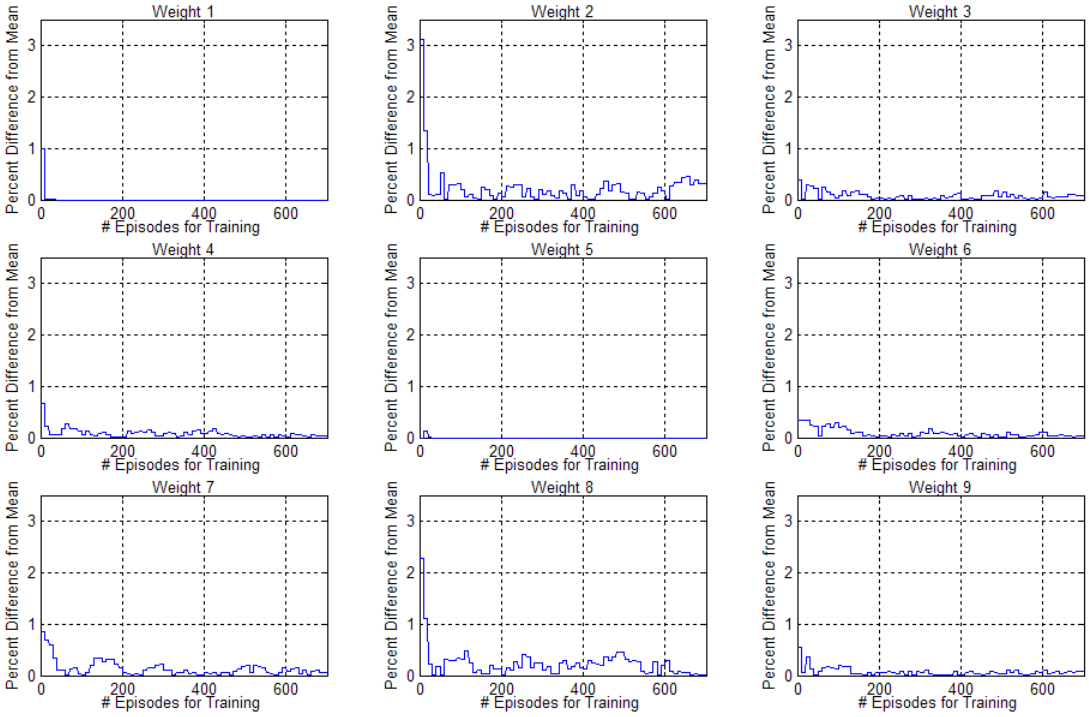


Figure 52. Percent Difference Between the Learned Weight Values During Training and the Weight Means for the Non-Fixed Weight, High Cost Condition and Self-Guided Experience

Table 29 presents the final grocery clusters after 700 episodes of training. These clusters are similar to those learned using random experience, however, of the final 30 training epochs (episodes 400:700) these clusters were only identified ~30% of the time. The remaining training epochs are nearly uniformly distributed across three additional partitions. Two of these partitions are presented in Tables 30 and 31. Each differ from the partition presented in Table 29 by just a small amount, and each occur ~25% of the time. The third partition appears 20% (6/30) of the time and is an inconsistent partitioning in which class C_0 and C_1 (Table 29) have been arbitrarily split into 3-4 new classes, however, of these arbitrary partitions only occur once in the final 15 training epochs and not at all in the final 10 epochs. The take home point regarding these learned partitions is that for the final 300 episodes (30 training epochs) as the learned weights fluctuate the system alternates between the three similar classification schemes presented in Tables 29, 30, and 31, approximately 80% of the time, with the classification scheme of Table 29 occurring most frequently (30%).

Table 29. Final Learned Partition Using 700 Episodes (30% Occurrence)

Class	Grocery
C_0	ice_cream, chicken, yogurt, cucumbers
C_1	granola, ziploc_bags, frozen_pizza, cereal, spaghetti, green_beans teriyaki_bowl, hot_soup
C_2	rotisserie
C_3	tuna
C_4	milk, frozen_fruit
C_5	oranges, potatoes
C_6	soda, fruit_juice
C_7	tissue, bread
C_8	chips
C_9	eggs
C_{10}	strawberries

Table 30. Learned Partition (25% Occurrence)

Class	Grocery
C_0	ice_cream, chicken, yogurt, cucumbers
C_1	granola, ziploc_bags, frozen_pizza, cereal, spaghetti, green_beans teriyaki_bowl, hot_soup
C_2	rotisserie
C_3	tuna
C_4	milk, frozen_fruit
C_5	oranges, potatoes
C_6	soda, fruit_juice
C_7	tissue, bread
C_8	chips
C_9	eggs, strawberries

Table 31. Learned Partition (25% Occurrence)

Class	Grocery
C_0	ice_cream, chicken yogurt, cucumbers eggs, tuna
C_1	granola, ziploc_bags, frozen_pizza, cereal, spaghetti, green_beans,
C_2	rotisserie
C_3	teriyaki_bowl
C_4	milk, frozen_fruit
C_5	oranges, potatoes
C_6	soda, fruit_juice
C_7	tissue, bread
C_8	chips
C_9	strawberries
C_{10}	hot_soup

The performance measures for the episodes generated from *GrocerySet-B* are provided in Figure 53. As in the fixed condition, there is a slight net decrease in the number of groceries per bag, however, unlike the fixed condition both constraints (1) and (2) show some improvement with learning. Figure 53(d) and (e) both show that the system exhibits a decrease in appraisal errors, which explains the improved performance in Figures 53(a) and (b). This is explained through the results presented in Figures 53(c) and (e), in which the system shows that its ability to appraise individual grocery bags generated from *GrocerySet-B* improves at a rate greater than that for the fixed condition. Furthermore, Figure 54 shows that the appraisal accuracy for

GrocerySet-B shows the same trend as that observed for *GrocerySet-A*. Examination of the final grocery partitions for *GrocerySet-B*, corresponding to the learned partition of Table 30, indicates that the errors which result from applying the relations learned for *GrocerySet-A* to *GrocerySet-B* has been reduced dramatically. This examination focuses on clusters C_1 and C_7 as these approximately correspond to the clusters that were used to analyze inter-cluster error rate for the fixed condition. Whereas in the fixed condition, the inter-cluster error rate was 29.6%, in the non-fixed condition this value has dropped to 8.3%. Therefore, the system has learned a set of weights that better enable generalization to new situations and experience than the fixed set of weights provided for the earlier condition. While this simple analysis does not cover all possible learned relations, comparisons between Figures 53(d), (e) and 49(d), (e) indicate that this improvement does extend beyond those relationships that include clusters C_1 and C_7.

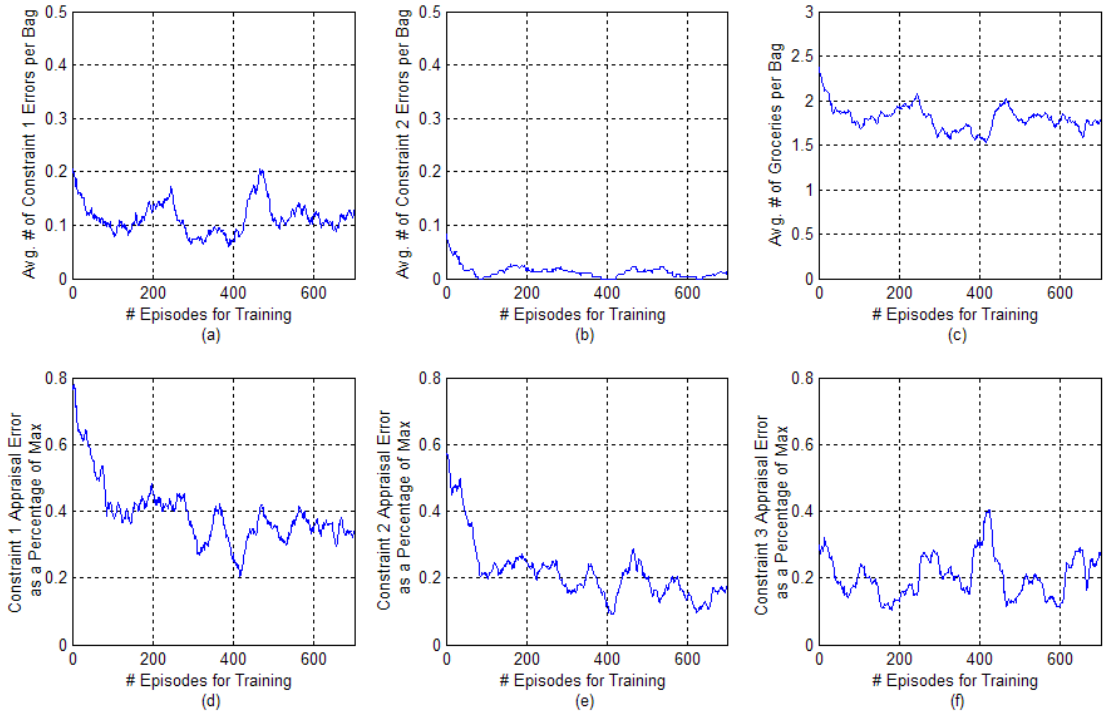


Figure 53. Evaluation Graphs for Episodes Generated with *GrocerySet-B* for the Non-Fixed, High Cost Condition and Self-Guided Experience

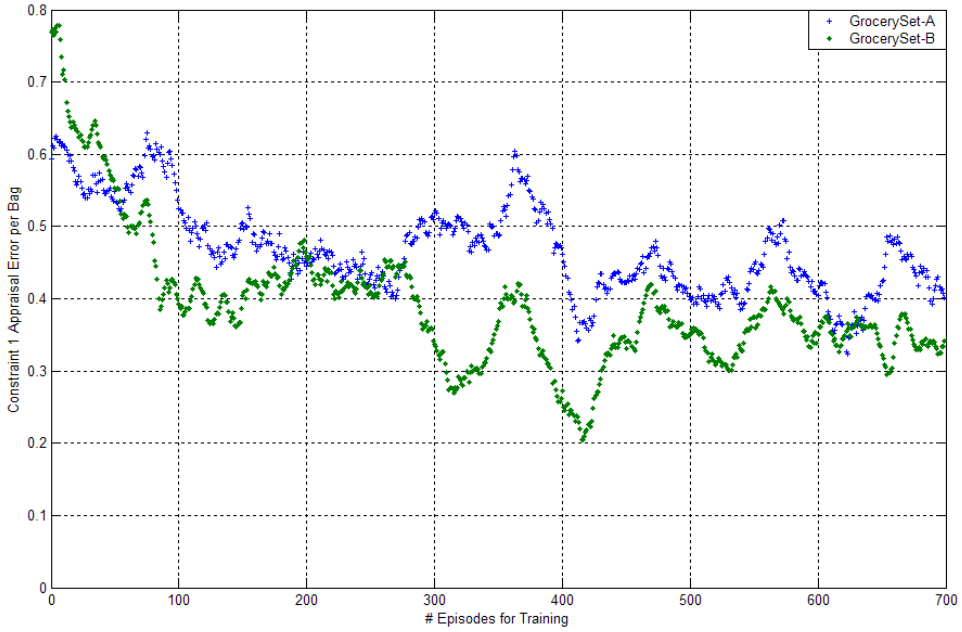


Figure 54. Constraint (1) Appraisal Error Rate for the Non-Fixed Weight, High Cost Condition and Self-Guided Experience

Results and Discussion for the Fixed Weight and Low Action Cost Condition

This condition investigates the effects of having a lower action cost, which allows the system to choose actions based only on its learned preferences. There is no significant punishment for choosing to get a new bag for every grocery that appears on the conveyor belt, other than the appraisal information provided for constraint (3). Figure 55 presents the results for the fixed, low cost condition for 700 episodes of training. These results are similar to those obtained for the fixed, high cost condition, however, the error rate for this condition has been shifted down and the oscillations are smaller. Figure 55(a) shows that for this condition performance on constraint (1) is, on average, better than it has been for any of the previous conditions. However, the ratio of groceries to bags also never approaches that of the fixed, high cost condition. The most obvious explanation for this behavior is that the lower action cost is not forcing the system to explore mixing groceries as much as the higher action cost. While such conservative behavior should have a detrimental effect on learning by providing fewer complex examples for training, this does not appear to be the case when analyzing the remaining plots of Figure 55. In these plots the errors rates and appraisal accuracies all show similar trends of improvement, but that it should be noted that the system is, by design, an experiential learner and that as it learns from experience the learned knowledge dictates the type of "new" experience it

acquires. Therefore, the appraisal accuracy should improve, but this improvement is for a different type of experience than that encountered in the earlier conditions: experience generated without the pressure of a high action cost and thus a lower ratio of groceries to bags. The results presented in Figure 56 validate this explanation by showing that the error rate per bag for each bag type decreases with experience for all types.

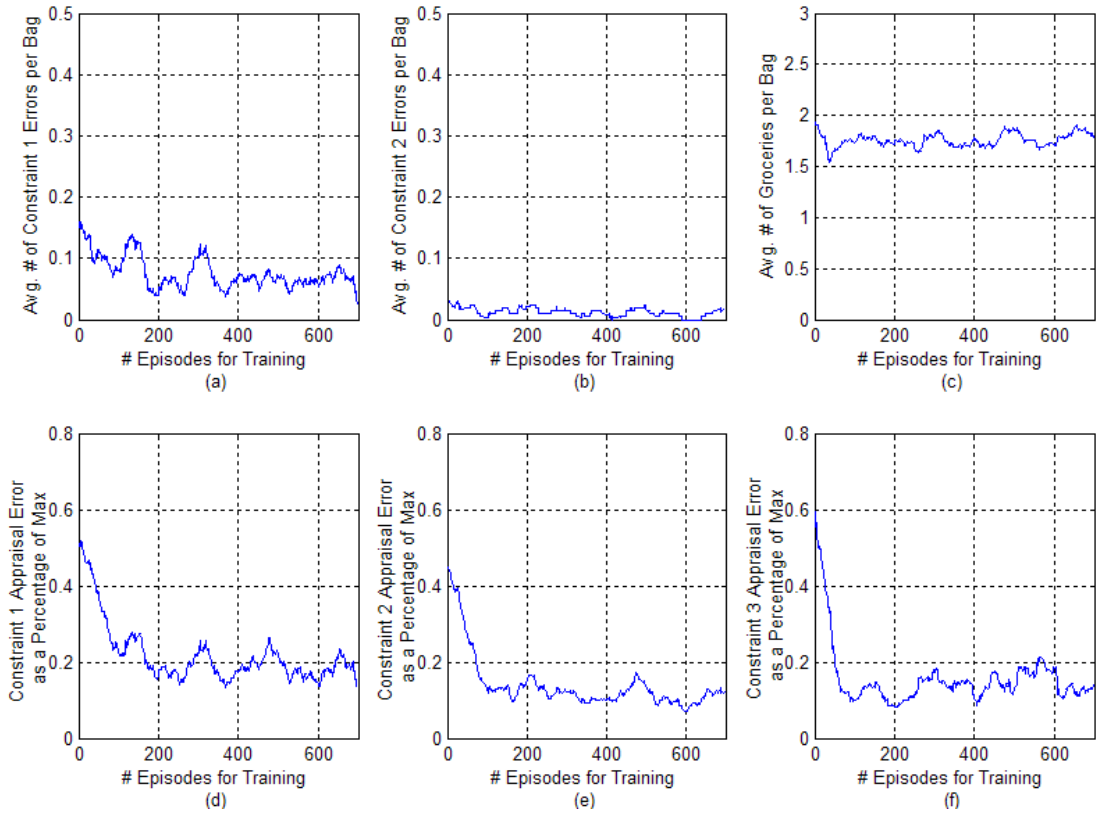


Figure 55. Evaluation Graphs for Episodes Generated with *GrocerySet-A* for the Fixed Weight, Low Cost Condition and Self-Guided Experience

Figure 56 shows that the rate of occurrence per episode for bags with two groceries steadily increases with training, and that the error rate on these bags steadily decreases. In addition, the rate of occurrence for bags with three and four groceries remains roughly constant, with the rate of occurrence for the three-grocery bags showing an increase at the end of training. Simultaneously, the number of errors *per bag* decreases with training, and thus it should be concluded that the system is learning to combine groceries in ways that enable the performance on constraints (1) and (2) to improve. Of these three bag types, the four-grocery bags show the least improvement but this is explained by noting that the rate of occurrence for these bags is

lowest of all bag types across all experimental conditions described thus far. Without the increased exploration (and subsequent errors) caused by the higher action cost, the system rarely explores risky situations and learning suffers. Interestingly, the error rate for constraint (2) on these bags steadily decreases with training. The most likely explanation for this is a crossover effect when creating bags with four groceries. To understand this effect, it is important to note that all four-grocery bags were, at on point, three-grocery bags, and that all three-grocery bags were, at one point, two-grocery bags, etc. If the system learned to appraise constraint (2) correctly for these earlier bags, then the system would be much more likely to "serendipitously" appraise the later bags correctly, because of the close proximity of these different bags in feature space. Furthermore, these extrapolated appraisals are also more likely to be correct for constraint (2) than for constraint (1) because the addition of a single grocery to a single bag is less likely to cause a constraint (2) error than a constraint (1) error if the previous bag was evaluated positively. This is because constraint (2) errors (overloading a grocery bag) are rarely determined by a single grocery, while constraint (1) errors can swing dramatically based on the inclusion of a single grocery (e.g., "potatoes with the eggs and bread").

Figure 57 presents the results from testing the system with *GrocerySet-B*. As with the fixed, high cost condition there is only a marginal improvement on the number of constraint (1) errors per bag. The same explanation is provided as in the previous condition: the fixed weights (which were not designed for *GrocerySet-B*) are causing errors when the system attempts to apply the same learned relationships across grocery sets. This is confirmed by noting the slow rate of improvement for the appraisal accuracy in this condition, i.e. Figure 57(c).

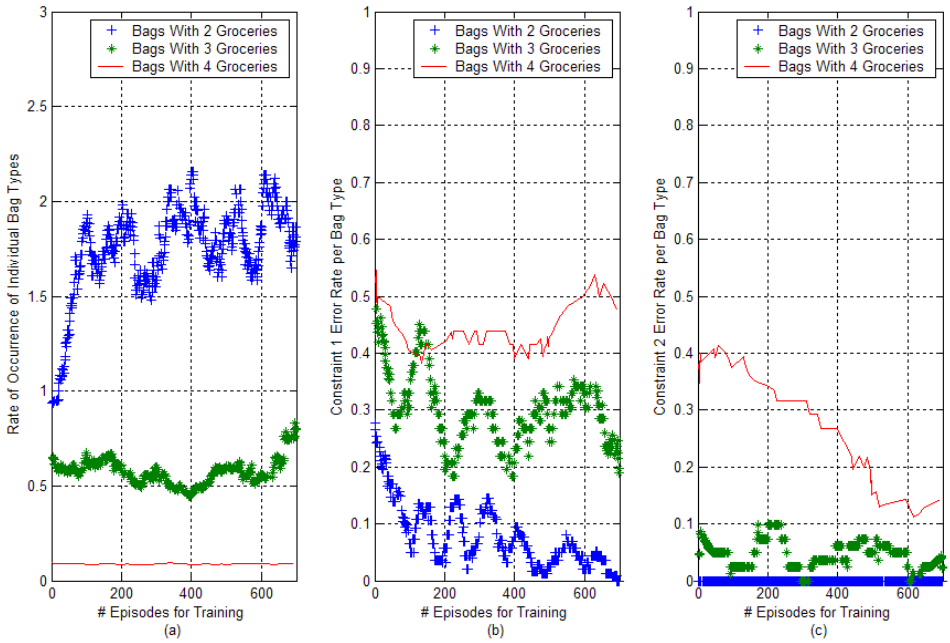


Figure 56. Breakdown of the Error Rate and Rate of Occurrence for Bags with Different Amounts of Groceries for the Fixed Weight, Low Cost Condition and Self-Guided Experience

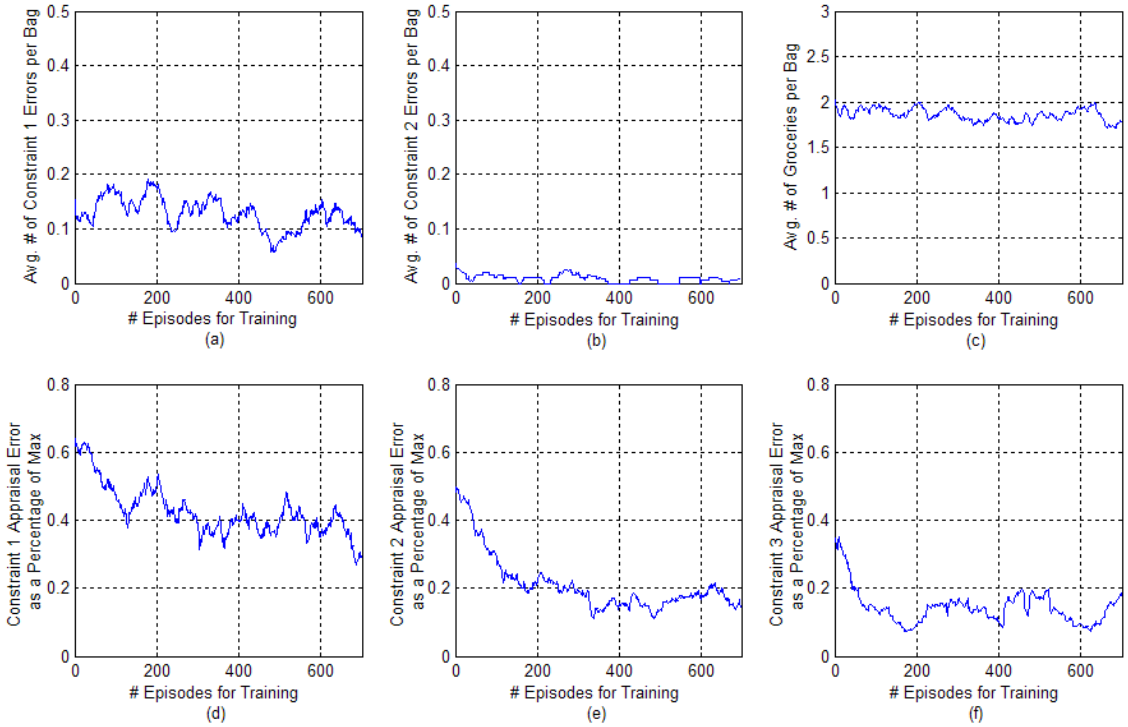


Figure 57. Evaluation Graphs for Episodes Generated with *GrocerySet-B* for the Fixed Weight, Low Cost Condition and Self-Guided Experience

Results and Discussion for the Non-Fixed Weight and Low Action Cost Condition

The final condition for Experiment 2 requires evaluating the system with non-fixed weights and a low action cost. Learning and improvement on the task should be much more difficult for this condition than for any other. The earlier non-fixed weight, high action cost condition indicated that non-fixed weights caused the system to perform (slightly) more errors than in the fixed condition. However, those results also indicated that the system performed better when exposed to *GrocerySet-B*. The earlier fixed, low cost condition indicated that a lower action cost slowed learning by not forcing the system to explore new options, and thus it is expected on this condition that the system will exhibit a much slower learning rate than in any previous condition, but that the system should perform better on *GrocerySet-B* than when trained on the fixed, low cost condition.

Figure 58 presents the results for the non-fixed, low cost condition for *GrocerySet-A*. This figure shows that the number of constraint (1) errors per bag remains constant as experience increases, even though the appraisal accuracy for each bag tends to be more correct. An interesting point about these results is that, while the system does not improve on constraint (1), its average performance is as good as it has been for any condition except for the fixed, high cost condition described earlier in Experiment 2.

Further investigation of these results requires analysis of the individual bag types. Figure 59(a) shows that the number of bags per episode with two groceries initially increases and then levels off, while the number of bags with three and four groceries remain constant throughout training. Figure 59(b) shows that the rate of constraint (1) errors decreases for bags with two and four groceries, but that this rate actually increases for bags with three groceries. This immediately explains why the rate of constraint (1) errors per bag (Figure 58a) remains constant: the decrease in error rate on two- and four-grocery bags is counterbalanced by the increased error rate in three-grocery bags.

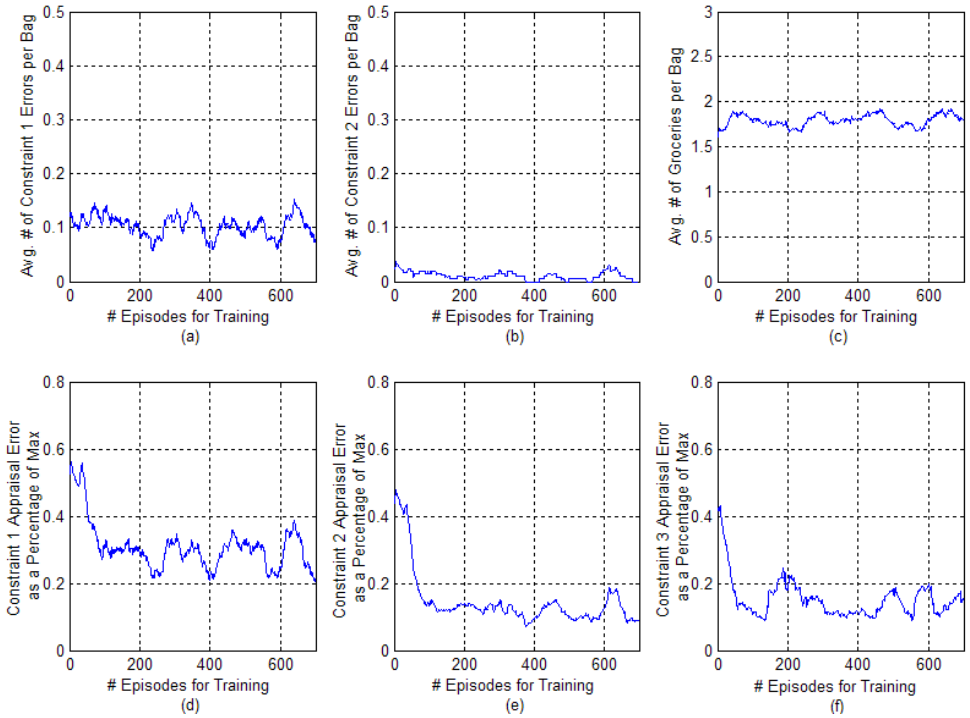


Figure 58. Evaluation Graphs for Episodes Generated with *GrocerySet-A* for the Non-Fixed Weight, Low Cost Condition and Self-Guided Experience

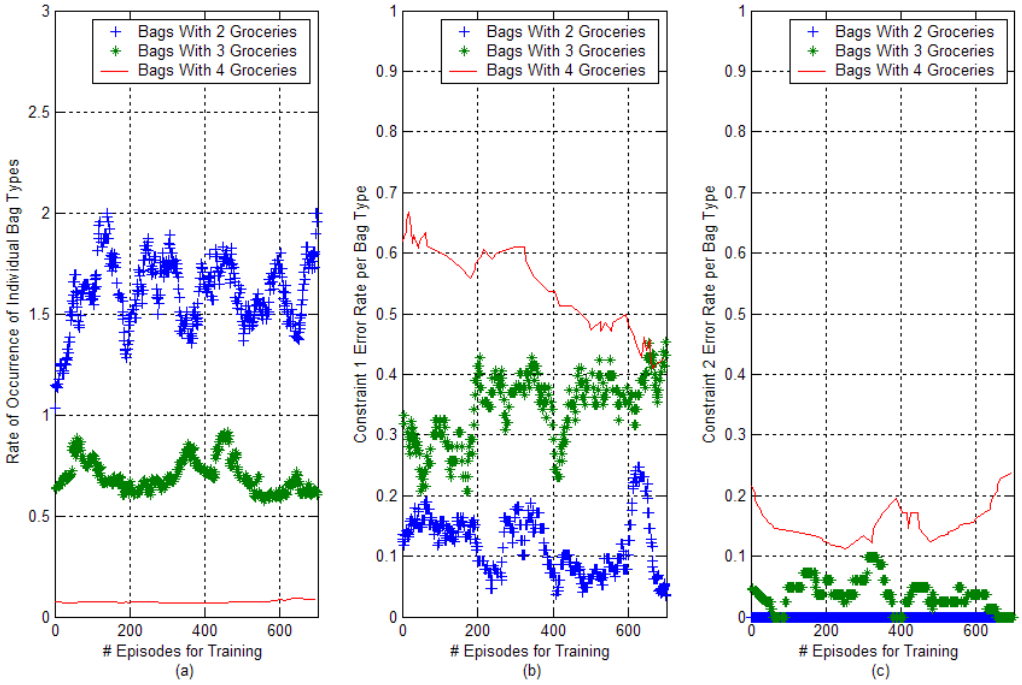


Figure 59. Breakdown of the Error Rate and Rate of Occurrence for Bags with Different Amounts of Groceries for the Fixed Weight, Low Cost Condition and Self-Guided Experience

Analysis of how the weights change with increase training does not shed any light on why system performance does not improve for bags with three groceries. The weight values are presented in Figure 60. The maximum oscillation/peak as a percentage of the mean is presented in Figure 61. As with the earlier non-fixed condition, the weight oscillations reduce in amplitude with increased training except for those attributes that are given a low weight value and are not relevant to the grocery-bagging task.

The developed partitions that resulted from the learned weight values were the same as those developed for the high cost condition, however, the distribution across those partitions was dramatically different. Whereas in the earlier condition three different partitions were created with an almost uniform distribution, for the current condition the partition presented in Table 29 was only created 10%, while the partition from Table 30 was created 53% of the time, and the partition presented in Table 31 was created 16.7% of the time. The remaining partitions involved arbitrarily splitting the larger clusters into multiple smaller clusters. These final three partitions, however, are not to blame for the poor performance on bags with three groceries. As noted earlier, these partitions appropriately clustered the groceries so as to reduce the number of constraint (1) errors within clusters while leaving it to the relational maps to learn the inter-cluster relationships. With the low action cost and subsequent lower exploration rate, it appear that the relational maps are simply not learning as well in this condition as in prior conditions.

Figure 62 presents the results of testing the trained system on experience generated from *GrocerySet-B*. Figures 62(d) and (e) show that just as in the earlier non-fixed condition, the system learns to correctly appraise each situation with respect to constraints (1) and (2), but there is no improvement on these constraints during task performance. The error rate, however, for constraint (1), as well as the error rate for constraint (2), are consistently lower than they are in any of the previous conditions. Unfortunately, the ratio of groceries to bags is also lower than it has been at any other point during these experiments. Thus, the lack of exploration greatly hinders the ability of the system to improve upon its task performance and the behavior appears to be unchanged while the appraisals increase in accuracy.

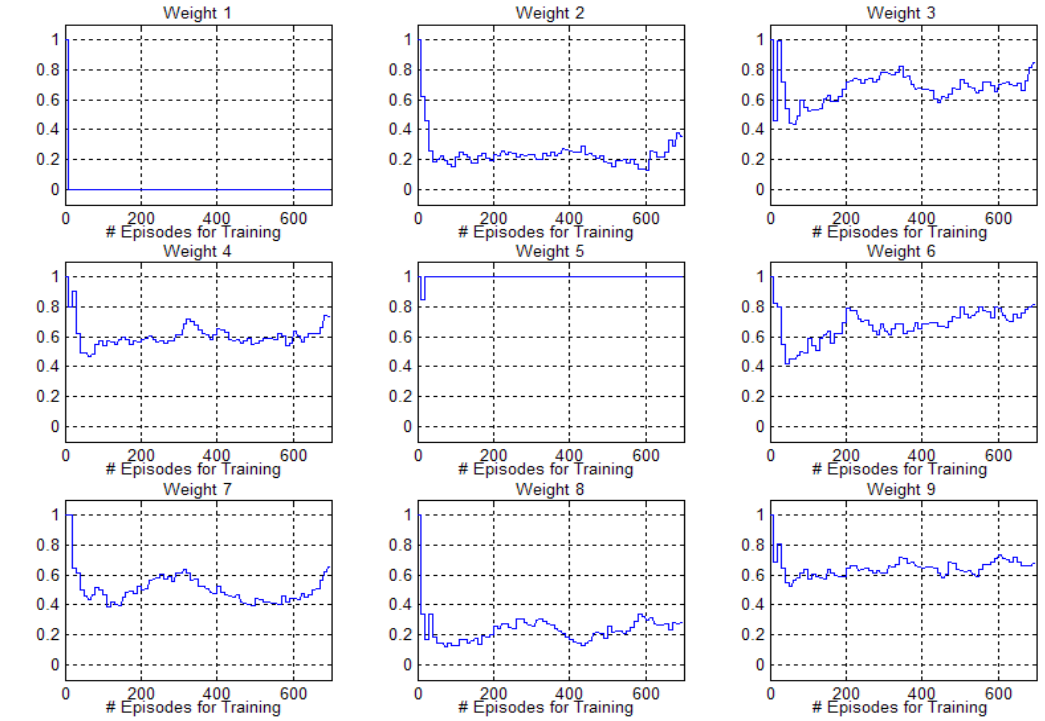


Figure 60. Learned Weights with Increased Training for the Non-Fixed Weight, Low Cost Condition and Self-Guided Experience

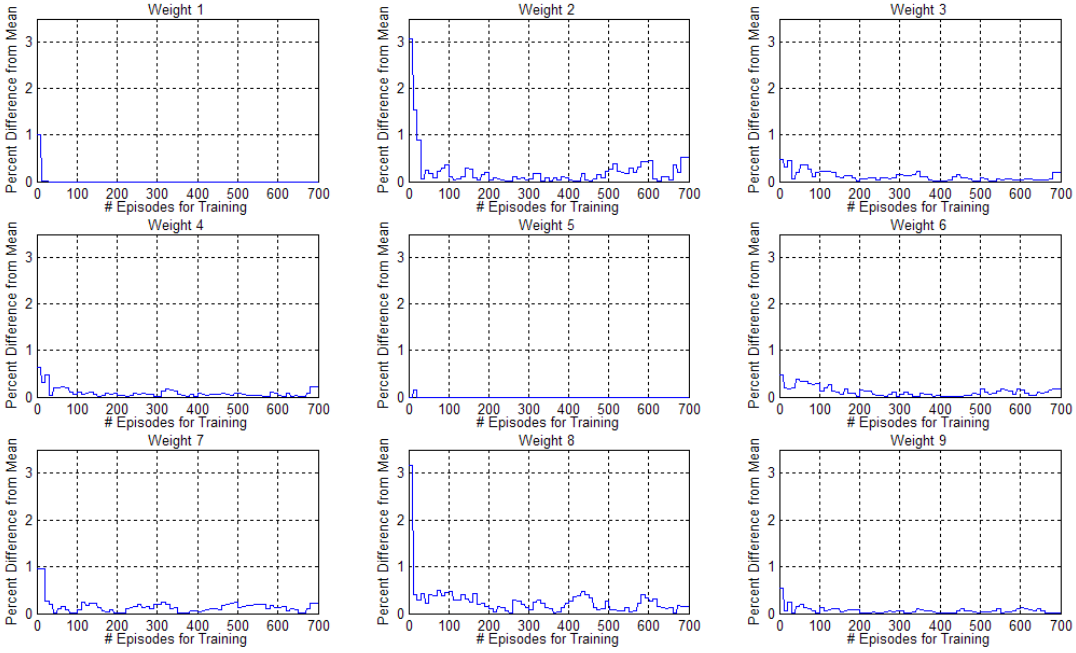


Figure 61. Percent Difference Between Maximum Peaks During Weight Training for the Non-Fixed Weight, Low Cost Condition and Self-Guided Experience

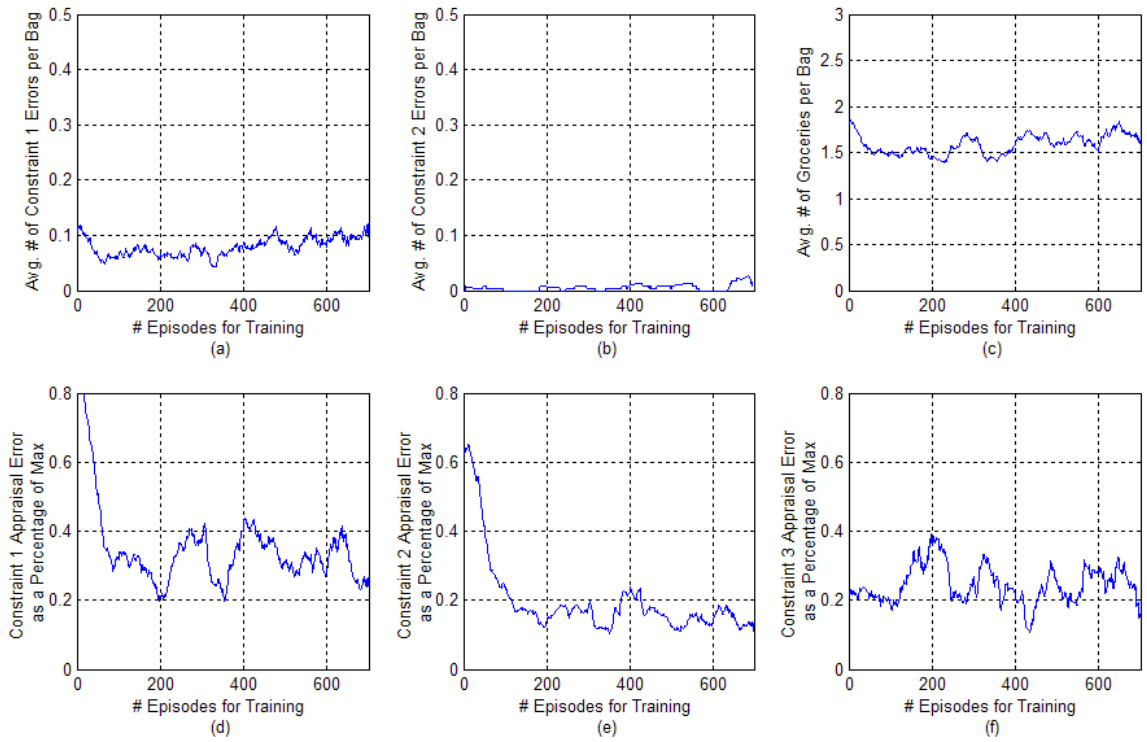


Figure 62. Evaluation Graphs for Episodes Generated with *GrocerySet-B* for the Non-Fixed, Low Cost Condition

Experiment 2 was designed to evaluate how well the system could extract knowledge from its own experience and then apply that knowledge to future iterations of the grocery-bagging task. Throughout each experimental condition, the system was evaluated and the results indicated that in order for the system to learn it must be forced to explore the wealth of possible situations available to it. Without this exploration, the system may improve on its ability to appraise situations, but this improvement is not reflected in it behavior. The rational is that without exploration, the system has fewer alternatives for it to "prefer". It should be noted that the appraisal error that has been measured throughout this experiment was based on the actions that were actually chosen by the system. In other words, the results presented here only indicate that appraisal accuracy improves for those situations that the system chooses to explore/pursue.

Certainly, the need for exploration is not a new concept in intelligent robotics, and thus it should be expected that when the system explores less the learning will suffer. However, it was not as expected that in order for the system to generalize it must be allowed to develop its *own* relevance appraisals. When the identification of goal-relevant features is not rooted in the system's own experience, the system runs the risk of creating goal-relevant concepts that are

overly specialized to whatever *a priori* knowledge was provided. If this knowledge is good, obviously performance will be good as well, but when this knowledge contains errors, the system is unable to generalize beyond the limited concepts that it was able to form at initialization. While the generalized performance presented here (*GrocerySet-B*) did not exhibit the same level of learning as the non-generalized performance (*GrocerySet-A*), when the system was allowed to create its own concepts the performance was better that when the system had to use the preset concepts.

Enabling the system to develop its own goal-relevant concepts, however, requires that the system have more flexibility in learning the correct utility appraisals. Throughout Experiment 2, as well as Experiment 1, the system demonstrates that when it is required to learn goal-relevant concepts, its utility appraisals do not degrade. In fact, in many of the conditions that have been discussed, the utility appraisals are as good as, and sometimes better, when the system is allowed to develop its own notions of relevance.

The results that have been presented thus far require the system to balance multiple constraints without any *a priori* knowledge of which constraints are more important. Therefore, as the system has learned to appraise each situation it must often perform a tradeoff between conflicting constraints. For example, while the average ratio of groceries to bags remained close to 2.0 across and throughout all conditions, a simpler solution to the grocery-bagging problem would have involved merely placing one grocery in each bag. This solution would have reduced the number of constraint (1) and (2) violations to zero, but the system would have been penalized by constraint (3). Without knowledge that constraint (1) was, perhaps, the most important, the system was forced to balance the ratio of groceries to bags with its expected confidence and appraisals that constraints (1) and (2) would or would not be met. Furthermore, for each grocery in which the system believed that the best course of action was separation (i.e., one grocery in one bag), the system necessarily had to create a bag with three groceries in order to maintain adequate performance on constraint (3) while still protecting the destroyable grocery.

Finally, for each experiment and condition that has been described in this section the system was allowed to perform a (somewhat) comprehensive search of the possible state space. This was done without concern for deliberation time. The experiment described in the following section will test how well the system performs the grocery-bagging task when it is must first appraise situations for deliberation time (i.e., urgency) and adjust its deliberation parameters

accordingly. In this experiment these appraisals are based on the system's ability to internally rehearse various situations during offline processing.

Experiment 3: Urgency Appraisal Learning Using Self-Guided Experience

Experiment Description

In this experiment, the system's ability to intelligently modify its search through the decision space is evaluated and tested. This includes the urgency appraisals that adaptively preset the search parameters *depth* and *breadth*. The generation of *interrupt* signals is evaluated during the experiment on the physical ISAC system. Therefore, this experiment tests the system's ability to generate useful performance profiles and to employ these profiles in future grocery-bagging situations.

The experimental procedure is the same as that used for Experiment 2, with the exception that a new step (process experience to learn urgency appraisals) has been added. Whereas the appraisals for *relevance* and *urgency* are trained after every 10 episodes, in this experiment the appraisal for urgency is trained every 100 episodes. The rationale for this discrepancy is the computation costs associated with internally rehearsing past experience to derive the performance profiles. At each training epoch only a subset of the possible episodes are used for training. In this experiment the system selects 10 states at random for every 100 episodes in episodic memory. This number was chosen because initial tests showed that examining 10% of past experience was typically sufficient to analyze performance without requiring extensive, lengthy deliberation.

It is important to recall from Chapter V, that during internal rehearsal each sampled state is analyzed for a variety of depth and breadth values. Thus each state is analyzed several times and if too many states are selected the internal rehearsal will be prohibitively slow; as shown in the following subsection deliberation time can be quite long for some states. During training, the system repeatedly analyzed the sampled state for all possible search depths (up to three), and for each 10% breadth (i.e., 10%, 20%, etc.). Here depth is taken to be the number of groceries bagged, therefore, for a search depth of three the system continues its search until three groceries have been bagged, or no more groceries can be bagged. Unlike depth, however, breadth is set as a percentage. Therefore, for a search breadth of 30% only the top 30% responses (i.e., best) are

kept at each depth, and the rest are pruned. The modified experimental procedure is listed as follows:

1. Train the system using the current experience in long-term memory.
2. If *num_episode* is a factor of 100, then select 0.1 * *num_episode* episodes and train the urgency appraisal maps for $d = \{1, 2, 3\}$ and $b = \{0.1, 0.2, 0.3, \ldots, 1.0\}$.
3. Select the number of groceries N uniformly from the range [5, 20].
4. Set the total number of groceries bagged, *total_count* = 0.
5. Select $0 < M \leq \min(6, N - \textit{total\_count})$ using a uniform distribution.
6. Select M groceries from *GrocerySet-A* or *GrocerySet-B* using a uniform distribution and then place those groceries on the conveyor.
7. Allow the system to bag each of the M groceries.
8. Set *total_count* = *total_count* + M.
9. If *total_count* < N return to Step (5), else go to Step (10).
10. Provide external feedback for the final situation.
11. Measure the error between the internal appraisals and the external feedback and record the number of constraint violations.
12. If *GrocerySet-A* was used, then add the new episode to long-term memory.
13. Set *num_episode* = *num_episode* + 1.0.
14. If *num_episode* is a factor of 10, add new episodes to long-term memory and go to Step (1), else go to Step (3).

Results and Discussion for Urgency Appraisals

Rather than re-test the system for all four conditions that have been used thus far, the urgency appraisals were evaluated for only the non-fixed weight, high action cost condition. The selection of this condition is based on the discussion from the end of Experiment 2 in which it was concluded that the non-fixed weights were necessary for generalization and that the high action cost was necessary for learning. Once the results are obtained for the urgency condition, those results will be compared to each of the four conditions from Experiment 2 in order to better understand the possible gains and risks in using internal rehearsal and urgency appraisals.

Figure 63 presents the averaged deliberation time for each of the four previous experimental conditions when tested using *GrocerySet-A*. This figure also presents the averaged deliberation time for the new urgency condition. As with Experiment 2, averaging is performed using a square filter of width 30 episodes, or three training epochs. In each of the four conditions from Experiment 2 the averaged deliberation time appears centered about a mean, but is also marked by very large oscillations that occur at random episodes. These oscillations are the result of the random nature of episode generation and the static nature of the planning algorithm. In none of the four conditions shown in Figures 63(a)-(d) is the mean deliberation time less than 30 seconds per episode, and it is not uncommon for deliberation to require more than 100 seconds per episode.

For the urgency condition, however, there is a significant decrease in the mean deliberation time over the entire experiment. For the first 100 episodes of this experiment, the system has not yet performed any internal rehearsal and Figure 63(e) shows that before the 100 episode mark, the averaged deliberation time is similar to that of other conditions. After the 100 episode mark, though, the urgency appraisals allow the system to modify its search and a subsequent decrease in deliberation time is observed. It should be noted that the urgency appraisals performed during this experiment, are only those appraisals related to maximizing the solution quality versus the deliberation time. The result of these appraisals is adaptive modification of the search parameters *depth* (*d*) and *breadth* (*b*).

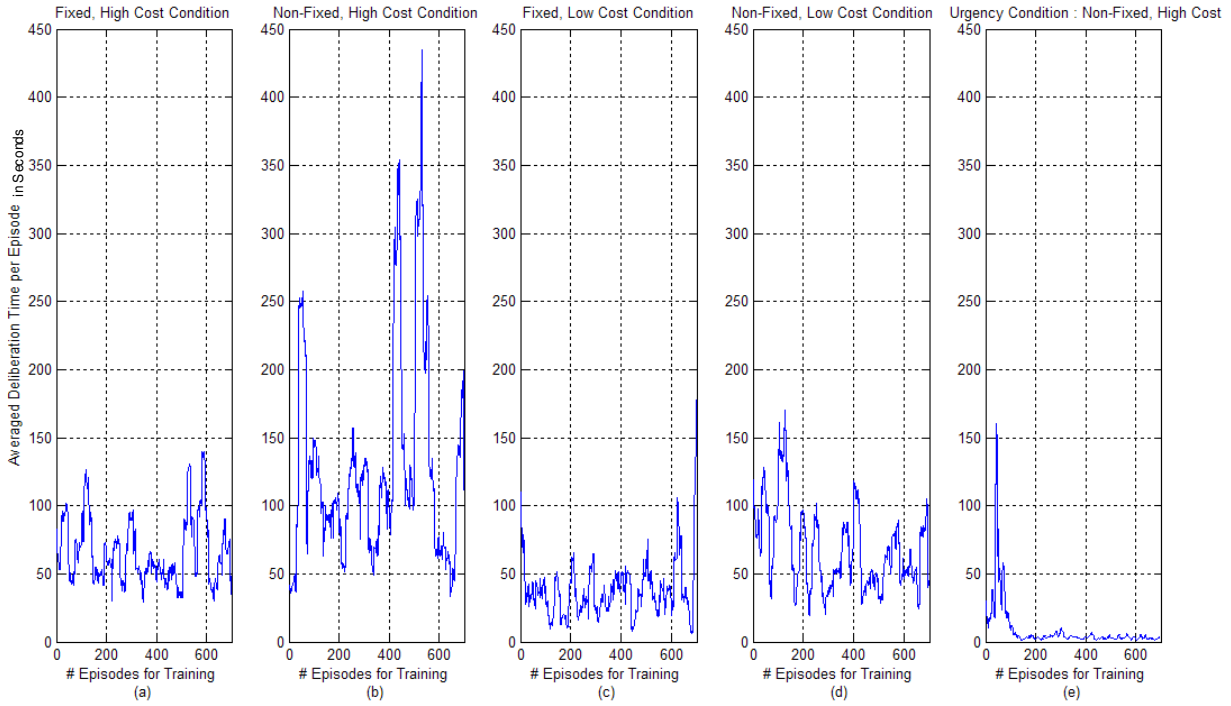


Figure 63. Comparisons of Deliberation Time Between the Four Domain Knowledge Conditions and the Urgency Condition

The adjustment of *d* and *b* is based on the performance profiles extracted by the system from the new internally-generated experience (i.e., the mentally simulation). As stated in Chapter V, the search algorithm that is implemented for this work is not a true anytime algorithm because final solution quality is not necessarily a strict, monotonically increasing function of time (or the amount of search). However, the search algorithm does approximate the anytime function with respect to the system's internal appraisals and knowledge representations, and this is what is necessary to perform internal rehearsal in the manner required by this dissertation. The claim that the search algorithm approximates the anytime property is supported experimentally by the results presented in Figure 64, which plots the learned performance profiles for each of the six internal rehearsal epochs. While there are some exceptions, Figure 64 shows that, in general, the solution quality with respect to the system's internal appraisals increases with both search depth and search breadth. Furthermore, the variance across these results can be explained by noting that:

1. The system does not rehearse the same states at each training epoch
2. The system can only use the most current knowledge, and that this knowledge is derived through unsupervised learning which necessarily injects some variability.

3. These plots, by design, only reflect the percentage of maximum solution quality achievable and not the actual maximum solution quality.

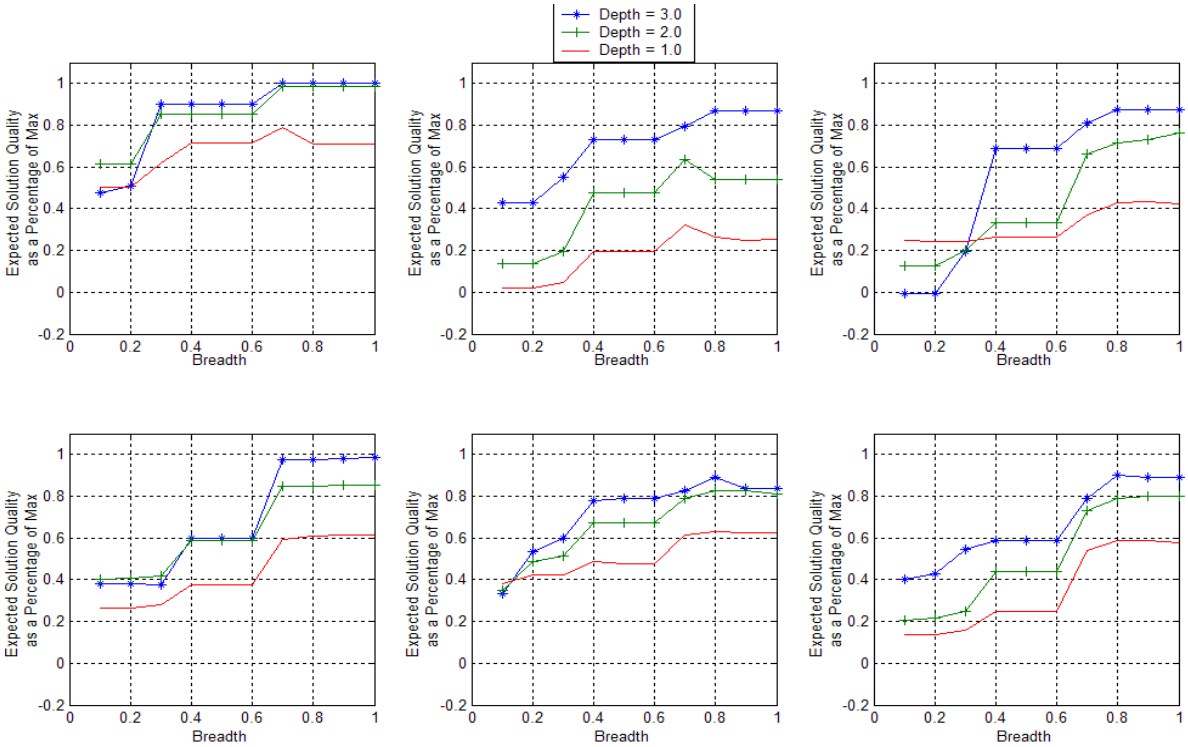


Figure 64. Learned Performance Profiles for Appraising Urgency and Adjusting the Search Parameters Depth and Breadth

As this experiment has thus far shown, the system is able to learn through internal rehearsal an urgency appraisal that allows adjustment of its cognitive response and deliberation time. However, it is important that the gains in deliberation time not come at the ultimate expense of solution quality. Due to the nature of the internal rehearsal approach used, which re-evaluates past experience using the latest learned knowledge, the learned performance profiles are biased towards being optimistic estimations of solution quality. This bias exists because the most current knowledge has, by design, been learned from the same experience that is later used as a baseline for internally generating new "practice" experiences. Furthermore, Figure 63(e) suggests that this bias *does* influence parameter setting to an extent, because the improvement in deliberation time is both very dramatic and immediate; hallmarks of an "optimistic" system. Such biased influence is only unwelcome, though, if it simultaneously sacrifices solution quality.

Figure 65 presents the performance evaluations for this experiment. Figure 65(a) and (b) show that the final error rate for both constraints (1) and (2) is actually *less* than that observed in the non-fixed, high cost condition of Experiment 2. Figure 65(c), however, shows that there is also a slight decrease in the ratio of groceries to bags, and thus performance is not better on every constraint. Further analysis of these results also shows that the rate of occurrence of bags with two or three groceries does not increase, as it has in earlier conditions, but that the error rate on these bag types does steadily decrease (Figure 66). Additionally, these results show that the system has a very difficult time avoiding errors on grocery bags with four groceries and that performance on bags of this type leaves much to be desired.

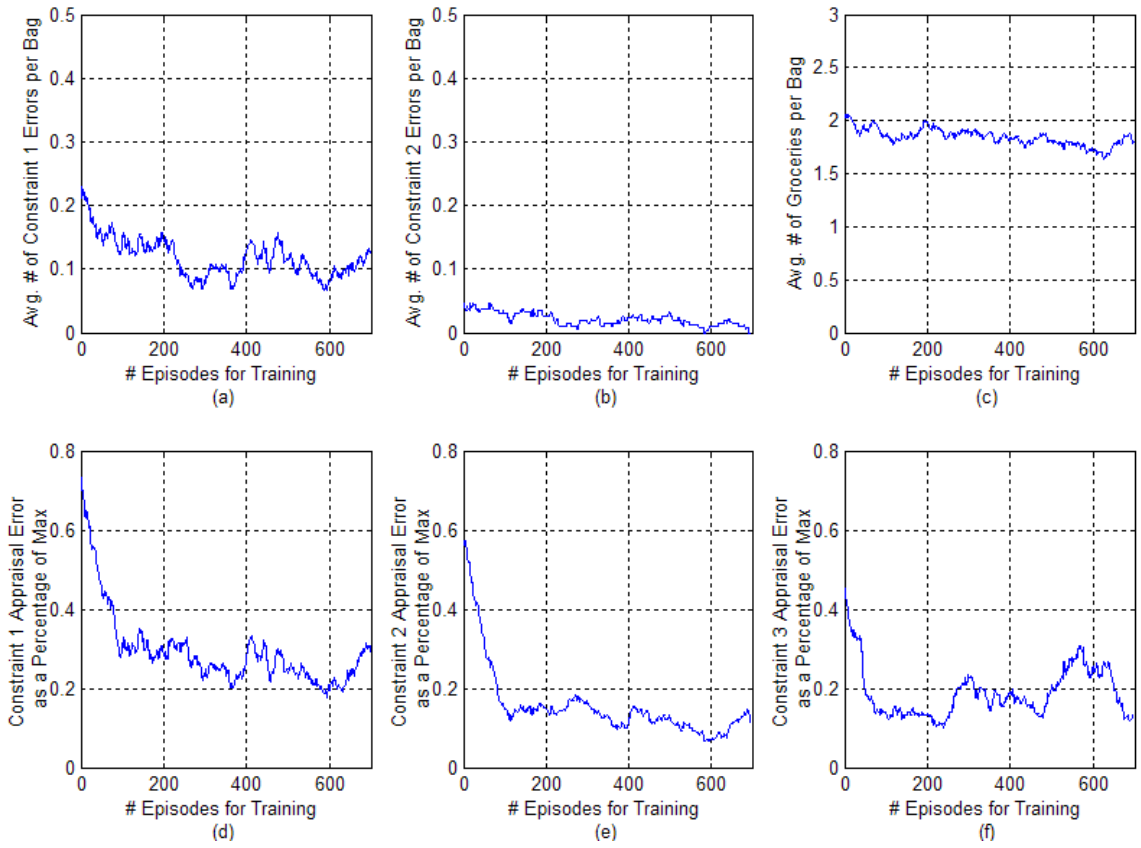


Figure 65. Evaluation Graphs for Episodes Generated with *GrocerySet-A* for the Non-Fixed Weight, High Cost, Urgency Condition

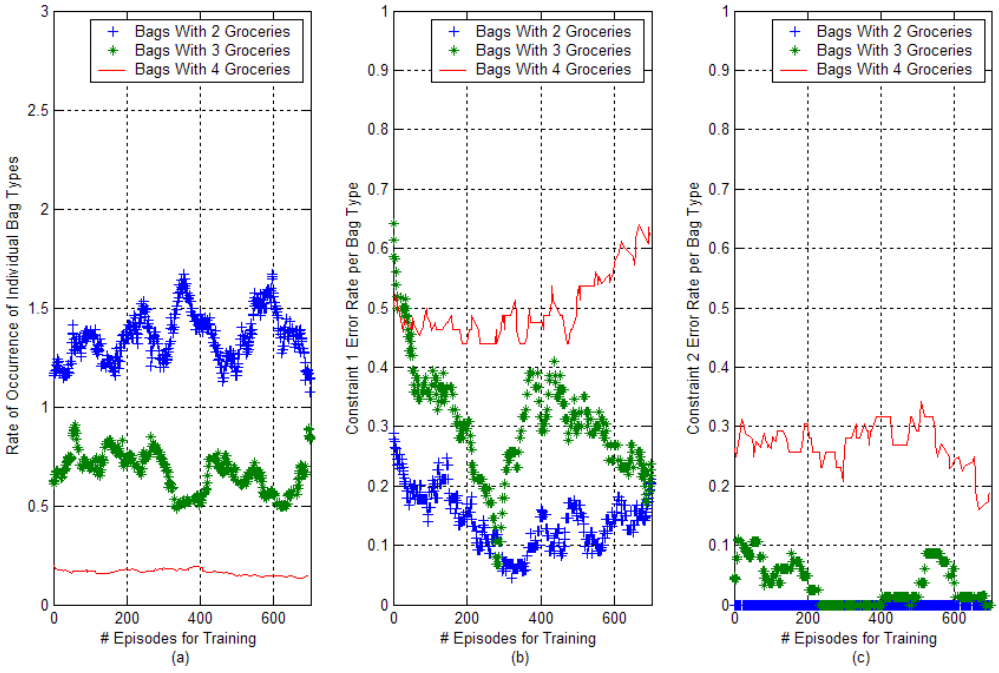


Figure 66. Breakdown of the Error Rate and Rate of Occurrence for Bags with Different Amounts of Groceries for the Non-Fixed Weight, High Cost, Urgency Condition

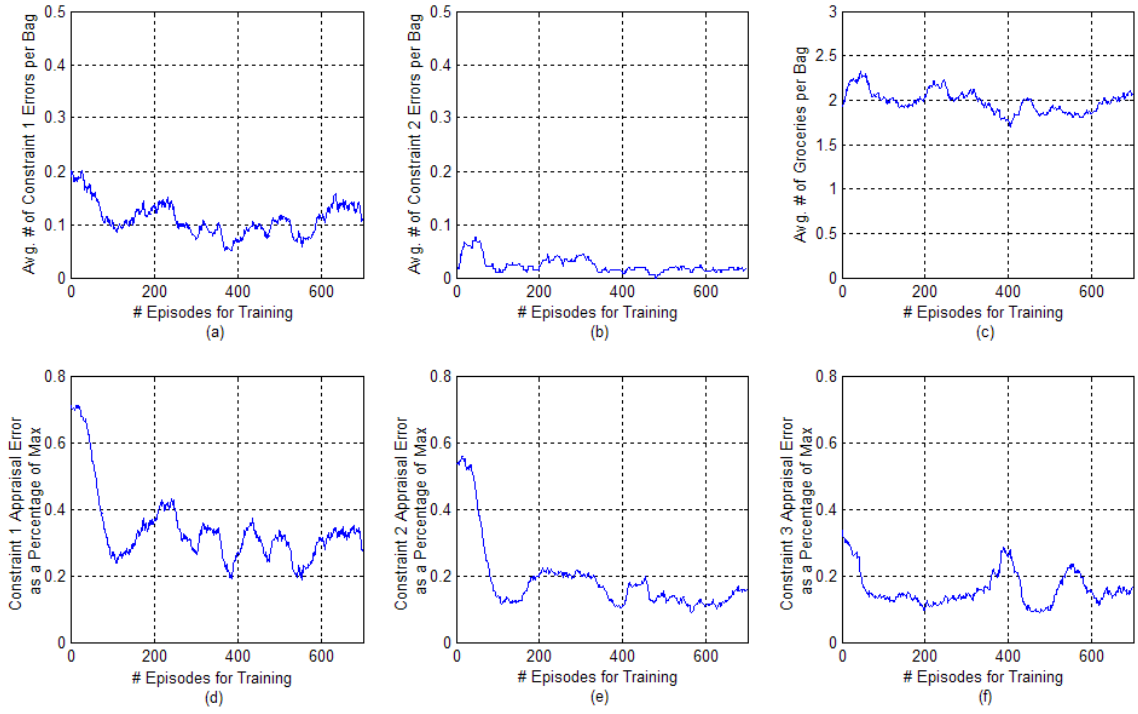


Figure 67. Evaluation Graphs for Episodes Generated with *GrocerySet-B* for the Non-Fixed Weight, High Cost, Urgency Condition

Figure 67 presents the results for the episodes generated using *GrocerySet-B*. As with the earlier non-fixed conditions the system exhibits the most improvement on its ability to correctly appraise situations with respect to constraints (1) and (2). However, this figure also indicates that there is a small improvement on the error rate for both constraints (1) and (2), and unlike the episodes generated with *GrocerySet-A*, the ratio of groceries to bags does not decrease. Figures 65-67 indicate that task performance does not suffer dramatically when the system is allowed to adjust its own search parameters in favor of quicker decision making.

Experiment 4: Fit Appraisal Learning Using Self-Guided Experience

Experiment Description

The final simulation experiment evaluated and tested the system's ability to monitor the levels of *fit* between its current knowledge and abilities and the task that is performed. While such knowledge could ultimately be used by the robot to mediate the cognitive cycle in a manner similar to that for urgency appraisals, the purpose in this research is simply to determine which knowledge structures are performing well and which require further training, or outside assistance. These appraisals can be viewed as assigning credit/blame when the robot succeeds or fails at the task; however, this is not merely a static process but is ongoing over a period of multiple episodes. Fit is only reset when the system retrains its knowledge structures.

For this experiment the fit appraisals were taken from the system trained during Experiment 3. In other words, the system was not retrained through an additional 700 episodes, but since the fit appraisals are simply measurements on the system and do not affect performance, these measurements were taken concurrently as the system was trained and evaluated in Experiment 3. The rational for choosing the urgency condition is based on the fact that the measured system had to appraise both relevance and utility, and therefore none of the earlier fixed conditions could be used. In addition, the system needed to learn well in order for fit to improve, and thus the low cost conditions were eliminated. These eliminations left the non-fixed, high cost condition as the one most suitable for this experiment. The urgency condition was used because the shortened deliberation time forced the system to make quicker decisions, which is more conducive to evaluating how well the system's knowledge fits the current situation.

As the system was evaluated and trained during Experiment 3, fit appraisals were made using the final state from each episode, the internal appraisals for this state, and the external reward given for that state. Using this information, the matrix E (Chapter V) was determined and used to update the vector $\boldsymbol{\phi}$. This was repeated for every episode and $\boldsymbol{\phi}$ was allowed to accumulate between training epochs. However, when the individual components were retrained, $\boldsymbol{\phi}$ was reset. The following subsection presents the final fit appraisals for each training epoch.

Results and Discussion

Figure 68 presents the fit values obtained as the system from Experiment 3 was trained and evaluated. As with the earlier conditions, these results have been filtered using a window size of three training epochs. Values of fit can range between 0 and 1, with 0 indicating good fit, and 1 indicating poor fit. However, with these appraisals the individual values are not as important as the order relative to each other as well as any trends present. These results indicate that as experience is acquired the fit between the system's knowledge and planning components and the new situations to which they are exposed becomes increasingly more accurate, but also that each component improves at its own rate.

Figure 68(a) presents the fit levels with respect to constraint (1) for the relevance and utility appraisals and the planning algorithm, i.e., $\phi_1{}^c$, $\phi_1{}^m$, $\phi_1{}^p$. These appraisals indicate that the components for relevance and utility improve the most and that a small, nearly constant amount of error can be attributed to the planning component. It should be noted that the planning component cannot improve its own performance and that the only aspect of its performance that changes with experience is the amount of search that it performs (and this is only the case in the urgency condition). This component is analyzed because as the system is required to balance multiple constraints this balance is ultimately achieved by the planning algorithm. It is possible that during deliberation, and in an attempt to balance all three constraints, the planning algorithm will sacrifice one constraint to preserve the other two. When this happens it is important to realize that the planning algorithm was the "culprit".

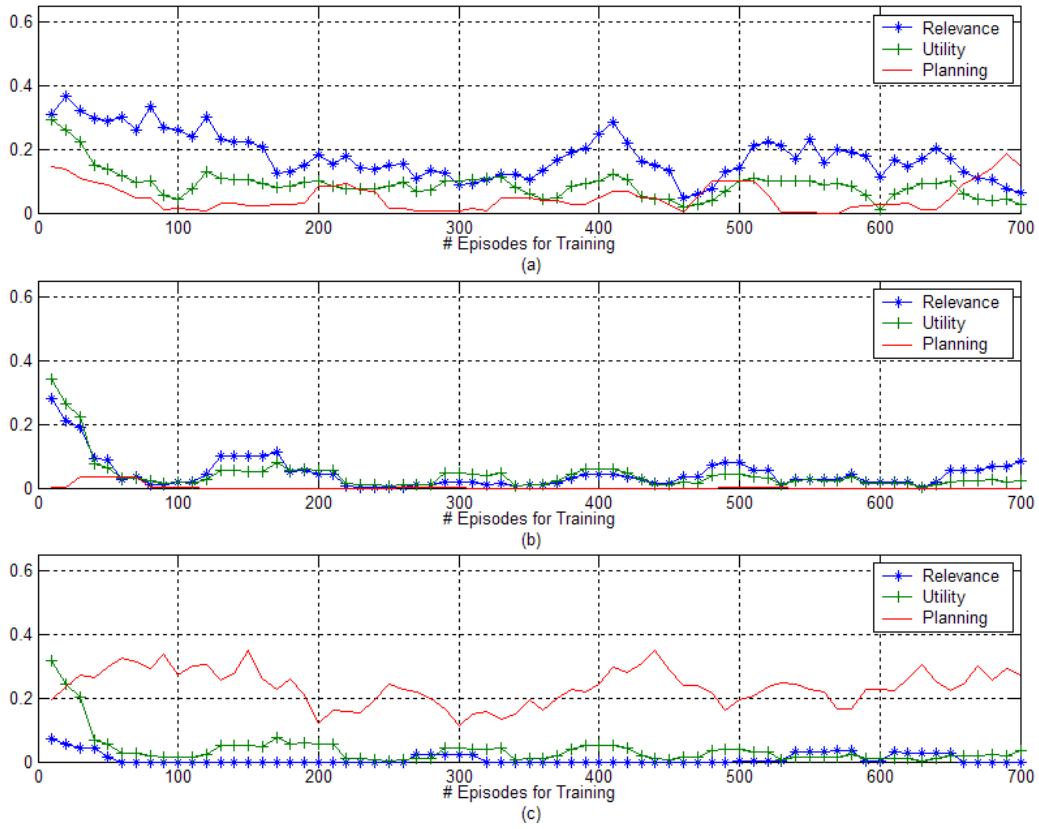


Figure 68. Fit Appraisals for Relevance, Utility, and Planning for Each Constraint and the Non-Fixed Weight, High Cost, Urgency Condition

Figure 68(a) shows that while the fit between the planning algorithm and success on constraint (1) remains constant, the planning algorithm is also the least responsible for errors on this constraint. The results indicate that even though the fit between both the relevance and utility appraisals and constraint (1) improves with training, these components deserve most of the blame for errors on this constraint, with the relevance appraisals receiving the lion's share. Analysis and verification requires evaluation of the individual episodes, the errors that were made, and the appraisals that caused those errors.

For this analysis the five training epochs 38, 39, 40, 41, and 42 were used. This window was chosen because it corresponds to a large "jump" in the fit appraisals for constraint (1). During these 50 episodes there was a combined total of 29 errors on constraint (1). What was analyzed with these errors was whether or not they could have been predicted, and if not – why? This analysis focused first on the individual groceries in each bag and the cluster to which those groceries belonged, in order to understand whether the grocery classifications caused the error.

To do this the groceries that caused each error were extracted and the clusters to which they belonged were analyzed to determine whether the remaining members of those clusters would also have caused the error, or was the error caused by an incorrect assignment. This is the same type of analysis that led to the conclusion in the fixed weight, high cost condition from Experiment 2 that the fixed weights forced groceries from *GrocerySet-B* into clusters when they did not obey the relationship properties of those clusters.

For the 29 errors, 10 were identified as having been caused by incorrect clustering. For these 10 errors, the groceries, classifications, and "cluster mix ratio" are listed in Table 32. The cluster mix ratio is a measure of how many pairs of groceries (one from each cluster) obey the same relationship as those two groceries that caused the error. For example, given two clusters, each with ten groceries, if nine of the groceries in the first cluster can successfully be mixed (i.e., bagged) with all ten groceries from the second cluster then the cluster mix ratio for success is 90%. The remaining grocery in the first cluster does not obey this relationship and would cause errors during task performance if the relational maps successfully learned the "mix is okay" relationship.

Table 32. Constraint (1) Errors as a Result of Misclassification

Grocery 1	Grocery 2	Class 1	Class 2	Cluster Mix Ratio
eggs	oranges	C_4	C_8	16.7%
eggs	oranges	C_4	C_8	16.7%
eggs	hot_soup	C_5	C_0	12.5%
hot_soup	ice_cream	C_0	C_{13}	15.0%
eggs	tuna	C_6	C_6	22.2%
milk	eggs	C_0	C_6	16.7%
hot_soup	yogurt	C_0	C_6	20.0%
milk	eggs	C_8	C_6	20.0%
eggs	yogurt	C_6	C_6	32.0%
ice_cream	hot_soup	C_0	C_6	20.0%

Table 32 shows an interesting trend: 100% of these errors involved either *hot_soup* or *eggs*. Further analysis revealed that when these two groceries were treated correctly, they were placed in their own category (sometimes *eggs* was grouped with *strawberries*). When this did not happen, however, these groceries were placed in a category in which they were the outlier. *Eggs* were often placed with other "cold" items, yet many of these cold items could be bagged with heavy items, such as *oranges*, while *eggs* could not. *Hot_soup* was often placed in a large cluster

with *spaghetti*, *green_beans*, *ziploc_bags*, *granola*, and sometimes *tuna*, but unlike these other groceries *hot_soup* could not be bagged with cold items, such as *ice_cream*.

Of the remaining 19 errors, six involved bags with four or more groceries and a confidence measure less than 50%. For these bags, the relational map was to blame either because it did not have enough experience, or because it did not correctly generalize the experience it had. Because the number of groceries in these bags was so high, however, the risk associated with these bags kept these errors from substantially modifying fit. In addition, six of the 29 errors were caused by the planning algorithm. Analysis of these bags revealed that the relational maps had given appraisals that were, at least approximately, correct but that the planning algorithm still chose an action that led to that error. Here "approximately correct" means that the appraisals had the correct sign and was with 0.5 of the true value. Thus these errors could be attributed to either a lack of search, the high action cost, or "fear" of a worse alternative. The final seven errors were attributed to the relational map because it failed to identify the relationship that would have predicted task success. Each of these bags, however, had three groceries which meant that the target relationship was more complex than in the simple two grocery case.

Figure 68(b) shows that the level of fit, with respect to constraint (2), between the same three components and the grocery-bagging task was much better. This should be expected, however, because the rate of constraint (2) errors was much lower and therefore the system *should* appraise fit better. Another conclusion that can be drawn from Figure 68(b) is that when errors do occur, the blame is almost equally divided amongst all of the components.

Figure 68(c) is intriguing. This figure shows that while the level of fit between the relevance and utility appraisals and the performance on constraint (3) improves with training, the fit between the planning algorithm and constraint (3) not only remains constant, but also that the planning algorithm deserves almost all of the blame for errors on this constraint. To understand why this is the case, it should be recalled that constraint (3) is evaluated with a fuzzy rule that is capable of returning values in the range $[\kappa_1, \kappa_2)$, where κ_1 and κ_2 are parameters set by the system designer; in this experiment they are set to 1.5. Due to the fuzzy nature of this rule, many of the values provided by the external critic are actually much less than these limits. Analysis revealed that it is not uncommon for the system to receive values of ± 0.5. Because of this, once the system learned to appraise all three constraints constraint (3) implicitly became "less

important" in most situations. Therefore, when choosing actions, the planning algorithm would be more likely to pursue paths in which appraisals for constraint (3) have a lower value because these appraisals are simply outweighed by the ±1.0 values returned by the other utility appraisals. This also provides more explanation as to why there was no substantial increase in the ratio of groceries to bags in the earlier experiments.

ISAC Integrated Experiments

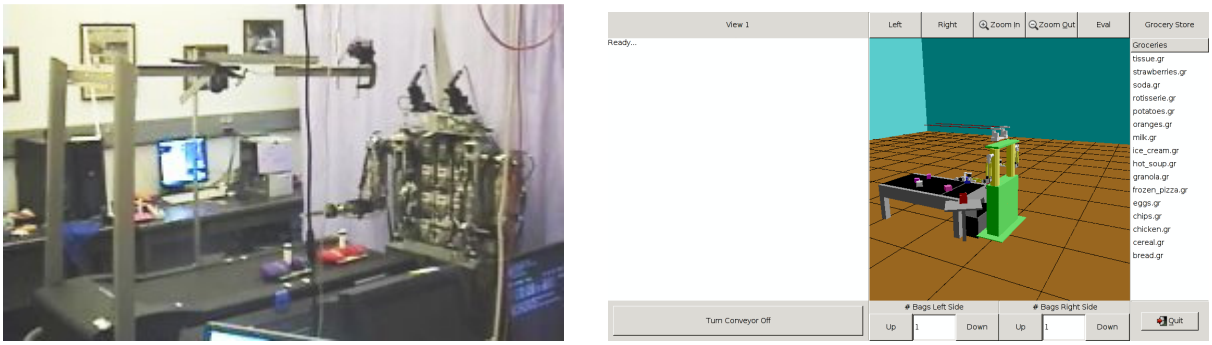
Experimental Design

The previous simulation-based experiments have focused on running multiple trials with large amounts of experience and numerous conditions. Through these experiments, the system has shown the ability to appraise relevance and identify useful concepts for goal accomplishment. The system has also learned to appraise utility along the various dimensions required for bagging groceries, and the system has shown that it can use internal rehearsal to evaluate its ability for the purpose of developing appreciations of itself. These appreciations were used to inform deliberation by mediating the cognitive cycle, or to identify for a human user which components require further training or assistance. In addition, the simulation experiments used an extended behavioral repertoire that included the *Wait(g)* behavior, which is not available with the physical robot. In effect, this behavior enabled the simulated system to select the desired order in which groceries should be bagged by allowing the system to bypass those groceries on the conveyor belt that *are* within reach in favor of groceries that *are not* yet in reach.

By including the extra *Wait(g)* behavior, the simulated grocery-bagging task was more complex with respect to planning. This is due to the fact that this behavior introduced additional options that then had to be appraised, weighed, and considered. Both the simulated and hardware experiments required that the system cope with groceries at the front of the conveyor belt, but the simulated system could also plan for groceries at the end of the belt as well. Therefore, ISAC had to be deal with groceries in the sequential order in which they appeared and this, simultaneously, limited the behaviors available to ISAC and eased the computational requirements associated with planning.

The interaction of a humanoid robot with the real world critically depends on the robot's morphology and on its environment. Therefore, simulation is only one aspect of system validation. The current experiments aimed to evaluate the integration of the cognitive control

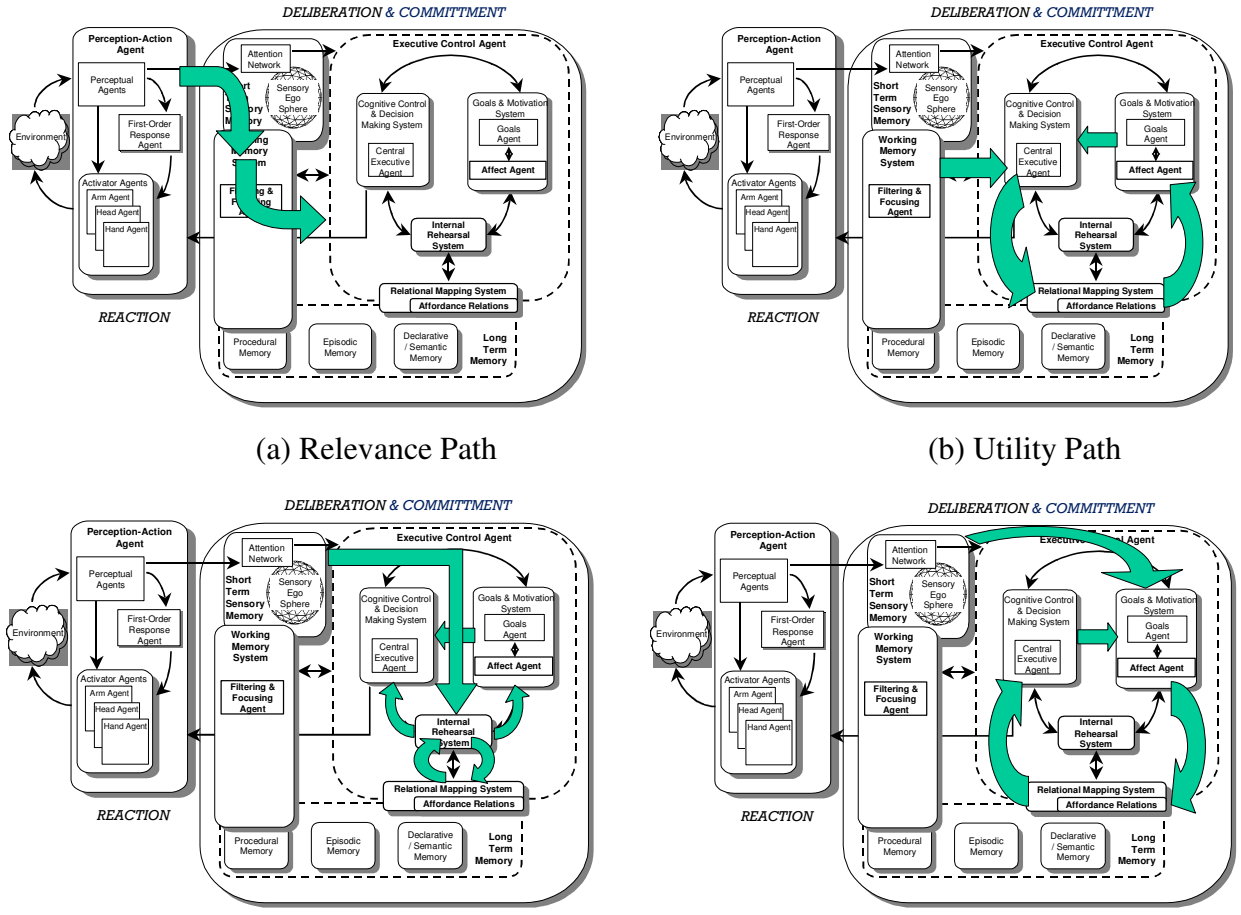
system described in Chapter V, and conceptually validated in Experiments 1-4, with the ISAC hardware system and peripheral software components. In particular, the objective is to integrate the designed control system with ISAC's Perceptual Agents and Activator Agents in order to complete the cognitive control process. Figure 69 shows the combined ISAC-Simulator integrated environment used.



(a) (b)

Figure 69. (a) ISAC Hardware System, and (b) Simulator Environment

To investigate system integration, each of the components described in Chapter V and documented in Appendex B (dynamic representations, relational maps, urgency appraisals, and fit evaluations) were used for these experiments. The operation of these components within the ISAC architecture can be visualized as a set of control paths through the cognitive architecture, as shown in Figure 70. Within the ISAC architecture, appraising relevance occurs along the path shown in Figure 70(a). This path should be considered a "preprocessing" step that filters and focuses incoming stimuli into a set of task-relevant categories that can then be passed to the deliberative control loop. Appraising utility occurs along the path shown in Figure 70(b), and ends in the *Central Executive Agent* (CEA) which is in charge of comparing options and making the final decision. Urgency appraisals occur along the path shown in Figure 70(c) and are used to inform both the CEA and the Affect Agent. While utility and urgency appraisals ultimately end in the CEA, fit appraisals (Figure 70d) end in the *Goals and Motivation System* due to the fact that these appraisals are only used as *post hoc* processing and evaluative measures.



(a) Relevance Path (b) Utility Path

(c) Urgency Path (b) Fit Path

Figure 70. Significant Paths within the ISAC Architecture for (a) Relevance, (b) Utility, (c) Urgency, and (d) Fit Appraisals

Integrating the designed system with the *Perceptual Agents* requires replacing the simulated perceptual input with input from the SES. Because the system was originally designed to operate on the symbolic output of the SES, this integration was a straightforward operation in which the data on the SES was accessed rather than the data stored in the simulated system. This can be conceptualized by the function:

```
GetPercepts(SState s, bool simulation)
{
      if(!simulation)
            ses.Get(s.percepts( ));
      else
            simulator.Get(s.percepts( ));
}
```

in which the flag *simulation* indicates whether the experiment is using real or simulated data, and the struct *s* holds the current state (see Appendix A for a description of *SState*).

Once the perceptual information has been received from the SES, it can be sent in two directions. The first direction involves sending that information to the Working Memory System (WMS) where it is filtered and used to create the necessary feature vectors that will later be used to access the relational maps. As described in Chapter V, in this work the WMS was not provided *a priori* with information indicating how percepts should be categorized into goal-relevant chunks. Therefore, the component for developing dynamic representations had to use the learned weights to create the goal relevant chunks. This can be seen as an addition to the Working Memory toolkit (WMtk) described in Chapter V. In the WMtk, chunks were preset by the system designer; here the chunks are learned from experience. Example code for this operation is given as follows:

```
//retrieve goal-relevant weights
double weights[9] = {0};
dynamic_representation.Retrieve(weights);

//group percepts into goal-relevant categories
dynamic_representation.Compress(s.percepts, weights);

//Fill in feature vectors with percepts
std::vector<SFeatureVector> fv;
dynamic_representation.InitializeFeatureVectors(fv, s.percepts);
s.fv = fv;
```

The second direction involves sending the perceptual information directly to the IRS and Goals and Motivation System. This enables the identification of interrupts and the development of fit appraisals. In both cases, only specific perceptual information is monitored. The IRS monitors grocery positions, and the Goals and Motivation System monitors external feedback. Example code to implement these operations is as follows:

```
std::vector<double> positions[3];
std::vector<SEvaluation> evaluation;
for(int i=0;i<s.percepts.size( );i++)
{
      if(s.percepts.at(i).identificationType == m_GrocID)
      {
            positions[0].push_back(s.percepts.xpos);
            positions[1].push_back(s.percepts.ypos);
            positions[2].push_back(s.percepts.zpos);
      }
      else if(s.percepts.at(i).identificationType == m_ExtID)
```

```
        {
                SEvaluation eval;
                eval.fv = s.percepts.GetFeatureVector( );
                eval.r = s.percepts.GetReward( );
                evaluation.push_back(eval);
        }
}

//IRS
bool interrupt;
irs.GetInterrupt(positions, interrupt);

//Goals and Motivation System
gms.AppraiseFit(evaluation);
```

The feature vectors are used to retrieve internal utility appraisals and to facilitate planning within the CEA. This process involves accessing the relational maps and passing the retrieved information through the Affect Agent back to the CEA. Once this has been accomplished the CEA can continue to search through the decision space, or can return the current best policy. The actual code to perform this recursive search is too long to present here, but is best summarized using the pseudo-code presented in Figure 25, in which all possible actions *A* that may be performed in a given state are used to create the set of possible states *S* that may be reached from the current state.

Once the policy has been developed, a behavior must be selected and executed. In this dissertation, the term behavior is used to indicate the action/motion that ISAC physically performs. In Chapter V and in the simulation experiments, behaviors have also been referred to as actions, but for the current experiment, once an action has been selected it is referred to as a behavior to maintain consistency with previous work on ISAC. In addition, some behaviors may be composed of sub-actions as described in the section Behavioral Repertoire. The lowest level of sub-action is referred to as an atomic action. As described earlier, this hierarchy exists to reduce the computational demands on planning and decision making.

The selection process requires choosing the behavior for the current state, but the execution process requires integration with the Activator Agents. This integration requires two steps: 1) parsing the desired behaviors into the properly sequenced atomic actions, and 2) sending the parsed action commands to the agents in charge of the physical hardware rather than to the simulated agents. The second step mirrors the straightforward integration between the

control system and the Perceptual Agents and can be conceptualized by the simple function *SendAction( )*:

```
SendAction(CAction action, bool simulation)
{
      if(!simulation)
            activatorAgents.Put(action);
      else
            simulator.Put(action);
}
```

The first step requires the use of procedural knowledge for each behavior, in particular the preconditions and postconditions for each behavior. This step also requires feedback from the Activator Agents in order to determine when the next behavior should begin. An example is the behavior *BagGroceryRight(b, g)* which, as described earlier, could be sequentially composed of multiple simpler behaviors that for the sake of deliberation time have been hidden from the planning and decision-making algorithms. Control of this behavior must appreciate that the order in which this sequence is executed is critical (i.e., ISAC should not reach for the bag before grasping the grocery) and that one sub-action cannot be initiated until the previous sub-action has finished. Therefore control of this behavior should be performed using a function similar to:

```
PerformBehavior(CAction action, bool sim)
{
      std::vector<CAction> action_seq;
      //Get the sequence of actions for the current behavior
      m_ProceduralMemory.RetrieveActionSequence(action, action_seq);
      //Execute each action in sequence
      for(int i=0;i<action_seq.size( );i++)
      {
            SendAction(action_seq.at(i), sim);
            int outcome = SUCCESS;
            while(!Response(action_seq.at(i), sim, outcome))
            {}
            if(outcome != SUCCESS)
                  break;
      }
}
```

where the *Response( )* function awaits confirmation from the Activator Agents that the sub-action has been completed. Information related to the success/failure of the behavior is stored in the variable *outcome* and can be used to stop the behavior at any point during execution if the previous action was unsuccessful.

Communication between the Perceptual Agents, Activator Agents, and the components housed in the Deliberation & Commitment block was handled using standard TCP/IP sockets, as shown in Figure 71. The communication between the Perceptual Agents and the Activator Agents and the hardware was handled using RS232 serial connections or PCI ports. Once information had been passed to the computer running the remaining software components (i.e., Deliberation & Commitment block), inter-computer communication was no longer an issue as these components were designed as object-oriented classes and were running within the same *main( )* loop.

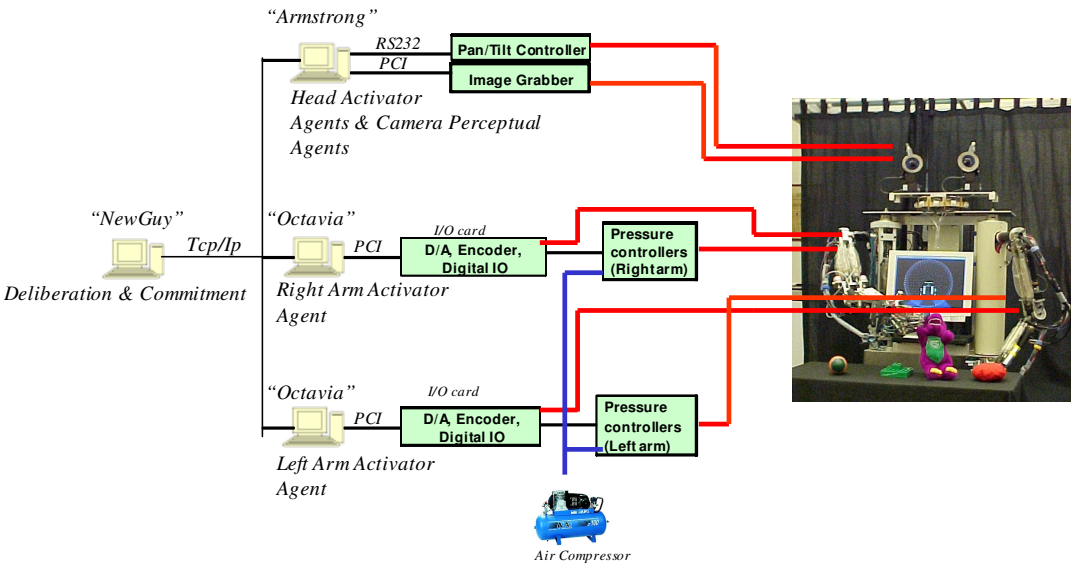


Figure 71. System Connections for the Integrated ISAC Experiments

The final integration step for these experiments was the most difficult to perform, and was important for proper, low-level (i.e., end-effector position) control. This was an integration step that did not involve any of the components specifically designed for this work. Rather, this step involved directly integrating the Perceptual and Activator Agents in order to perform the behaviors necessary to grasp items on the conveyor belt. Currently, there are two basic methods for executing arm behaviors on ISAC:

1. Joint angle control based on inverse kinematics. In this case, a desired end-effector position and orientation is input to the system, and inverse kinematics are used to determine the appropriate joint angles for this pose. The arm can be driven directly to this point, or can traverse a pre-defined set of waypoints before reaching the final position.

2. Joint angle control based on pre-recorded files. In this case, the arm is moved through a pre-defined set of points. These pre-defined points may be either previously recorded motions, or new interpolated motions, but in either scenario the motion is completely defined before ISAC begins execution.

Figure 72 illustrates a user-interface (UI) developed to perform basic arm control. This user interface is based on a neural network controller developed by Ulutas, et al., [2008].

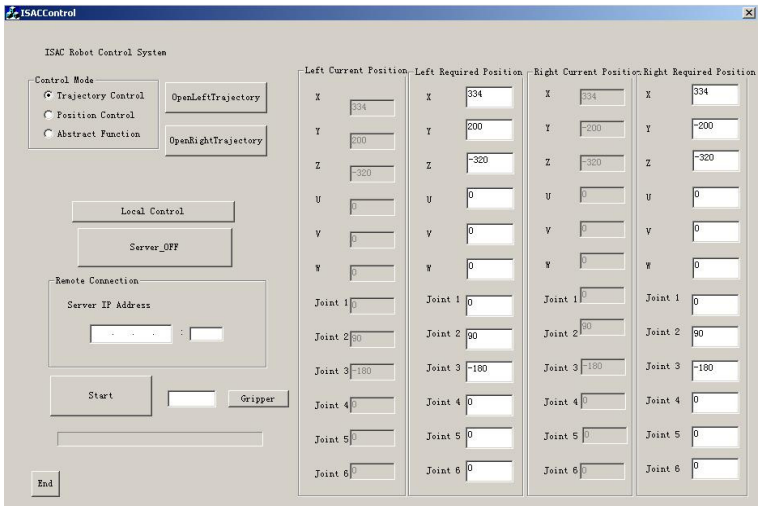


Figure 72. User-Interface for ISAC Arm Control

Each method relies on joint angle information, but differs in the manner in which the joint angles are determined. In the first case, inverse kinematics are used to derive the joint angles from desired end-effector cartesian coordinates, while in the second case the joint angles are loaded directly from a file. Grippers can be used with both control methods. In the first method, the gripper should be operated (i.e., opened/closed) once the arm has reached the desired final position. The second method, however, requires that a pause be inserted into the recorded motion file so that during this pause the grippers may be opened or closed as necessary. Gripper control was enabled by using the pressure control from ISAC's outermost "tricep" muscle. This muscle had been previously disabled, as it was not necessary for control. (NOTE: This is because the downward motion powered by the triceps was already assisted by gravity and did not require flexing of the triceps but rather merely the controlled relaxing of the biceps). Therefore, the Arm Agents were used to control the grippers, and this control function is described in Appendix B.

Without direct integration between the Perceptual and Activator Agents the coordinates used by either of the described methods may not reflect the most recent, or most accurate, information regarding desired end-effector location. An ideal solution to this problem would be the implementation of visual-servoing control [Taylor and Kleeman, 2001] [Hosoda and Asada, 1994], however, such control was beyond the scope of this dissertation. A less ideal, but more practical solution, involves combining the percept's perceived location with the measured position of ISAC's end-effectors. Unfortunately, there is not a linear transformation between these two coordinate frames because of the different sensors used (cameras and joint encoders), and thus this technique is associated with a certain amount of error that is a nonlinear function of object location, color, size, orientation, lighting conditions, and camera angles. Therefore, rather than attempt to solve these problems within this dissertation, pre-recorded (open-loop) motions were used to demonstrate the integration between the designed control system and ISAC's Activator Agents. The integration between ISAC's Perceptual and Activator Agents was reserved for future work.

Finally with respect to system integration, only a subset of ISAC's information control pathways was used. Examples of these pathways are provided in Figure 70, but are also described in Chapter V. The primary control pathways that were not used were those associated with ISAC's reactive and routine (1^{st} order response) systems. For these experiments, it was desired to isolate ISAC's cognitive control processes (i.e., working memory, central executive, internal rehearsal, and long-term memory) and therefore the reactive and routine components were omitted.

Once integration was complete, the components were connected as described in Chapter V (and shown in Figures 14 and 26). The inputs/outputs for each component were unchanged, with the exception that the percepts retrieved from the SES were used as the perceptual inputs rather than the percepts present on the simulated conveyor belt, and that the behavior outputs were parsed and sent to the Activator Agents rather than to the simulated ISAC system. The code to implement each component is summarized and described in Appendix B.

ISAC Experiment 1: Integration of Knowledge and Processes Developed in Simulation

The objective of this experiment was to evaluate how well ISAC could deploy the knowledge learned in simulation, and related to the appraisals for *relevance*, *utility*, *urgency*, and

fit, to the grocery-bagging task. Complete system integration was required with the exception that a pre-recorded motion was used to for the executed behavior. This motion involved reaching across the conveyor belt, closing the gripper, moving the gripper to a preset location, opening the gripper, and then returning the arm to the home position. This motion was recorded through manual teleoperation on ISAC, and the six individual joint angles were stored in an ASCII character file labeled "motion.txt". The code method to capture motions is shown in Appendix B. This motion was executed in lieu of the complex, closed-loop pick and place behavior required to place groceries in individual bags. Because of the pre-programmed nature of this motion, once the behavior was completed the planning algorithms were instructed to update the current state "as if" the motion had been completed correctly. If the grocery was still on the conveyor belt, then the grocery was removed manually from ISAC's visual space, so that the task could continue. This insured that ISAC had to continually plan for more complex situations, by updating bags regardless of behavior outcome. Because this experiment required the use of the same external critic that was used in the simulated Experiments 1-4, all of the knowledge learned in simulation was applied to this task without additional processing. However, because this experiment used the actual conveyor belt, it was necessary for ISAC to develop additional appreciations for *how long* it could deliberate before a decision must be made. This was done using the Bayesian network approach described in Chapter V. The significant events were defined to be those states in which groceries were located at the edge of the conveyor belt, i.e., about to fall off of the conveyor belt.

For this experiment the conveyor belt was divided into 14 evenly distributed bins, each 10" long, and 10" wide. The rational for these settings was to keep bin size uniform. The dimensions of the conveyor were 52" x 20", however, the cameras had an effective viewing area ~70" long at the height of the conveyor. The representation of bins is shown in Figure 73. Bin positions 7 and 14 corresponded with the end of the conveyor belt (and the floor) for the left and right sides of the conveyor belt, respectively. The speed of the conveyor belt was set at the lowest possible setting, which was approximately 0.1 miles per hour. This setting was used to ease the computational pressure placed on ISAC's perception algorithms, as robust perception was not within the scope of this dissertation. As in Chapter V, a sampling rate of one second was used to measure grocery positions, and the resulting transition model was obtained (Figure 74).

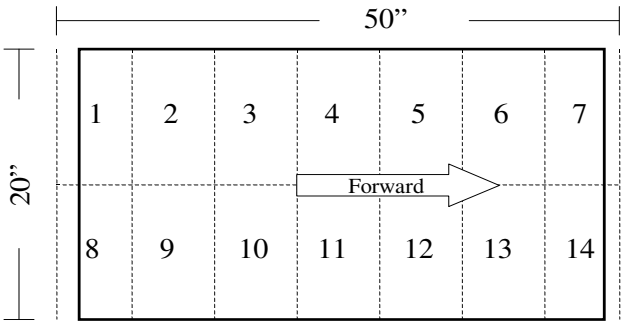


Figure 73. Dimensions and Bin Distribution for Conveyor Belt

Bin_{i-1}	Bin_i 1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	**0.500**	**0.500**	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	0.000	**0.615**	**0.385**	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
3	0.000	0.000	**0.500**	**0.500**	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4	0.000	0.000	**0.029**	**0.714**	**0.200**	0.000	**0.029**	0.000	0.000	0.000	**0.029**	0.000	0.000	0.000
5	0.000	0.000	0.000	**0.037**	**0.630**	**0.333**	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
6	0.000	0.000	0.000	0.000	0.000	**0.600**	**0.400**	0.000	0.000	0.000	0.000	0.000	0.000	0.000
7	**0.074**	0.000	0.000	**0.037**	0.000	**0.037**	**0.704**	0.000	0.000	0.000	0.000	0.000	0.000	**0.111**
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**0.667**	**0.333**	0.000	0.000	0.000	0.000	0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**0.556**	0.000	**0.444**	0.000	0.000	0.000
10	0.000	0.000	0.000	**0.043**	0.000	0.000	0.000	0.000	**0.043**	**0.652**	**0.261**	0.000	0.000	0.000
11	0.000	0.000	0.000	**0.033**	**0.100**	0.000	0.000	0.000	0.000	0.000	**0.733**	**0.100**	**0.033**	0.000
12	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**0.250**	**0.250**	**0.500**	0.000
13	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**0.750**	**0.250**
14	0.000	0.000	0.000	0.000	0.000	0.000	**0.050**	0.000	0.000	0.000	0.000	0.000	0.000	**0.950**

Figure 74. Learned Transition Model, i.e., $P(Bin_i \mid Bin_{i-1})$ Using Hardware Obtained Observations

Figure 74 indicates that there is much more noise in the measurements taken by the physical system than the ideal measurements used in Chapter V. This is due to residual noise in the color tracking algorithms and errors in the frame capture for the individual USB webcams. The color tracking algorithms employed simple blob detection to identify target colors and worked very well *when* the color to be tracked was present in input image. However, if the target color was not present, these active vision algorithms would occasionally identify background noise as the target color. This was only a problem when the percepts were not actually present, but during these situations caused percepts to randomly appear in the input image. A preset threshold was used to reduce this error by forcing the algorithms to detect a minimum number of pixels for the target color, and while this worked well it could not completely remove all noise

effects. Errors in frame capture were the result of a random latency with the USB device drivers. All efforts were made to remove this problem, but it was still possible for the USB cameras to "freeze" for 1-2 seconds. The investigation of this phenomenon identified the device driver software as the source of the problem, but was unable to fix the problem. Because such noise could potentially affect the urgency appraisals, ISAC continuously monitor the conveyor belt while deliberating, and based interrupt generation on the "worst-case scenario".

During prediction, a preset threshold of 80% was used to signify that a grocery was "confidently" believed to be in specific bin. This threshold was chosen because empirical tests suggested 80% was a sufficient value for prediction and such a high value forced ISAC to only rely on short-term predictions (i.e., 1-2 bins in the future). If the Bayesian networks predicted that a grocery had an 80% chance of being in either of the next to last bins (i.e., just before the edge of the conveyor) an interrupt signal was generated, deliberation was stopped, and the behavior began. Because a pre-recorded motion was used for the behavior, the conveyor belt was stopped so that this behavior could be performed on a stationary grocery. This also prevented new groceries from entering ISAC's workspace (and potentially falling off of the conveyor) while the pre-recorded behavior was being performed. The conveyor, however, was *only* stopped after deliberation had been completed (or interrupted), and ISAC was not allowed to perform any additional planning during this time period. In other words, during behavior execution (and *only* during action execution), ISAC's cognitive processes were halted while the Activator Agents executed the desired behavior. This ensured that while ISAC's cognitive control algorithms had to cope with changing environments (i.e., conveyor belt on), the Activator Agents could execute a pre-recorded motion on a stationary environment. The interrupt process generated by the detection of significant events follows the control path illustrated in Figure 69(c).

Experimental Procedure

The experimental procedure for this experiment is listed as follows:

1. ISAC is allowed to observe the motion of 15 groceries on the conveyor belt in order to train the Bayesian networks. 15 groceries were chosen because no significant changes were observed in the Bayesian networks after this amount of training. The Bayesian networks were trained using the discretized positions of the individual groceries on the

conveyor belt. This does not require action, but merely observation and then training on these observations.

2. *N* groceries are selected uniformly from the range [5, 10] from the combined grocery set presented in Table 14. This range was chosen to ensure that ISAC had to cope with a minimum number of groceries, while keeping expected completion time for episodes within a reasonable time limit. Furthermore, accrued encoder error limited the number of times ISAC could successfully and *safely* perform the pre-recorded motion.
3. Groceries were continuously placed on the conveyor belt at regular intervals so that no more than three groceries were present at any one time. This was repeated until all groceries had been presented to ISAC. The size of the conveyor belt dictated that no more than three groceries be present at any one time. An example is shown in Figure 75.
4. Using knowledge learned from simulation about relevance, utility, urgency, and fit, as well as the Bayesian networks trained at the beginning of the experiment, ISAC develops a policy for the perceived state space. This knowledge was represented by the individual weights learned for perceptual attributes, the grocery clusters developed by these weights, the trained relational maps, and the stored performance profiles, and was taken directly from training epochs in Experiment 3. This is described further below. The input to the Bayesian networks was the perceived location of individual groceries on the conveyor belt. The policy associated a specific action with each individual state encountered during the planning process. The action stored by the policy was the action considered to be best, given the current search.
5. Once deliberation is finished, ISAC bags the groceries using the developed policy by sending a flag to the Activator Agents that the pre-recorded motion should be executed. This was done using TCP/IP, as shown in Figure 71.
6. Just as in the simulation experiments, ISAC is provided with external feedback from the critic related to evaluations of constraints (1), (2), and (3) for specific situation components (i.e., bags and collections bags).
7. Steps (2) – (5) are repeated for 20 episodes as described below.

In addition to these experimental steps, the *Wait(g)* behavior from Table 17 was removed from the list of possible behaviors. This was because the physical conveyor belt did not have a

collection bin, and therefore ISAC could not choose to wait for the next grocery in lieu of bagging the nearest grocery. This ensured that groceries did not fall off of the conveyor belt because ISAC had chosen to wait and bag another grocery first.

Because this experiment required using all of the learned knowledge structures and components, i.e., attribute weights, conceptual clustering, relational maps, performance profiles, and Bayesian networks, how these components were connected was critical for experimental validity. As percepts were acquired (from the SES) they were used to fill in, or update, the current state representation. This state representation was described in both this chapter and Chapter V. The individual c++ variables and structures for the state representation are described in Appendix A. In addition, Appendix A also describes the classes used to represent percepts and actions, as well as lists of the major function used to operate on this information.

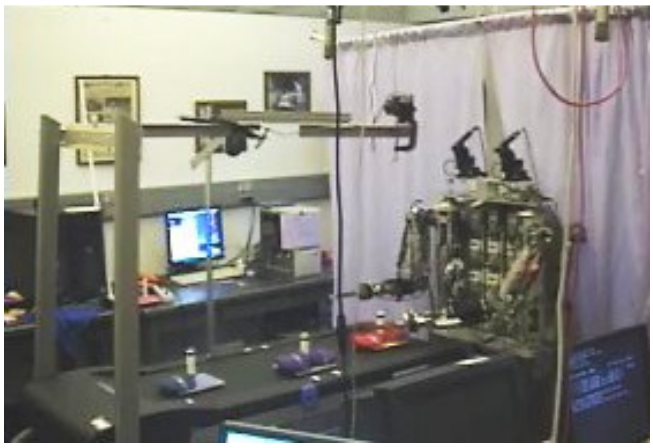


Figure 75. ISAC Coping with Three Groceries

Once the state had been updated with the latest perceptual information, the state was passed to the Working Memory System (WMS). In the WMS the conceptual clustering algorithm (Chapter V) used the learned attribute weights to compress the percepts into the feature vectors that would be used to retrieve evaluations for the current state. The member functions to do this are described in Appendix A, while example code is presented in Appendix B. The feature vectors were then appended to the state representation, and the state was passed to the Executive Control Agent (ECA). During deliberation utility and urgency appraisals were retrieved from the relational maps and performance profiles, respectively. This was performed using the processes described in Chapter V. The necessary functions are described in Appendix A, while example

code is presented in Appendix B. Planning was performed by the planning algorithm (Figure 25, Chapter V), which was instantiated within the CEA. As planning was performed a local policy was updated that associated specific states with actions. The representation for this policy is provided in Appendix A. The Bayesian network was used to determine if an interrupt should be generated. This was performed by making predictions about future states using the perceptual information stored in the current state. This also required the learned information shown in Figure 74. Finally, fit appraisals were made by comparing the external feedback (from the external critic) to the internal appraisals for a given state. Both Appendix A and B describe this process.

Results and Discussion for ISAC Experiment 1

ISAC was tested and evaluated for 20 episodes using derived knowledge of each appraisal acquired and used during simulation. This knowledge was stored in the learned attribute weight values, groceries clusters, relational maps, and performance profiles. In addition, the newly acquired Bayesian network (specifically Figure 74) was also used for this experiment. For each episode, a different trained knowledge set was selected from simulation so that error rates were not a function of a single faulty component. This re-sampling, however, did not include the Bayesian network, which was the same throughout. This was because the Bayesian networks were not trained using unsupervised learning while many of the other components (e.g., clustering and relational maps) were. In this experiment, the simulation-based knowledge was sequentially selected from the final 20 training epochs of the urgency experiment (i.e., Experiment 3). This experiment was chosen because it incorporated all of the appraisals investigated in this dissertation with the exception of the Bayesian network. Only 20 episodes were run due to the time constraints associated with using the physical system. Unlike the simulation experiments, each episode with ISAC took approximately 10 minutes to perform. This includes experiment set-up time as well as the time for groceries to move down the conveyor and ISAC to perform the pre-recorded motion (multiple times). The results for these 20 episodes are presented in Figure 76. The same smoothing window that was used in the simulation experiments (three training epochs) has also been used here. Figure 77 shows images of ISAC taken during two separate episodes from this experiment.

Figure 76 indicates that the knowledge learned in simulation transfers well to the physical hardware. This is determined by noting that the error rates in Figures 76(a) and (b) are equal to or better than the error rates shown in Figure 65(a) and (b) for the final 20 training epochs. In addition, the appraisal error rate for constraints (1) and (2), Figures 76(d) and (e) respectively, is also better than that shown in Figure 65. The appraisal error rate for constraint (3), however, is just slightly worse. The fact that simulation knowledge transferred well to the physical hardware is expected because the system, as designed, is a planning and evaluation system that relies on an external critic for evaluation, and the external critic was the same for both the simulation and the experiment. For this experiment, with the exception of the Bayesian networks the operations performed by the Deliberation & Commitment block were independent of the source of the input to the system (i.e., either simulation or physical hardware). While the perceptual input for this experiment was generated from the physical hardware and was less than ideal, it represented by the same post-perception symbols (i.e., groceries not pixels) that were passed by the SES to the WMS and into the cognitive cycle. Because the Perceptual Agents were continuously monitoring the external state, the Bayesian networks was not allowed to accrue, and this did not substantially affect performance. During these episodes, none of the groceries fell off of the conveyor as a result of not receiving an interrupt signal.

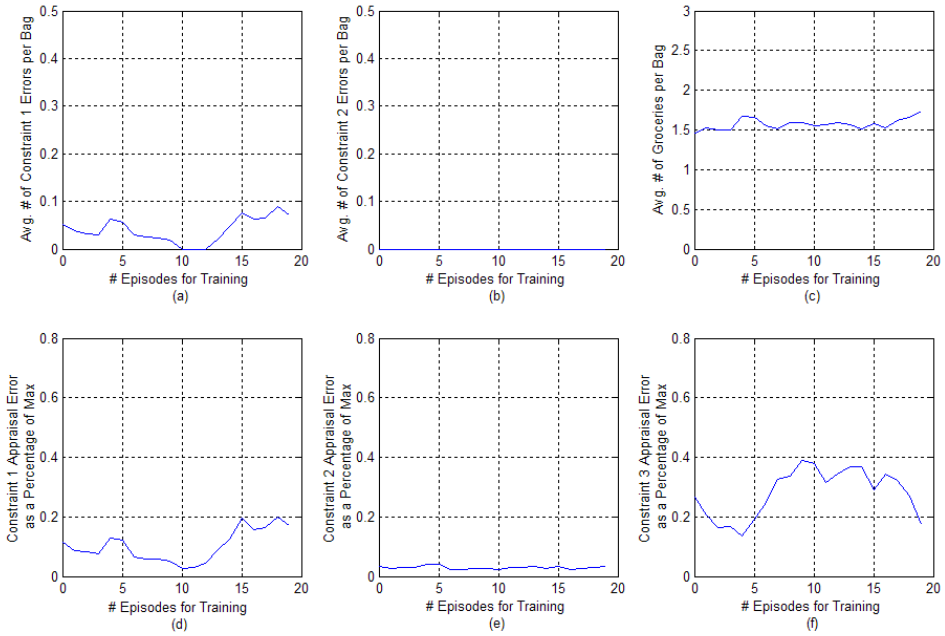


Figure 76. Performance Results for the ISAC Experiment Using Knowledge Derived from Simulation and Trained Bayesian Networks

(a) (b)

Figure 77. ISAC Interacting with Groceries Using Pre-Recorded Motions

The primary difference between the simulation and ISAC experiments is in the manner in which groceries were perceived and actions were performed. While errors in perception and actuation are frequent issues in robotics research, this experiment showed that the error rate per bag did not increase over simulation when groceries were tracked using the physical hardware. However, it is interesting to note that the ratio of groceries : bags is less than it was in simulation. This is explained by the fact that fewer groceries were used and that groceries had to be dealt with in the sequential order in which they appeared on the conveyor belt. It was observed that ISAC preferred to create several bags in which initially very few groceries were placed. This behavior was frequently observed for approximately the first four-five groceries that appeared on the belt. Then, if possible, ISAC would attempt to place the new groceries into these bags. By reducing the combined number of groceries presented to ISAC, this behavior should result in a lower grocery : bag ratio. This was further compounded because ISAC could only bag groceries in a fixed, sequential order and thus it could be the case that ISAC would have to create many bags initially, while waiting on those groceries that could be placed in multiple bags. But by reducing the number of groceries per bag, perceptual errors actually made the grocery-bagging task easier, and this could be taken as an explanation on why the error rate was actually better than in the simulation condition rather than simply equal to it.

The explanation for why the knowledge learned from simulation can be so readily applied to the physical system is rooted in the fact that the reward rule (which ultimately

determined the error rate) was derived by an external critic, and the same critic was used for simulation and the current hardware experiment. True analysis of this crossover, therefore, must be reserved for future work in which the appraisals and planning components (the focus of this dissertation) have been integrated with additional sensors and hardware that can alleviate the responsibilities and perform the role of the external critic. This could include force and touch sensors on the grippers as well as more robust vision tracking algorithms for feedback control.

ISAC Experiment 2: Integrated Cognitive Control Experiment

The objective of this experiment is to evaluate integrated cognitive control on ISAC. In this task, ISAC is required to learn "from scratch" an easier form of the grocery-bagging task, and then to deploy this knowledge successful completion of future tasks.

Experimental Condition and Assumptions

The reason this experiment used a simpler task (described below) was that the designed system was based on statistical learning from experience, in which multiple trials were assumed to be obtainable. As with the first experiment, ISAC was resctricted to 20 episodes and thus ISAC had to learn from fewer examples. Again, this is due to the fact that each episode took approximately 10 minutes to perform. The nearly 3.5 hours required to run these experiments placed stress on the physical system and required nearly static lighting and environmental conditions to ensure robust color tracking. Coupled with the fact that ISAC's real world experiences are often limited to much shorter durations, it was desired to show through this experiment that ISAC could learn on tasks of the type and duration that may be expected in the future, i.e., a limited number of episodes. Figure 78 shows the path of information through the ISAC architecture as experience is placed in episodic memory. The episodic memory system stores a record of what happened. For ISAC this includes both states and actions, and therefore information must enter episodic memory from both perception and deliberation, as shown by the two arrows entering episodic memory in Figure 78. This information is not fused in episodic memory but rather is stored in the order in which it appears. Once stored, information in episodic memory can mediate future deliberation through the Relational Mapping System, Internal Rehearsal System, and the Goals and Motivation System, as shown by the arrow leaving episodic memory in Figure 78.

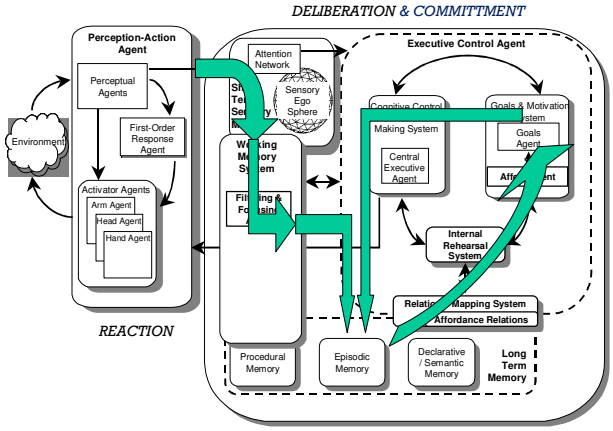


Figure 78. Paths Used for Recording Experience

In addition, the following assumptions were made for this experiment:

1. Integration of the Perceptual and Activator Agents (with each other) was not necessary for ISAC to learn from its experience. This assumption is based on the fact that planning was performed based on information provided by the Percetual Agents, and that the external critic provided feedback to the planning system based on the developed plans and selected behaviors, not on the results of those behaviors.
2. Randomly generated experience was sufficient for learning. The simulation-based experiments indicated that the system could learn even when groceries were selected at random.
3. For the sake of evaluating the decision-making and planning techniques, current behaviors were assumed to be successful, regardless of outcome. Without this assumption, the task may not grow in complexity. For example, if the first grocery on the conveyor was “eggs”, it would not matter in which bag the eggs were placed. If physical system, however, failed to successfully place the eggs in the bag, this failure could not be attributed to the planning system, but the failure also made the task easier on the next iteration when, perhaps, the grocery “potatoes” would be considered (i.e., all bags are still empty because the eggs are on the floor, and thus bag selection is inconsequential). In this experiment, once a behavior had been selected for a particular grocery, that grocery was manually removed from the conveyor belt.

The simpler version of the grocery-bagging task only required ISAC to sort groceries using the attributes *temperature* and *healthy*, and therefore the evaluation rules used by the external critic were different than those stated in the earlier section *Performance Measures*. For this experiment, ISAC was given negative reward on constraint (1) if, and only if, a bag contained two groceries of different temperature (see Table 14), and negative reward on constraint (2) if, and only if, a single bag mixed "healthy" and "nonhealthy" groceries. The evaluation for constraint (3) was based on the absolute difference between the average number of groceries per bag and a preset optimum (described in the next section). It is argued here that this task was easier because each individual constraint was evaluated using only one attribute and this evaluation was based on the presence/absence of attributes.

The experimental process for the second condition is the same as that listed for the first experiment, except that the ISAC was allowed to reuse the trained Bayesian networks and that training was performed every five episodes. This is contrasted with the simulation experiments in which training was performed every 10 episodes. The allowable behaviors for this experiment are similar to those listed in Table 17, with the exception that the *Wait(g_i)* behavior was no longer applicable. The rational for removing *Wait(g_i)* is because the collection area had been removed to ensure the possibility that groceries could fall off of the conveyor. Finally, the pre-recorded motions were not used for this experiment due to the lack of integration between the Perceptual Agents and the Activator Agents. Therefore, groceries were allowed to fall off of the conveyor belt *after* ISAC had chosen the bag in which to place them.

Experimental Procedure

The experimental procedure for this experiment is the exact same as it was for the previous experiment with three exceptions

1. ISAC was allowed to re-use the trained Bayesian network.
2. The remaining components (attribute weights, grocery clusters, and relational maps) were re-trained every 5 episodes because no learned knowledge was re-used from simulation. It should be noted that the training in this experiment occurs every 5 episodes rather than every 10 episodes as was done in simulation. The shorter duration between training steps was used because only 20 episodes were acquired and a 10 episode training window would have only evaluated one round of training before the experiment ended. Reducing

the training window to 5 episodes allows the system to train itself multiple times during the experiment, while simultaneously attempting to maximize the amount of new information included at each training step. Further rational for the training window size was based on empirical knowledge that had been acquired over many months of testing and evaluating the learning system.

3. Behaviors were not performed, and therefore groceries were allowed to fall off of the conveyor belt after ISAC had selected in which bag they should be placed. The rational for this was because this experiment was testing the system's ability to plan, not act. Combined with the difficulties (noted earlier) involving integration between the Perceptual and Activator Agents, it was desired to leave this complexity for future work.

During this experiment, the same integration between components (specifically the input/output described in Chapter V) was used as in the previous experiment. Therefore, the techniques for implementing this experiment are those listed in Appendices A and B. While the procedure for this experiment differed slightly from the previous experiment, these differences did not affect component integration.

Results and Discussion

This experiment demonstrated the application of the designed system for cognitive control on ISAC. ISAC was required to cognitively process its experience to develop internal appraisals and then had to use those appraisals to improve task performance. As mentioned previously, the task used for this experiment was a modified form of the grocery-bagging task in which ISAC had to learn to separate groceries based on *temperature* (constraint 1) and to separate *healthy* and *unhealthy* groceries (constraint 2). The reward rule for constraint (3) gave negative reward proportional to the absolute difference between the average number of groceries per bag and a preset optimum number of groceries per bag. In this experiment, the preset optimum value was set to 3.0. This value was chosen because it was the closest integer that was not equal to the grocery : bag ratio from the simulated Experiments 1-4, and was also greater than 1.0.

Figure 79 shows ISAC performing the cognitive control experiment. The performance results are presented in Figure 80. These results indicate that ISAC's performance on this task

improved with experience, and that 20 episodes were sufficient for improvements of at least 50% on constraints (1) and (2). Therefore, while the limitations of the physical hardware dictated that a task be selected which could be learned quicker, the transition from simulation to the physical hardware was not so difficult to require a re-design of the learning algorithms. Given a task in which ISAC was required to not mix groceries with different temperatures or of differing nutritional value (i.e., *healthy* vs. *nonhealthy*), and to only place exactly three groceries in a bag, ISAC could learn to simultaneously improve on all three constraints.

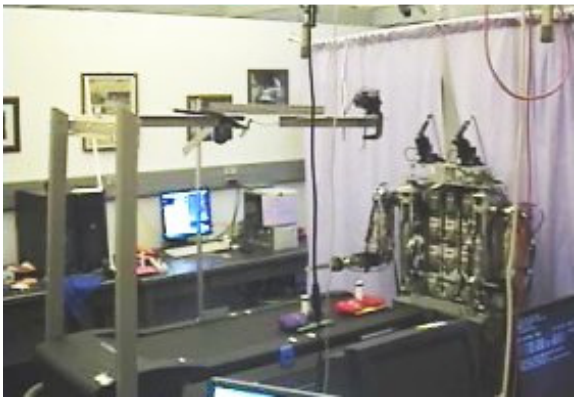


Figure 79. Integrated Cognitive Control Experiment

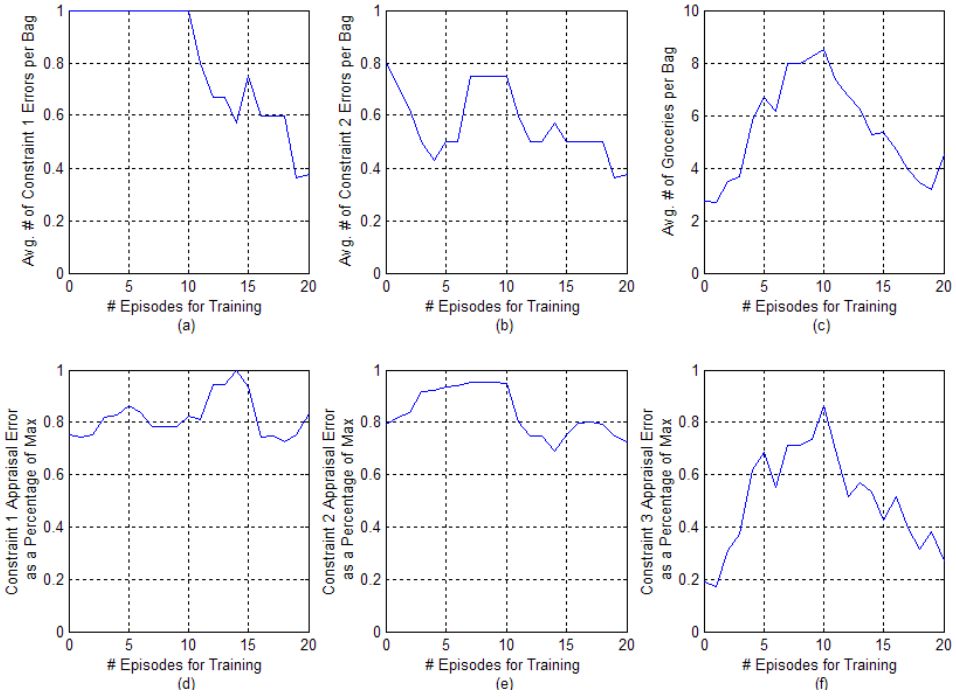


Figure 80. Performance Results for the ISAC Cognitive Control Demo

It should also be noted that while the task was intended to be simpler to learn, violations of constraint (1) and (2) occurred with higher frequency during the early stages of this experiment than in previous experiments, and that this should have a positive effect on learning. One explanation for this is that the new reward rules naturally had a higher error rate for random selection than the previous reward rules, which is in part due to the diversity of the attributes in question and the simplicity of the reward rules. The second explanation is obtained by noting that in Figure 80, ISAC starts with a grocery : bag ratio of approximately 3.0. This would result in ISAC not learning any negative evaluations for constraint (3), and subsequently, ISAC could choose to place increasing more groceries in bags. This is, in fact, the behavior observed. It is not until the next training iteration (episodes 10-15) that ISAC realizes the mistake in this approach and is able to take corrective action. Even though the grocery : bag ratio eventually drops back to the range in which it started, the error rate per bag for both constraints (1) and (2) continues to decrease. Interestingly, however, the accuracy of ISAC's appraisals does not improve at the same rate as the number of errors. The explanation for this is based on the fact that ISAC simply does not have the amount of experience necessary to learn appropriately fill-in the relational maps, and thus the improvements in performance are based on the fact that ISAC is learning (from experience) what *not to do* rather than what *to do*. This is supported by Figures 80(a) and (b) which show that while the error rate per bag has improved dramatically, it is still ~40% for both constraints, which is only slightly better than chance. In other words, ISAC is learning to avoid dangerous situations, but still has to guess a lot when selecting actions that might lead to good situations.

Finally, the learned attribute weights are presented in Table 33 and the learned grocery clusters is presented in Table 34. These clusters indicate that not only is ISAC capable of learning a different task (other than that learned in simulation), which involves learning different utility appraisals, but also that ISAC can learn distinct grocery clusters as well. This validates that, given a task and experience on that task, the designed system is capable of learning goal-specific appraisals for relevance, utility, and urgency.

Table 33. Learned Attribute Weights

Attribute	Weight
healthy	1.0000
temperature	0.9110
firmness	0.7052
size	0.6480
weight	0.5386
price	0.2243
type	0.1944
color	0.0000
name	0.0000

Table 34. Final Partition Using *GrocerySet-A*

Class	Grocery
C_0	ice_cream, eggs, yogurt, cucumbers, chicken strawberries
C_1	granola, ziploc_bags, tuna spaghetti, green_beans
C_2	rotisserie
C_3	hot_soup
C_4	teriyaki_bowl
C_5	milk, frozen_fruit
C_6	oranges, potatoes
C_7	soda, cereal, fruit_juice
C_8	tissue
C_9	bread
C_{10}	chips

Final Discussion of Results

The system, as designed, is complex and expansive. It is argued, however, that such a design is necessary to investigate the type of intelligent, cognitive behavior that must be realized if robots are to operate successfully in real-world environments. The approach described in this dissertation investigates how the cognitive processing of experience can be used to enable intelligent control at various levels of deliberation and planning. The processing of experience is based on cognitive and psychological theories of human intelligence, processing, and emotion. In this dissertation, this processing of experience focuses on a variety of appraisals that have been identified in biological systems (Chapter IV) and are known to be necessary for artificial systems (Chapter III). While each appraisal has received attention in the literature, few approaches tie each of the individual theories together within a full cognitive architecture to affect system

performance. Furthermore, it is argued that the system's unique experience is key to its development, and that only by the processing of its own experience can it improve in the areas encountered during its normal routine.

The experiments described in this chapter evaluate and validate this system and its design, but also point to many areas in which more specific focus is needed. The results show that the system is capable of learning individual appraisals for *relevance*, *utility*, and *urgency*. The appraisal for relevance allows the system to create perceptual categories that reflect the goal significant properties of items. This allows compression of the perceptual space and enables the dynamic development of useful feature vectors to represent a situation. Experiments 1 and 2 showed that through the processing of experience the system was able to identify what perceptual attributes should be focused on during task execution. Once these features were identified, the system was able to create perceptual categories using a well-known and effective conceptual clustering algorithm. In the fixed weight conditions, these categories resulted in a 64% compression, while in the non-fixed conditions the amount of compression varied between 64% and 56%.

The interplay between the learned attribute weights, the derived grocery clusters, and the trained relational maps can be viewed as similar to that between the working memory, representational, and evaluative processes that occur (primarily) within the prefrontal and orbitofronal cortex, hippocampus, and amygdala [Rolls, 1999] [Braver and Cohen, 2000] [Richter-Levin, 2004]. In particular, the attribute weights construe the attribute space to reflect the goal-relevance, or irrelevance, of specific perceptual features. The conceptual clustering algorithm uses this information to create classes in which goal-relevant attributes are highly predictable. The result of these two processes is that the system learns what to focus on (i.e., what is important about different percepts) and then uses this knowledge to filter incoming stimuli into goal-relevant sets. While this does not, necessarily, reduce the total number of percepts that must be considered, it provides a more compact representation for the percepts. Furthermore, this technique could be used to reduce the total number of percepts considered for planning through the use of an 'irrelevant' perceptual class. Using the compressed perceptual representation, the relational mapping system learns evaluations for different combinations between the individual classes. This provides utility appraisals about the current situation, which can then be used for planning.

The relevance-based clustering also provided the system with the ability to generalize to new percepts. While, expectedly, the results obtained from generalization were not as good as those obtained in the non-generalized case, the results did indicate that such a method could be used and that this ability would enable the system to expand beyond the horizons embedded in it by the programmer. This type of appraisal is critical for systems that could be deployed in environments in which it is not known *a priori* what information is important and should be focused on, and what information should be filtered out. This ability is provided for by the ISAC cognitive architecture and is partially realized through the work described here.

Experiments 1 and 2 analyze how well the system learns utility appraisals given the current relevance-based feature vectors. The utility appraisals were based on the identification of relational information in the environment that could be connected and associated with each other as well as goal-specific rewards. These experiments showed that as training increased, the system's utility appraisals became more accurate, and that the number of errors per bag simultaneously decreased. For most conditions, the error rate was decreased by about 50%, however, in the nonfixed weight, low cost condition the reduced exploration and the noise injected by errors in the derived clusters caused performance on constraint (1) to remain constant over time rather than exhibiting a net decrease.

The noise injected by the derived clusters was discussed in Experiment 2 and further described in Experiment 4. While the role of the relational maps is to identify relations between the derived clusters, errors will exist if there are outliers in each cluster that do not obey the statistical trends between members of that cluster and the remaining clusters. In Experiment 4 it was noted that these types of errors occur frequently with the groceries *eggs* and *hot_soup*.

Because the system did not have an *a priori* representation of the desired feature vectors it was important that the system be able to learn the utility appraisals "from scratch". In addition, because the system was an experience-based learner, these appraisals must be derived from and reflect an episodic long-term memory system. Within humans and other mammals such processing is believed to take place in an area of the brain known as the hippocampus, which processes relational and contextual information. The relational maps used in this dissertation are inspired by this research and are thus situated in the ISAC cognitive architecture at the "gateway" between the higher cognitive processes and ISAC's long-term memory systems.

Experiment 3 built on the results from Experiments 1 and 2, and showed that through internal rehearsal the system could develop a basic understanding of its performance and abilities and use this understanding to modify its deliberation strategy. Using this approach, episodes were selected at random and re-evaluated using the current knowledge as the reward rule. This evaluation was performed for a variety of decision-making settings, specifically different depth and breadth, and for each situation the system recorded how well it believed it could perform on the task given more, or less, time. The results indicated that the system could achieve a substantial savings in deliberation time without a prohibitive loss in overall solution quality. The appraisals were based on the notion of urgency, yet for this experiment, immediate action was never demanded but rather the system only had to maximize the tradeoff between expected search time and expected solution quality.

The fourth experiment investigated whether degrees-of-fit could be evaluated for a system composed of the components and appraisals included in this research. Of the four fundamental appraisals described throughout this dissertation, fit was the only appraisal that, when implemented, did not process past experience but instead processed the current experience using the system's other internal appraisals and confidence values along with the external feedback. This experiment showed that a basic fit evaluation could be performed and that this evaluation provided information regarding credit assignment for task success/failure. This appraisal was not designed to enable the system to mediate its deliberation or improve its performance. While there are intriguing potential directions for such a signal, in this research fit was intended to provide the user with a basic understanding of *which* component was to blame when errors were committed and *how bad* was that component's performance.

The ISAC experiments demonstrated that the designed system could be integrated with the physical hardware, but this integration required that additional attention be given to the input/output processes of perception and arm control, as well as to the separate coupling of these two systems. Due to the nature of the designed system, the symbolic percepts were the easiest to integrate, however, there was noise associated with perceptual tracking that, if not continuously monitored, would cause inappropriate behavior in the form of mistimed interrupts and deliberation cycles. This is not a fundamental failure of the designed system, because the system (as designed) mediates deliberation based on the robot's current best knowledge of the external state, or the beliefs about that state. Yet, this limitation does currently constrain ISAC to those

tasks in which the external state can be continuously monitored. Errors in perception could also result in false positive grocery identification. If a false positive occurred in ISAC's workspace, ISAC would deliberate and plan for an imaginary grocery and then attempt to execute a behavior on that grocery. The end result made the grocery-bagging task easier along constraints (1) and (2) by reducing the actual number of groceries per bag (unbeknownst to ISAC).

The use of interrupts also affected task performance during the ISAC experiments. The interrupts reduced the amount of searching performed by the deliberative process, and thus lowered the quality of the final solutions. This was primarily observed with respect to constraint (3), in which fewer groceries were placed in more bags (i.e., ISAC performed less exploration). This behavior should be expected because the planning algorithm was structured to expand new states in order of decreasing appraisal values; the states with the highest values were expanded first. In addition, the system was designed to be an experience-based learner, and thus it should be natural for the system to have higher utility evaluations for those states that have already been explored and are deemed "safest". As the system bootstraps itself, the safest states are those with a low grocery : bag ratio. When interrupts were generated, the system was naturally forced towards safe behaviors, and this explains the extremely low error rate on the first ISAC experiment. But, because constraint (3) was intrinsically designed to punish safe behaviors, solution quality suffered on this one constraint.

Finally, the integration with ISAC's hardware and actuation systems was not difficult from the perspective of issuing behavior commands or reading information from the SES, however, integration between these components (perception and actuation) was extremely difficult due to the real-time feedback control issues associated with visual-servoing and the nonlinear errors associated with translating from the visual coordinate frame to the arm coordinate frame. Furthermore, the system was designed as a deliberation and planning algorithm and is not suited to the type of real-time feedback control and monitoring required for low-level behavior execution. In other words, while the system must continuously monitor the external state for changes that may affect the next planning step (i.e., interrupts and new states), it does not monitor the external state for changes that merely effect low-level control parameters for end-effector position. This is comparable to the notions in psychology that deliberative control is well suited for cognitive, executive behavior (i.e., attention, deliberation, and planning) but poorly suited for automatic, procedural behavior (i.e., arm control). Therefore, additional

integration between ISAC's Perceptual and Activator Agents was required (e.g., visual servoing), but this was beyond the scope of the current research.

Once further system integration has been performed, though, the results presented here demonstrate that the designed system is able to cognitively process experience in order to develop internal appraisals and to apply these appraisals for improvement on the task. The system is able to learn in both simulation environments and environments built for complex, humanoid robots. While learning and cognitive control on ISAC requires that the task be tailored, to an extent, to ISAC's physical capabilities, ISAC was able to learn from its own experiences, even though they may be few in number.

CHAPTER VII

CONCLUSIONS AND FUTURE WORK

The goal of this research was to investigate how mutiple appraisals could be designed and integrated to enable decision making in a cognitive robot. The appraisals that were the focus of this work were *relevance*, *utility*, *urgency*, and *fit*. Learning these appraisals required that the robot cognitively process its own unique experience, which was stored in episodic, long-term memory. This research has found that cognitive robot which possessed these tools could learn a task, by developing its own appreciations of what was relevant for that task, how much utility should different situations elicit, how much urgency should be attached to each situation, and whether or not the current knowledge (i.e., learned appraisals) had a good degree-of-fit to the current task.

A robot designed around the appraisals *relevance*, *utility*, *urgency*, and *fit* is not *per se* an "emotional" robot; however, such a robot does possess some of the "mental underpinnings" that, in humans, have been argued to develop into emotional states. These underpinnings are based on the cognitive processing of experience and enable the robot to focus on what is important, arbitrate amongst competing responses, and respond in a time sensitive manner. These abilities are present (at least to some extent) in all mammals, which suggest that they may be linked through evolution to our own notions of human emotion. Fittingly, each of these abilities has been individually studied within the robotics literature, but what is now necessary is to implement and integrate these abilities in a cognitively inspired system. Each component should be trained (as much as possible) from the robot's unique experience, allowing that robot to develop its own unique appraisals for the environment in which it is deployed. As research in cognitive robotics advances, the internal states (including appraisals) will become more complex and of higher cognitive order, and the systems that utilize such states will be deployed in increasingly more complex environments. As these systems develop, their internal states will become more complex and adaptive, as well as more useful in everyday tasks. In fact, many systems already possess specific internal mechanisms that alone serve some of the same adaptive purposes as the internal appraisals linked to emotion and described in this dissertation. What is missing in the literature, however, is an investigation into how these components can be learned

from experience and then combined, especially within the framework of a larger cognitive architecture.

In robotics research there are a vast multitude of challenges that exist today and must eventually be solved before any autonomous robotic system will ever truly succeed outside of the laboratory. Within in the myriad challenges are the decision-making issues related to real-time responsiveness and adaptability. These issues require that the robot possess the ability to filter and focus on what is relevant in the current situation, develop adaptive, flexible representations of the environment, evaluate these representations quickly and accurately, and then adjust its responses based on internal measures of degree-of-fit as well as resource and time demand. These problems are often simplified in order to focus on optimality and to make tasks more tractable, however, this simplification risks ignoring the interdependencies between each problem and the decision-making process as a whole. Robots will eventually need to break from using preset mappings from state to state representation, and in so doing will need to learn and represent increasing more complex relational knowledge that has been abstracted out of their own experiences. Tabular representations will need to be abandoned in favor of more techniques that favor generalization, re-use, and finite search time. Planning algorithms will constantly need to be adjusted to meet the resource imposed by the environment.

Because no amount of innate knowledge can completely prepare a system for true non-deterministic nature of the real world, it is important to continue taking steps towards systems that can build their knowledge up from experience in increasingly more complex ways. This requires integration of multiple components that filter and focus, evaluate and interrupt, rehearse, plan, and meta-manage. Interestingly, as current emotion research continues to evolve it is beginning to be understood that the control of the higher-order processes is not necessarily done by an even higher-order processes, but rather much of the control exerted on these processes actually finds its origins in the evolutionarily older neurological substrates [Sloman, 2001a] [Richter-Levin, 2004] [Gazzaniga, et al., 2002]. Therefore, part of the solution for adaptable, flexible robot control must be found in the interplay between the high-level cognitive processes and the lower-level, internal appraisal mechanisms, neither of which dominates the other.

Summary of Contribution of Work

The contribution of this dissertation has been the investigation of theories of cognitive processing and emotion, and how such theories can be used to mediate decision making in an artificial, embodied system. This dissertation research has required investigating methods for flexibly representing experience (as well as the current situation), using this experience to develop internal evaluations for new states, rehearsing experience to enable appreciations of the robot's own ability, and then matching performance against expectations to determine how the robot's experience and knowledge compares to that robot's performance on the current task. Through this investigation a lot of material has been presented related to neuroscientific and psychological theories about human cognition and mammalian abilities. The investigation has led repeatedly and extensively to the areas of emotion and appraisal mechanisms, episodic memory and relational learning, working memory and filtering and focusing, executive decision making and meta-management, as well as mental simulation and internal rehearsal. This entire approach has been framed within a larger cognitive architecture, and the attempt has been made to ground the various theories (e.g., episodic memory, emotion-based appraisals, alarm mechanisms and motivation, spotlight, and common currency) in known engineering methods (e.g., self-organizing maps, function approximators, interrupts and anytime algorithms, goal significance, and utility).

A cognitive system has been developed that uses each of these techniques to perform an "everyday" task, that like the simple act of human perception is much more difficult than it seems. The results have shown that the system is able to develop each of the appraisals concurrently and that during this process performance on the task improves. The amount of improvement is predicated on the learning performed by each component. However, improvement is also based on the amount and type of experience acquired by the system. As the system acquires experience and learns from that experience, it develops a bias towards pursuing new experiences that closely match the previous experience. This is due to the fact that the similar experience is most likely to be appraised correctly, and thus the high confidence in responses is often enough to outweigh low confident, yet potentially very beneficial alternative responses. Thus, the robot becomes comfortable in its own environment.

Through the experiments with the physical hardware it was shown that the appraisals relevance, utility, urgency, and fit are not only necessary for appropriate, adaptive behavior, but

can also be integrated with a complete cognitive architecture. While research on cognitive architectures either investigates deliberative control, or attempts to compartmentalize control within specific layers, the approach described in this dissertation integrated the appraisal processes within the recursive, cognitive loop. Therefore, no one component was at the top of the hierarchy, but rather each component exerted an influence on another, with the combined result of adaptive cognitive behavior. This behavior was based on the robot's own experience, as well as the specific techniques (derived from psychology and neuroscience) to process this experience. While more work remains to be done within the ISAC cognitive architecture (e.g., integrating more robust feedback control between the Perception and Activator Agents), this research showed that through the cognitive architecture, ISAC could simultaneously learn multiple appraisals and improve task performance.

Further Directions

The research presented in this dissertation is the first step towards a cognitive robot that can truly bootstrap itself to learn a complex task. The designed system has only begun to scratch the surface of the type of appraisals that cacn usefully inform cognitive control and thus there are many future directions for this work. In addition, each of the appraisals discussed in this dissertation warrant more comprehensive, individual attention. In particular, the appraisal for relevance could be expanded to include fuzzy notions of class membership in which individual percepts may belong to multiple classes to varying extents. This would enable the system to better track multiple conflicting goals where each goal dictates different internal perceptions of objects. The attributes used for classification could also be expanded to include inter-attribute relationships, e.g., *weight* is only important when *size* is greater than '*x*'. Such a complex ability would truly enable the robot to build up notions of goal-relevant classification from scratch and allow for the type of complex relationships often taken for granted in human decision making. Furthermore, it would be interesting to investigate whether an associative network could be used to create the fundamental goal relevant classes. Such an investigation might involve training a network of interconnected perceptual attributes via Hebbian learning based on the contents of individual bags and episodes.

The utility appraisals could certainly provide directions for future research. As highlighted at various points in this dissertation, the ability to access and use generalized

abstracted relations is critical for experience-based learning. This dissertation investigated one type of relation, i.e. sequential, which could be viewed as a constrained form of temporal or spatial relation. For example, the notion that "potatoes" were placed in the bag *after* "bread", but *before* "milk" is a temporal interpretation of these relations, while the notion that "potatoes" were placed *on top* of "bread", but *under* "milk" is spatial interpretation. Other tasks, however, do not cross over as well and it is currently an open question whether a system, such as the one described in this dissertation, could bootstrap itself to learn the relationship structure as well as the relationship content. Coupled with an extended ability to identify goal-relevant classes, such a system would be capable of learning in a wide variety of environments with comparatively little pre-programming (or re-programming) on the part of developers and engineers.

Another open question would be whether or not the learned relationships stored within the *Relational Mapping System* could be extended to tasks for which they were not originally defined. This is an extremely complex problem that would also require highly flexible and reusable identification of goal-relevant classes. Consider, for example, an autonomous robotic vehicle driving along a crowded city street; it would certainly be advantageous for this system to understand that "large heavy objects do *not* go on top of smaller softer objects". The interesting question is whether this important relationship could be learned merely by bagging groceries and then noting the attribute and relationship similarity across tasks? Researchers have begun to ask such questions (for an excellent review see [Thrun and Pratt, 1998]), however, few employ the type of complex architectural approach that ultimately may be required.

The process of internal rehearsal to develop appreciations between situations and the behavioral and cognitive constraints that each affords is also an interesting problem that has recently seen some very intriguing results [Erdem, et al., 2008] [Sahin, et al., 2007]. As the environments in which robots are deployed become increasingly more complex, the need for offline pre-processing to mine a reusable set of easily deployable appreciations and affordances will become increasingly more important. This necessitates that a full investigation of internal rehearsal, including its biases and drawbacks, be performed.

The process of appraising fit has only been lightly examined in this dissertation, and the individual fit appraisals have not been used to mediate control. However, error detection and error correction are critical for robotic systems [Lyons et al., 1989] [Halder and Sarkar, 2007]. This extends beyond measuring sensor error, and must encompass all aspects of system

functionality, including the cognitive processes ultimately responsible for decision making. As robots find themselves in increasingly more complex situations, it could be disastrous if these robots were to mistakenly believe that their current knowledge is a good fit to the new tasks.

Much future work also remains to be done on the cognitive architecture used in this dissertation. The architecture has been an on-going project, and is constantly under revision and being taken in new directions [Kawamura, et al., 2008]. One such direction being considered is ubiquitous robotics [Rusu, et al., 2008], in which the architecture enables the robot to plug-and-play into various sensing and effector-equipped environments. The research described in this dissertation would be an ideal starting point for such future work because in such environments the robot would not know *a priori* what information is going to be presented to it, and thus what information is relevant for its current goal. As the robot learns what is relevant, the robot will need to develop relational maps that associate the learned information with goal-based rewards. It would then be useful for the robot to appreciate relationships between the current situation and the need for fast commitment and urgency, as well as how well its current knowledge fits the new sensor suite or effector set that it has acquired.

The system that has been described throughout this dissertation was developed based on the concepts of emotion and emotion-based appraisals. The discussion and design approached these issues from the point-of-view that low level "emotional" control would be highly advantageous for a system required to be adaptive and flexible in real-world environments. This discussion intentionally focused upon the commonly accepted parallels (at least within engineering) between emotion and utility, and attempted to describe robotic emotion as the type of multi-component process envisioned by Frijda [1986], Scherer [1997], Sloman [2001a], and a host of others. However, this dissertation tried to stop well short of pursuing these emotional concepts to the full extent with which they are realized in humans, and this necessarily opens the door for much future work in this area. While it is expected that some of this research will have immediate and practical uses for modern robots, some directions may best be left to science-fiction writers.

It is strongly contended that any robotic approach in which the goal of emotion is to enable more general, adaptive control, *must* include the robot's experience and allow the robot to develop its own internal states. Attempts to create adaptive control mechanisms based on subjective notion of proto-typical emotions (e.g., "happy/sad" states) are destined for trouble

[Damasio, 1994], as robots are fundamentally different than humans, and this is not likely to change soon. Investigations into how such labeled states may improve human-robot interaction, however, are very intriguing and warrant future researcher, yet it is argued that this is conceptually distinct from the type of control and information processing investigated here.

Ultimately, as robot technology advances the control systems that enable these robots to adaptively function in complex environments will require increasingly more complex systems, components, and appraisals. These appraisals will perform a multitude of tasks, including but not limited to: detecting relevance, appraising utility, identifying urgency, and measuring fit. The continuous and recursive interaction of these signals with the cognitive processes of the system will define its own set of control states, and within this set will lie the robot's potential "emotions". Some of these states may be familiar to us, while others may be alien, however, each will share a few common traits with our own concepts of emotion and internal control: each state (and the signals that underlie it) will be critical for the perceived proper functioning of the system, and each state will in part have been developed by that system cognitively processing its own experience.

APPENDIX A

LIST OF MAJOR COMPONENTS, CLASSES, FUNCTIONS, AND VARIABLES

State Representation: state.h

States were represented as a C++ structure, *SState*. This was because states did not require any member functions, but only state variables. These variables were a combination of numeric, symbolic, and true/false relationships that were represented by associating enumerated types with boolean operators. In addition, vectors could be used to represent lists of any of these types of information. The state structure was designed for this dissertation to have just those variables described in Chapter VI. The important variables are listed as follows:

1. *vector<CPercept> percepts* – A list of percepts (described next).
2. *vector<string> fv* – This is the set of symbolic feature vectors.
3. *vector<double> u* – This is the corresponding set of numeric utility appraisals.
4. *vector<double> x* – This is the confidence vector for the numeric utility appraisals.
5. *bool* $flag_i$ – This is the interrupt flag (true = interrupt; false = keep going)
6. *int d* – This is the search depth
7. *double b* – This is the search breadth
8. *double dt* – This is the expected/allowable deliberation time
9. *vector<double> r* – This is the external reward provided by the critic

There was a separate structure *SRelation* defined for the individual logical relations. The variables within this relation are:

1. *vector<CPercept> percepts* – This is a list of the percepts (described next) that form the relation.
2. *enum type* – This is a type identifier for the relation.
3. *bool value* – This is the true/false value of the relation.

There was also a structure for maintaining the current, local policy, *SPolicy*. This policy associated states with actions, and therefore this structure had two variables:

1. *SState state* – Specific state representation with which a single action is associated.

2. *CAction action* – Action representation (described below) for the specific action associated with the input state.

This structure was used as a vector, (i.e., vector<SPolicy> policy) so that only one action would be represented with a single state.

Percept Representation: percept.h, percept.cpp

A C++ class was designed for percepts called *CPercept*. The member variables for this class were the individual attributes possible for a percept. For example, *weight* was a member variable as was *xyz* position. Each variable was specified as either numeric or symbolic. There were three primary member functions for the percept class *CPercept*:

1. *Initialize( )* – This function initialized the member variables to default values, which could be set based on the task.
2. *GetSemanticInformation(ref)* – This function retrieved all known semantic information for that percept, given a reference pointer. The most common reference pointer is the percepts name (e.g., "bread", "potatoes", etc.) but any information related to the percepts attributes can be passed through this function.
3. *GetRewardInformation(fv, u)* – The external reward values provided by the critic were treated as a type of percept. This function retrieved those values and stored them in the variables *fv* and *u*.

Action Representation: action.h, action.cpp

A C++ class was designed for actions called *CAction*. The member variables for this class were:

1. *double cost* – Cost for performing that action.
2. *vector<CPercept> percepts* – List of *CPercept*'s that could be used to represent the targets of that action.
3. *vector<int> subactions* – Integer list of any other actions that may hierarchically form the current action. I.e., *CAction(4)* may be formed by performing *CAction(2)*, *CAction(3)*,

CAction(1) in sequence. Such examples included *BagGrocery(b,g)* which could also be written *ReachTo(g)* → *Grasp(g)* → *ReachTo(b)* → *Release(g)*.

4. *vector<SRelation> preconditions* – Set of true/false relationships that must hold before that action can be performed.
5. *vector<SRelation> postconditions* – Set of true/false relationships that should hold after that action has been performed.

There were two member functions for the class *CAction*:

1. *CheckPossibility(current_state)* – This function checked whether the given action could be performed in the current state.
2. *DetermineEffect(current_state, new_state)* – This function determined the new state as a result of applying the given action to the current state.

Episode Memory: episodic_memory.h, episodic_memory.cpp

There was nothing new implemented in the C++ episodic memory class, *CEpisodic*, for this dissertation. However, this class is described here because episodic memory was critical for this research and this description will help the discussion in Appendix B. There was only one public member function for this class:

1. *current_episode* – This episodes was a list that recorded every state and action encountered during the current episode.

The current episode was implemented as a generic structure with two components: *SState* and *CAction*. As described in Chapter's V and VI, goals were implicit for this dissertation. In addition, outcomes were implicit in the next state stored in the list. The public member functions for this class were:

1. *AddState(state)* – This function added an input state to the current episode.
2. *AddAction(action)* – This function added an input action to the current episode.
3. *SaveEpisode( )* – Saves the current episode to a file.
4. *RetrieveEpisode(id)* – Retrieves a specific episode from the saved files.
5. *GetNumberOfEpisodes( )* – Retrieve the number of episodes stored in long-term memory.

Working Memory: working_memory.h, working_memory.cpp

For this dissertation, most of the working memory functions were incorporated within the Dynamic Representation block, which included the functions described for weight learning and conceptual clustering. However, two critical functions of working memory (*CWorking*) that remained were to get the current state from short-term memory, and to maintain the local policy. The policy was one public variable that was represented as a structure, *SPolicy* with the following individual member variables:

1. *vector<SState> states* – This was a list of states.
2. *vector<CAction> actions* – This was the associated list of actions for each state.
3. *vector<double> u* – This was the list of expected utilities for each state.

The public member functions were:

1. *GetState(state)* – This function retrieved the current information in the short-term memory buffer
2. *UpdatePolicy(state, action, u)* – This function updated the current policy with the information *state*, *action*, *u*.

Weight Learning: weight.h, weight.cpp

The weight learning algorithm was designed to load sets percepts from states along with associated reward values, and derive the learn what value should be assigned to each perceptual attribute. The number of percept sets does not have to be preset, however, the number of reward dimensions does need to be preset. This algorithm was designed as a C++ class *CWeight*. The two public member variables were:

1. *m_percepts* – Vector list of percepts.
2. *m_rewards[N]* – *N* dimensional Vector list of associated rewards, where *N* is the reward dimension.

The member variables for the weight learning algorithm were:

1. *Load(file)* – Load the sets percepts from a file along with the associated reward values. The files used for this research were the individual episodes.
2. *Train( )* – Learn the weight values from the percept/reward combinations.
3. *Save(file)* – Save the learned weights to a file.

Conceptual Clustering: ccluster.h, ccluster.cpp

The conceptual clustering algorithm was a C++ class, *CCluster*, that implemented the COBWEB algorithm [Fisher, 1987] using the learned weight values for each percept. The training instances for this algorithm were the individual percepts used for the experiments (*GrocerySet-A*). The algorithm did not require any public member variables, only public functions. Theses functions were:

1. *LoadTrainingInstances(file)* – Loaded a set of percepts stored as *CPercept.*
2. *LoadWeights(file)* – Load the learned weights.
3. *Cluster( )* – Run the algorithm and develop the clusters.
4. *Classify(percept)* – Place an input percept in the best cluster and return a number associated with that cluster.
5. *SaveClusters(file)* – Save the learned clusters to file. This saves future computation by alleviating the need to continually re-calculate the clusters.
6. *AppendFeatureVector(fv, id)* – This function appended a percept's cluster identification onto a given feature vector.

Self-Organizing Maps: som.h, som.cpp

The self-organizing map algorithm was implemented as a C++ class *CSom*. As with the conceptual clustering algorithm, there were no public member variables associated with this class, only public member functions. These functions were:

1. *LoadTrainingInstance(episode)* – This function loaded an entire episode from a file, but had to be called for each episode needed for learning.
2. *Initialize( )* – This function cleared the current SOM had initialized a new random SOM.
3. *LoadSOM(file)* – This function loaded a previously saved SOM from a file.

4. *SaveSOM(file)* – This function saved the current SOM to a file.
5. *Train( )* – This function trained the current SOM using the training instances.
6. *RetrieveNode(input)* – This function retrieved the nearest node for a given input vector, and returned the complete vector stored at that node.

Urgency: urgency.h, urgency.cpp

The algorithm for determining urgency, stored in class *CUrgency*, matched the current state feature vectors to a list of examples stored in a text file. Using this list as a guide, the algorithm attempted to set the search parameters *d* and *b* to maximize solution quality while minimizing deliberation time. Trained Bayesian networks were used to trigger interrupt flags. The public member functions are listed as follows:

1. *RetrieveMatches(fv, n)* – This function retrieves the *n* best matches from the stored file.
2. *SetDepthBreadth(d, b)* – This function sets the values *d* and *b* based on the stored matches.
3. *LoadTrainingInstance(episode)* – This function loads an entire episode from a file, but must be called for each episode required for training.
4. *TrainBayesian( )* – This function uses the loaded episodes to train a Bayesian network.
5. *LoadBayesian(file)* – This function loads a Bayesion network from a file.
6. *SaveBayesian(file)* – This function save the Bayesian network to a file.
7. *CheckFlags(state, flag)* – This function uses the current state to determine whether an interrupt should be generated.
8. *InternallyRehearse(state)* – This function is an exact copy of the planning function given in Figure 25, but is used here to rehearse situations without continually monitoring the external states.
9. *AppendResults(file)* – This function appends the results from the last rehearsal to a file.

APPENDIX B

DESCRIPTION OF COMPONENTS AND IMPLEMENTATIONS

Dynamic Representation

This component was used to learn weights and to create clusters, or groups, or percepts using these weights. Specifically, the percepts that were groups were the individual groceries described in Table 14. The basic code to perform the training operation is shown as follows:

```
/*********************************************************/
CEpisodic epm;
CWeight wgt;
int num_episodes = epm.GetNumberOfEpisodes( );
for(int i=0;i<num_episodes;i++)
{
      char file[100] = epm.RetrieveEpisode(i);
      wgt.Load(file);
}
wgt.Train( );
wgt.Save("weights.txt" );
/*********************************************************/
```

The code to create the individual clusters is:

```
/*********************************************************/
CCluster clstr;
clstr.LoadTrainingInstances("percepts.txt" );
clstr.LoadWeights("weights.txt" );
clstr.Cluster( );
clstr.SaveClusters("clusters.txt");
/*********************************************************/
```

The code for classifying percepts using the current clusters is:

```
/*********************************************************/
CCluster clstr;
clstr.LoadClusters("clusters.txt");
CPercept percept;
percpet.GetSemanticInformation("bread"); //as an example
int id = clstrs.Classify(percept);
/*********************************************************/
```

The code snippets described here can be inserted, or used wherever they are needed. The first two are related to the offline cognitive processing of experience, while the last one deals performs online appraisals for relevance.

Relational Mapping

This component used experience stored within a traditional episodic memory system (i.e., state, action, state, action), to create a relational map that associated sequential aspects of each episode with goal-dependent reward values. The basic code to train the relational maps is as follows:

```
/*********************************************************/
CEpisodic epm;
CSom som;
som.Initialize();
int num_episodes = epm.GetNumberOfEpisodes( );
for(int i=0;i<num_episodes;i++)
{
      char file[100] = epm.RetrieveEpisode(i);
      som.LoadTrainingInstance(file);
}
som.Train( );
som.SaveSOM("som.txt" );
/*********************************************************/
```

The code to use the SOM during online task performance is:

```
/*********************************************************/
CSom som;
som.Initialize();
som.LoadSOM("som.txt");
SState state;
wms.GetState(state);
int id = som.RetrieveNode(state.fv);
som.GetUtilityAppraisal(id, state.u, state.x);
/*********************************************************/
```

Urgency

The urgency component used a trained Bayesian network to set the search parameters *d* and *b*, or to generate interrupts $flag_i$ (note: the interrupts were referred to as *i* in the text, but have been changed to $flag_i$ to avoid confusion with the common nomenclature "`for(int i=0;i<somelimit;i++)`"). The code used to train the urgency component is:

```
/*********************************************************/
CEpisodic epm;
CUrgency urg;
int num_episodes = epm.GetNumberOfEpisodes( );
for(int i=0;i<num_episodes;i++)
{
      char file[100] = epm.RetrieveEpisode(i);
      urg.LoadTrainingInstance(file);
}
urg.TrainBayesian( );
urg.SaveBayesian("bayes.txt" );

int num_episodes_for_rehearsal = rand()%num_episodes;
for(int i=0;i<num_episodes_for_rehearsal;i++)
{
      char file[100] = epm.RetrieveEpisode(i);
      SState state = epm.current_episode.pop_front();
      urg.InternallyRehearse(state);
      urg.AppendResults("urgency.txt");
}
/*********************************************************/
```

The code to use the urgency appraisals is:

```
/*********************************************************/
CUrgency urg;
urg.LoadBayesian("bayes.txt");
SState state;
CWorking wms;
wms.GetState(state);
int d;
double b;
bool flag;
urg.RetrieveMatches(state.fv, 5); //for example
urg.SetDepthBreadth(d, b);
urg.CheckFlags(state, flag);
/*********************************************************/
```

Fit

The algorithm for determining fit does not have a separate class, but rather this algorithm simply uses the functionality provided by the other classes. The code to calculate fit is provided. For brevity, the fit equations have been omitted.

```
/*********************************************************/
SState state;
CWorking wms;
wms.GetState(state);
vector<double> E;
```

```
for(int i=0;i<state.percepts.size();i++)
{
      if(!strcmp(state.percepts.at(i).name,"External Reward"))
      {
            vector<string> fv;
            vector<double> u;
            state.percepts.at(i).GetRewardInformation(fv, u);
            for(int j=0;j<u.size();j++)
            {
                  double fit = /* Calculate fit as shown in Chapter V */
                  E.push_back(fit);
            }
      }
}
/*********************************************************/
```

Communication with the Arm Agent

Standard TCP/IP was used to communicate with the Arm Agent. Through this communication, the signals that were sent included flags for which behavior to execute. There were two flags used, one to initiate the pre-recorded motion, and one to shut the arms down at the end of the experiment. The gripper was closed/opened at preset points during the execution of the pre-recorded motion. Code to initiate the motion is provided, along with code to shut the arms down.

```
/*********************************************************/
//Initiate the arm behavior
int flag = INITIATE;  //#defined to be 1
char msg[255];
sprintf(msg, "%d", flag);

//send msg to computer at 129.59.72.55 (Octavia) and port 30000
sock.send(msg, "129.59.72.49", 30000);
/*********************************************************/


/*********************************************************/
//Shut the arm down
int flag = SHUTDOWN; //#defined to be 0
char msg[255];
sprintf(msg, "%d", flag);

//send msg to computer at 129.59.72.55 (Octavia) and port 30000
sock.send(msg, "129.59.72.49", 30000);
/*********************************************************/
```

The code to open/close the gripper using the Arm Agent is:

```
/*********************************************************/
//example preset position for closing the gripper
double pose1[3] = {-30, 80, 70};
//example preset position for opening the gripper
double pose2[3] = {10, 80, 75};
if(trajectory[0] == pose1[0]
      && trajectory[1] == pose1[1]
      && trajectory[2] == pose1[2])
{
      CloseGripper( );
}

if(trajectory[0] == pose2[0]
      && trajectory[1] == pose2[1]
      && trajectory[2] == pose2[2])
{
      OpenGripper(7);
}
/*********************************************************/
```

where the functions *CloseGripper( )* and *OpenGripper( )* are designed to activate or deactivate the pressure valve of one of ISAC's triceps muscles that has been disconnected from the arm and connected to the gripper.

Recording motions with the Arm Agent required initializing the arms, and then closing the valves that connect to the arms, and finally reading the encoder data from the arms (as the arms were being manually driven through the desired motion).

```
/*********************************************************/
InitializeCards(); InitializeAllValves();
Sleep(5000); //msec
ResetLeftEncoders(); ResetRightEncoders();
CloseValves();
Sleep(5000); //msec
ofstream out("motion.txt");
getchar();
while(1)
{
      ReadRightEncoders(); RealRightAngles();
      ReadLeftEncoders(); RealLeftAngles();
      out << leftAngles[0] << " " << leftAngles[1] << " " << leftAngles[2] <<
endl;
      Sleep(50);
} //note: this program must be manually stopped
/*********************************************************/
```

REFERENCES

[Albus, 2002] J.S. Albus, "4D/RCS, A Reference Model Architecture for Intelligent Unmanned Ground Vehicles", in *Proceedings of the SPIE 16th International on Aerospace/Defense Sensing, Simulation, and Controls*, 2002

[Albus and Barbera, 2004] J.S. Albus and A.J. Barbera, "RCS: A cognitive architecture for intelligent multi-agent systems", in *Proceedings of the 5th IFAC/EURON Symposium on Intelligent Autonomous Vehicles*, 2004

[Aggleton and Pearce, 2002] J.P. Aggleton and J.M. Pearce, "Neural Systems Underlying Episodic Memory: Insights from animal research", in A. Baddeley, M. Conway, and J. Aggleton (eds.), Episodic Memory: New directions in Research, Oxford University Press, New York, 2002

[Ahn and Picard, 2005] H. Ahn and R. Picard, "Affective-Cognitive Learning and Decision Making: A Motivational Reward Framework For Affective Agents", *The 1st International Conference on Affective Computing and Intelligent Interaction*. Beijing, China. October 22-24, 2005

[Anderson, 1976] J.R. Anderson, Language, Memory, and Thought, Lawrence Erlbaum Associates, 1976

[Anderson, 1983] J.R. Anderson, The Architecture of Cognition, Harvard University Press, Cambridge, 1983

[Anderson and Lebiere, 1998] J.R. Anderson and C. Lebiere, The Atomic Components of Thought, Lawrence Erlbaum Associates, 1998

[Arbib and Fellous, 2004] M.A. Arbib and J-M. Fellous, "Emotions: From brain to robot", *Trends in Cognitive Sciences*, Vol. 8, No. 12, 2004

[Arkin, 1998] R.C. Arkin, Behavior-Based Robotics, MIT Press, Boston, MA, 1998

[Arkin, 2004] R.C. Arkin, "Moving Up the Food Chain: Motivation and emotion in Behavior-Based Robots", in J-M. Fellous and M. Arbib (Eds.), Who Needs Emotions? The Brain Meets the Robot, Oxford University Press, 2004

[Arkin and Vachtsevanos, 1990] R.C. Arkin and G. Vachtsevanos, "Techniques for Robot Survivability", in *Proc. 3rd International Symposium on Robotics and Manufacturing*, 1990

[Baddeley, 1998] A.D. Baddeley, Human Memory: Theory and practice –Rev. Ed.,Allyn and Bacon, 1998

[Baddeley, 2000] A.D. Baddeley, "The Episodic Buffer: A New Component to Working Memory?", *Trends in Cognitive Science*, Vol. 4, No. 11, 2000

[Baddeley and Hitch, 1974] A.D. Baddeley and G.J. Hitch, "Working Memory", In G.A. Bower (Ed.), *Recent Advances in Learning and Motivation*, Vol. 8, pp. 47-90, Academic Press, New York, NY., 1974

[Baddeley, et al., 2002] A. Baddeley, M. Conway, and J. Aggleton (Eds.), Episodic memory: New Directions in Research, Oxford University Press, New York, 2002

[Barbera, et al., 1979] A.J. Barbera, J.S. Albus, and M.L. Fitzgerald, "Hierarchical Control of Robots Using Microcomputers", in *Proceedings of the 9th International Symposium on Industrial Robots*, 1979

[Barrett, 2006] L.F. Barrett, "Are Emotions Natural Kinds?", *Perspectives on Psychological Science*, Vol. 1, No. 1, pp. 28-58, 2006

[Barto, et al., 1995] A.G. Barto, S.J. Bradtke, and S.P. Singh, "Learning to Act Using Real-Time Dynamic Programming", *Artificial Intelligence*, Vol. 72, No. 1, pp. 81-138, 1995

[Baum, 1900] L.F. Baum, The Wonderful Wizard of Oz, George M. Hill, 1900

[Baumeister, et al., 2007] R.F. Baumeister, K.D. Vohs, C.N. DeWall, and L. Zhang, "How Emotion Shapes Behavior: Feedback, Anticipation, and Reflection, Rather Than Direct Causation", *Society for Personality and Social Psychology*, Vol. 11, No. 2, pp. 167-203, 2007

[Bechara, et al., 1997] A. Bechara, H. Damasio, D. Tranel, and A.R. Damasio, "Deciding Advantageously Before Knowing the Advantageous Strategy", *Science*, Vol. 275. pp. 1293-1295, 1997

[Beer, 1995] R. Beer, "A Dynamical Systems Perspective on Agent-Environment Interaction", *Artificial Intelligence*, Vol. 72, pp. 173-215, 1995

[Beer, 2000] R. Beer, "Dynamical Approaches to Cognitive Science", *Trends in Cognitive Sciences*, Vol. 4, No. 3, 2000

[Bellman, 1957] R.E. Bellman, Dynamic Programming, Princeton University Press, Princeton, NJ, 1957

[Bentham, 1789] J. Bentham, An Introduction to the Principles of Morals and Legislation, (1948), New York: Hafner Press, 1789

[Bezdek, 1981] J.C. Bezdek, "Pattern Recognition with Fuzzy Objective Function Algorithms", Plenum Press, New York, 1981

[Biswas, et al., 1995] G. Biswas, J. Weinberg, and C. Li, "ITERATE: A conceptual clustering scheme for knowledge discovery in databases", in B. Braunschweig and R. Day (Eds.), Artificial Intelligence in the Petroleum Industry, Paris, France, Editions Technip, pp. 111-139, 1995

[Botvinick, et al., 2001] M.M. Botvinick, T.S. Braver, D.M. Barch, C.S. Carter, and J.D. Cohen, "Conflict Monitoring and Cognitive Control", *Psychological Review*, Vol. 108, No. 3, pp. 624-652, 2001

[Braver and Cohen, 2000] T.S. Braver and J.D. Cohen, "On the Control of Control: The Role of Dopamine in Regulating Prefrontal Function and Working Memory", In S. Monsell and J. Driver (Eds.), *Control of Cognitive Processes*, Vol. 18 of *Attention and Performance*, chapter 31, pp. 713-737, MIT Press, 2000

[Breazeal, 2002] C. Breazeal, Designing Sociable Robots, The MIT Press, 2002

[Breazeal and Brooks, 2004] C. Breazeal and R.A. Brooks, "Robot Emotion: A functional perspective", in J-M. Fellous and M. Arbib (Eds.), Who Needs Emotions? The Brain Meets the Robot, Oxford University Press, 2004

[Brooks, 1986] R.A. Brooks, "A Robust Layered Control System for a Mobile Robot," *IEEE Journal of Robotics and Automation*, Vol. RA-2, No. 1, pp. 14-23. 1986

[Burgess, et al., 2002] N. Burgess, S. Becker, J.A. King, and J. O'Keefe, "Memory for Events and Their Spatial Context: Models and experiments", in A. Baddeley, M. Conway, and J. Aggleton (eds.), Episodic Memory: New directions in Research, Oxford University Press, New York, 2002

[Cabanac, 1992] M. Cabanac, "Pleasure: The Common Currency", *Journal of Theoretical Biology*, Vol. 155, pp. 173-200, 1992

[Cacioppo and Bernston, 1999] J.T. Cacioppo and G.G. Bernston, "The Affect System: Architecture and Operating Characteristics", *Current Directions in Psychological Science*, 1999

[Canamero, 1997] D. Canamero, "Modeling Motivations and Emotions as a Basis for Intelligent Behavior", *Proc. of the First International Symposium on Autonomous Agents*, 1997

[Carbonell, et al., 1990] J.G. Carbonell, C. Knoblock and S. Minton "Prodigy: An Integrated Architecture for Planning and Learning", in K. VanLehn (Ed.), Architectures for Intelligence, Erlbaum, 1990

[Carver and Scheier, 1998] C.S. Carver and M.F. Scheier, On the Self-Regulation of Behavior, New York: Cambridge University Press, 1998

[Clayton and Dickinson, 1998] N.S. Clayton and A. Dickinson, "What, Where, and When: Episodic-Like Memory During Cache Recovery by Scrub Jays", Nature, Vol. 395, pp. 272-274

[Clayton, et al., 2002] N.S. Clayton, D.P. Griffiths, N.J. Emery, and A. Dickenson, "Elements of Episodic Memory in Animals", in A. Baddeley, M. Conway, and J. Aggleton (eds.), Episodic Memory: New directions in Research, Oxford University Press, New York, 2002

[Cos-Aguilera, et al 2005] I. Cos-Aguilera, L. Cañamero, G.M. Hayes, and A. Gillies, "Ecological Integration of Affordances and Drives for Behaviour Selection", *Proc. of MNAS2005*, Edinburgh, 2005

[Damasio, 1994] A. Damasio, Descartes' Error: Emotion, reason, and the human brain, New York: Grosset/Putnam, 1994

[Damasio, 1999] A. Damasio, The Feeling of What Happens: Body, Emotion, and the Making of Consciousness, Heinemann, London, 1999

[Dautenhahn, 2002] K. Dautenhahn, "Design Spaces and Niches of Believable Social Robots", in *Proceedings of the International Workshop on Robots and Human Interactive Communication*, 2002

[Darwin, 1872] C. Darwin, The Expression of the Emotions in Man and Animals, 1872, New York: Philosophical Library, 3rd Ed, 1998, with Introduction, 1998

[Dean and Boddy, 1988] T.L. Dean and M. Boddy, "An Analysis of Time-Dependent Planning", in *Proceedings of the Seventh National Conference on Artificial Intelligence*, pp. 49-54, 1988

[Dean, et al., 1993] T. Dean, L. Pack Kaelbling, J. Kirman, and A. Nicholson, "Planning with Deadlines in Stochastic Domains", in *Proceedings of 11th National Conference on Artificial Intelligence*, pp. 574-579, 1993

[Dearden and Boutilier, 1994] R. Dearden and C. Boutilier, "Integrating Planning and Execution in Stochastic Domains", in *Proceedings of Tenth Conference on Uncertainty in Artificial Intelligence*, 1994

[DeLancy, 2002] C. DeLancey, Passionate Engines, New York: Oxford University Press, 2002

[Ekman, 1992] P. Ekman, "Are There Basic Emotions?", *Psychological Review*, Vol. 99, pp. 550-553, 1992

[Ekman, 1999] P. Ekman, "Basic Emotions", in T. Dalgeish and M. Power (eds.), Handbook of Cognition and Emotion, John Wiley and Sons, Ltd., Sussex, UK, 1999

[Erdemir, et al., 2008] E. Erdemir, C.B. Frankel, K. Kawamura, S.M. Gordon, S. Thorton, and B. Ulutas, "Towards a Cognitive Robot that Uses Internal Rehearsal to Learn Affordance Relations" *International Conference on Intelligent Robots and Systems*, 2008

[Fellous and Arbib, 2004] J-M. Fellous and M. Arbib (Eds.), Who Needs Emotions? The Brain Meets the Robot, Oxford University Press, 2004

[Fisher, 1987] D.H. Fisher, "Knowledge Acquistion via Incremental Conceptual Clustering", *Machine Learning*, Vol. 2, pp. 139-172, 1987

[Food Pyramid – MIT] http://web.mit.edu/athletics/sportsmedicine/wcrfoodpyr.html

[Franklin, 1997] S. Franklin, "Global Workspace Agents", *Journal of Consciousness Studies*, Vol. 4, No. 4, pp. 322-334, 1997

[Frijda, 1986] N.H. Frijda, The Emotions, Cambridge University Press, Cambridge, UK, 1986

[Frijda, 1995] N.H. Frijda, "Emotions in Robots", in H.L. Roitblat and J-A. Meyer (Eds.) Comparative Approaches to Cognitive Science, MIT Press, Cambridge MA, 1995

[Frijda and Moffat, 1994] N.H. Frijda and D. Moffat, "Modeling Emotion", *Cognitive Studies*, Vol. 1, No. 2, pp. 5-15, 1994

[Frijda and Swagerman, 1987] N.H. Frijda and J. Swagerman, "Can Computers Feel? Theory and Design of an Emotional System", *Cognition and Emotion*, Vol. 1, No. 3, 1987

[Gadanho and Hallam, 1998] S.C. Gadanho and J. Hallam, "Exploring the Role of Emotions in Autonomous Robot Learning", *Proc. of AAAI Fall Symposium on Emotional Intelligence*, 1998

[Gadanho, 2003] S.C. Gadanho, "Learning Behavior Selection by Emotions and Cognition in a Multi-Goal Robot Task", *Journal of Machine Learning Research*, Vol. 4, pp. 385-412, 2003

[Gardiner, 2001] J.M. Gardiner, "Episodic Memory and Autonoetic Consciousness: A First-person Approach", *Philosophy Transactions Royal Society of London*, Vol. 356, pp. 1351-1361, 2001

[Gathercole, 1999] S.E. Gathercole, "Cognitive Approaches to the Development of Short-Term Memory", *Trends in Cognitive Science*, Vol. 3, No. 11, 1999

[Gazzaniga, et al., 2002] M.S. Gazzaniga, R.B. Irvy, and G.R. Mangun, Cognitive Neuroscience: The biology of the mind, 2nd Ed., W.W. Norton, New York, 2002

[Geffner and Bonet, 1998] H. Geffner and B. Bonet, "Solving Large POMDPs Using Real-Time Dynamic Programming", *Working Notes Fall AAAI Symposium on POMDPs*, 1998

[Gluck and Corter, 1985] M.A. Gluck and J.E. Corter, "Information, Uncertainty, and the Utility of Categories", *Program of the 7th Annual Conference of the Cognitive Science Society*, pp. 283-287, 1985

[Gockley, et al., 2006] R. Gockley, R. Simmons, and J. Forlizzi, "Modeling Affect in Socially Interactive Robots", *Proc. of 15th Annual IEEE International Workshop on Robot and Human Interaction*, London, UK. 2006

[Gordon, 1995] G.J. Gordon, "Stable Function Approximation in Dynamic Programming", in *Proceedings of the 12th International Conference on Machine Learning*, 1995

[Gordon and Hall, 2006] S.M. Gordon and J.F. Hall, "System Integration with Working Memory Management for Robotic Behavior Learning", *International Conference on Development and Learning*, 2006

[Griffiths, 1997] P.E. Griffiths, What Emotions Really Are: The Problem of Psychological Categories, University of Chicago Press, Chicago, 1997

[Griffiths, 2004] P.E. Griffiths, "Emotions as Natural and Normative Kinds", *Philosophy of Science*, Vol. 71, pp. 901-911, 2004

[Grupen and Huber, 2005] R.A. Grupen and M. Huber, "A Framework for the Development of Robot Behavior", *AAAI Spring Symposium Series: Developmental Robotics*, 2005

[Haddawy, 1996] P. Haddawy, "Focusing Attention in Anytime Decision-Theoretic Planning", *AAAI Spring Symposium on Planning with Incomplete Information for Robot Problems*, 1996

[Halder and Sarkar, 2007] B. Halder. and N. Sarkar, "Robust Fault Detection of Robotic Manipulator", *International Journal of Robotics Research*, Vol. 26, No.3, pp. 273-285, 2007

[Hall, 2007] J. Hall, "Internal Rehearsal for a Cognitive Robot Using Collision Detection", M.S. Thesis, Vanderbilt University, December 2007

[Haykin, 2008], S. Haykin, Neural Networks and Learning Machines, Pearson Education Inc., New Jersey, 2008

[Hebb, 1949] D.O. Hebb, The Organization of Behavior, New York: Wiley, 1949

[Hesslow, 2002] G. Hesslow, "Conscious Thought as Simulation of Behavior and Perception", *Trends in Cognitive Science*, Vol. 6, pp. 242-247, 2002

[Horsch and Poole, 1998] M.C. Horsch and D. Poole, "An Anytime Algorithm for Decision-Making Under Uncertainty", in *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, Vol. 5, pp. 162-173, 1998

[Horvitz, 1987] E.J. Horvitz, "Reasoning About Beliefs and Actions Under Computational Resource Constraints", in *Proceedings of the 1987 Workshop on Uncertainty in Artificial Intelligence*, 1987

[Hosoda and Asada, 1994] K. Hosoda and M. Asada, "Versatile Visual Servoing without Knowledge of True Jacobian", in *Proc. IEEE/RSJ International Conference on Intelligent Robots & Systems*, pp. 186-191, 1994

[James, 1884] W. James, "What Is an Emotion?", *Mind*, Vol. 19, pp. 188-205, 1884

[Jirenhed, et al., 2001] D.A. Jirenhed, G. Hesslow, and T. Ziemke, "Exploring Internal Simulation of Perception in Mobile Robots", *Lund University Cognitive Studies*, Vol. 86, pp. 107-113, 2001

[Kahneman and Tversky, 1979] D. Kahneman and A. Tversky, "Prospect Theory: An Analysis of Decision Under Risk", *Econometrica*, Vol. 47, pp. 263-291, 1979

[Kalman, 1960] R.E. Kalman, "A New Approach to Linear Filtering and Prediction Problems", *Journal of Basic Engineering*, Vol. 82, No. 1, pp. 35-45, 1960

[Kapur, 1999] N. Kapur, "Syndromes of Retrograde Amnesia: A conceptual and empirical synthesis", *Psychology Bulletin*, Vol. 125, pp. 800-825, 1999

[Kawamura, et al., 2008] K. Kawamura, S.M. Gordon, P. Ratanaswasd, E. Erdemir, and J. Hall, "Implementation of Cognitive Control for a Humanoid Robot", *International Journal of Humanoid Robotics*, 2008

[Kawewong, et al., 2008] A. Kawewong, Y. Honda, M. Tsuboyama, and O. Hasegawa, "A Common-Neural-Pattern Based Reasoning for Mobile Robot Cognitive Mapping", *INNS-NNN Symposia*, 2008

[Keiras and Meyer, 1997] D.E. Keiras and D.E. Meyer, "An Overview of the EPIC Architecture for Cognition and Performance with Application to Human-Computer Interaction", *Human-Computer Interaction*, Vol. 12, pp. 391-438, 1997

[Kleinginna and Kleinginna, 1981] P.R. Kleinginna and A.M. Kleinginna, "A Categorized List of Emotion Definitions, With Suggestions for a Consensual Definition", *Motivation and Emotion*, Vol. 5, pp. 345-379, 1981

[Koenig and Likhachev, 2001] S. Koenig and M. Likhachev, "Incremental A*", *Neural Information Processing Systems*, 2001

[Koenig and Likhachev, 2002] S. Koenig and M. Likhachev, "D* Lite", in *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems*, pp. 476-483, 2002

[Koenig and Likhachev, 2006] S. Koenig and M. Likhachev, "Real-Time Adaptive A*", *Autonomous Agents and Multi-Agent Systems*, 2006

[Kohonen, 1988] T. Kohonen, Self-Organization and Associative Memory, New York, Springer-Verlag, 1988

[Kohonen and Somervuo, 1998] T. Kohonen and P. Somervuo, "Self-Organizing Maps of Symbol Strings", *Neurocomputing*, Vol. 21, pp. 19-30, 1998

[Koku, et al., 2003] A.B. Koku, A. Sekmen, and D.M. Wilkes, "A Novel Approach for Robot Homing", in *Proceedings of the IEEE Conference on Control Applications*, Vol. 2, pp. 1477-1482, 2003

[Korf, 1990] R.E. Korf, "Real-Time Heuristic Search", *Artificial Intelligence*, Vol. 42, pp. 189-211, 1990

[Krichmar and Edelman, 2005] J.L. Krichmar and G.M. Edelman, "Brain-Based Devices for the Study of Nervous Systems and the Development of Intelligent Machines", *Artificial Life*, Vol. 11, pp. 63-78, 2005

[Kuipers, et al., 2004] B. Kuipers, J. Modayil, P. Beeson, M. Macmahon, and F. Savelli, "Local Metrical and Global Topological Maps in the Hybrid Spatial Semantic Hierarchy", *ICRA*, 2004

[Landauer, 1986] T.K. Landauer, "How Much do People Remember? Some Estimates of the Quantity of Learned Information in Long-term Memory", *Cognitive Science*, Vol. 10, pp. 477-493, 1986

[Larsen and Diener, 1992] R.J. Larsen and E. Diener, "Promises and Problems with the Circumplex Model of Emotions", in M.S. Clark (ed.), *Review of Personality and Social Psychology: Emotion*, Vol. 13, pp. 25-50, Newbury Park, CA: Sage

[Lehman, et al., 2006] J.F. Lehman, J. Laird, and P. Rosenbloom, "A Gentle Introduction to Soar, an Architecture for Human Cognition: 2006 Update", online at: http://sitemaker.umich.edu/soar/home, 2006

[Leventshtein, 1966] V.I. Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions, and Reversals", *Soviet Physics Doklady*, 1966

[Liu, et al., 2007] C. Liu, K. Conn, N. Sarkar, and W. Stone, "Affect Recognition in Robot Assisted Rehabilitation of Children with Autism Spectrum Disorder", in *Proceedings of IEEE International Conference on Robotics and Automation*, 2007

[Loewenstein, 1996] G. Loewenstein, "Out of Control: Visceral Influences on Behavior", *Organizational Behavior and Human Decision Processes*, Vol 65, No. 3., pp272-292, 1996

[Logan, 1988] G.D. Logan, "Toward an Instance Theory of Automatization", *Psychological Review*, Vol. 94, No. 4, pp. 492-527, 1988

[Lyons et al., 1989] D.M. Lyons, R. Vijaykumar, and S.T. Venkataraman, "A Representation of Error Detection and Recovery in Robot Task Plans", *SPIE Symposium on Advances in Intelligent Robotics Systems, Intelligent Control and Adaptive Systems*, 1989

[Marr, 1971] D. Marr, "Simple Memory: A theory of archicortex", *Phil. Trans. Royal Society of London*, Vol. B 262, pp. 23-81, 1971

[Martinez, et al., 1990] T.M. Martinez, H. Ritter, K.J. Schulten, "Three-Dimensional Neural Net for Learning Visuomotor Coordination of a Robot Arm", *IEEE Transactions on Neural Networks*, Vol. 1, pp. 131-136, 1990

[Matthies, et al., 2000] L. Matthies, Y. Xiong, R. Hogg, D. Zhu, A. Rankin, B. Kennedy, M. Herbert, R. Maclachlan, C. won, T. Frost, G. Sukhatme, M. McHenry, and S. Goldberg, "A Portable, Autonomous, Urban Reconnaissance Robot", in *Proceedings of the International Conference on Intelligent Autonomous Systems*, 2000

[McCarthy, 1959] J. McCarthy, "Programs With Common Sense", in *Proceedings of the Teedington Conference on the Mechanization of Thought Processes*, pp. 756-791, 1959

[McCauley and Franklin, 1998] L. McCauley and S. Franklin, "An Architecture for Emotion", *AAAI Fall Symposium Emotional and Intelligent: The Tangled Knot of Cognition*, 1998

[Mellers, et al., 1999] B. Mellers, A. Schwartz, and Ilana Ritov, "Emotion-Based Choice", *Journal of Experimental Psychology: General*, Vol. 128, No. 3, pp. 332-345, 1999

[Mellers, 2000] B. Mellers, "Choice and the Relative Pleasure of Consequences", *Psychological Bulletin*, Vol. 126, No. 6, pp. 910-924, 2000

[Melo, et al., 2008] F.S. Melo, S.P. Meyn, and M.I. Ribeiro, "An Analysis of Reinforcement Learning with Function Approximation", in *Proceedings of the 25th International Conference on Machine Learning*, 2008

[Miller, 1956] G.A. Miller, "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information", Psychological Review, Vol. 63, pp. 81-97, 1956

[Mitchell, 1997] T. Mitchell, Machine Learning, McGraw Hill, 1997

[Montague and Berns, 2002] R.P. Montague and G.S. Berns, "Neural Economics and the Biological Substrate of Valuation", *Neuron*, Vol. 36, pp. 265-284, 2002

[Morris, 2002] R.G.M. Morris, "Episodic-like Memory in Animals: Psychological criteria, neural mechanisms and the value of episodic-like tasks to investigate animal models of

neurodegenerative disease", in A. Baddeley, M. Conway, and J. Aggleton (eds.), Episodic Memory: New directions in Research, Oxford University Press, New York, 2002

[Morris, et al., 1990] R.G.M. Morris, F. Schenk, F. Tweedie, and L.E. Jarrard, "Ibotenate Lesions of Hippocampus and/or Subiculum: Dissociating Components of Allocentric Spatial Learning", *European Journal of Neuroscience*, Vol. 2, pp. 1016-1028, 1990

[Moser and Moser, 1998] M.B. Moser and E.I. Moser, "Functional Differentiation in the Hippocampus", *Hippocampus*, Vol. 8, pp. 608-619, 1998

[Moshkina and Arkin, 2003] L. Moshkina and R.C. Arkin, "On TAMEing Robots", in *Proc. IEEE International Conference on Systems, Man, and Cybernetics*, Vol. 4, pp. 3949-3959, 2003

[Nadel and Moscovitch, 1997] L. Nadel and M. Moscovitch, "Memory Consolidation, Retrograde Amnesia and the Hippocampal Complex", *Current Opinions in Neurobiology*, Vol. 7, pp. 217-227, 1997

[Newell, 1990] A. Newell, Unified Theories of Cognition, Harvard University Press, 1990

[Newell and Simon, 1963] A. Newell and H.A. Simon, "GPS: A Program that Simulates Human Thought", in E.A. Feigenbaum and J. Feldman, (Eds.) Computers and Thought, McGraw-Hill, New York. 1963

[Norman, 2002] D.A. Norman, The Design of Everyday Things, Basic Books, New York, 2002

[Nuxoll and Laird, 2004] A. Nuxoll and J. Laird "A Cognitive Model of Episodic Memory Integrated With a General Cognitive Architecture", *International Conference on Cognitive Modeling*, 2004

[Nyberg, et al., 1996] L. Nyberg, A.R. McIntosh, R. Cabeza, R. Habeb, and E. Tulving, "General and Specific Brain Regions Involved in Encoding and Retrieval of Events: What, Where, and When", *Proc. of National Academy Of Science*, Vol. 93, pp. 11280-11285, 1996

[OpenCV] http://sourceforge.net/projects/opencvlibrary/

[Ortony and Turner, 1990] A. Ortony and T.J. Turner, "What's Basic About Basic Emotions?", *Psychological Review*, Vol. 97, pp. 315-331, 1990

[Ortony, et al., 1988] A. Ortony, G.L. Clore, and A. Collins, "The Cognitive Structure of Emotion", Cambridge University Press, Cambridge UK, 1988

[Ortony, et al., 2004] A. Ortony, D.A. Norman, and W. Revelle, "Affect and Proto-Affect in Effective Functioning", in J-M. Fellous and M. Arbib (Eds.), Who Needs Emotions? The Brain Meets the Robot, Oxford University Press, 2004

[Panksepp, 1998] J. Panksepp, Affective Neuroscience, Oxford: Oxford University Press, 1998

[Panksepp, 2000] J. Panksepp, "Emotions as Natural Kinds Within the Brain", in M. Lewis and J.M. Haviland-Jones (eds.), Handbook of Emotions, 2nd Ed., New York: Guilford Press, pp. 137-155, 2000

[Papdimitriou and Tsisiklis, 1987] C. Papadimitriou and J.N. Tsisiklis, "The Complexity of Markov Decision Processes", *Mathematics of Operations Research*, Vol. 12, No. 3, pp. 441-450, 1987

[Paquet, et al., 2005] S. Paquet, L. Tobin, and B. Chaib-draa, "An Online POMDP Algorithm for Complex Multiagent Environments", *Autonomous Agents and Multi-Agent Systems*, 2005

[Perkins and Precup, 2003] T.J. Perkins and D. Precup, "A Convergent Form of Approximate Policy Iteration", in S. Becker, S. Thrun, and K. Obermayer (eds.), Advances in Neural Information Processing Systems, MIT Press, Cambridge, MA, 2003

[Peters, 2006] E. Peters, "The Functions of Affect in the Construction of Preferences", in S. Lichtenstein and P. Slovic (eds.), The Construction of Preference, pp. 454-463, Cambridge University Press, New York, 2006

[Peters, et al., 2001] R.A. Peters II, K.A. Hambuchen, K. Kawamura, and D.M. Wilkes, "The Sensory Ego-Sphere as a Short-term Memory for Humanoids", *Proc of the IEEE-RAS Int'l Conf. on Humanoid Robotics*, pp. 451-459, 2001

[Pfister and Böhm, 2008] H. Pfister and G. Böhm, "The Multiplicity of Emotions: A Framework of Emotional Functions in Decision Making", *Judgment and Decision Making*, Vol. 3, No. 1, pp. 5-17, 2008

[Phillips and Noelle, 2005] J.L. Phillips and D. Noelle, "A Biologically Inspired Working Memory Framework for Robots", *Proc. of the 27th Annual Conf. of the Cognitive Science Society*, pp. 1750-1755, 2005

[Picard, 1997] R.W. Picard,. Affective Computing, MIT Press, Cambridge, MA., 1997

[Picard, et al., 2004] R.W. Picard, S. Papert, W. Bender, B. Blumberg, C. Brazeal, D. Cavallo, T. Machover, M. Resnick, D. Roy, and C. Strohecker, "Affective Learning – A Manifesto", *BT Technology Journal*, Vol. 22, No. 4, pp. 253-269, October, 2004

[Pinker and Mehler, 1988] S. Pinker and J. Mehler, Connections and Symbols, Cambridge MA, MIT Press, 1988

[Posner and Snyder, 1975] M.I. Posner and C.R. Snyder, "Attention and Cognitive Control", in R.L. Solso (ed.), Information Processing and Cognition: The Loyola symposium, Hillsdale, N.J., Erlbaum Associates, 1975

[Provost, et al., 2006] J. Provost, B.J. Kuipers, R. Miikkulainen, "Developing Navigation Behavior Through Self-Organizing Distinctive State Abstraction", *Connection Science*, Vol. 18, No. 2, 2006

[Purdy and Olmstead, 1984] J.E. Purdy and K.M. Olmstead, "New Estimate for storage time in sensory memory", *Percept Mot Skills*, Vol. 59, No. 3, pp. 683-686, 1984

[Ratanaswasd, 2007] P. Ratanaswasd, "Implementation of Cognitive Control in a Humanoid Robot", Ph.D. Dissertation, Vanderbilt University, December 2007

[Ratanaswasd, et al., 2006] P. Ratanaswasd, C. Garber, and A. Lauf, "Situation-Based Stimuli Response in a Humanoid Robot", *International Conference on Development and Learning*, 2006

[Richter-Levin, 2004] G. Richter-Levin, "The Amygdala, The Hippocampus, and Emotional Modulation of Memory", *The Neuroscientist*, Vol. 9, No. 7, pp. 1-9, 2004

[Rolls, 1999] E.T. Rolls, The Brain and Emotion, Oxford, Oxford University Press, 1999

[Rolls, 2004] E.T. Rolls, "The Functions of the Orbitofrontal Cortex", *Brain and Cognition*, Vol. 55, pp. 11-29, 2004

[Russell, 2003] J.A. Russell, "Core Affect and the Psychological Construction of Emotion", *Psychological Review*, Vol. 110, No. 1, pp. 145-172, 2003

[Russell and Norvig, 2003] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 2nd Edition, Prentice Hall, 2003

[Sahin, et al., 2007] E. Sahin, M. Cakmak, M.R. Dogar, E. Ugur, and G. Ucoluk, "To Afford or Not to Afford: A new formalization of affordace-based robot control", *Adaptive Behavior*, Vol. 15, No. 4, pp. 447-472, 2007

[Scherer, 1981] K.R. Scherer, "Speech and Emotional States", in J. Darby (Ed.), Speech Evaluation in Psychiatry, Grune and Stratton, New York NY, 1981

[Scherer, 1997] K.R. Scherer, "Profiles of Emotion-Antecedent Appraisal", *Cognition and Emotion*, Vol. 11, No. 2, pp. 113-150, 1997

[Schneider, 1999] W. Schneider, "Working Memory in a Multi-Level Hybrid Connectionist Control Architecture (CAP2)", in A. Miyake and P. Shah (Eds.) Models of Working Memory: Mechanisms of active maintenance and executive control, pp 340-374. Cambridge, UK: Cambridge University Press, 1999

[Schneider and Shiffrin, 1977] W. Schneider and R.M. Shiffrin, "Controlled and Automatic Human Information Processing: Detection, search, and attention", *Psychological Review*, Vol. 84, pp. 1-66, 1977

[Schneider and Chein, 2003] W. Schneider and J.M. Chein, "Controlled and Automatic Processing: Behavior, Theory, and Biological Mechanisms", *Cognitive Science*, Vol. 27, pp. 525-559, 2003

[Schwarz and Clore, 1988] N. Schwarz and G.L. Core, "How Do I Feel About It? The Informative Function of Affective States", in K. Fiedler and I. Forgas (eds.), Affect, Cognition, and Social Behavior, pp. 44-62, Gottingen: Hogrefe, 1988

[Selfridge, 1959] O.G. Selfridge, "Pandemonium: A Paradigm for Learning", *Proc. of the Symposium on Mechanisation of Thought Process*. National Physics Laboratory, 1959

[Sehad and Touzet, 1994] S. Sehad and C. Touzet, "Reinforcement Learning and Neural Reinforcement Learning", *ESANN*, 1994

[Shanahan, 2006] M.P. Shanahan, "A Cognitive Architecture That Combines Internal Simulation with a Global Workspace", *Consciousness and Cognition*, Vol. 15, pp. 433-449, 2006

[Shani, et al., 2005] G. Shani, R.I. Brafman, and S.E. Shimony, "Adaptation for Changing Stochastic Environments Through Online POMDP Policy Learning", *Workshop on Reinforcement Learning in Non-Stationary Environments*, 2005

[Shen and Hasegawa, 2005] F. Shen and O. Hasegawa, "An Incremental Network for Online Unsupervised Classification and Topology Learning", *Neural Networks*, Vol. 19, No. 1, pp. 90-106, 2005

[Shrobe, et al., 2006] H. Shrobe, P. Winston, J. Tennenbaum, P. Shaftoe, S. Massaquoi, P. Robertson, B. Williams, I. Eslick, S. Rao, M. Coen, and R. Bobrow, "CHIP: A Cognitive Architecture for Comprehensive Human Intelligence and Performance", Electronic Resource: http://www.darpa.mil/ipto/programs/bica/phase1.htm, 2006

[Sloman, 2001a] A. Sloman, "Varieties of Affect and the CogAff Architecture Schema", *Proc. of the AISB'01 Symposium on Emotion, Cognition, and Affective Computing*, 2001

[Sloman, 2001b] A. Sloman, "Beyond Shallow Models of Emotion", *Cognitive Processing*, Vol. 2, No. 1, pp. 177-188, 2001

[Sloman, et al., 2004] A. Sloman, R. Chrisley, and M. Scheutz, "The Architectural Basis of Affective States and Processes", in J-M. Fellous and M. Arbib (Eds.), Who Needs Emotions? The Brain Meets the Robot, Oxford University Press, 2004

[Slovic, et al., 2003] P. Slovic, M. Finucane, E. Peters, and D.G. MacGregor, "The Affect Heuristic", in T. Gilovich, D. Griffin, and D. Kahneman (eds.), Intuitive Judgment: Heuristics and Biases, Cambridge University Press, Cambridge, MA, 2003

[Smith and Simmons, 2004] T. Smith and R. Simmons, "Heuristic Search Value Iteration for POMDPs", in *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, 2004

[Smith, 2002] A.J. Smith, "Applications of the Self-Organizing Map to Reinforcement Learning", *Neural Networks*, Vol. 15, pp. 1107-1124

[Sondik, 1971] E.J. Sondik, "The Optimal Control of Partially Observable Markov Decision Processes", PhD Thesis, Stanford University, 1971

[Sperling, 1960] G. Sperling, "The Information Available in Brief Visual Presentations", *Psychological Monographs: General and Applied*, Vol. 74, pp. 1-29, 1960

[Stentz, 1995] A. Stentz, "The Focussed D* Algorithm for Real-Time Replanning", in *Proceeding of International Joint Conference on Robotics and Automation*, pp. 3310-3317, 1995

[Strosslin, et al., 2005] T. Strosslin, D. Sheynikhovich, R. Chavariaga, and W. Gerstner, "Robust Self-Localization and Navigation Based on Hippocampal Place Cells", *Neural Networks*, Vol. 18, No. 9, pp. 1125-1140, 2005

[Sudo, et al., 2008] A. Sudo, M. Tsuboyama, C. Zhang, A. Sato, and O. Hasegawa, "Pattern-Based Reasoning System Using Self-Incremental Neural Network for Propositional Logic", *Lecture Notes in Computer Science*, Springer Berlin, 2008

[Sutton, 1988] R.S. Sutton, "Learning to Predict by the Method of Temporal Differences", *Machine Learning*, Vol. 3, pp. 9-44, 1988

[Sutton, 1999] R.S. Sutton, "Open Theoretical Questions in Reinforcement Learning", *Lecture Notes in Computer Science*, Vol. 1572, pp. 11-17, 1999

[Sutton and Barto, 2000] R.S. Sutton and A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, 2000

[Szepesvári and Smart, 2004] C. Szepesvári and W.D. Smart, "Interpolation-Based Q-Learning", in *Proceedings of the 21st International Conference on Machine Learning*, 2004

[Taylor and Kleeman, 2001] G. Taylor and L. Kleeman, "Flexible Self-Calibrated Visual Servoing for a Humanoid Robot", in *Proc. of Australian Conference on Robotics and Automation*, 2001

[Tesauro, 1990] G. Tesauro, "Neurogammon: A neural network backgammon program", in *IJCNN Proceedings III*, pp. 33-39, 1990

[Thayer, et al., 2000] S. Thayer, B. Digney, M. Diaz, A. Stentz, B. Nabbe, and M. Herbert, "Distributed Robotic Mapping of Extreme Environments", in *Proceedings of the SPIE: Mobile Robots XV and Telemanipulator and Telepresence Technologies VII*, Vol. 4195, 2000

[Thrun and Pratt, 1998] S. Thrun and L. Pratt (Eds.), Learning to Learn, Kluwer Academic Publishers, 1998

[Thrun, et al., 2005] S. Thrun, W. Burgard, and D. Fox, Probabilistic Robotics, MIT Press, Cambridge, 2005

[Tulving, 1983] E. Tulving, Elements of Episodic Memory, Oxford University Press, New York, 1983

[Tulving, 2002] E. Tulving, "Episodic Memory: From Mind to Brain", *Annual Review of Psychology*, Vol. 53, pp. 1-25, 2002

[Tversky and Kahneman, 1986] A. Tversky and D. Kahneman, "Rational Choice and the Framing of Decisions", *Journal of Business*, Vol. 59, No. 4, pp. S251-S278, 1986

[Ulutas, et al., 2008] B. Ulutas, E. Erdemir, and K. Kawamura, "Application of Hybrid Controller with Non-Contact Impedance to a Humanoid Robot", 10th International Workshop on Variable Structure Systems, 2008

[Veloso and Carbonell, 1993] M.M. Veloso and J.G. Carbonell, "Derivational Analogy in PRODIGY: Automating Case Acquisition, Storage, and Utilization", *Machine Learning*, Vol. 10, No. 3, pp 249-278, 1993

[Watkins, 1989] C. Watkins, "Learning from Delayed Rewards", Dissertation Thesis, University of Cambridge, England, 1989

[Winkielman and Trujillo, 2007] P. Winkielman and J.L. Trujillo, "Emotional Influence on Decision and Behavior: Stimuli, States and Subjectivity", in K. Vohs, R. Baumeister, and G. Loewenstein (eds.), Do Emotions Help or Hurt Decision Making?, New York: Russell Sage, 2007

[Witten, 2000] I.H. Witten, Data Mining: Practical machine learning tools and techniques with Java implementations, San Francisco, CA, Morgan Kaufman, 2000

[Zeelenberg and Pieters, 2006] M. Zeelenberg and R. Pieters, "Feeling is for Doing: A Pragmatic Approach to the Study of Emotions in Economic Behavior", in D. De Cremer, M. Zeelenberg, and K. Murnighan (eds.), Social Psychology and Economics, Erlbaum, Mahwah, NJ, 2006

[Zilberstein, 1996] S. Zilberstein, "Using Anytime Algorithms in Intelligent Systems", *AAAI*, 1996

[Zilberstein and Russell, 1995] S. Zilberstein and S.J. Russell, "Approximate Reasoning Using Anytime Algorithms", in S. Natarajan (ed.), Imprecise and Approximate Computation, Kluwer Academic Publishers, 1995