Enhancing Intelligent Agents with Episodic Memory

by

Andrew M. Nuxoll

A dissertation submitted in partial fulfillment
of the requirements of the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2007

Doctoral Committee:

Professor John E. Laird, Chair
Professor Martha E. Pollack
Associate Professor Satinder Singh Baveja
Associate Professor Richard L. Lewis
Robert Wray, Soar Technology

For Sarah, Isaac and Josephine

Table of Contents

List of Figures

Chapter 1

Introduction

One advantage that humans have over current artificial intelligence (AI) systems is a personal history of specific events from their past that they can draw upon to improve their learning and decision making. This episodic memory was first described in depth by Endel Tulving (1983, 2002). Tulving's focus is phenomenological and in particular distinguishes episodic memory from semantic memory. In particular, episodic learning remembers events and history that are embedded in experience, while semantic learning extracts facts from their experiential context. Thus a memory of looking out over the Grand Canyon during your last family vacation is an episodic memory; however, if someone asks what state the Grand Canyon is in, we would typically use semantic memory to answer (unless the answer relies on the recall of a specific episode where someone is telling you where the Grand Canyon is).

The ability to remember where you have been, what you have sensed and what actions you have taken in various situations provides a knowledge base of information that is invaluable for acting in the present. Knowing your personal history facilitates your ability to perform several cognitive capabilities in the context of sensing, reasoning and learning:

Sensing:

- **Noticing Significant Input** – detecting what is important about a given situation by its relative familiarity

- **Detecting Repetition** – realizing when you are repeating the same series of actions and altering your behavior as a result
- **Virtual Sensing** – retrieving past sensing of features outside current perception that is relevant to the current task

Reasoning:

- **Action Modeling** – predicting the immediate outcome of your actions
- **Environment Modeling –** using past experience to predict how the environment will change
- **Recording Previous Successes/Failures** – using past performance to guide future behavior
- **Managing Long Term Goals –** keeping track of a plan and what steps in that plan have been accomplished so far
- **Sense of Identity –** understanding one's own behavior in relation to other agents

Learning:

- **Retroactive Learning** – reviewing experiences and learning from them when sufficient time (or another resource) becomes available
- **Reanalysis Given new Knowledge** – relearning from experience upon receiving new knowledge
- **Explaining Behavior** – reviewing your past actions to others for mutual benefit
- **"Boost" other Learning Mechanisms –** provide a database of knowledge that can be manipulated by other learning mechanisms

Despite this array of benefits, the vast majority of AI agents lack an episodic memory. This does not mean that they lack all the cognitive capabilities listed above. Specialized algorithms can provide individual cognitive capabilities, but if all of these capabilities are to be supported, doing so with specialized algorithms would lead to redundant functionality that could be provided more efficiently by a single episodic memory. Therefore, a general, task independent episodic memory is a summarily efficient approach to providing an agent with a wealth of functionality.

As an example, consider an agent that is navigating in a two dimensional maze. Using only its own local percepts, it navigates gathering resources and using them to

attack and defend itself from other agents in the maze[1]. Episodic memory can be invaluable to this agent. It can use its episodic memory to evaluate the resource cost vs. gain of taking certain actions. When the agent is about to run out of an essential resource, it can use its virtual sensors to recall and navigate to the location of that resource and replenish it. Upon sensing a new object in its world, the agent's episodic memory allows it to recognize that it has not seen the object before and investigate the object. When it meets an enemy, the agent can examine the past success or failure that resulted from similar situations to determine how best to engage the enemy.

A flexible way for an agent to benefit from its experiences is to simply record them so that they can be examined and reexamined when they become appropriate. This appears to be the approach that evolution has taken with humans and episodic memory. Episodic memory does not preclude humans (or intelligent agents) from distilling experience into knowledge. Instead, episodic memory can assist that process and ultimately improve cognition. Certainly, there is evidence that human cognition is severely crippled by the loss of episodic memory. The difficulties that amnesiacs face have been documented (Tulving 2002) and were dramatically portrayed in the movie Memento (Nolan 2001). Therefore, we have conducted an investigation of algorithms for artificial episodic memory.

This is not a trivial task. The prospect of recording everything that an agent experiences is problematic. Any attempt to reduce the size of the memory store faces the problem of selecting what is "important" in a task independent manner. Being able to efficiently retrieve a single, appropriate experience from such a knowledge base is daunting. The first goal of this research is to engage these issues and explore how an effective, general purpose, task independent episodic memory system can be built and built to operate efficiently.

Concurrently, the second goal of this research is to explore how such an episodic memory system can be exploited by an agent. Specifically, we attempt to investigate whether episodic memory is sufficient to support a subset of the previously-listed cognitive capabilities across a range of tasks in two environments. This research suggests

[1] TankSoar, a domain that meets this description, will be introduced in the next chapter.

that episodic memory can be applied in a wide variety of ways and can enhance the performance of AI agents with a range of goals and behaviors.

Investigating whether episodic memory is the best learning mechanism for any given domain is not a goal of this research. In particular, some of this research demonstrates that episodic memory works effectively in concert with other learning mechanisms and does not necessarily compete with them.

Modeling human behavior (cognitive modeling) is also not a goal of this research. However, much of the inspiration and some ideas for this research come from the field of psychology. Human memory is complex and has many characteristics whose necessity for effective behavior is unknown (see Appendix A). Moreover, at this time there is inadequate data and theory to form the basis for a detailed model. For now, the functional computational constraints provide sufficient direction for the research. To date, this approach has been to introduce new functionality as demanded by the tasks that the agents perform and to temper open-ended design decisions with input from psychology.

Throughout this research, we have alternated between these two goals: the engineering problem of building the episodic memory and the research problem of investigating its effectiveness.

As you will see, the majority of this dissertation follows this paradigm. Chapter 2 begins with a framework for building an episodic memory and describes a set of requirements for such a memory system. Then, Chapter 3 goes into detail about the cognitive capabilities listed above and how they might manifest both in humans and intelligent agents. Chapter 4 discusses the prior work that has influenced and inspired this research. In Chapter 5, we present the Soar architecture and why it is well suited for this research. Chapter 6 provides a detailed description of the two environments that were used for both goals of this research. Chapter 7 describes the evolution of this episodic memory system with the framework defined by Chapter 2 using examples from those environments. Chapter 8 returns to the goal of evaluating the effectiveness of episodic memory by describing the agents we have built to demonstrate the cognitive capabilities granted by episodic memory. The behavior of these agents also exhibits the motivation behind some of the design decisions described in Chapter 7. We conclude in Chapter 9

with a discussion of the results as a whole, their implications, and how we expect them to drive future progress in episodic memory research.

Chapter 2

Requirements for an Episodic Memory System

This chapter presents a comprehensive examination of the design decisions and challenges that must be considered when constructing an episodic memory. Doing so provides two benefits. First, this analysis motivates effective comparison between disparate episodic memory systems both within the context of this research as well as future research. For example, we use this benefit in Chapter 7 when we present the evolution of the episodic memory system used for this research. Second, identifying the major decisions we faced provides motivation to explore other decisions at the same choice points and thus leads naturally into some of the experiments discussed in section 7.4.

In this chapter, we also discuss the functional and architectural requirements that influenced the design choices made as part of this research. Functional requirements are derived from the definition of episodic memory (e.g., versus another memory or learning mechanism). The architectural requirements stem from the decision to embed this episodic memory system into an architecture. This decision also motivates the discussion of where to separate agent from architecture at the conclusion of this chapter.

2.1 Framework for an Episodic Memory System

Episodic learning can be decomposed into the following major phases

- **encoding** – how a memory is captured and stored

- **storage** – how a memory is maintained
- **retrieval** – how a memory is retrieved.

Additionally, there is an external (but compulsory) phase: How a retrieved memory will be used by the agent. Some of the most basic design decisions, such as the structure of the representation of a memory, have impact across all phases.

By refining these three high level phases, we arrive at a larger set of steps that comprise a framework of all the points where a major design decision must be made in the implementation of an episodic memory. In effect, each of these steps is an axis, and each decision, selects a point on that axis. Together, the axes define a space of episodic memory implementations. This original framework was based upon the work of Endel Tulving (1983) and then further refined as we encountered issues or other research. There is not a definitive framework, but it does encompass all implementations and ideas we have considered to date.

Encoding

- **Encoding initiation** – when an episode is recorded. There are many possible events that could trigger its encoding. This might be as simple as recording a new memory at regular intervals. Alternatively, a new episode might be recorded whenever the agent takes an action in the world, whenever there is a significant change in its sensing, or when something unexpected occurs.
- **Episode determination** – what information is stored in an episode. An agent's state consists of a set of features that represent its current sensing and information derived from its processing. The content of an episode consists of a subset of these features from a particular point in time. This subset could consist of only the sensing information or only the internally derived features. The episode might consist of only those features that have been attended to by the agent. It is at this stage that the episodic memory system must select what is "important" about the agent's current state that should be stored for possible later retrieval.
- **Feature selection** – which features in an episode will be available for matching against a cue during the retrieval phase. The selection of features may include the entire episode or as little as one or two key features. If a subset is selected, this

reflects the episodic memory system's prediction about what features will be useful when matching future memory cues.

Storage

- **Episode structure** – how the encoded episode is stored. The structure of the episode is often dependent upon the computational architecture upon which it is built. The structure must support the other phases of the memory system within the constraints of the requirements of the architecture. In particular, the structure must support efficient and stable storage as well as expedient episode retrieval.
- **Episode dynamics** – how stored episodes change over time. This may manifest as cross-indexing with other memories. It may include a form of memory decay such as a loss of detail, a decline in activation, a merging with other memories or outright removal from the episodic store.

Retrieval

- **Retrieval initiation** – how memory retrieval is triggered. The agent may deliberately initiate memory retrieval in order to accomplish a specific subtask where it deems that previous experience may be useful. Episodic retrieval may also be triggered spontaneously by the presence of strongly familiar elements or unexpected sensing.
- **Cue determination** – how data are selected or created to cue the retrieval of an episode. Memory cues can consist of a partial memory that is matched directly to other memories in the store. They can also consist of only a few key features that the retrieved episode should or should not contain. Finally, a memory cue can also contain meta-information that is relative to the current state or the most recent retrieval (e.g., retrieve a memory of an episode that occurred after some other episode).
- **Selection** – how the retrieved episode is selected based upon the given cue. This is the critical matching phase of memory retrieval. Matching can be exact or partial. Different features of the cue can be given different weight in the match.

The match algorithm can also be influenced by the agent's current sensing and internal state. Multiple memories can be retrieved or just a single episode.

- **Retrieval** – what aspects of the episode(s) are retrieved and how they are represented. The result of a match can be as simple as a Boolean yes/no (i.e., recognition) or it can include all the data structures in the matching episode. Typically, the episodes are recreated or referenced in some location where the agent can examine them.
- **Retrieval meta-data** – what meta-information about the retrieved episode is available. Meta-data that could be available includes: information about the strength of the match between the episode and the cue; temporal information describing when the episode occurred; and information about when or how often the episode has been retrieved in the past.

Use

Once the episode is retrieved, how it is used to aid reasoning? This is not a part of the design of the episodic learning system, but depends on the capabilities of the embedding architecture, general methods, and task knowledge. In this dissertation, we demonstrate how episodic memory can be used to enhance an agent's cognitive capabilities. We discuss these capabilities in detail in Chapter 3.

2.2 Functional Requirements

Episodic memory has several characteristics that define it and may have an impact on its implementation. Below are some of the most distinguishing features of episodic learning and how those features impact the design of an artificial episodic memory:

- **Architectural** – Episodic memory is a functionality that should not change from task to task. (However, agent reasoning can impact episodic memory indirectly by affecting the determination of what is stored such as through deliberate rehearsal.) As such, it should be part of the underlying architecture with which an agent is built.
- **Automatic** – Memories are created without a deliberate decision by the agent. The nature of the episodic memory is that it records experience without

adjustment or distillation into knowledge. The underlying assumption is that the agent does not know which aspects of its experiences will be relevant to decisions it makes in the future. If an agent must decide when or how to record each episode, then intellectual power must be expended that could be spent on the task itself.

- **Autonoetic** – A retrieved memory is distinguished from current sensing. Clearly, an agent that believes a retrieved episode is actually part of its current situation will make decisions that are not necessarily consistent with its current situation. To prevent this, retrieved episodic memories should be marked as such by the architecture or occupy a reserved space in the agent's working memory.
- **Temporally Indexed** – The agent has a sense of the time when the episode occurred. Because an episodic memory describes a particular, unique moment in time, some temporal information is a part of any episodic memory. This need not be an exact time but it should convey a sense of the relative time of the episode with respect to other episodes.

2.3 Architectural Requirements

The first functional requirement (above) is that we create an architectural episodic memory. To meet that requirement an episodic memory must function across many tasks and behave in real time. This section introduces and describes the specific requirements that allow this. All of these requirements were either met or at least addressed as part of the research.

- **Task Independent** – Task independence has obvious value but seems particularly critical for episodic memory. The programmer of a task-specific episodic memory must decide in advance which features of an episode will be important and which features can be ignored. However, the features that are not recorded may be features that are needed in situations that the agent cannot predict in advance. A task-independent episodic memory can be used in domains that the agent encounters but for which it is not programmed.

  However, implementing a task-independent episodic memory is difficult and has been (and will likely continue to be) a central theme of this research. Among

the steps that are difficult to perform in a task-independent manner are the following:

- Deciding when to record a new episode
- Deciding what features of the agent's state should be recorded in an episode
- Deciding how to weigh the features of the cue and candidate memory during a match
- Deciding what memories or features might be discarded by a forgetting mechanism

An episodic memory can be task-independent and still be informed about the task by the agent. During the course of this research, we examined multiple mediums for communication between the episodic memory system and the agent. The most concrete of these are the ability for the agent to provide commands along with an episodic memory query (see section 7.4.3.1) and the additional meta-information provided with a retrieved memory (see section 7.4.3.2).

- **Low Resource Demand** – If the episodic memory system interrupts, or severely slows the agent's reasoning, then the cost of the system can outweigh its value. As a result, there should be a bound on the computational overhead for recording, maintaining and retrieving episodes. We describe the work we have done so far to reduce the resource load of the baseline episodic memory system in section 7.4.2
- **Non-Invasive Integration** – When an episodic memory system is added to an existing agent, the system should not require any changes to that agent beyond those that actually use the episodic memory. This means the episodic memory system observes the agent and records episodes without being told to do so and episodic retrieval occurs only when it is triggered by the agent. All of the implementations of episodic memory presented in this research conform to this requirement.

When building an architectural episodic memory, the issue of division of functionality between the agent and the architecture must also be addressed. This division is not a

requirement, per se, but throughout this research, we have had to face this subtle yet pervasive issue. Some functionality clearly belongs in the architectural implementation of the episodic memory module (e.g., matching). Some functionality clearly belongs in the agent (e.g., processing the retrieved memory). But some functions can occur in one or both places. For example, an agent might construct the cue for a deliberate retrieval but the architecture must construct the cue for a spontaneous retrieval.

There are clear tradeoffs between faster, task-independent architectural implementations and flexible, more precise knowledge-based implementations. Specifically, the match can be improved by allowing the agent to influence this process (see section 7.4.3.1) but the architecture cannot rely upon this influence without violating the requirement for task independence.

The divide between architecture and agent also has implications for agent performance both in terms of memory usage and response time (see section 7.4.2). An architectural implementation is more efficient but the agent has less control over it. An agent implementation is slower but more precise.

Chapter 3

The Promise of Episodic Memory: Cognitive Capabilities

To design a useful episodic memory system, we must be aware of how it will be used. For this thesis, this process began as an analysis of environments where episodic memory would be effective. It evolved into a catalog (below) of the cognitive capabilities that are supported by episodic memory. Episodic memory has the potential to support all these cognitive capabilities across sensing, reasoning and learning:

Sensing:

- **Noticing Significant Input** – An episodic memory can provide a measure of recognition of the features of an environment. This occurs when the agent retrieves memories of past situations that are similar to the current situation. This capability is particularly important in environments that change independently of the agent's actions. If a situation has unexpectedly changed, that is a signal that those aspects that changed may deserve attention.
- **Detecting Repetition** – Computers are notorious for repeating their mistakes. Even learning AI agents are rarely immune to situations wherein they repeat the same sequence of actions repeatedly. Avoiding this situation is possible if an agent can compare its current situation and proposed actions to its memory of its past can detect and avoid repeating an error.

- **Virtual Sensing** – Events or sensing that may not have been relevant to the task when experienced may unexpectedly become relevant in the future (e.g., Where did I last see my car keys?). Episodic memory provides an avenue for expanding an agents sensing beyond its immediate perceptions by retrieving past sensing of areas outside its immediate perception. In effect, it provides another sensory input to the decision process.

  We demonstrate this cognitive capability in section 8.4.

Reasoning:

- **Action Modeling** – Episodic memory allows an agent to predict immediate changes in its environment that result from a given action. It does this by recalling episodes where it took the same action in a similar situation and then retrieving the situation that immediately followed it. In addition, if an expected outcome does not occur it can signal to the agent that additional learning is necessary.

  In this research, we examine the value of action modeling with two different domains. See section 8.1.

- **Environment Modeling –** In many domains, the environment has its own dynamics (e.g., "Sunset has been around 6:30pm lately."). If an environment includes other agents, these agents can also change the environment in ways that are significant to the task (e.g., "Bob likes to play chess aggressively and usually brings his queen out early."). An episodic memory provides a record of these changes and, thus, allows the agent to predict them in similar situations in the future.

- **Recording Previous Successes/Failures** – Many tasks can only be accomplished by learning a series of steps to perform in an appropriate order. Often, success or failure is not apparent until the entire sequence is completed. For example, an agent navigating a maze must learn a best action to take at each point in the maze. Such an agent does not know it has succeeded until it exits the maze.

  As a result, an agent that is attempting to learn the right steps to accomplish a task cannot rely on action modeling alone to succeed. Instead, an

agent must be able to recall sequences of actions that led to a goal (or alternatively, failure to reach that goal).

In section 8.5, we introduce an agent that demonstrates this cognitive capability in a complex domain and discuss the requirements for an agent to be successful.

- **Managing Long Term Goals –** An agent that has multiple goals must often switch between them because of environmental demands and opportunities. In addition, long term goals often require multiple-step plans with their own subgoals to successfully complete. To schedule these goals, an agent needs to remember the progress it has made toward each one, and whether they are still active. When an agent with episodic memory considers a long term goal, it can use its memory to determine which subgoals have already been accomplished by retrieving memories of a goal being completed or undone and comparing the relative times that they occurred.
- **Sense of Identity** – For humans, one's sense of identity is rooted in memories of past experiences. As a result, granting an episodic memory to an intelligent may help grant it a similar sense of identity. An agent with a sense of identity gains greater ability to be introspective and analyze its behavior compared to the behavior of other agents. This, in turn, allows the agent to better model other agents' behavior and improve its own behavior by imitating successful differences or exploiting perceived weaknesses.

Learning:

- **Retroactive Learning** – Often, it is not possible to learn while an event is occurring because the agent lacks the specific information or resources that it needs to learn. For example, an agent in a real-time environment may not have time to apply an iterative learning algorithm while it is performing a task. However, when time becomes available, the agent can replay the events and learn from them then. Episodic memory allows previous experiences to be relived or rehearsed once the resources are available so it can be reanalyzed with new knowledge or additional experiences.

  We demonstrate this cognitive capability in section 8.2.

- **Reanalysis Given new Knowledge** – When a learning agent receives new knowledge about its environment, inferences and behavior it has learned in the past may no longer be valid. An episodic memory allows an agent to review its experiences that relate to the new knowledge and change its behavior accordingly. For example, an agent that finds a skeleton key can reconsider the actions it took when it encountered a locked door.
- **Explaining Behavior** – The ability to remember what you did in the past allows you to explain your actions to others and allow them to instruct you or you to instruct them (e.g., Why did you go left instead of right?). An agent can use its episodic memory to recall the situation in question as well as the decisions it made in that situation. A human user or an external learning mechanism can use this information to improve future behavior.
- **"Boost" other Learning Mechanisms –** An episodic memory store provides a wealth of material for other learning mechanisms.

  In section 8.3, we discuss experiments designed to demonstrate this cognitive capability with Soar's chunking mechanism.

Chapter 4

Related Work

The concept of memory is a familiar part of human experience. In the vernacular, saying that you "remember" something refers to episodic memories. In contrast, semantic memory and procedural memory are what you "know" or "can do" respectively. Nonetheless, acknowledging the distinction between long term memory systems has been a long time coming. Perhaps as a result, there is a dearth of literature on the subject in the field of artificial intelligence. While psychology has many recent studies of human episodic memory, few of them offer insights that aid in the creation of an artificial episodic memory. This section discusses the research we could find that has been done to date in both fields and describes how this research builds upon that progress.

4.1 Psychology

The term "episodic memory" was first coined by Endel Tulving in 1972 (Tulving 1972). His study of the distinction between episodic and semantic memory in humans was covered in more depth about a decade later when he published a complete phenomenological study of episodic memory including high level description of the episodic memory architecture (Tulving 1983). This work was the seed for this study of the space of implementations for an episodic memory (see section 2.1). Tulving's phenomenological examination of human episodic memory helped us define the behaviors and functions that are inherent to an episodic memory and provided hints about possible future design alternatives.

In the more recent past, Martin Conway proposes a distinction between short term episodic memory (which spans seconds or minutes) and longer term autobiographical memory which can persist for hours or even a lifetime (Conway 2001). Conway's phenomenological basis for this distinction is based primarily on memory scope and level of detail. No research we know of attempts to model this difference though it may be rooted in an integration of a semantic and episodic memory system (see section 9.2.1.1).

Modern technology has allowed psychologists to begin to look for the anatomical basis of episodic memory (Tulving 2002, Baddeley et al. 2002). Among neuroscientists, it has long been believed that the region of the brain called the hippocampus is the seat of memory. Memories are registered here and then migrate over time to the neo-cortex (Marr 1971). Models of long term memory that are based upon this theory have been built. Some of these models exhibit multiple, familiar properties of episodic memory (Moll and Miikkulainen 1997). While we are not aware of any of these models being used by an AI agent, these experiments provide further evidence for a distinction between episodic and semantic.

In the field of cognitive science, Altmann & John (1999) built a model of episodic memory model that was based upon the recorded behavior of a computer programmer. While informative, the model was built specifically for that task (computer programming). The model was also not architectural or automatic (two requirements we defined in section 2.2). Specifically, it required deliberate processing that could compete with the task at hand).

ACT-R (Anderson & Lebiere, 1998) creates long term memories via its declarative learning mechanism. Each chunk contains partial contents of the agent's working memory. This approach bears some similarities to the implementations presented in this research but it is most accurately described as a semantic memory mechanism. In particular, the chunks cannot be used to form a complete picture of a past event, nor do they contain any temporal information. Nonetheless, ACT-R's concept of using buffers to store retrieved long term memories was the model for the interface in this own system.

4.2 Artificial Intelligence

Almost no artificial intelligence research is focused directly on episodic memory for intelligent agents. The most notable exception (Ho, et al. 2003) describes multiple experiments with an agent that uses its episodic memory to backtrack to previously seen locations. Their work demonstrates the cognitive capability we describe as virtual sensors (see section 8.4). The system, while strongly task specific, provides some support for the hypothesis that episodic memory is an essential ingredient for achieving general intelligence.

In instances where researchers attempted to build a general purpose agent it sometimes became necessary to include an episodic memory. In particular, Vere and Bickmore (1990) implemented a limited episodic memory of events for their Basic Agent. Because of its peripheral nature, the effectiveness, efficiency and completeness of their agent's episodic memory implementation was not investigated. However, the fact that their attempt at a general AI required an episodic memory is telling. The Basic Agent also encountered problems with the performance of its episodic memory as the episodic store grew in size. In sections 7.4.2, we present one approach to reducing this effect. In section 9.2.4, we discuss additional future research that could address it.

Similarly, researchers attempting to build believable non-player characters for roleplaying games found that their agents needed an episodic memory to provide consistently effective interactions with human players (Brom, et al. 2007).

Episodic memory research is closely related to case-based reasoning (CBR) (Kolodner 1993, Shank 1999). In CBR, each case describes a problem that the agent faced and a specific solution to that problem. By maintaining a database of these cases, an agent can act effectively by re-using them in situations that recur or adapt them to new situations. Cases are often described by filling a fixed number of contextual fields which are designed by a human. While individual cases can be adapted new problems, no CBR systems that we know of adapt cases intended for one task for use in an entirely different task.

Nonetheless, research in CBR highlights some important research that is equally relevant to episodic memory. Goodman (1993) describes using a CBR systems' case library to predict the future. He found that given a particular case, he could predict future

cases into the near future and individual variables even further into the future. This alternate use of a case library demonstrates the cognitive capabilities we call action modeling and learning from past success and failure (see section 8.5).

Episodic memory is, in effect, the ultimate case database. The episodic memory system presented in this research does not rely upon its episodes having any particular structure or content. Nor does the episodic memory system assume its stored episodes will be used in any particular way. That decision is left entirely to the agent. Instead, we have focused on creating a task independent, architecture episodic memory that can be used to support many different cognitive capabilities (see Chapter 3) including those that can be demonstrated via case-based reasoning.

In the field of machine learning, instance-based techniques for identifying the agent's current state bear a strong resemblance to episodic memory (McCallum 1994). McCallum's approach is to record the agent's percepts at given time intervals and encode the agent's state as a sequence of these percepts. While this episodic memory is fairly simplistic and task specific, it demonstrates that giving an agent a personal history can allow it to overcome the extreme difficulty of a partially observable environment. McCallum's experiments demonstrate the cognitive capability we call virtual sensing.

Another machine learning technique reminiscent of episodic memory is locally weighted learning (Atkeson, et al. 1997) and the closely related approaches called lazy learning (Sheppard and Salzberg 1997) and continuous case-based reasoning (Ram & Santamaría 1997). All these learning mechanisms are a form of unsupervised learning in which the agent makes no effort to generalize its experiences. Instead, the agent simply stores them in much the same way than an episodic memory stores episodes. When a test case is presented, the agent selects its action by taking a weighted average of the actions taken in training cases that are "near" the test case. Thus, these algorithms rely upon both the cases and the output consisting of a fixed number of continuous values and are thus, by definition, task specific. Nonetheless, these algorithms are particularly effective at control tasks. Episodic memory can be thought of as providing a vehicle for the most general form of lazy learning. Many of the advances in finite nearest neighbor search that have been made as part of nearest neighbor search are relevant to the matching algorithms used in this research (see section 7.4.1).

Rhodes (1997) describes a prototype for an episodic memory aid for humans via a wearable PC. His agent collects data from the user's environment and then prompts the user with appropriate information based upon where the user is and what he is doing. Rhodes' episodic memory uses a database that is indexed with a word vector. The agent continuously calculates a cue vector based on its textual content. Whichever database entry's word-vector most closely matches the cue is summoned and presented on a head-up display. This AI agent effectively provides virtual sensors to a human agent.

	Noticing Significant Input	Detecting Repetition	Virtual Sensing	Action Modeling	Environment Modeling	Recording Previous Successes/Failures	Managing Long Term Goals	Sense of Identity	Retroactive Learning	Reanalysis Given new Knowledge	Explaining Behavior	"Boost" other Learning Mechanisms
Altmann & John 1999			✓									
Ho, et al. 2003			✓									
Vere & Bickmore 1990			✓									
Brom, et al. 2007											✓	
Goodman 1993				✓		✓						
Atkeson, et al. 1997 Sheppard & Salzberg 1997 Ram & Santamaría 1997				✓								
Rhodes 1997			✓									
Nuxoll 2007			✓	✓		✓			✓			✓

Figure 4-1: Cognitive Capabilities Demonstrated by Previous Research

Figure 4-1 summarizes existing research in terms of the cognitive capabilities that are defined in the previous chapter. The related AI research focuses on one or two particular cognitive capabilities that require episodic memory. The research in this thesis attempts to demonstrate multiple cognitive capabilities in a task independent manner.

In addition, this research takes a step back and examines episodic memory for its own sake. What is an episodic memory? How can it be integrated into an agent architecture? What abilities does an agent gain by virtue of having an episodic memory? These are the questions we attempt to address in this dissertation.

Chapter 5

The Soar Architecture

Our research does not consider episodic memory in isolation, but instead we explore episodic memory as a component of a broader cognitive architecture for the following reasons:

First, there are already existing arguments for using architecture to develop artificially intelligent agents in general. In particular, an architecture provides a basis for comparison between multiple implementations. It allows existing research to be more easily reused and applied to other areas and thus speeding development and allowing researchers to produce incremental results. An architecture also puts clear requirements on the overall performance of its constituent subsystems. (This issue that is particularly important to episodic memory and is discussed in sections 7.4.2 and 9.2.4.)

Second, there is evidence that episodic memory is required for general human-level intelligence and thus will have to be integrated into a larger system. Specifically, the research of Vere and Bickmore (1990) as well as (Brom, et al. 2007) both suggest that episodic memory is a required functionality for achieving a general AI agent. We've also established that the cognitive capabilities presented in Chapter 3 do not necessarily require an episodic memory but that to implement all of them will require some form of recording of events. Given that many of these cognitive capabilities are necessary for a general intelligence, this suggests that episodic memory is a requirement for general AI.

Finally, some of the research presented in this dissertation suggests that episodic memory is at its most effective when used in conjunction with other learning and memory

systems (see section 9.2.1). To make these disparate systems work together implies a medium for communication and cooperation (i.e., an architecture).

For this research, we selected Soar (Newell 1990) as the architecture in which we will embed episodic memory. Soar is a general cognitive architecture that has been used to model a wide variety of phenomena. It shares many of the features of other architectures such as ACT-R (Anderson & Lebiere 1998) and EPIC (Kieras & Meyer 1997). However, more than any other cognitive architecture, Soar is used for both cognitive modeling and artificial intelligence research. As such, it provides an excellent test bed for this research.

Soar represents procedural knowledge as production rules. Like most other architectures of this type, Soar has two types of knowledge: working memory and procedural memory. In addition, it uses a concept for decision making called operators. Finally, the version of Soar used for this research includes a working memory activation mechanism. The following sections provide an overview of the details of Soar necessary to understand many of the implementation details and experiments presented in this dissertation.

5.1 Working Memory

Working memory is a short-term declarative memory that encapsulates the agent's entire state including external sensing, internal inferences and architectural information. In Soar, working memory consists of working memory elements (WMEs). Each WME consists of three symbols: an identifier, an attribute and a value. The identifier is a symbol generated by the architecture. The attribute and value are symbols defined by a programmer and usually have descriptive names. The value of a WME can be either a constant or the identifier of one or more other WMEs. As a result, WMEs are connected together so that working memory is a directed graph. This graph is rooted (in the same way that a tree data structure is rooted) with the state identifier. Thus, all WMEs are connected directly or indirectly to the state identifier.

A reserved portion of working memory describes the agent's current sensing (i.e., input). Changes in this sensing can cause rules to fire or retract. Conversely, a reserved portion of working memory is reserved for issuing agent action commands (i.e., output).

If specific environment-defined WMEs are created in this location, the agent takes the prescribed actions in its environment. These actions can, in turn, cause changes in the environment, which can result in additional changes to working memory via its sensory input.

5.2 Procedural Memory

Procedural memory (or production memory) is a long term memory that consists of a set of production rules that encapsulate the agent's knowledge about how to act in its environment. Production rules are similar to the if-then statements used in many programming languages. They consist of a set of conditions and actions. If the conditions of a production are satisfied by the contents of working memory then that production "fires" by performing its actions. The actions usually create or remove one or more elements in working memory. When the conditions are no longer met, the production "retracts" and the WMEs it created are removed from working memory unless they have operator support (described below).

Changes to working memory made by production rules may in turn trigger the firing (or retraction) of additional productions so that this "match-fire" cycle can repeat indefinitely. When the agent reaches a state where no more productions can match, it is called quiescence.

5.3 Operators

In Soar, all matching productions fire simultaneously. To avoid conflicting behavior that might result from multiple simultaneous actions, Soar supports a special entity called an operator. A production can propose an operator by creating a specific type of structure in working memory. Multiple operators may be proposed at one time but once no more rules are ready to fire, the architecture has reached quiescence and only one operator is selected. Selection is controlled by the creation of preferences: directives created by operators that test working memory and tell the architecture which operator to select. The selection of an operator can trigger the firing of additional productions.

Soar operators extend the match-fire cycle of Soar to a five-phase decision cycle which repeats indefinitely as the agent executes:

- **Input** – The agent's sensory input from environment is updated in working memory
- **Propose** – Productions fire based upon the changes in working memory. Operators are proposed and preference rules compare them. This phase runs until the architecture reaches quiescence.
- **Select** – The architecture uses the preferences to select a single operator. This selection is notated in working memory.
- **Apply** – Production rules fire based upon the selected operator. This phase runs until the architecture reaches quiescence.
- **Output** – Any action commands created by the productions are relayed to the environment.

5.4 Persistence of Working Memory Elements

Soar distinguishes between three types of persistence for working memory elements:

- **O-supported** – If a production includes a selected operator as one of its conditions then any WMEs created by that production persist indefinitely; they remain in working memory until they are explicitly removed. These WMEs are called o-supported (operator supported) WMEs.
- **I-supported** – Any WME created by a production that does not test a selected operator persist only as long as the conditions of production continue to be met. These are known as i-supported (instantiation supported) WMEs.
- **Architectural** – Any WME created by the Soar architecture (e.g., WMEs that describe the agent's sensory input) persist until the architecture removes them again.

5.5 Subgoals

There are situations when a Soar agent does not know what to do next. For example, the agent could reach a state in which no rules fire or it could select an operator for which no rules exist to apply it. This situation is known as an impasse.

When a Soar agent enters an impasse, the architecture creates a new subgoal of the agent's original goal. This subgoal is specifically to resolve the impasse that was created. The subgoal is represented in Soar's working memory by creating a new state structure (i.e., a new root) for the entire graph. The new state contains a WME that links it to the old state so that the new state encompasses all the information in the old state.

While attempting to resolve the impasse, the agent may enter another impasse. This impasse will, in turn, generate yet another subgoal which in turn may generate other impasses and resulting subgoals and so on. As a result, working memory contains a stack (as in the fundamental data structure) of states.

An agent can resolve an impasse by making changes to the original, parent state that allow processing to continue in that state. Once an impasse is resolved in a particular state, all subgoals and corresponding substates are removed and processing continues in the goal that created the recently resolved impasse.

Entering an impasse and creating a subgoal is not an error. Among other benefits, subgoals provide a way for an agent to hierarchically decompose a task into manageable parts. For example, the agent might begin with a subgoal to "win this game of chess." However, in order to accomplish that goal it must decompose that task into a subgoal (e.g., "gain control of the center of the board") which in turn may have other subgoals that eventually lead to atomic actions (e.g., "move king's pawn to the 4^{th} rank").

5.6 Chunking

An impasse is also an indication that the Soar agent needs to learn. If the agent doesn't know what to do next, but subsequently discovers an appropriate action in a subgoal then it has obviously learned something. Chunking formalizes this fact by converting the learned knowledge into a new production that will fire on all subsequent occasions that the agent reaches the state that caused the original impasse.

To do this, the architecture records two things:

- the working memory elements in the parent state that were tested by rules that fired while in the subgoal.

- the changes that an agent made to the working memory elements in the parent state while in a subgoal.

Collectively, these two sets changes represent what needs to be done to resolve the impasse in the future. When an impasse is resolved, Soar creates a set of new productions (called chunks) whose conditions are the WMEs that were tested and whose actions are to create the new WMEs that were created. When the agent reaches the same state again, the new chunk fires and prevents the impasse from occurring again.

5.7 Working Memory Activation

The version of Soar used by this research has been extended to include working memory activation (Nuxoll, et al. 2004). All architectural and o-supported working memory elements in this version of Soar have an associated activation level. The activation level of these WMEs changes as follows:

- WMEs receive an initial, fixed activation when they are first created.
- Any time that a WME is tested by a production that fires, it receives an activation boost.
- Any time an action attempts to add an already existing WME, the existing WME receive an activation boost.

- WME activation levels decay over time using an exponential decay formula:

$$A_i = \beta + \ln\left(\sum_{j=1}^{n} t_j^{-d}\right)$$

A_i is the activation of a WME at time i. Time is measured in Soar by decision cycles. β is a base level constant. t_j is the number of cycles since the WME was referenced for the j^{th} time. d is a learning rate parameter which we set at a fixed value (0.8) for all of this research.

Chapter 6

Evaluating Episodic Memory

This chapter presents the high level approach to evaluation that supports the pursuit of both research goals: exploring the design space of episodic memory systems and demonstrating the cognitive capabilities granted by an episodic memory. The chapter begins by presenting a general methodology for evaluating these episodic memory systems. The remainder of the chapter describes the two environments that were used for all this experiments and used throughout this description of the implementation of episodic memory.

6.1 Evaluation Methodology

Given the exploratory nature of this part of the research, the effectiveness of the episodic memory systems we build are difficult to measure directly. As a result, we evaluated these episodic memory systems indirectly by comparing the performance of an episodic memory agent versus one or more control agents. The approach meets the goal of this research by providing qualitative evidence that the agent benefits from the cognitive capabilities it gains by virtue of having an episodic memory.

We have also identified secondary, qualitative goals that are essential for an effective episodic memory system. The complete set of evaluation criteria are as follows:

- **Demonstrably Improved Agent Behavior** – For each domain, we should be able to demonstrate that an agent with episodic memory can use its episodic memory to perform better than a control agent that begins with some task-specific knowledge. In some tasks, the control agent has only enough knowledge to act randomly in the world. For other tasks, we use fully functional and effective agents as this control. Regardless of the point of comparison, the addition of episodic memory should never cause the agent to perform worse than it would without that memory. Performance will be measured in terms of the quality of its actions in the domain.
- **Architectural and Task-Independent** – The same episodic memory system should work with all the agents without need for modification.
- **Acceptable Resource Usage** – For all the agents in these tests, this system should be able to operate over a long term without exceeding reasonable limits for resource usage, especially processing time and memory.
- **Simple Integration** – Adding episodic memory to an existing agent should require no changes to the agent other than adding the functionality to use the episodic memory system.

6.2 Environments for Episodic Memory

An episodic memory is of no use by itself. To demonstrate the efficacy of this system it was necessary to build agents with specific tasks set in specific environments. Given that task independence is one of these goals, it was essential to have multiple tasks and, ideally, multiple environments. We started with an action modeling task in a relatively simple environment and moved to a much more complex environment as we graduated to progressively more difficult cognitive capabilities.

This section describes both of these selected environments in detail so that we can refer to them throughout the remainder of this dissertation whenever we describe an agent's behavior within one of these tasks.

6.2.1 Eaters

The domain we selected for this first experiment is called Eaters. An "eater" is a Pac-Man[2]-like agent that moves around a 16x16 grid world. Each cell in the grid is either empty or contains a wall, normal food (● = 5 points) or bonus food (■ = 10 points). The eater is able to move in each of the four cardinal directions unless there is a wall in its way. Each time it moves into a cell containing food; it automatically eats the food (receiving the appropriate increase in score). When an eater leaves a cell, the cell becomes empty. The eater's goal is to get the highest score it can, as fast as it can. The eater's sensory input includes the contents of nearby cells, its current score, its color, and number of moves taken so far.

Figure 6-1 contains an image of the Eaters playing board (on the left) as well as a graphical depiction of the input available to an eater (on the right). The eater's actual input is presented to it symbolically (not graphically).

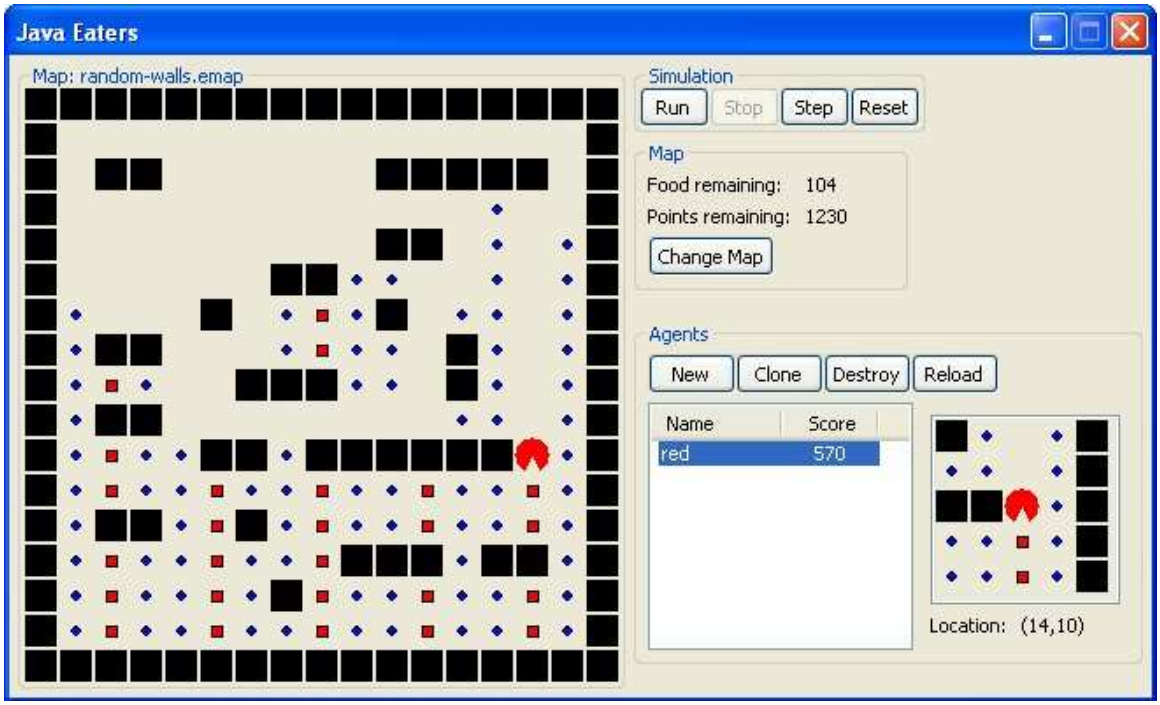

Figure 6-1: The Eaters Environment

[2] "Pac-Man" is a registered trademark of NAMCO BANDAI Games America Inc.

For most of this research, the eater knew what actions it could take in the environment but did not know the significance of the content of the cells surrounding it. In section 8.1.1, we use this task to demonstrate the cognitive capability called action modeling. Changes to agent behavior while performing this task are also used to measure the impact of various design alternatives as we explore the space of episodic memory systems in section 7.4.

Among the merits of Eaters as this first environment were the following properties:

- The domain is very simple in nature with only a limited set of four possible objects in the world and only four possible actions for the agent. This allowed us to concentrate on episodic memory and not the environment or agent implementation.
- The environment's features are repeated over and over again in varying patterns. This is, in effect, a hostile environment for an episodic memory agent because all the agent's episodic memories are similar but with small, significant differences. As a result of this property, we were able to gain significant insights into the effectiveness of various matching algorithms. In particular, we examined using activation to bias the match within the Eaters environment (see section 7.4.1).
- The simplicity of the environment and relatively fast pace of the task requires that many episodic memories be recorded and made it easy to quickly gather data about the performance of a given agent.

6.2.2 TankSoar

TankSoar is a two-dimensional, tile-based implementation of the computer game genre known as a "first person shooter". The agent is a tank moving in a two-dimensional, tile-based maze. Figure 6-2 shows a typical TankSoar map.

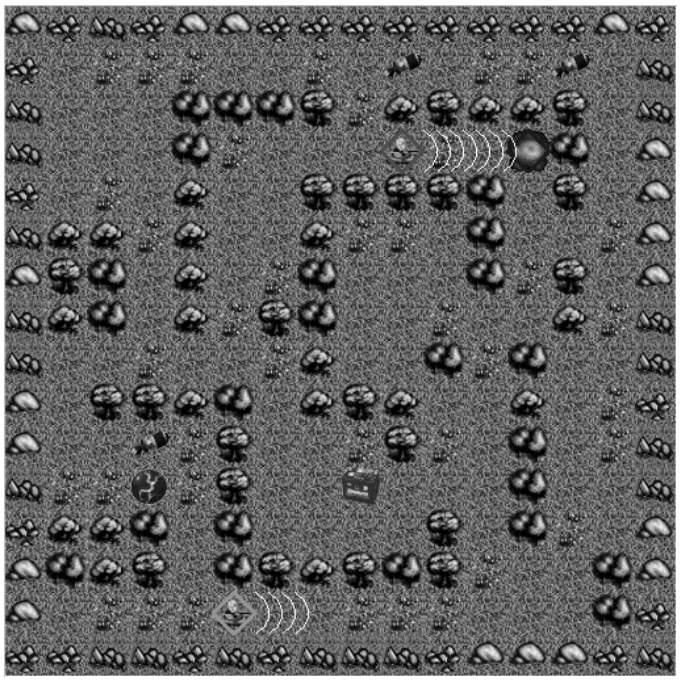

Figure 6-2: The TankSoar Environment

The tank has four types of actions it can take in the world:

- **Turn** – The tank can rotate in place to the left or right.
- **Move** – The tank can move north, south, east or west. A tank that moves in a direction perpendicular to its bearing is effectively taking a side-step.
- **Fire** – The tank can fire a missile from its turret. The missile moves in the same direction as the tank is pointed and travels in a straight line until it hits an obstacle or a tank.
- **Shields** – The tank can turn on its shields to prevent missile impacts. This preserves the agent's health at the expense of energy (see below). The agent can turn off its shields to conserve energy.
- **Radar** – The tank can turn on its radar to better sense the world around it at the expense of energy (see below). The radar's range can be set by the agent up to a fixed maximum. The agent can turn off its radar to conserve energy.

The agent can take multiple actions in the same turn as long as they are of a different type. (Exception: The agent cannot move and turn at the same time.) Each action except "Fire" has a variable argument. As a result, there are between 200 and 700 different unique actions available to the agent at any one time.

While the human observer can see the entire maze, the tank's sensors are much more limited though quite varied. The following senses are available to a TankSoar agent. With the exception of the radar, they are always on and require no energy:

- **Smell** – The agent can smell the shortest-path distance to the nearest enemy tank.
- **Hearing** – If an enemy tank is nearby and it is moving or turning, the agent can hear it and knows how far away the enemy is. Again, this the shortest-path distance.
- **Blocked** – The agent knows which of the four spaces adjacent to it are clear or blocked. If a space is blocked, it does not know if the space is occupied by an obstacle or another tank.
- **Radar Waves** – If another tank is detecting the agent with its radar, the agent can sense the radar waves and knows which direction (north, south, east or west) the radar waves are coming from.
- **Incoming** – If the agent is in the path of a missile, it can sense this and knows which direction (north, south, east or west) the missile is coming from.
- **Radar** – The agent can turn on its radar to a particular range that is specified by the agent. This is the most useful of the agent's sensors but it requires that the agent expend energy. Radar allows the agent to see the contents of a three-tile-wide corridor of spaces directly in front of it. The range of this sensing is specified by the agent (up to a fixed maximum). The greater the range, the more energy is expended. The radar is also blocked by solid objects (i.e., an obstacle or another tank).

Succeeding in the TankSoar domain requires that the agent be successful at several subtasks. One of the most critical of these is managing its three finite resources:

- **Missiles** – The agent begins with a limited supply of missiles. It can increase its supply by locating stationary missile packs that are placed at random locations in the maze. The tank automatically picks up a missile pack by moving into its square.
- **Energy** – The agent begins with a default amount of energy. Energy is expended in three ways:

- Radar - Each turn that the radar is on, the agent loses energy proportionate to the radar's range.
- Shields - Each turn that the tank's shields are on, the agent loses a fixed amount of energy.
- Deflected Missile - If a missile strikes the tank's shield then the tank loses a fixed amount of energy.

Each turn spent upon the battery (a unique stationary object in the maze) increases the tank's energy. The location of the battery is randomly selected and the agent must discover it during play and subsequently recall its location when needed. The agent's energy cannot exceed a preset maximum. Once the tank's energy reaches zero, it is no longer able to turn on its radar or its shields. This leaves it effectively blinded and vulnerable.

- **Health** – An agent begins with a default amount of health. Being hit by a missile lowers the agent's health by a set amount. Each turn spent upon the health charger (a unique stationary object in the maze) increases the tank's health. The location of the health charger is randomly selected and the agent must discover it during play and subsequently recall its location when needed. The agent's health cannot exceed a preset maximum. An agent that reaches zero health is destroyed and recreated (resetting health, energy and missile count) and placed at a random location in the maze.

Within the TankSoar environment, we implemented agents that used episodic memory to accomplish three different tasks:

- **Radar Setting Task** – In this task, the agent is attempting to reduce its energy usage by using only the minimum required setting for its radar. (The higher range that the tank sets its radar, the more energy is required.) See section 8.1.2 for a more detailed description of this task and how it is used to demonstrate the cognitive capability called action modeling.
- **Energy Search Task** – In this task, the agent is low on energy and must locate the battery (a static object in the environment) in order to recharge. Section 8.4 provides a more detailed description of this task and how it is used to demonstrate the cognitive capability called virtual sensors.

- **Combat Task** – This is the complete task that TankSoar agents are normally developed for. The agent's goal is to maximize its score. The agent scores points for hitting an enemy tank with a missile. It scores additional points for destroying an enemy tank. Conversely, the agent loses points for being hit by a missile. It loses additional points when it is killed. The agent also must maintain its supply of the three resources described above (energy, health and missiles). An agent that is low on any resource is at a disadvantage vs. an agent that has an adequate supply of all of the resources. The game ends when one of agents scores enough points to reach a fixed threshold (usually 50 points). Section 8.5 describes how we use this task to demonstrate the cognitive capability called learning from past success and failure. In addition, we use this task to demonstrate several different design alternatives in Chapter 7.

As is apparent, the TankSoar environment is much more complex than Eaters. It requires the agent not only be skilled in tactical combat, but also resource management and navigation. A rich, complex environment such as this is well-suited for episodic memory research because an agent requires multiple cognitive capabilities to be effective. In Chapter 8 we discuss the experiments we have performed to demonstrate multiple cognitive capabilities within the TankSoar environment.

Chapter 7

Implementations of Episodic Memory

In section 2.1, we described this framework for the space of episodic memory systems and also described how one goal of this research is an exploration of this space. In this chapter, we describe the results of that exploration. We reserve the discussion of the work towards this second goal, demonstrating the cognitive capabilities granted to an agent by virtue of having an episodic memory, for Chapter 8. For the sake of clarity, we describe the two goals separately rather than attempting to continually switch between them in roughly chronological order. In reality, this work toward these two goals was heavily interleaved and, as a result, this chapter makes multiple forward references to Chapter 8.

To describe this exploration of the space of episodic memory systems, this chapter describes how each of our major implementations has evolved from this original pilot implementation to the current, fully task-independent implementation and beyond. We begin by describing the pilot implementation and the baseline implementation by defining their locations in the framework. By doing so, we hope to provide a clear picture of the evolution of the system as well as its future trajectory.

We follow with a description of each of the major explorations of that implementation space by describing the experiments, results and conclusions for each experiment. Collectively, these experiments represent further evolution of this episodic memory system and provide a basis for some of the future work discussed in section 9.2.

7.1 Pilot Implementation

Compared to more recent implementations, the pilot implementation had limited functionality and a lack of task independence. Despite these limitations, the pilot implementation provided an essential first step into the design space of episodic memory systems. It provided valuable insights into the existence of several critical design decisions. In other words, this pilot implementation and this definition of the design space evolved together. The pilot implementation also was used to perform a few of the experiments that are discussed in this dissertation. Finally, the pilot provided a point for comparison with all future implementations. As a result, it is useful to understand some of the details of this implementation in order to understand future implementations.

At a high level, the pilot implementation stored each element of an episodic memory as a Soar production. These productions fired (effectively retrieving the episodic memory) when a special retrieval operator was selected by the agent. This approach allowed us to focus on episode encoding while using Soar's existing storage and retrieval functionality. As we discuss at the end of this section, this parsimony came at the cost encoding specificity issues and some loss of functionality.

The pilot implementation fit within this framework as follows:

Encoding

- **Encoding initiation** – New episodes were recorded whenever there was a significant change in the working memory of the agent. The rationale for this approach was that the agent should record more episodes when the agent's situations or focus of attention was changing and fewer episodes when the agent's state was relatively stable. We defined a significant change in working memory to be a change in the top N most-activated working memory elements (where multiple values of N were tried). Working memory activation in Soar is discussed in section 5.7.
- **Episode determination** – We hand-selected the content of the episode using a task-specific method. This decision represented a deliberate departure from task independence that allowed us to concentrate on other aspects of the design. In particular, we began by studying the efficacy of activation-based feature selection

(see section 7.4.1). Future implementations of the episodic memory system have been task independent.

- **Feature selection** – The parts of a memory that would match the cue were determined by selecting the N most-activated working memory elements. We hypothesized that the most activated parts of working memory would provide a good approximation of the locus of the agent's cognitive activity. As a result, we expected that these features would be most relevant when selecting a memory to retrieve.

Storage

- **Episode structure** – Episodes were encoded as collections of rules in Soar. In particular, each rule was responsible for recreating a particular working memory element (WME) that existed in the episode.
- **Episode dynamics** – The original implementation had no episode dynamics. Once an episode was recorded it was never altered or removed from memory.

Retrieval

- **Retrieval initiation** – In the pilot implementation, retrieval was triggered when the agent deliberately constructed a cue in working memory.
- **Cue determination** – The agent constructed a cue by creating a special substate specifically for this purpose. The agent determined the content of the cue, which could include not only surface features but also their relations to each other.
- **Selection** – Since episodic memories were recorded as collections of Soar productions, episodic match and retrieval were performed as part of a normal Soar decision cycle. An operator proposal rule would match the cue. In effect, the operator was proposing the retrieval of a particular episodic memory. If that operator was selected, the operator application rules would recreate the memory. If multiple memories matched the cue, multiple operators would be proposed and the agent would select the best match via operator selection.

- **Retrieval** – In the initial implementation, the recall operation directly overwrote the retrieval cue structure that the agent had created. The resulting structure was a duplicate of the original state from which the episode was drawn.
- **Retrieval meta-data** – The only metadata provided to the agent was a numeric ID that was assigned to each episode sequentially as it was encoded. This provided the agent with information about the temporal order of episodes without providing a specific time that the episode occurred. This information, while present, was never used by the agents in our experiments.

Use

An array of experiments was performed with the pilot implementation in the Eaters environment (which is described in section 6.2.1). In particular, we used the pilot implementation to demonstrate three cognitive capabilities: action modeling, retroactive learning, and boosting other learning mechanisms.

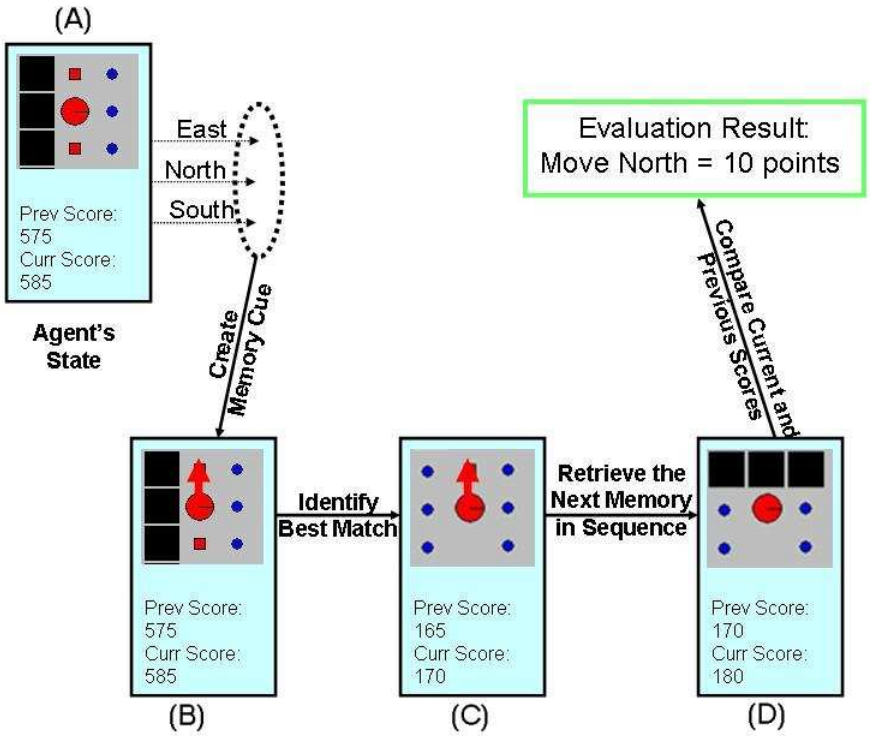

Figure 7-1: An Episodic Memory Eaters Agent (Pilot Implementation)

The agent used in all of these experiments followed this basic algorithm (refer to Figure 7-1):

1. The agent's current state includes its current sensing and its current score. The state also contains elements related to the agent's reasoning including a record of what its score was before the previous action. This state is represented by the box labeled (A) in the figure.
2. In any given state, the agent can move in at least two and as many as four different directions (north, south, east or west). The agent determines which of these actions are available and proposes an operator to move in each of these directions. In the figure, the agent can move east, north or south (it is blocked by a wall to the west).
3. Since the agent does not have sufficient knowledge to decide which direction to move next, it must evaluate each possible movement direction. It does this as follows:
    a. First, the agent creates an episodic memory cue that contains the agent's entire current state plus action to be evaluated. In other words, the agent is querying for an episodic memory of taking the to-be-evaluated action in a similar situation. In the figure, the box labeled (B) represents the cue the agent would create to evaluate the action "move north."
    b. Once the cue is created, the episodic memory system locates the episode that best matches this cue. An example of the identified memory is labeled (C) in the figure. Note that this memory is not a perfect match but the situation is similar enough in this case since the agent is moving north to a cell that has the same content. In other situations, the memory may not be similar enough and, ultimately, the agent is likely to make a bad decision as a result of this poor retrieval.
    c. Once the episodic memory system identifies the best matching episode, it retrieves the episodic memory of the state that occurred directly after the best matching episode. This is the episodic memory

that the agent sees as a result of the cue it created. An example of what this might look like is the box labeled (D) in the figure.

d. The agent then compares the current and previous scores recorded in the retrieved episodic memory. It uses the difference between these two scores as a quantitative evaluation of the movement direction being considered.

4. Once all available actions have been evaluated in this manner, the agent selects the action with the highest evaluation. (Ties are resolved by selecting randomly.)

7.2 Lessons Learned from the Pilot Implementation

This section describes each of the insights we gained from this work with the pilot implementation. As might be expected from a pilot implementation, many of these insights were fundamental in nature.

7.2.1 Definition of the Episodic Memory Space

The most valuable insight gained from the pilot implementation was its use as a guide for defining the scope of this research. What initially began as a catalog of important decisions made during the pilot's construction, evolved into the axes of what we now define as the space of episodic memory systems (which was introduced in section 2.1).

7.2.2 The Importance of Matching

This work with the pilot implementation in the Eaters environment quickly taught us that the most important stage in the episodic memory system is selecting the best match for a given memory cue. The manner in which memories are encoded and stored is important, but if the wrong episode is retrieved for an agent cue, agent decisions based upon that retrieval are suboptimal.

With the pilot implementation, we discovered that this decision to use Soar's built-in matching mechanism hurt the performance of episodic memory agents because of encoding specificity issues. To work effectively, the recalled episode had to exactly match the cue. Too many cue entries meant that the memory would never be recalled.

Too few entries would lead to too many matches and (usually) random behavior. Therefore, selecting the correct entries for the agents to place in the memory cue was difficult.

Even in the simple environment we used for most of this research, this requirement was difficult to meet. As a result of this insight, the next episodic memory system implementation used a partial matching algorithm. As is described in sections 7.4.1 and 7.4.2, the importance of an effective matching algorithm drove us to try several variations and additions to this algorithm as part of this research.

7.2.3 The Importance of Episodic Memory Sequences

In the Eaters environment, the agent is focused on the immediate result of its actions (i.e., the cognitive capability we call action modeling). The pilot implementation accomplished this by determining which memory best matched the cue but then retrieving the next memory (in chronological order) rather than the one that was the best match. This behavior combined with the agent's deliberate recording of its previous score at each step allowed the agent to evaluate the outcome of a particular action.

In addition to requiring the agent to record critical information, this approach made it difficult to use the system for some of the other cognitive capabilities since a best-matching episodic memory could not be directly retrieved. A better way to acquire the same information is to retrieve a sequence of two episodic memories. The first memory is retrieved when the agent constructs a cue consisting of the agent's current situation and a proposed action. The agent then issues a command or special query to the system in order to determine what happened next.

This approach (which was used in future implementations) not only allows the agent to directly retrieve an episode, but also to retrieve sequences of episodes bounded only by the extent of the episodic store. As will be shown in section 8.5, this functionality is essential for an agent to learn from previous successes and failures in the TankSoar environment.

7.2.4 The Importance of an Autonoetic Episodic Memory

To create an episodic memory cue, the pilot implementation required the agent to create a special Soar state for that purpose. The retrieved episodic memory would then overwrite the current state. While this approach is parsimonious, it ended up creating multiple problems due to the fact that productions that were designed to match against a state would also match against a retrieved memory. If the agent became confused in this manner, its behavior usually became irrevocably aberrant. In effect, we were violating the autonoetic property of episodic memory described in section 2.2.

As a result of this insight, subsequent episodic memory systems created separate, reserved areas in working memory for the cue and retrieved episodic memory.

7.3 Baseline Implementation

The lessons learned from the pilot implementation were applied to what we now call the baseline implementation. The most significant difference between these two implementations is that the baseline implementation uses an independent episodic memory store rather than using Soar productions.

Several changes stem from this difference. Most importantly, a separate memory store allowed us to implement a partial matching algorithm that eliminated the encoding specificity issues we encountered in the pilot implementation. Given what we had learned about the importance of matching in the episodic memory process, a custom matching algorithm allowed us to explore multiple approaches. In addition to the approach used by the baseline implementation, we discuss two major modifications in section 7.4.1 and 7.4.2.

This implementation is called baseline because it forms the core of all future modifications we have made to the episodic memory system. In other words, if we view this research as a search through the space of implementations, the baseline implementation is near the center of all the points we have explored since the pilot implementation.

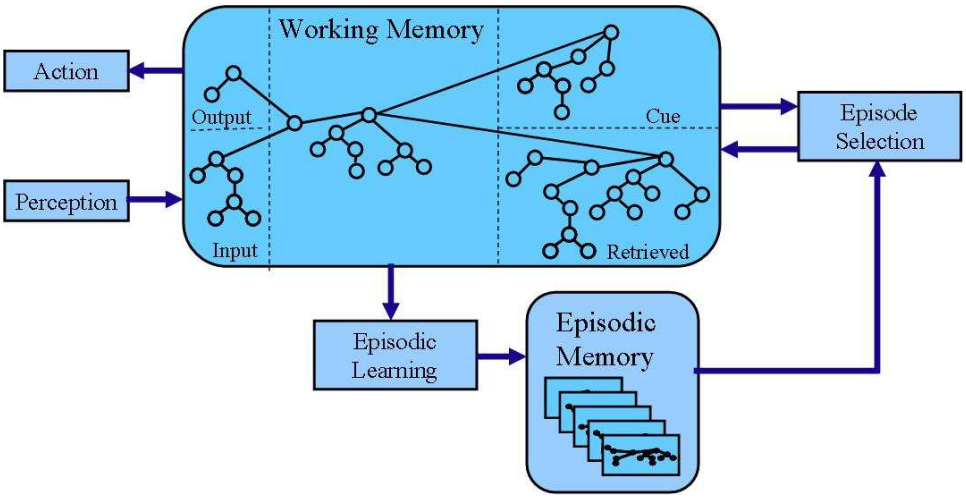

Figure 7-2: Baseline Implementation Architecture

Figure 7-2 depicts an architectural diagram of the baseline implementation of episodic memory. The central rectangle in the figure represents Soar's working memory. The nodes in the rectangle represent the working memory elements[3]. Working memory contains some reserved areas that contain the agent's input (perceptions) and output (actions). The baseline implementation created two special areas in working memory for episodic memory cue construction and episodic memory retrieval respectively. These areas are strongly reminiscent of the buffers used by ACT-R 5.0 (Anderson and Labiere 1998). These reserved areas are delineated in the figure with dashed lines.

The baseline implementation includes an episodic learning module that monitors the agent's behavior and, at prescribed times, records a new episodic memory. This memory is effectively a snapshot of working memory taken at the time of recording. This snapshot does not contain elements that are in the areas of working memory reserved for cue construction and episodic memory retrieval.

If at any time the agent wishes to retrieve an episodic memory, it constructs a cue in the area reserved for that purpose. The cue nominally consists of working memory elements that the agent would like to be in the retrieved episode.

Once a cue is constructed, the episodic selection routine compares the cue to the stored episodic memories and determines which memory best matches the cue. (This

[3] An actual Soar agent is likely to have many more elements than are depicted here.

happens at the end of every cycle that a cue is present in working memory.) The selected episode is then added to working memory by reconstructing a copy of the original working memory in the reserved area of working memory.

The baseline implementation falls into the design framework as follows:

Encoding

- **Encoding initiation** – A new memory is encoded each time the agent takes an action in the world. The approach used by the pilot implementation — recording new episodes when there was a significant change in working memory — led to the same frequency of recording. This straightforward approach was simply more efficient and easier to implement. This new approach has since proved effective for multiple tasks and, as a result, we have not changed it.
- **Episode determination** – For the baseline implementation, we abandoned the task dependent approach used by the pilot implementation. The content of an episode now consists of a large portion of the agent's working memory which includes its input (sensing), internal data structures and output (actions in the world). Each working memory element is selected or rejected for an episodic memory based upon its activation level. WMEs which are recorded in a new episodic memory are those whose activation level has not decreased to the point where the element would be removed in a strict psychological model[4].
- **Feature selection** – Since the baseline implementation incorporates a partial matching algorithm, we elected to allow all features of the episode to participate in retrieval.

Storage

- **Episode structure** – The episodic store was completely re-implemented for the baseline implementation which consists of two structures: a working memory tree and a set of episodic memories (refer to Figure 7-3):

[4] Working memory activation was originally added to Soar to support memory decay models. Thus, when a particular working memory element dropped below an activation threshold it was removed from working memory. In this experiments, memory activation was active but memory decay was not used.

- The working memory tree holds a single instance of all elements that have ever been in an episodic memory throughout the agent's existence. By using a tree structure, we preserve most[5] of the structure of each episode as it existed in working memory. This reduces the processing time required to store and reconstruct an episodic memory as compared to a system that recorded the entire structure. This benefit comes at the cost of a loss of some detail in retrieved memories.
- Each episodic memory consists of a list of pointers to each of the working memory elements it contains, in a canonical order. In addition, all elements in the tree have pointers to the memories that contain them.

In section 7.4.2, we discuss an alternate approach to the structure of the episodic store and an associated alternate matching algorithm.

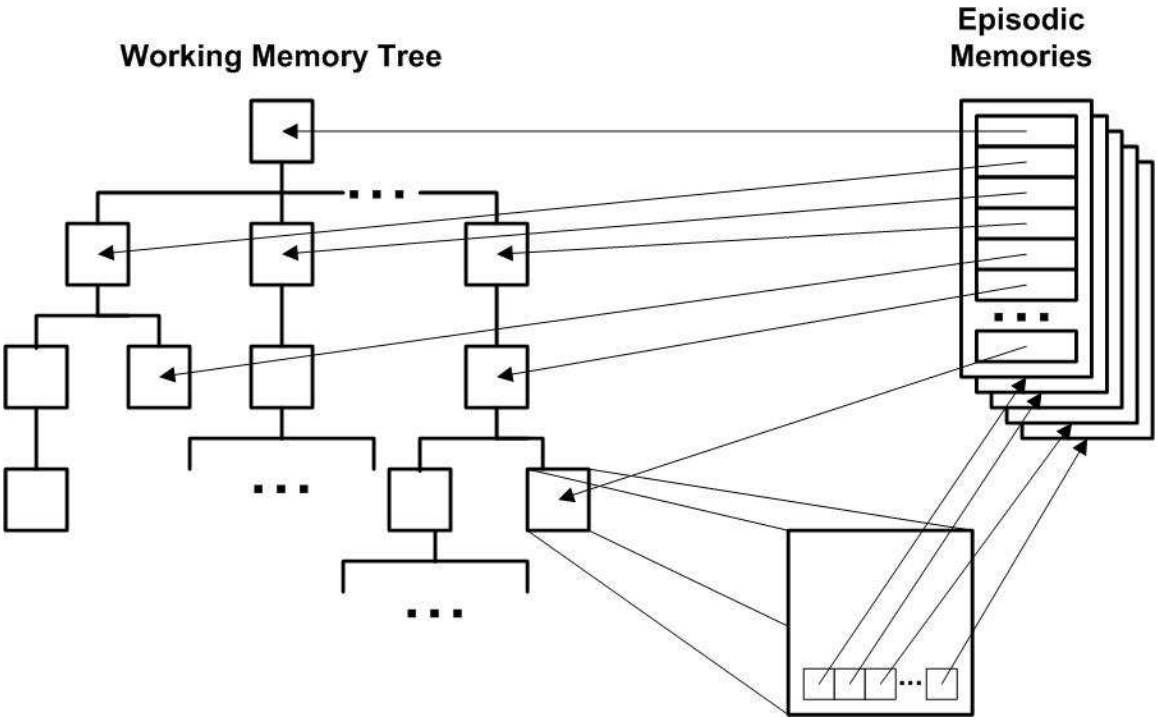

Figure 7-3: Episodic Memory Data Structures

[5] Two things are lost with this approach. First, since a Soar state is a rooted graph (not a tree) recursive links between elements are skipped in order to force the graph into a tree. Second, multi-valued attributes with non-constant values are stored in the tree as a single WME.

- **Episode dynamics** – The baseline implementation still has no episode dynamics. The most obvious dynamic is a forgetting mechanism. However, we performed some rough performance measurements that indicated forgetting is unnecessary for the environments we have selected. As a result, we have chosen to ignore episode dynamics in favor of other experiments. Nonetheless, forgetting remains a prime candidate for future explorations of the episodic memory space (see section 9.2.4.2).

Retrieval

- **Retrieval initiation** – Just as in the pilot implementation, retrieval is triggered when the agent deliberately constructs a cue in working memory.
- **Cue determination** – A cue is constructed by the agent (using rules) in a reserved location in working memory. This location no longer corresponds to an actual Soar state but instead is a link similar to the output-link used for I/O (an established paradigm in the Soar architecture). The cue can have any number of working memory elements. For purposes of the match, the root of the link corresponds to the root of the state.
- **Selection** – During episodic retrieval, the cue is compared to all stored episodes in order to select the episode that "best matches" the cue. In the baseline implementation, the best match is determined by totaling the number of working memory elements that are shared between the cue and the episode. Once an episode has been retrieved, the agent can also retrieve the next episode in temporal order via a special "next" command. This allows the agent to retrieve sequences of episodes of indefinite length.

  In section 7.4.1, we discuss the effectiveness of using working memory activation to bias the match. In section 7.4.2, we discuss an alternate implementation of the baseline matching algorithm.
- **Retrieval** – The complete episode is retrieved in a reserved area of working memory to avoid confusion with the current state of the agent. As with the episodic memory cue, the root of the retrieval area corresponds to the root of the state from which the memory was originally recorded.

- **Retrieval meta-data** – The baseline implementation had no metadata that was retrieved with an episode. However, as part of this research we have added several different types of meta-data (see section 7.4.3.2).

Use

The remainder of the research took place within the baseline implementation or (more frequently) with additions and modifications made to it. The remainder of this chapter introduces the agents we used for these experiments and discusses the results of experiments in which we tested the modifications. Chapter 8 describes how we demonstrated multiple cognitive capabilities with this implementation.

7.4 Additions to the Baseline Implementation

As part of this exploration of the space of episodic memory system implementations, we experimented with several modifications to the baseline implementation. An underlying question of this research was: Given that you have an imperfect memory system, what are the techniques you can use to overcome that and how effective are they? However, the large number of modifications, environments, tasks and agents made it impractical to test all possible combinations. To cope with this complexity, our experiments reduced the question to: Does a particular agent performing a particular task in a particular environment perform better or worse with a given modification? Modifications that were effective were usually kept for future experiments and modifications that were ineffective were removed and set aside for future work. Thus, this search was, in some ways, a hill climbing approach.

This section describes, in roughly chronological order, the modifications that were tried and their outcomes.

7.4.1 Improving Episodic Memory Selection

The baseline implementation included a new partial matching algorithm used in episodic memory selection. Effective partial matching is an issue faced by many researchers in artificial intelligence tasks. Some approaches to this task include:

- **Nearest Neighbor** – In the purest form of this approach, the features of the query are compared to each instance. The instance which has the most features in

common with the query is selected for retrieval. Nearest neighbor is processing intensive and is also highly sensitive to the amount of irrelevant features in the instance or query. As a result, a feature weighting heuristic is often applied to focus the search on a few particular features of the instances or query. A thorough review of nearest neighbor algorithms can be found in (Dasarathy 1990).

- **String Matching** – Recent advances in rapid string matching (all based on Smith and Waterman 1981) focus on providing a near-best match rapidly. If the query and instance can be condensed into a string of symbols, then these algorithms can effectively be used for partial matching. The conversion to a string, however, can result in loss of information (Pardo, et al. 2004).
- **Hashing** – Hash tables can be used for partial matching provided an effective hash function can be found for the query and instances. As with string matching, there is a risk of too much information being lost via the hash function (Burkhard 1979).
- **Classifier Systems** – Various approaches to classifiers can also be applied to partial matching. In effect, the individual instances become categories for the classifier to assign the query to. Examples of classifier algorithms applied to partial match include rule induction (Gryzmala-Busse and Wang 1996), N-Gram matching (Tian, et al. 2006) and Bayesian learning (Meek and Birmingham 2004).
- **Hybrid Approach** – The combination of a fast inaccurate approach with a slow accurate approach can result in a sum that is better than its parts. In particular, classifier systems can be used to narrow the field for a nearest neighbor search (Cercone, et al. 1999).

Although we weren't aware of all this options at the time, we selected a nearest neighbor search for this baseline implementation because it offered the most accuracy. As we learned from this pilot implementation, retrieving the correct memory is a critical requirement for an effective episodic memory system (see section 7.2.2). We began with a simple nearest neighbor approach with no feature weighting.

7.4.1.1 The Baseline Eaters Agent

This first experiment with this new implementation was to implement an Eaters agent and compare its behavior to the pilot. This new Eaters agent used the following algorithm (refer to Figure 7-4):

1. The agent's current state includes its current sensing and its current score. The state also contains elements related to the agent's reasoning. This state is represented by the box labeled (A) in the figure.
2. In any given state, the agent can move in at least two and as many as four different directions (north, south, east or west). The agent determines which of these actions are available and proposes an operator to move in each of these directions. In the figure, the agent can move east, north or south (it is blocked by a wall to the west).

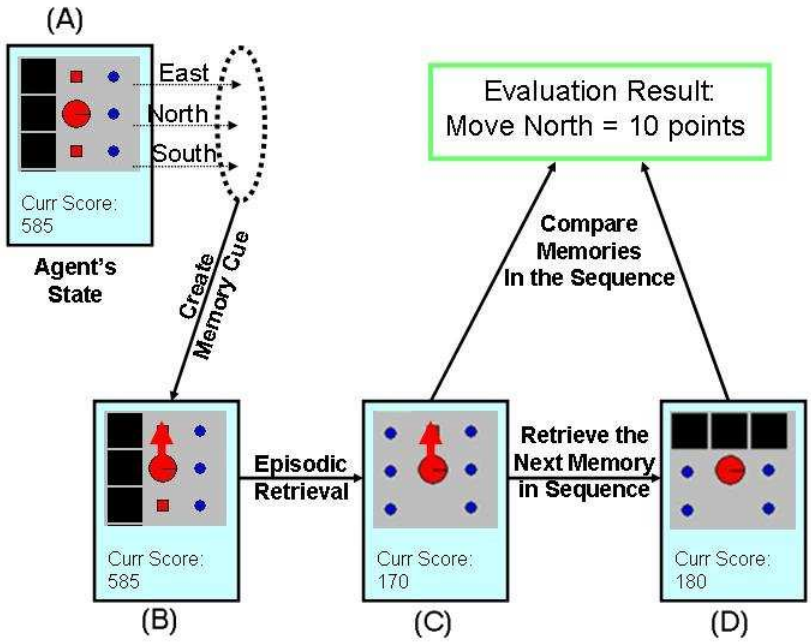

Figure 7-4: An Episodic Memory Eaters Agent (Baseline Implementation)

3. Since the agent does not have sufficient knowledge to decide which direction to move next, it must evaluate each possible movement direction. It does this as follows:
    a. First, the agent creates an episodic memory cue that contains the agent's entire current state plus action to be evaluated. In other words, the agent is querying for an episodic memory of taking the to-be-evaluated action in a similar situation. In the figure, the box labeled (B) represents the cue the agent would create to evaluate the action "move north."
    b. Once the cue is created, the episodic memory system locates the episode that best matches this cue and retrieves it for the agent to examine. An example of the retrieved memory is labeled (C) in the figure. Note that this memory is not a perfect match but the situation is similar enough in this case since the agent is moving north to a cell that has the same content. In other situations, the memory may not be similar enough and, ultimately, the agent is likely to make a bad decision as a result of this poor retrieval.
    c. The agent records the score that it had in that episode and then requests the memory that occurred next in temporal order. The episodic memory system responds to this command and retrieves the outcome of that situation and action. In the figure, this second retrieval is represented by the boxed labeled (D).
    d. The agent then compares the scores from both memories and uses the difference between these two scores as a quantitative evaluation of the movement direction being considered.
4. Once all available actions have been evaluated in this manner, the agent selects the action with the highest evaluation. (Ties are resolved by selecting randomly.)

7.4.1.2 Comparing the Pilot and Baseline

While the baseline implementation's partial matching immediately alleviated this encoding specificity issues, the baseline agent's performance did not improve. Figure 7-5 compares the behavior of a typical Eaters agent from the pilot and baseline implementations. In each case, the agent begins with no episodic memories but records more and more of them as it takes more and more actions in the world. The x-axis is the number of successive evaluations that the agent has made of a given action in a given situation. The y-axis indicates the fraction of the last ten evaluations that the agent attempted which resulted in success. The dashed line at the bottom of the graph indicates the predicted performance of random behavior. Due to the nature of the logging mechanism used with the pilot implementation, data gathered from that agent is sparser than from the baseline. The data in this graph for both agents is an average of five distinct runs.

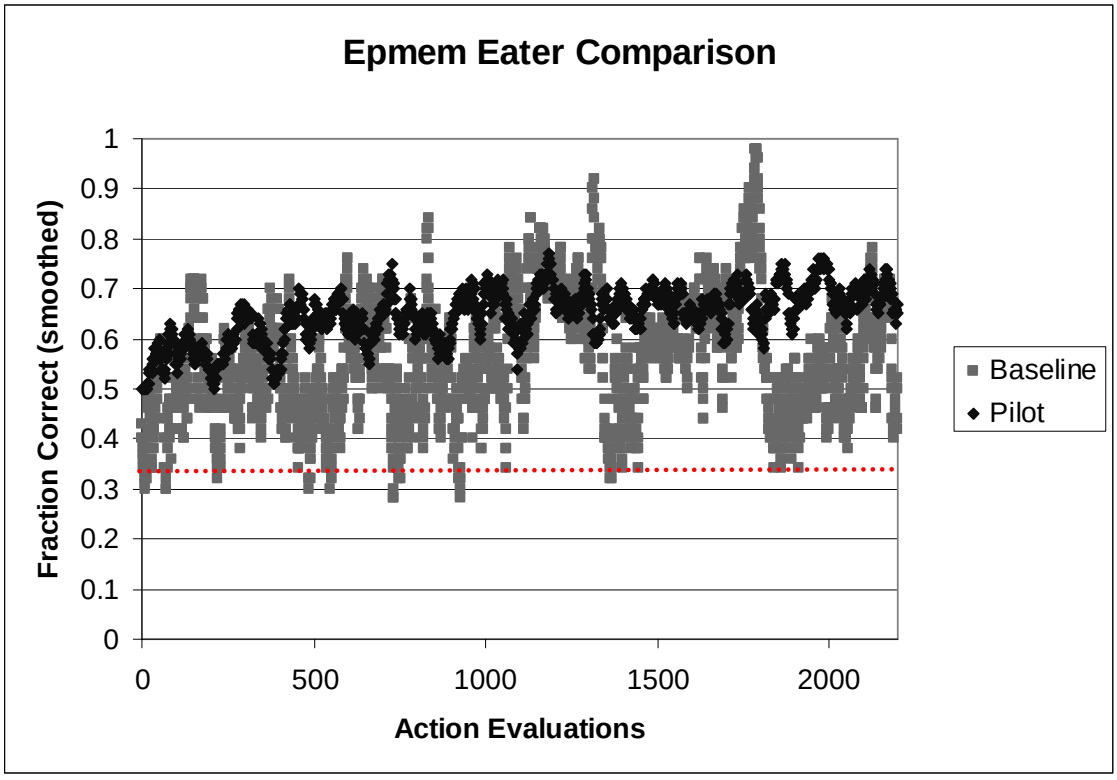

Figure 7-5: Comparison of Eaters Agents for the Pilot Implementation vs. the Baseline Implementation

Examination of this data indicates that the baseline agent's performance was somewhat worse as well as more erratic. Given that the pilot implementation was using a task-specific method for episode determination and feature selection while the baseline's approach is task independent, we hypothesized that the lack of this knowledge in the baseline implementation was the cause of the poor performance.

7.4.1.3 Analysis of Baseline Memory Selection

Close examination of agent behavior confirmed the hypothesis that the baseline implementation was suffering from lack of knowledge. Without task-specific information, the new Eaters agent was required to create episodic memory cues that contained all of its current sensing as well as its proposed action. These memory cues typically contained between 31 and 36 features[6]. However, unbeknownst to the agent, only two of those features are actually relevant to the task: the direction that the agent is considering moving and the content of the cell in that direction. Given so many features to work with, the "best" matching memory often did not contain a match for those two features. The agent's behavior suffered as a result.

This observation combined with what we learned from the pilot implementation about the importance of matching led us to devote repeated efforts to understand and improve the task independent matching algorithm.

7.4.1.4 Feature Selection with Working Memory Activation

Given more than thirty features in the memory cue used by the agent in the Eaters task, the "best" episodic memory selected by the baseline matching algorithm would often fail to match one or both of the most relevant features from the cue. This, in turn, led to poor decisions by this agent. More generally, as the ratio of relevant to irrelevant entries in the cue decreased the likelihood of an effective retrieval also decreased.

---

[6] Specifically, features are working memory elements whose values are leaves of the cue's tree structure. These leaf WMEs are the only ones that directly affect that match since the information provided by the non-leaf WMEs is implicit in the identity of the leaf WMEs.

To counteract this, we needed a mechanism to weight the features of the cue to bias the retrieval toward memories that were likely to be relevant to the task. An excellent review of feature weighting for nearest neighbor algorithms can be found in Wettschereck, et al. (1997). From that paper, feature weighting varies upon these dimensions:

- **Preset vs. Performance Bias** – In some algorithms the feature weights are set at agent creation and do not vary. In others, the feature weights are adjusted based upon agent performance.
- **Feature Weighting vs. Selection** – In some algorithms, only a small subset of features is selected for retrieval. Feature weighting subsumes feature selection if the only weights that can be applied are 0 and 1.
- **Given vs. Transformed Representation** – In some algorithms, the features of the instances and query are used as is. In others, they are transformed into another set of features that capitalize on correlation between the given features.
- **Global vs. Local Weights** – In some algorithms, weights are applied globally to all features with the same attribute. In others, the weights vary from instance to instance.

Because selection occurs in the architecture, the most important criterion was for this weighting mechanism to be task-independent. We had had some success with the pilot implementation using working memory activation to select cue entries for retrieval. In effect, the most activated WMEs seem to reflect the agent's focus and, thus, we hypothesized that it also measured what was relevant to the current task.

We modified the baseline implementation to record the activation level of each WME in an episodic memory as its feature weight. At selection, these activation levels were summed to create a match score for each episodic memory and the episode with the highest match score was retrieved. According to the criteria presented by Wettschereck et al., (1997) this approach is performance biased (activation levels are determined at run time), feature weighted (no features are excluded during the match), uses the given features without transformation and applies the weights locally.

As far as we have determined, this approach to feature weighting is unique. The most closely related research involves using explanation-based learning to guide feature

selection in case-based reasoning (Cain, et al. 1991). In addition, the ACT-R architecture (Anderson and Labiere 1998) uses activation to select among matching chunks during retrieval from semantic memory.

Figure 7-6 compares the behavior of the agent before and after this activation-based feature weighting was added. As with Figure 7-5 above, the x-axis is the number of successive evaluations that the agent has made of a given action in a given situation and the y-axis indicates the fraction of the last ten evaluations that the agent attempted which resulted in success. The data in this graph for both agents is an average of five distinct runs.

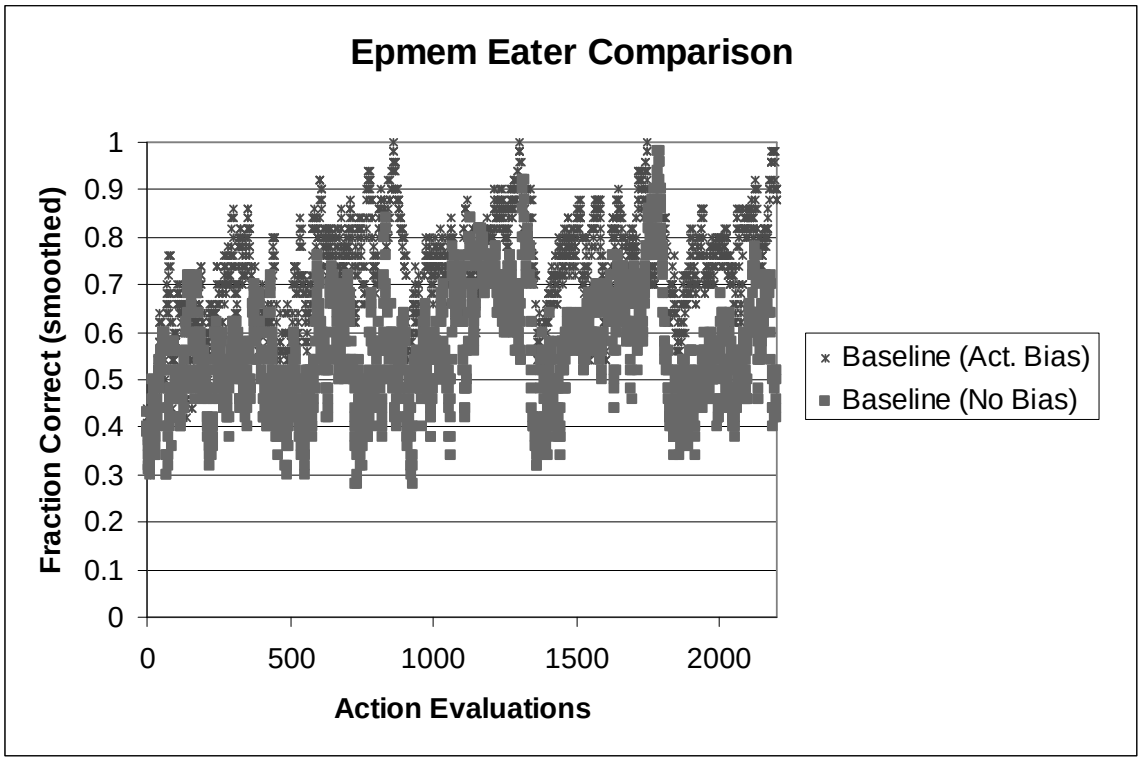

Figure 7-6: The Effects of Activation Bias on the Eaters Agent

This approach created a clear improvement in the agent's action evaluations that, in turn, were directly dependent up on the quality of retrieved memories.

7.4.1.5 Improvements to Working Memory Activation

Success using working memory activation as a feature weight led us to examine Soar's working memory activation algorithm more closely. We made two observations about the design of the activation system that led us to believe we could improve its ability to identify salient features of an agent's state.

The first observation was that the activation system was designed to ignore instantiation supported (i-supported) WMEs. (Recall that these are WMEs that are created via elaboration rules and remain on the state as long as the relevant rule continues to match.) The working memory activation implementation was originally accomplished by James (unpublished) which was in turn based upon research by Chong (2003) and Anderson & Labiere (1998). In all cases, the memory activation had been implemented as part of a memory decay mechanism. Removing an instantiation supported WME due to decay would have no effect since the relevant production would simply fire and recreate it. Thus, for James' purposes activation on i-supported WMEs was unnecessary.

For the purposes of agent focus, the activation level of i-supported WMEs also seemed unnecessary since the existence of architectural and operator supported (o-supported) WMEs that led to the i-supported WME's creation would have activation. (Recall that o-supported WMEs are created by operators and persist indefinitely.) Upon closer examination, we realized that i-supported WMEs can "mask" the activation of the underlying o-supported WMEs responsible for the creation of the i-supported WMEs. Figure 7-7 shows a graphical depiction of this situation. Consider the case where a production tests an i-supported WME. That WME does not receive any activation boost since it is not activated. However, the o-supported WME(s) that were tested in order to create that WME *also* do not receive a boost. Thus i-supported WMEs are masking o-supported WMEs from activation.

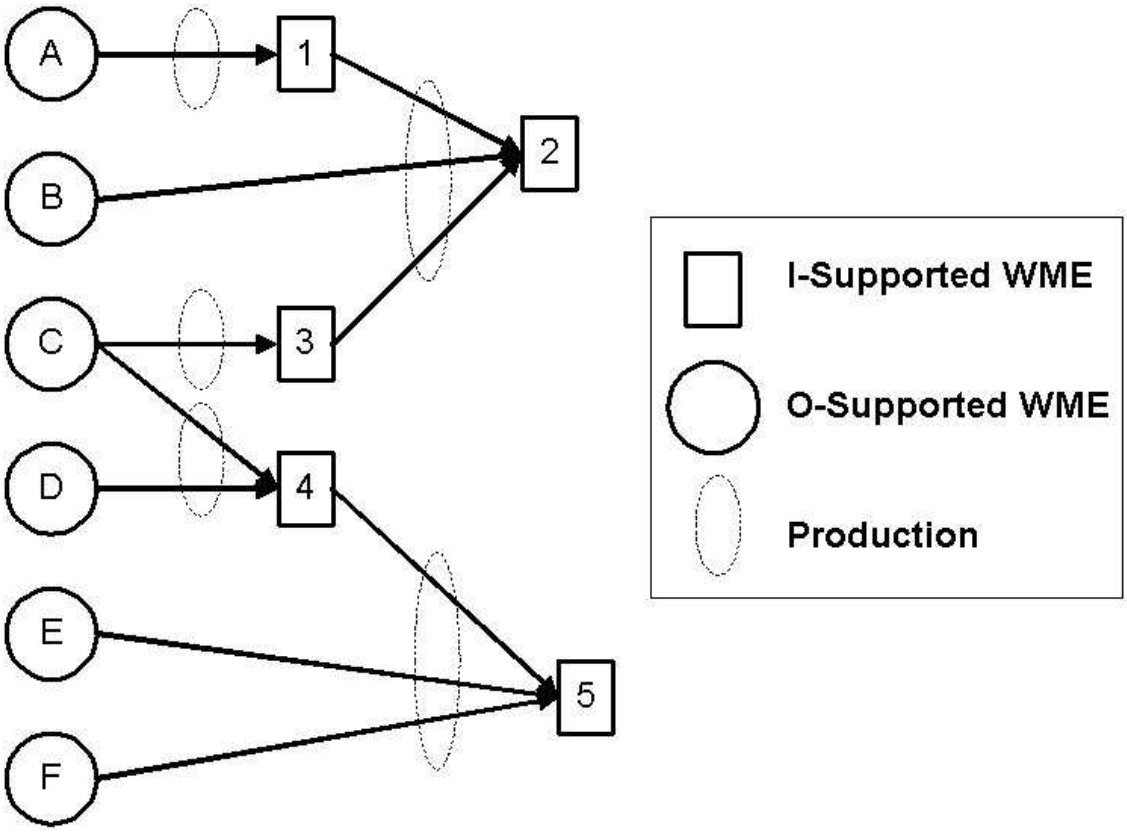

Figure 7-7: Example of Activation Masking in Working Memory

In response to this problem we added a "pay it backward" approach for passing references from i-supported WMEs to o-supported ones. The activation system now calculates the *set of support* for any referenced i-supported WME and then boosts the activation of WMEs in the set. In Figure 7-7, the set of o-support for WME 5 includes WMEs E and F, which are directly tested by the rule that creates 5, as well as C and D, which are indirectly tested by the WMEs that create 5 (via WME 4).

The second observation we made was in regard to the initial activation for newly created WMEs. In the original implementation, these WMEs receive a fixed initial boost equivalent to a single reference. In terms of decay and removal, this is sufficient because newly created WMEs will usually immediately lead to additional rule firings and thus the WME will receive a boost in activation. If, instead, a newly created WME is not relevant to the situation it will not be tested and, thus, will be removed after a short time. However, when using memory activation as a measure of the importance of features of

the agent's state, this flat level of initial activation can be misleading. An agent might test multiple WMEs with high activation and create one new WME that would have much lower activation. Thus, at creation time – when this feature will most likely be very important as a cue for future retrieval – its activation is much lower than the activation of the features that led to its creation.

This may, in fact, be a new manifestation of an old problem. Chong noted that in both his model and some ACT-R models, newly created WMEs/chunks can decay too rapidly and, as a result, never have a chance to participate in reasoning. To ameliorate this problem, the second modification we implemented was a "pay it forward" approach for setting the activation level of new WMEs wherein the activation of a new WME is based on the activation levels of its set of support.

Figure 7-8 depicts the Eaters agent's behavior before and after these modifications had been applied. As with Figure 7-6 above, the x-axis is the number of successive evaluations that the agent has made of a given action in a given situation and the y-axis indicates the fraction of the last ten evaluations that the agent attempted which resulted in success. The data in this graph for both agents is an average of five distinct runs.

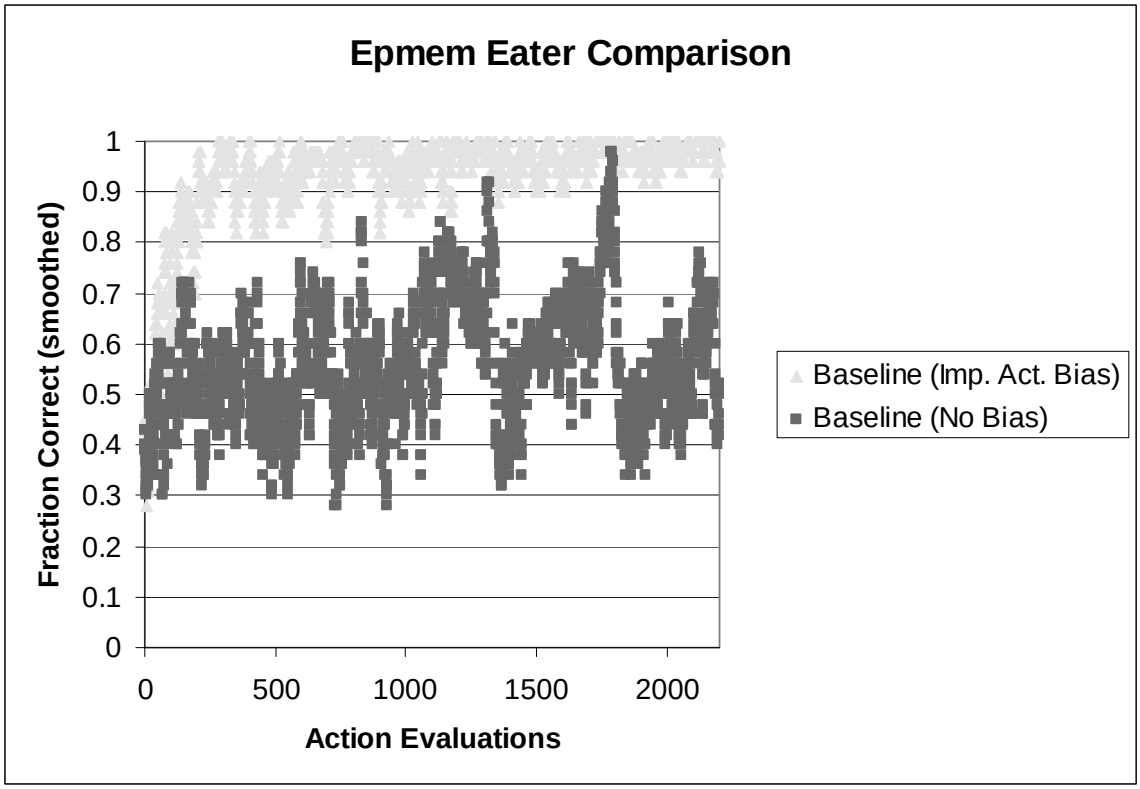

Figure 7-8: The Effects of Improved Activation Bias on the Eaters Agent

This data strongly suggests that working memory activation is an effective method for feature weighting in a task independent episodic memory system.

7.4.1.6 Balancing Activation Bias vs. Match Cardinality

Success with activation as a feature weighting mechanism led us to leave that functionality in the baseline implementation for future research. However, when we began working in the TankSoar environment (described in section 6.2.2), we began to see instances where the system was retrieving an incompletely matching episode when a complete match to the given cue was available. This behavior was because the sum of the activation level on the lesser match was higher despite the missing elements. Clearly, there is a conflict between the activation level of the match and the overall cardinality of the match.

As we furthered our investigation, we noted that some of the TankSoar agents we created used episodic memory cues with as few as four features. (For a full description of the TankSoar agent used in this investigation, see section 8.5.) Since these cues were much smaller than that used by this Eaters agent, it was much more likely that a

particular episodic memory would be an exact match to a given cue. However, due to the activation bias on the match, a non-exact match to a more highly activated episode would sometimes be selected instead.

As a result of this observation, we modified the baseline episodic memory system to calculate two match scores. One was based upon the sum of the activation levels of all matching features. The second was based upon the strict cardinality of the match. These two scores would then be combined at a user defined ratio to determine a final match score. A ratio of 1.0 would weigh each score equally. A ratio below 1.0 weighs the activation level more heavily. A ratio above 1.0 weighs the cardinality of the match more heavily.

Using this new functionality, we compared the performance of multiple TankSoar agents. Each agent was identical except for the value of this cardinality vs. activation ratio.

Figure 7-9 depicts the performance of this TankSoar agent with each setting of the cardinality vs. activation ratio. The agent begins with no episodic memories but records more of them as it plays more games (the x-axis). The y-axis is the agent's average margin of victory for each game given a specific number of previous games played. A negative margin of victory represents a losing game and a positive margin of victory is a winning game. For all curves, the data presented is the average of ten independent runs of the same agent.

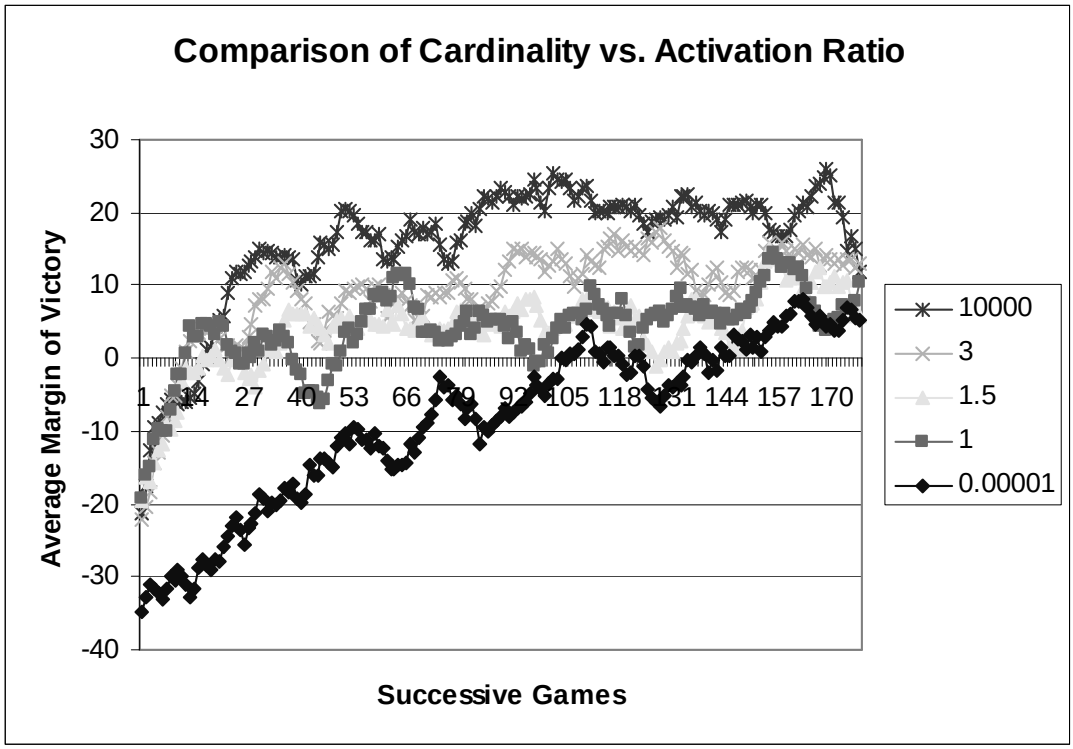

Figure 7-9: Effects of Various Settings of the Cardinality vs. Activation Ratio in the TankSoar Environment

Figure 7-10 depicts a summary of the overall results of Figure 7-9. Each column in the graph depicts the average overall margin of victory of the agents with the cardinality vs. activation ratio specified on the x-axis.

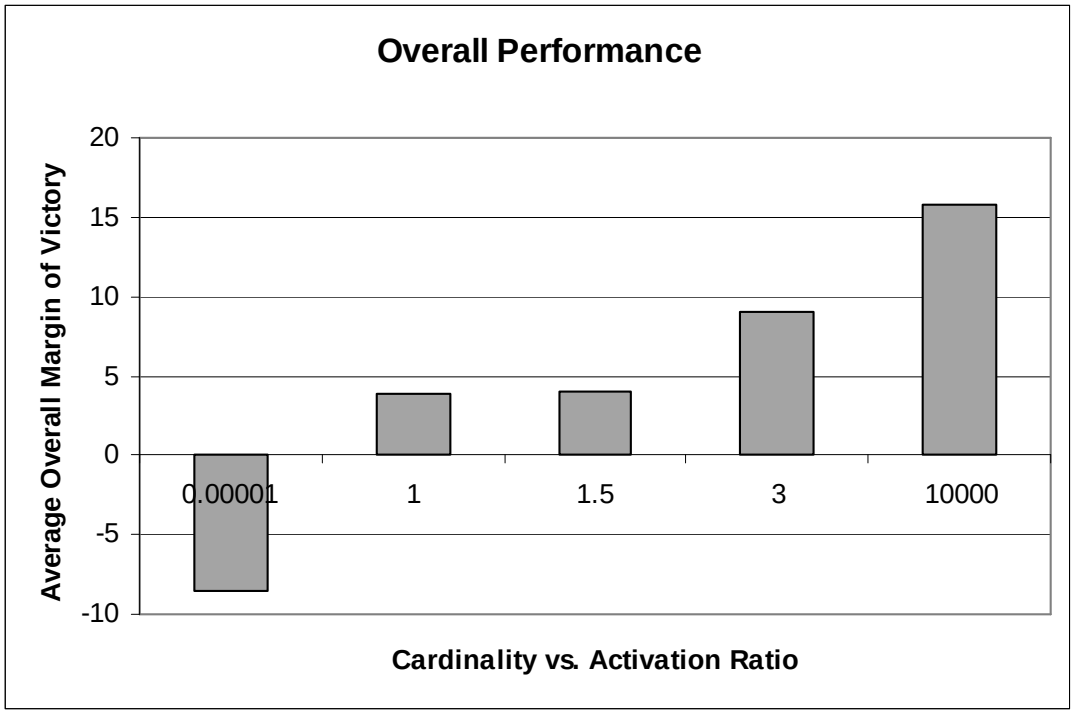

Figure 7-10: Summary of the Impact of Cardinality vs. Activation Ratio (Small Cue)

As the graphs show, the TankSoar agent learned faster and performed better when the cardinality of the match outranked the sum of the activation level of the match. These results are in contradiction to the activation bias results with the Eaters domain. Given that the only thing that was varied in this particular experiment was the cardinality vs. activation ratio we can hypothesize that the essential difference between these two agents was the sizes of the respective cues and episodic memories. The Eaters agent has a large cue (approximately 35 features) and a comparable episodic memory size (approximately 50 features). The TankSoar agent has a small cue (approximately 9 features) and a much larger episodic memory size (approximately 120 features).

To test this hypothesis we modified the TankSoar agent to use a much larger cue (approximately 55 features) and repeated the experiment with the same values for the cardinality activation ratio. The results are shown in Figure 7-11. While the extraneous cue entries hurt the performance of the agent, it is readily apparent that retrievals based upon large cues perform better when the activation bias is strong.

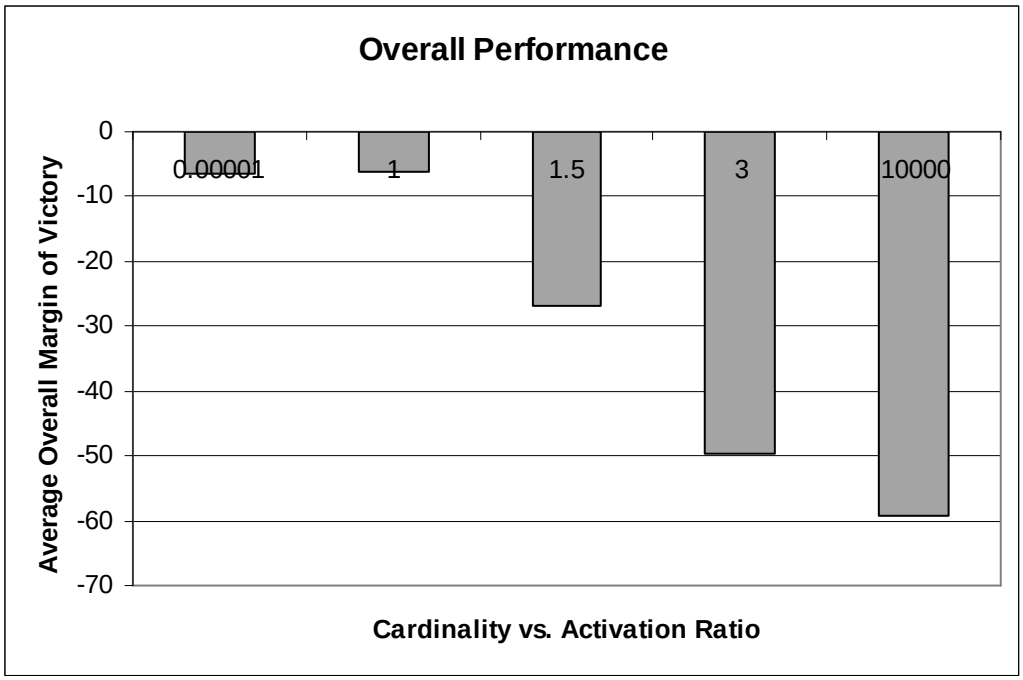

Figure 7-11: Summary of the Impact of Cardinality vs. Activation Ratio (Large Cue)

These data show clear evidence that a key factor in the matching algorithm for a task independent episodic memory system is to respond to the size of the cue and the episodic memory when performing episode selection. An obvious approach to this is to adjust the cardinality vs. activation ratio based upon the size of the retrieval cue. More data from more agents and environments will need to be gathered before this approach can be proven.

7.4.2 Improving Performance: Interval Based Matching

The impact that the episodic memory system has on the surrounding architecture is a crucial consideration. In section 2.3, one of the requirements we defined for an effective episodic memory implementation is low resource demand. To that end, we performed a close examination of the impact of the baseline episodic memory system on Soar performance.

7.4.2.1 Instance-Based Matching Algorithm

Initial profiling of the baseline system indicated that the largest fraction of processing time was spent during episode selection. As a result, we started with the baseline

implementation's data structures and matching algorithm (which we will henceforth refer to as the instance-based algorithm). Refer to Figure 7-12 which is a replica of Figure 7-3 from section 7.3. This algorithm has the following steps:

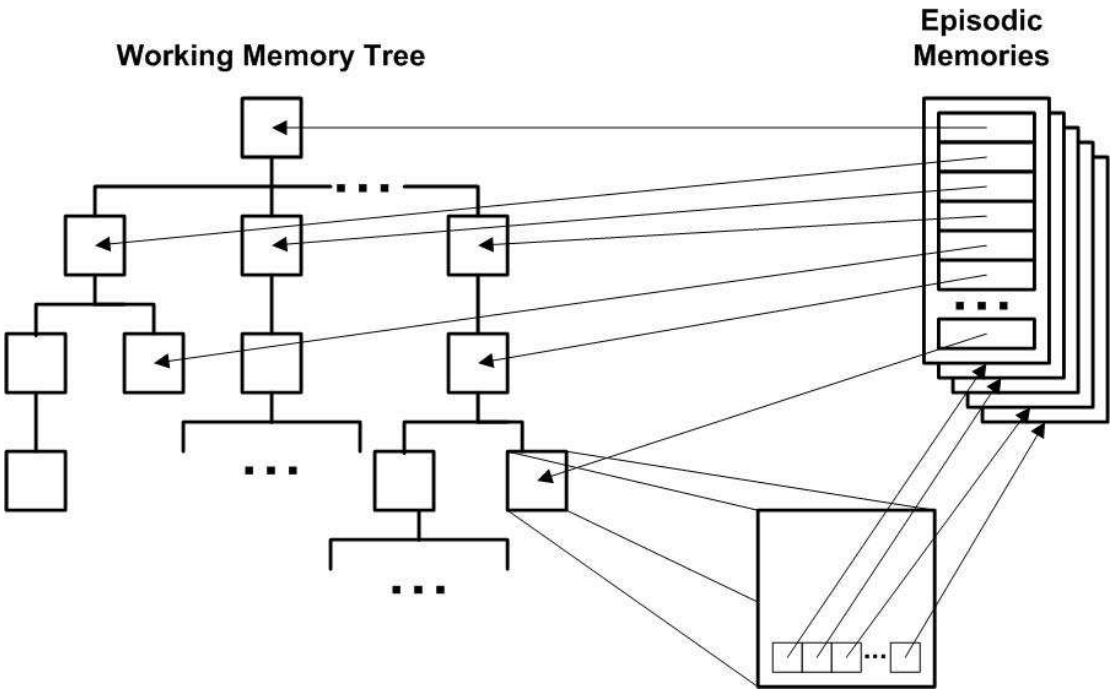

Figure 7-12: Data Structures used by the Instance-Based Algorithm

1. The system simultaneously traverses the given episodic memory cue and the working memory tree. For each entry in the cue, the corresponding entry in the working memory tree is located.
2. Each matching entry that is found in the tree contains a list of references to every episode that contains it. A list of all episodic memories that match at least one entry from the cue is created by merging the references from each matched part of the cue.
3. The complete cue is then compared to each episodic memory in the newly created episodic memory list and the one that best matches the cue is selected.

This algorithm requires O(nm) comparisons to find the best match to a given cue (where 'n' is the number of entries in the cue and 'm' is the number of episodes that

entries of the cue appear in). Note that the size of the cue will be much smaller than the number of episodes and does not grow over time. As a result, we expect the growth in processing time to grow linearly over time.

While the cardinality of 'm' could equal the size of the entire episodic store, in practice it is much smaller because a given entry in the cue is unlikely to appear in all the stored episodes. Moreover, the size of the cue limits the number of memories that need to be directly compared to the cue. In addition, if the agent is moving from task to task, so that the number of episodes with common features does not grow, the time required will be much smaller than O(nm) implies.

7.4.2.2 Measuring the Performance of the Instance-Based Match

Figure 7-13 depicts the memory usage required for the baseline implementation using the instance-based match in the Eaters task and how it is distributed among the various data structures used by the system. Figure 7-14 shows the same graph for the TankSoar task over a longer span of time. In both graphs, the agent begins with no episodic memories and the y-axis measures the amount of memory used as the agent gains more and more memories.

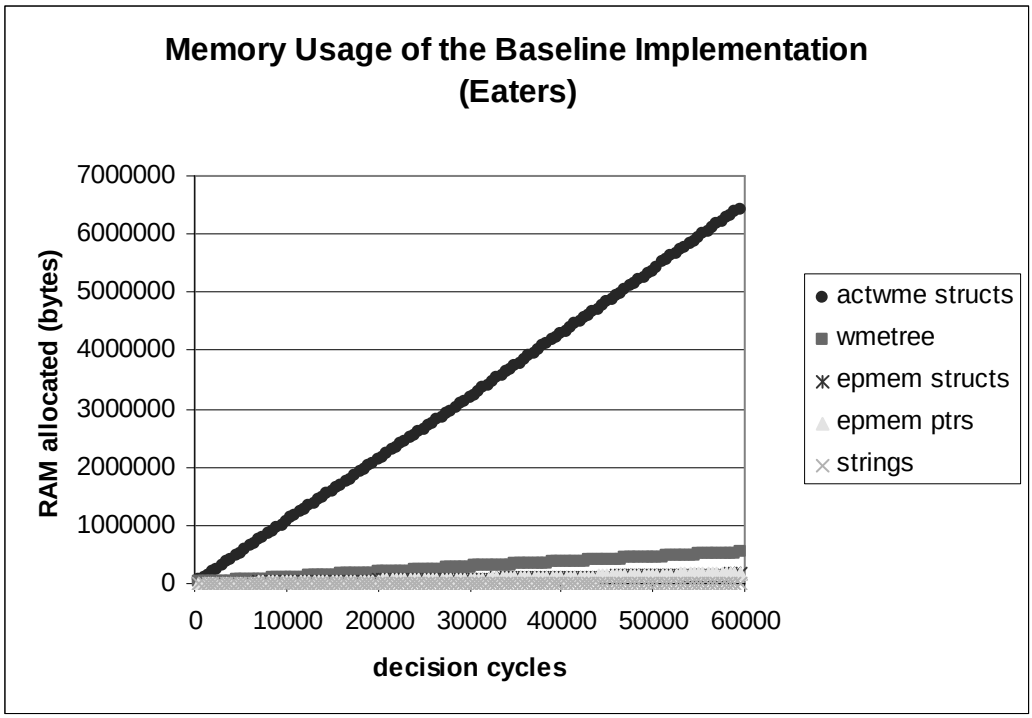

Figure 7-13: Baseline Implementation Memory Usage (Eaters)

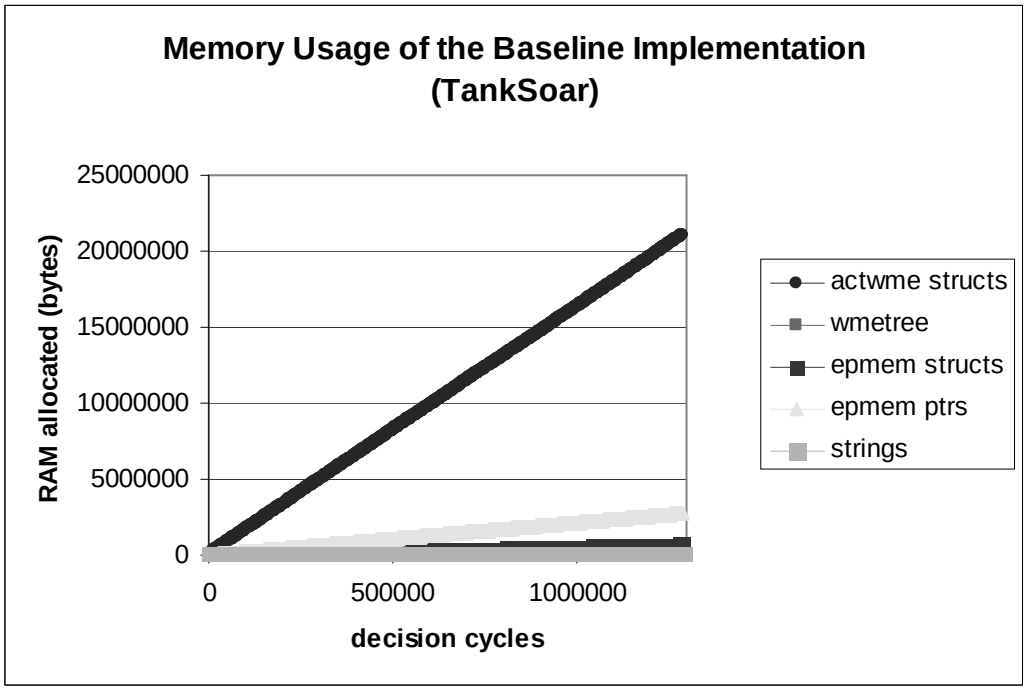

Figure 7-14: Baseline Implementation Memory Usage (TankSoar)

Predictably, the memory usage grows linearly as more and more memories are added to the episodic store. In particular, memory spent on "actwme structs" are the predominant contributor. Lists of these structures comprise an episodic memory in the baseline system. (These structures contain a pointer to a node in the working memory tree along with an associated activation.) In terms of the graphs shown in Figure 7-13 and Figure 7-14 (above), the vast majority of storage is used for the episodic memories and not the working memory tree.

Figure 7-15 depicts processing time required by the baseline implementation using the instance-based match in the Eaters task and how it is distributed among the three most expensive operations of the system: memory selection (match), episode installation (retrieving the episode into working memory) and episode removal (removing the episode from working memory). Figure 7-16 shows the same chart for the TankSoar task. In both graphs, the agent begins with no episodic memories and the y-axis measures the amount of processing time used by the agent as it gains more and more memories.

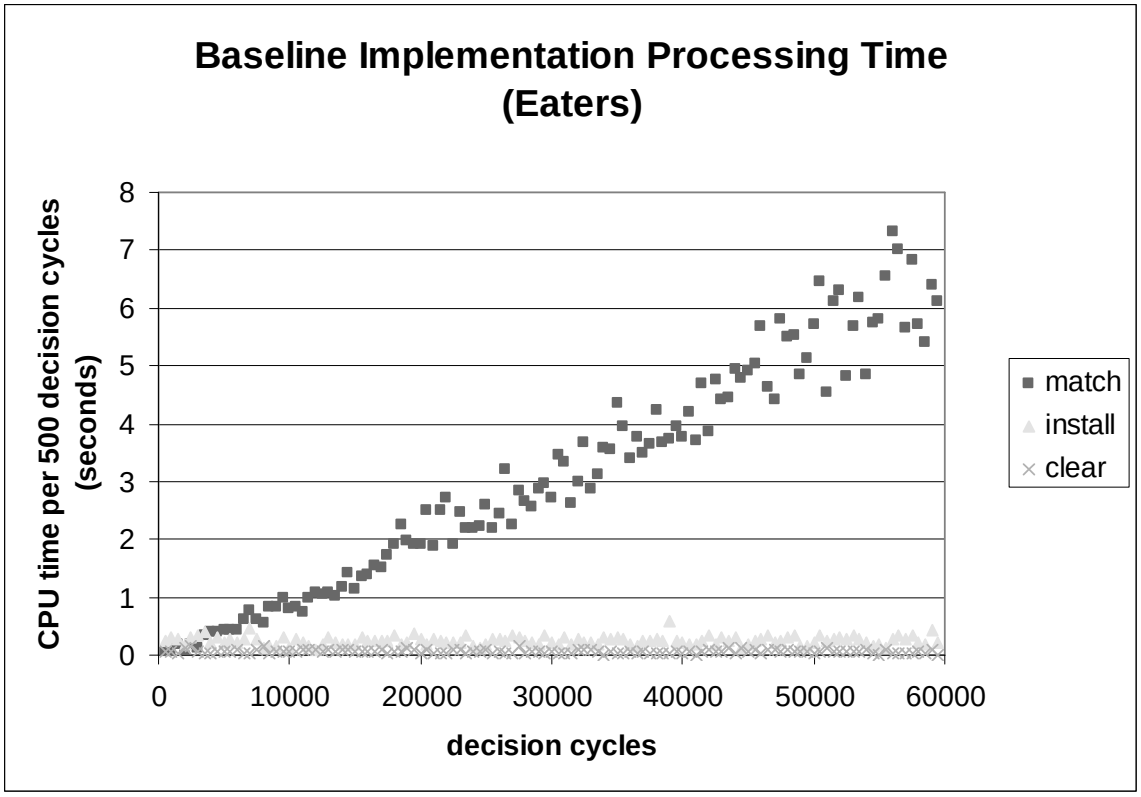

Figure 7-15: Baseline Implementation Processing Time (Eaters)

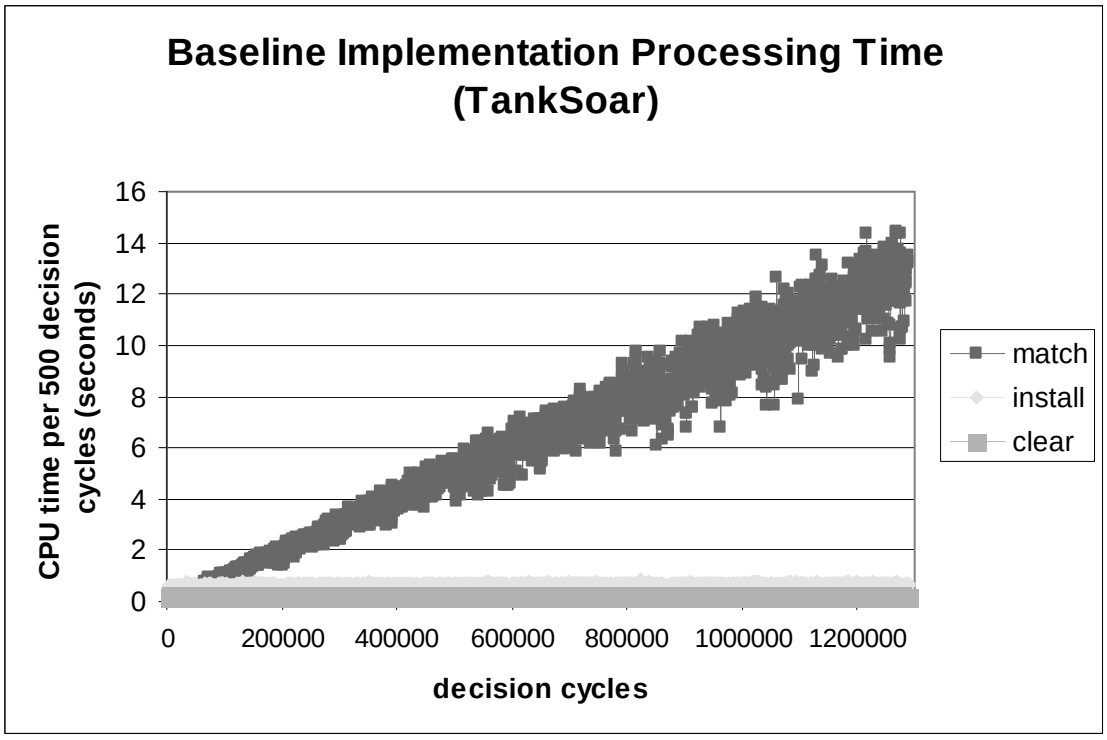

Figure 7-16: Baseline Implementation Processing Time (TankSoar)

This analysis of processing time shows that the amount of processing time also grows linearly and that the vast majority of time is spent on the match algorithm. Clearly, as more memories are added to the episodic store more time is required to select memories for retrieval.

7.4.2.3 Interval-Based Matching Algorithm

To improve the system's performance we began an investigation of alternative data structures for the episodic memories that would reduce its size at minimal cost in processing time. Examination of the content of the episodic memories revealed that each episode was usually only slightly different from the one that had been recorded just before it. This insight led us to consider an approach in which each episodic memory contained only the changes from one episode to the next. However, to implement such an algorithm, the activation values used for feature weighting during the match (see section 7.4.1) would be lost. Unlike the WMEs themselves, these values change every cycle and thus are not conducive to the same data compression scheme.

To compensate, we considered using the activation values from the episodic memory cue instead of each candidate episodic memory. We modified the baseline implementation to do this and re-ran the Eaters agent over a similar experiment. The results are shown in Figure 7-17. The x-axis measures successive actions taken by the agent. The y-axis records the fraction of evaluations performed during the last ten actions that were correct. The data shown is an average of five runs of each agent.

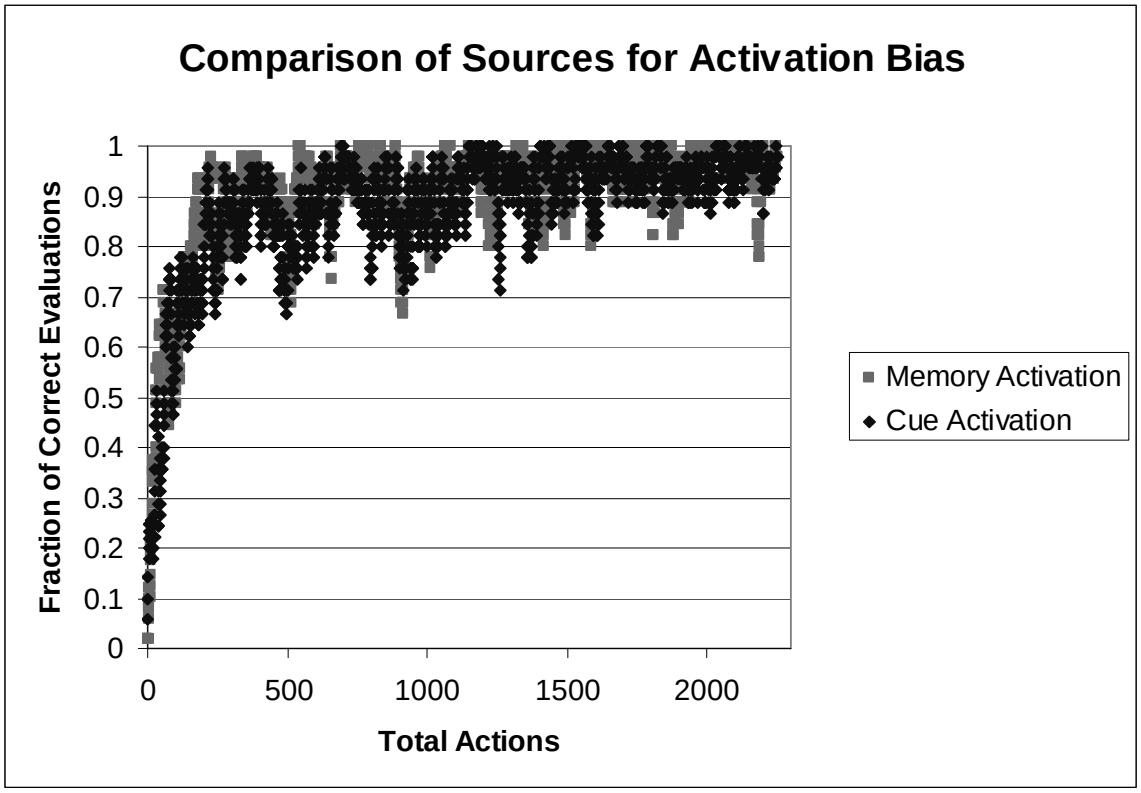

Figure 7-17: Impact of using Cue Activation vs. Memory Activation to Bias the Match

As the data shows, the performance of the two agents is nearly identical. While we could not conclude this would be true for all domains, this was enough support for that hypothesis that we proceeded with a new data structure that eliminates the need for an explicit representation of episodic memories. This structure is depicted in Figure 7-18 and should be compared to Figure 7-12.

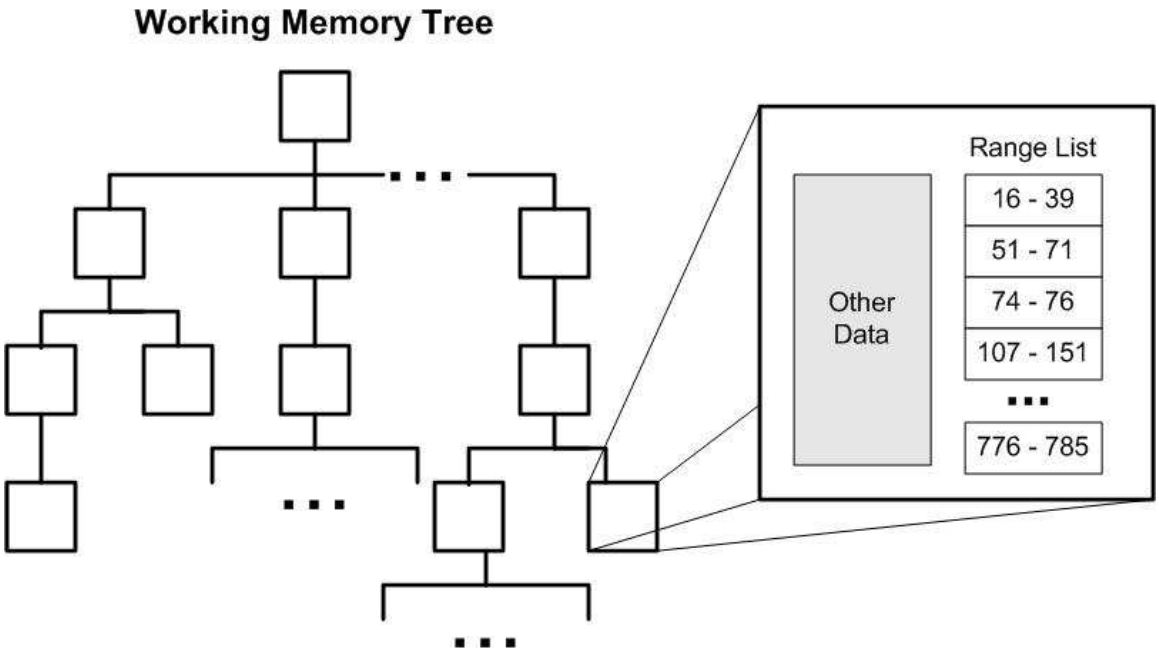

Figure 7-18: Data Structures used by the Interval-Based Algorithm

In this approach, the episodes are stored implicitly in the working memory tree via a series of time increments (ranges of decision cycles) on each node in the tree. The ranges indicate the cycles when the associated working memory element was in the agent's working memory. This approach makes the required storage linear in the number of changes to working memory instead of the number of elements in working memory.

We modified the matching algorithm to use this structure and to use the activation values from the episodic memory cue to weight the features of the candidate episodes. The resulting algorithm is as follows:

1. The system simultaneously traverses the episodic memory cue and the working memory tree. For each entry in the cue, the corresponding entry in the working memory tree is located.
2. Each entry that is matched in the tree contains a list of ranges. Each of these lists is set aside and each range in the list is assigned a match score equal to the activation level of the associated cue entry.
3. All of the selected lists are merged together into a single list of ranges. If two ranges partially overlap, they are split into two or more separate ranges. For example, if one list contains two ranges (1-10, 15-20) and the other list contains

one range (8-18) the merged list will contain six ranges (1-7, 8-10, 11-14, 15-18, 19-20). Each range in the merged list has a match score equal to the sum of the activation levels of all the ranges that entirely covered that range.

4. The merged list is traversed to locate the range with the highest match score (activation level). The number(s) in that range represent the cycles in which the best-matching episode was recorded. (In the event of a tie, the most recent cycle is selected.)
5. The episode can be recreated by traversing the working memory tree and creating working memory elements for each node that contains the selected cycle in one of its ranges.

The complexity of this algorithm is O(nr) where 'n' is the number of items in the cue and 'r' is the total number of ranges that must be examined. This complexity can also be expressed as $O(n^2l)$ where 'l' is the average number of ranges in each list that is merged in the matching process. However, these two variables are not independent in practice. At worst, we expect the growth to be linear in the number of episodes because the size of the cue ('n') is relatively constant. As in the prior algorithm, the growth will be minimized if the same features in the environment are not continually encountered (so that r is small). However, this algorithm has the additional advantage in that it is sensitive to only changes in features, so that the growth could be significantly less than the instance-based algorithm if environmental features change slowly over time.

7.4.2.4 Measuring the Performance of the Instance-Based Match

Using this new data structure and algorithm, we repeated the experiments we had performed with the baseline implementation. Figure 7-19 and Figure 7-20 depict the memory usage and processing time (respectively) required for this new implementation on the same task and using the same y-axis and x-axis at the same scale as used in the previous figures.

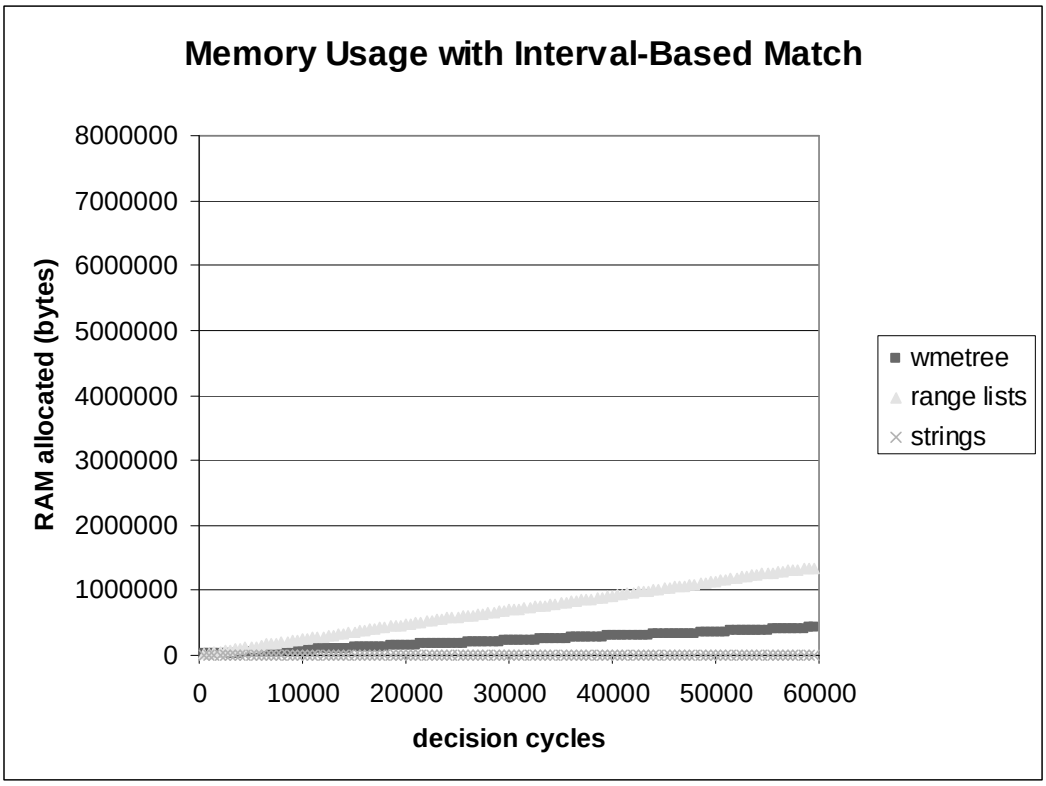

Figure 7-19: Impact of Interval-Based Match on Memory Usage

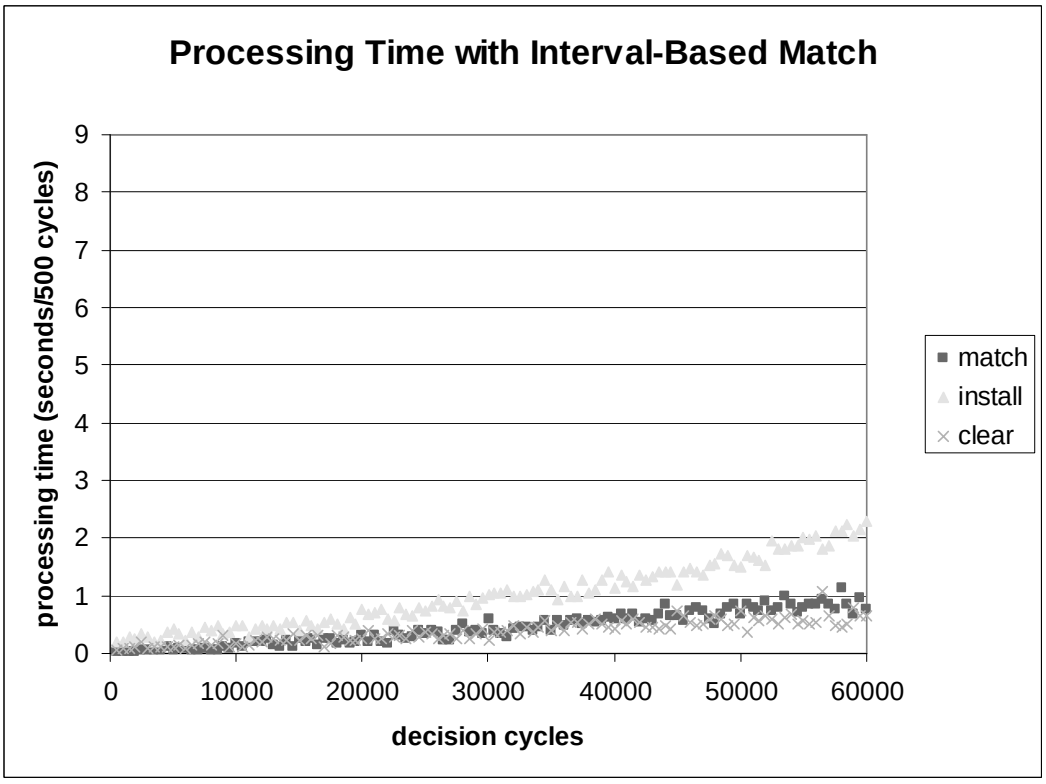

Figure 7-20: Impact of Interval-Based Match on Processing Time

Both resources still show linear or near-linear growth as predicted by this analysis but the amount of each resource used is significantly less. In the area of memory usage, the majority of space is consumed by the new range lists that implicitly represent the individual episodic memories. In the area of processing time, the most time is now required for reconstructing and installing an individual episodic memory rather than the match itself.

7.4.3 Flexible Agent-Architecture Interface

While the episodic memory system itself must be task independent, there is no such restriction on the knowledge in the agent. As a result, the episodic memory system can benefit from providing as flexible an interface as possible for the agents that use it so that the system's behavior can benefit from this knowledge. As we developed agents for the Eaters and TankSoar environments, we also encountered situations where the agent could benefit from having access to a more flexible interface either for specifying these cues or for receiving information about memory retrieval. Given these opportunities, we often implemented the requisite functionality. This section describes these additions to the baseline implementation and, where possible, demonstrates their impact on agent behavior.

7.4.3.1 Expanded Cues

The episodic memory cue that the agent constructs to trigger retrieval contains the WMEs that the agent believes may be relevant to the task. In the pilot implementation, we discovered that the need for retrieving sequences of episodes required a "next" command (which was added in the baseline implementation). As this work progressed, other commands were added. Below is a complete list of the commands supported by the most recent implementation:

- `query` - This is the base command for retrieving an episodic memory. All WMEs specified by this command (and their structure) are used to create an episodic memory cue.
- `neg-query` - Sometimes, the best way for an agent to describe what type of episode it wants is by specifying the features that it does *not* want. This command allows the agent to specify a negative cue. We used the negative query in some of these experiments as a way for the agent to emphasize the importance of certain entries of a cue by putting alternative entries into the negative cue. An example of this usage (and its impact) directly follows this list.
- `before/after` - When used in conjunction with `query` or `neg-query`, these commands restrict the resulting retrieval to episodes that occurred before and/or after a given episode. In other words, they allow the agent to set a chronological context for retrieval. These commands were not used for any of the experiments presented in this dissertation.
- `prohibit` - When used in conjunction with `query` (or `neg-query`) this command prevents the system from retrieving a specified episode. Among other uses, by prohibiting the episode that was just retrieved for a given cue, the agent is able to retrieve the "second best" match for a give query. Subsequent uses allow for "third best", "fourth best", etc. indefinitely. In the most functional of this TankSoar agents (see sections 8.4 and 8.5), this command allowed some of this agents to reject episodes that did not match the most critical entries in the cue.

While the potential value of these commands is apparent, there is usually no way to directly demonstrate their value in terms of an improvement in agent behavior. However, we can demonstrate that a specific use of one of these commands was essential for a specific agent. When this TankSoar agent (described in 8.5) is attempting to evaluate an action, it uses the `neg-query` command to bias the episodic memory system against retrieving episodes that do not contain the to-be-evaluated action. Without this ability, the agent is less likely to get a useful memory and, as a result, is less likely to select an effective action.

Figure 7-21 depicts the performance of the combat tank both with and without this ability. The agent begins with no episodic memories but records more of them as it plays more games (the x-axis). The y-axis is the agent's average margin of victory for each game given a specific number of previous games played. A negative margin of victory represents a losing game and a positive margin of victory is a winning game. For both curves, the data is the average of ten independent runs of the same agent. The error bars in the figure represent the range of 95% confidence.

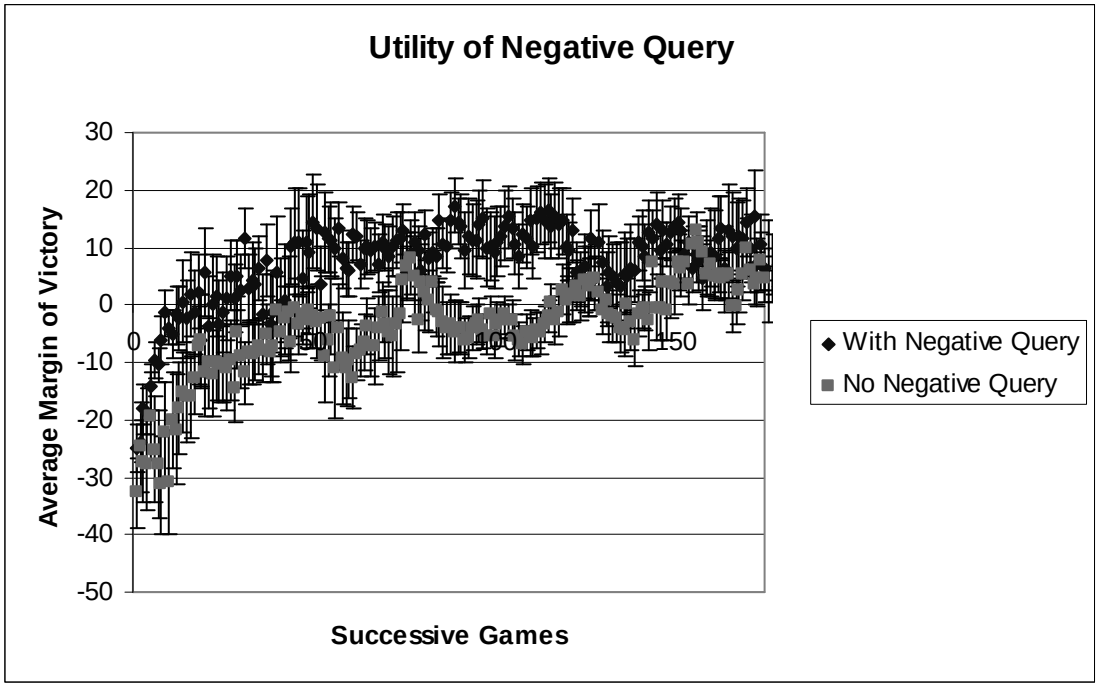

Figure 7-21: Impact of Removing the Negative Cue from the TankSoar Agent

As is apparent in the graph, the inability to create a negative cue means that the agent's rate of learning is diminished.

7.4.3.2 Retrieval Meta-Data

When an agent retrieves an episode, the architecture has the opportunity to supply metadata about the episode and the selection process that the agent can then use to guide its behavior in a task-specific manner. To support the agents used in this experiments, the following metadata are now provided whenever an episode is retrieved by the system:

- Retrieval Count - If the agent is retrieving a series of episodes (via the "next" command) the system reports how many episodes in that sequence have been retrieved so far. This retrieval count allows the agent to stop the sequence when "enough" episodes have been retrieved, where "enough" is a task-specific value determined by the agent. In the TankSoar agent we built to demonstrate learning from past success and failure, the agent used this information to detect when it had retrieved the maximum memories in a sequence (an arbitrary maximum) and should stop.
- Match Score - The raw match score is calculated during memory selection by the architecture and the range of possible values will vary depending upon the cue (and thus upon the current task). By providing this number to the agent, it can monitor these values to gain a measure of its confidence in a specific retrieval for a given "type" of cue. (Again, "type" must be a task-specific measure defined by the agent.) This value was also used to calculate the Normalized Match Score (below).
- Cue Size - The number of features that were in the cue. The agent can use this value to normalize quantities or classify cue types. The architecture uses this value to generate the Normalized Match Score (below).
- Normalized Match Score - This value is the match score divided by the cue size. While the agent could do this math, the architecture simply provides the value as a convenience. This value was never used by an agent but was used by the architecture to decide when to allow chunking in one experiment (see section 8.3.2).
- Match Cardinality - The cardinality of a match is the number of entries in the cue that were actually matched by the retrieved memory. This number can be important (particularly with small cues) because one highly activated WME can overwhelm the importance of other cue entries. (See section 7.4.1.6 for some experiments with balancing bias from activation versus bias from match cardinality.)
- Memory ID - Each episode is assigned a unique number when it is recorded. By assigning these numbers sequentially and reporting them to the agent, the episodic

memory system gives the agent a sense of the relative time when the episode occurred. These IDs also facilitate cue commands like `before`, `after` and `prohibit`.

- Present ID - By telling the agent which ID will be assigned to the next episodic memory that is recorded, the agent can compare this with the memory ID to gain a sense of the recency of a particular episode. This value was not used for any of the experiments presented in this thesis.

Chapter 8

Cognitive Capabilities

In Chapter 3, we introduced and discussed a set of cognitive capabilities that episodic memory might facilitate. One of the two major goals of this research is to investigate whether, in fact, episodic memory can play a role in supporting a subset of those cognitive capabilities using this episodic memory system. This chapter presents the work we have performed toward accomplishing this goal.

Each section in this chapter begins with a cognitive capability. We then describe a specific task where that cognitive capability can improve an agent's performance in one of the two environments used in this research (Eaters or TankSoar). Finally, for each task, we present and discuss the results from this investigation, as well as any lessons learned. The experiments described here are presented in roughly chronological order.

8.1 Action Modeling

Action modeling is an obvious case where episodic memory can be useful. Given a set of experiences in an environment, an agent can use those prior experiences to predict changes in the environment following a given action.

8.1.1 Action Modeling in the Eaters Environment

To demonstrate this capability we created an agent for the Eaters environment (see section 6.2.1) with an episodic memory. Aside from the goal of maximizing its score, the only knowledge given to this agent was an understanding of what actions it could take in the world (movement in a cardinal direction). The agent was not aware of

the semantic meaning of those actions or the relative importance of the different information that was available from its senses.

Given this lack of knowledge, we designed the agent to use its episodic memory to learn how various situation-action pairs affect its score and, thus, what actions are best in a given situation. In any given situation, the agent is aware of what actions it can take. To evaluate a proposed action, the agent creates a memory cue composed of its current sensory input and the action of moving in the proposed direction. If the cue resulted in a successful retrieval of performing the same action in a similar situation, the agent would then ask for the next memory in chronological order. If both memories include the agent's score, then the agent can determine what immediate change in score (if any) resulted from that past action. This change in score can, in turn, be used to quantitatively evaluate each proposed action. The action with the highest score is the action that agent actually selects. This agent is described in more detail in section 7.3.

Figure 8-1 depicts the accuracy of this agent's action evaluations as it gains more and more episodic memories. This particular experiment was run five times using this baseline implementation with this improved working memory activation mechanism providing feature weights for the memory selection routine (see section 7.4.1.5). The results were averaged and a mean smoothing with a windows size of ten was applied to achieve the final results that are shown in the figure.

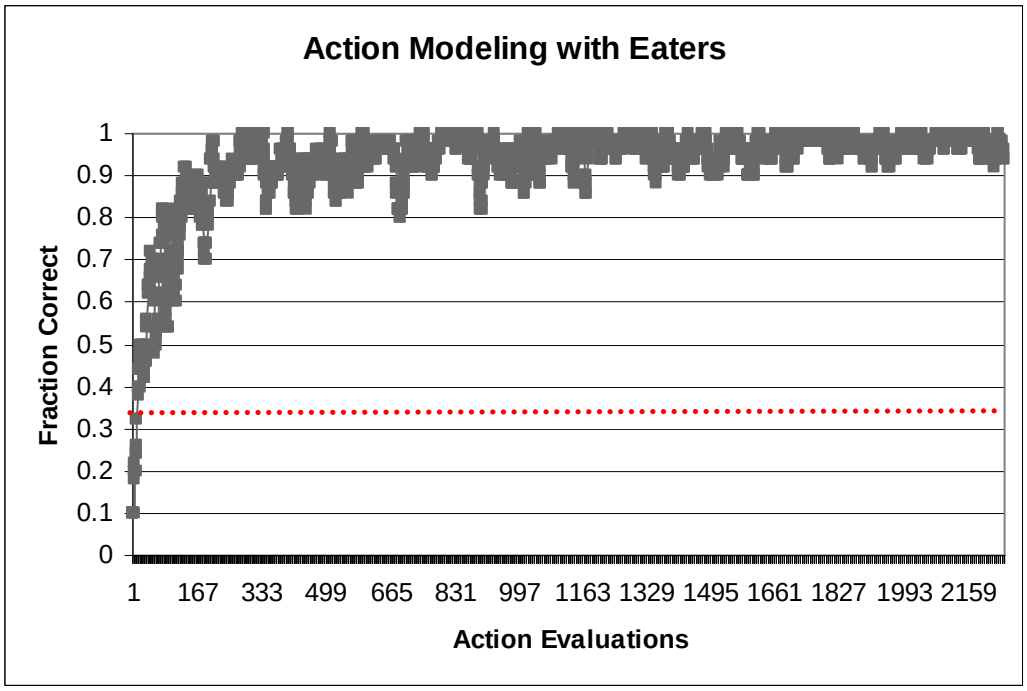

Figure 8-1: Action Modeling Results (Eaters)

The x-axis is the number of predictions that the agent has made and, thus, is a rough measure of time. The y-axis indicates the fraction of the last ten predictions that were correct. For this experiment, a correct evaluation consists of predicting the correct change in score that will result for a given action. If the agent makes an incorrect prediction (or is unable to make a prediction at all due to a failed episodic memory retrieval) then the outcome is considered a failure.

The agent begins with no episodic memories and, as a result, its accuracy begins at 0%. (The smoothing makes this appear closer to 10%.) As the agent acts in the world, its ability to model the results of its actions improves rapidly, approaching perfect behavior in the limit.

8.1.2 Action Modeling in the TankSoar Environment

The simplicity of the action modeling cognitive capability made it a good target for demonstration in multiple environments. Because the task in the TankSoar domain is complex, we instead focused on a smaller subtask. While a human watching the domain can see it in its entirety, the agent's sensing is limited. An agent has many senses but the

most useful one is its radar which allows it to sense the environment immediately in front of it. The radar can be set to different distances with further distances allowing the agent to see farther but also requiring more energy. Thus, energy is wasted if the radar is blocked by an obstacle (e.g., a wall or another tank). Figure 8-2 depicts what an agent can see with its radar from the same position and direction using three different radar settings.

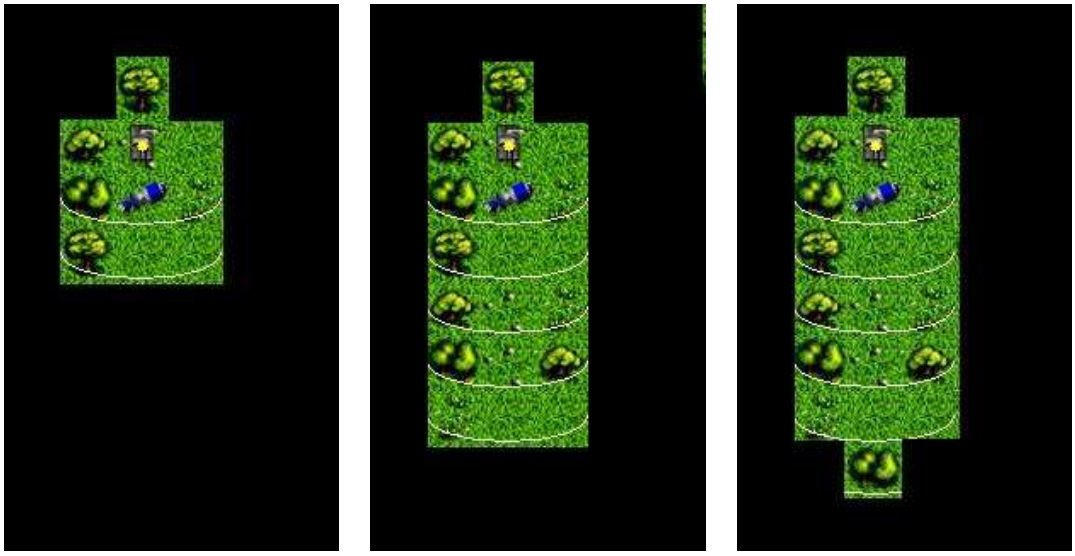

Figure 8-2: Outcome of Three Different Radar Settings

To demonstrate action modeling, this agent uses its episodic memory to predict what it will see when it turns on its radar, and uses that information to set the most efficient radar distance. As with the Eaters task, it is essential that the episodic memory system retrieve a relevant memory for a given cue. Unlike the Eaters task, the size of the memory cue was much smaller: consisting of the agent's current x, y coordinates, the direction it was facing and the maximum radar setting. As with Eaters, we used working memory activation to weight the features of episodes during the match but, due to this small cue, we found that the agent performed best when an exact match (even with lower activation) was preferred over the best activation-biased match. Section 7.4.1.6 has details on investigations into the balance between match cardinality vs. activation bias on memory selection.

Figure 8-3 depicts the agent's performance over one hundred radar settings while the agent explores a map. The y-axis is the fraction of the last ten settings that were

correct. A failed setting was given a partial score based upon how close it was to the best setting. Each data point is the average of five runs. The dashed line at the bottom of the graph indicates the performance of an agent that selects its radar setting randomly. As the graph shows, the agent quickly learns to make effective radar settings as it navigates the maze.

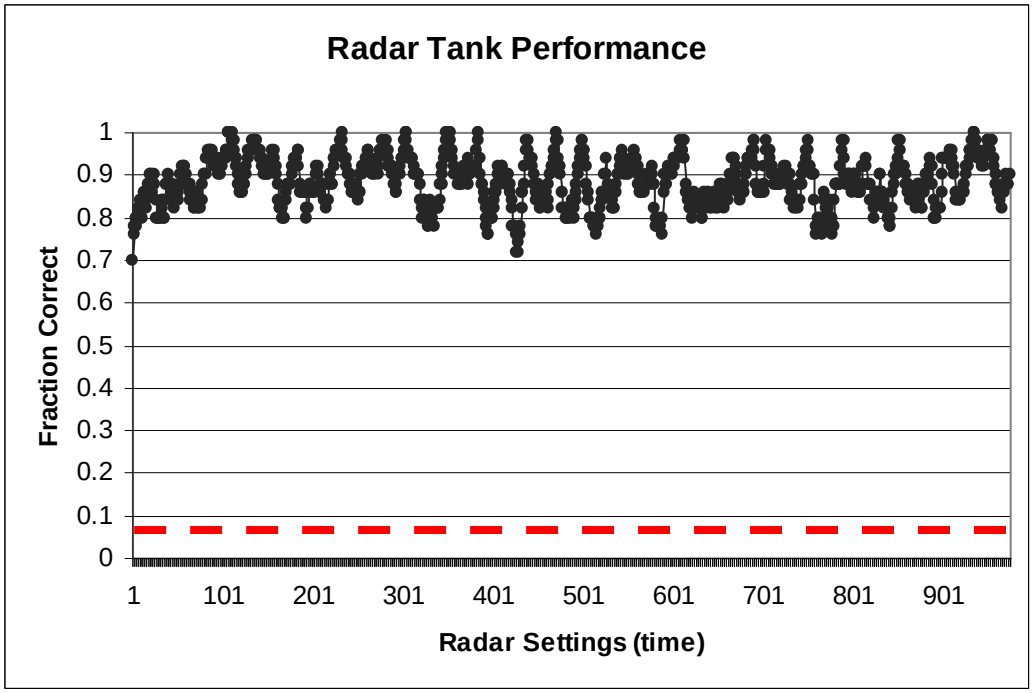

Figure 8-3: Action Modeling Results (TankSoar)

Based on the results from both the Eaters and TankSoar environments, it appears that episodic memory can be an effective resource for action modeling. The learning appears more rapid here than in the Eaters experiment due to the fact that partial successes are possible.

8.2 Retroactive Learning

Retroactive learning is the ability to relive an experience when more resources (usually time) are available in order to learn things from those experiences that could not be learned when they occurred. At the general level, a retroactive learning algorithm has these steps for an agent:

0. Gather episodic memories at a time when the agent can not learn from them fully.
1. Retrieve an episodic memory that the agent has not retrieved before for this learning task. If no such memory can be retrieved, exit.
2. If the agent already has knowledge informing it of how to get to the goal state from the recalled state, go to step 1.
3. Does the agent recall performing an action that got it to the goal state from the current state? If so, record the necessary state-action combination as semantic knowledge. Go to step 1.
4. Do the agent recall taking an action in this state that gets it to another state wherein it already have semantic knowledge of how to get to the goal? If so, record the action taken along with a sufficiently unique description of the recalled state as new knowledge.
5. Go to step 1.

We demonstrated this cognitive capability in the Eaters domain using this pilot implementation. The agent we created used a simplified version of the general algorithm:

0. We created an agent that acted randomly for a fixed period of time gathering episodic memories.

1,5. After this time, the agent would stop and retrieve its memories by constructing cues consisting of the agent in all possible combinations of neighboring cells. This is significant because it reflects additional information given to the agent: that the contents of cells are important.

2-4. Based upon its retrievals, the agent gained new knowledge (chunks) describing which action to take based upon the contents of neighboring cells.

After this period of reflection, the agent would resume its movement through the maze using this new semantic knowledge (additional episodes were also recorded).

Figure 8-4 depicts the performance of this retroactive learning agent in the Eaters environment. The y-axis measures the fraction of the last ten actions that were correct while the x-axis represents successive actions in the world. The data is an average of five runs of the agent.

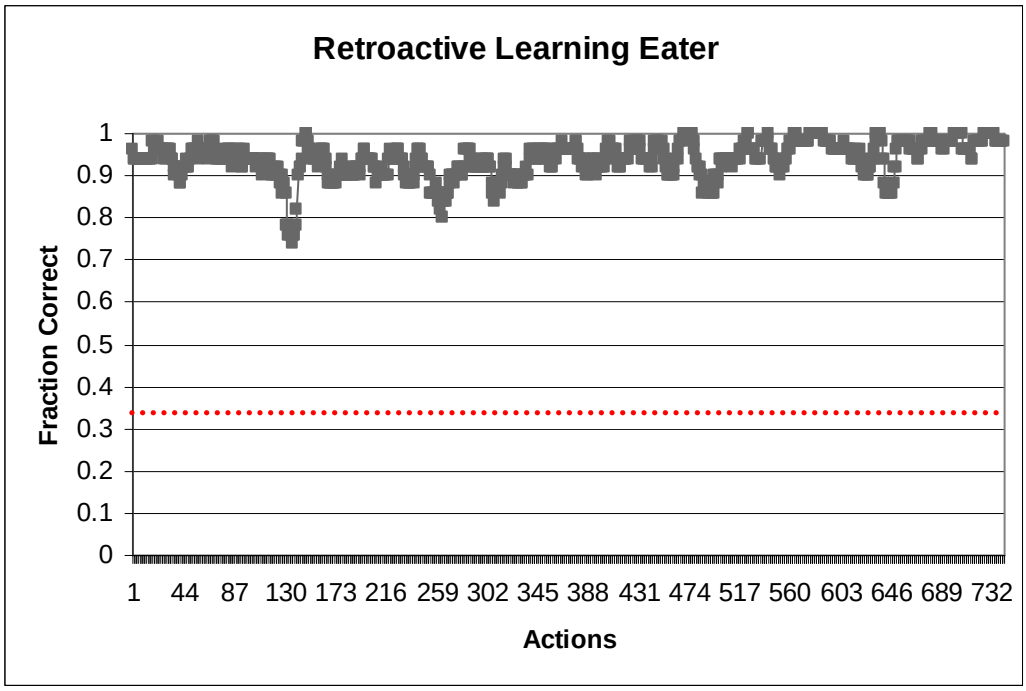

Figure 8-4: Retroactive Learning Results

This graph indicates that the retroactive eater has learned near-ideal behavior via retroactive learning.

8.3 Boosting Other Learning Mechanisms

The ability to provide "grist" for the "mill" of other learning mechanisms is potentially an important ability for episodic memory. Similar combinations of "lazy" and "eager" learners are demonstrated by locally weighted learning (Atkeson, et al. 1997), lazy learning (Sheppard and Salzberg 1997) and continuous case-based reasoning (Ram & Santamaría 1997).

8.3.1 Demonstrating Boosting with Eaters

To demonstrate an episodic memory system's ability to boost other learning mechanisms we combined this pilot implementation of episodic memory with Soar's built-in learning mechanism: chunking. (You may wish to refer to section 5.6 for an overview of chunking.) When the agent lacks the knowledge to choose among multiple

possible actions this creates an impasse. In the resulting subgoal, episodic memory retrieval is used to evaluate each action. These evaluations became results of the subgoals and lead to the creation of rules (chunks). These rules test WMEs that were used to create the episodic memory cue and create preferences for the actions that were selected based on the retrieved episodes. In other words, these chunks are compiling the processing that was used in the episodic memory retrievals so that the agent can skip these retrievals in the future. This expectation was that this would increase the speed at which the agent was able to make decisions.

Figure 8-5 depicts a comparison of an episodic memory agent with chunking and the same agent without the chunking mechanism. The figure includes data from a hypothetical agent with ideal behavior that requires only one decision cycle per action and always takes the correct action. The y-axis measures each agent's score while the x-axis represents successive Soar decision cycles (instead of successive actions as in Figure 8-4 above). This means that the graph is weighing both the effectiveness of the agent's actions and the speed at which the agent makes decisions. The data is an average of five runs of the respective agents.

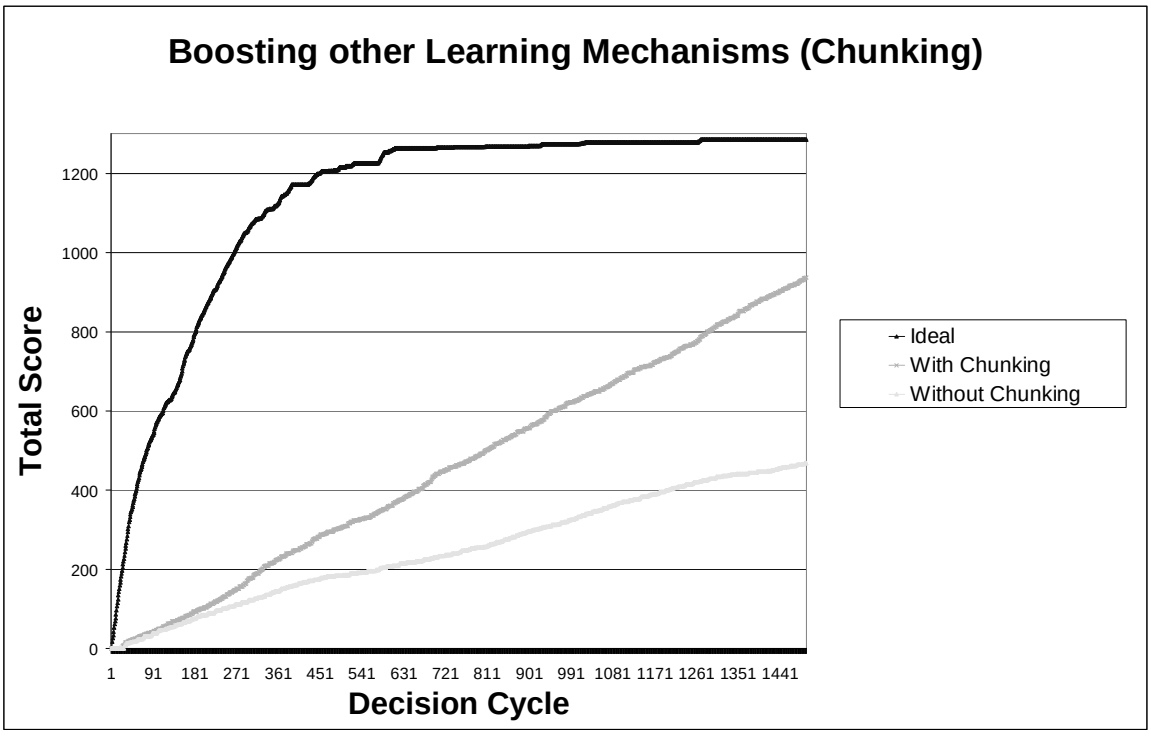

Figure 8-5: Boosting other Learning Mechanisms Results

As the figure depicts, chunking creates a significant decrease in the amount of time required for the agent to make decisions. This improvement in behavior can only result because both mechanisms (chunking and episodic memory) are present. In other words, this synergy demonstrates episodic memory's ability to boost another learning mechanism.

8.3.2 Chunking with Confidence

One issue that arose when using chunking with episodic memory is that chunking is a "one-shot" learning mechanism. In other words, once a chunk has been learned it cannot be unlearned even if knowledge it contains is erroneous. Based on these observations, we concluded that an agent might benefit from a chunking mechanism that had a measure of confidence in the results of an episodic retrieval and used that confidence to decide when to learn.

An obvious way to measure this confidence is to use the match scores that the matching algorithm uses to rank candidate episodes for retrieval. By normalizing the raw score with the size of the memory cue, we acquire a hypothetical measure of the agent's confidence. To explore this hypothesis, we gathered data about the correlation between the correctness of a retrieved episodic memory (where "correctness" is measured by whether the agent makes a correct decision based upon the memory). We did this from two environments using both of this action modeling agents (see section 8.1) by showing the correlation of these normalized match scores to the correctness of the memory (where "correctness" is measured by whether the agent makes a correct decision based upon the memory).

Figure 8-6 and Figure 8-7 depict a bar graph of the number of instances of each match score (normalized) for the Eaters agent and the TankSoar agent respectively. The x-axis has a bracket for each possible match score (binned to whole numbers). The range of the x-axis is larger for the TankSoar graph than for Eaters. The y-axis measures the number of instances in which the episodic memory system retrieved an episode with each

match score. Finally, the bars are labeled (and color coded) to indicate whether the retrieved memory resulted in a correct action by the agent.

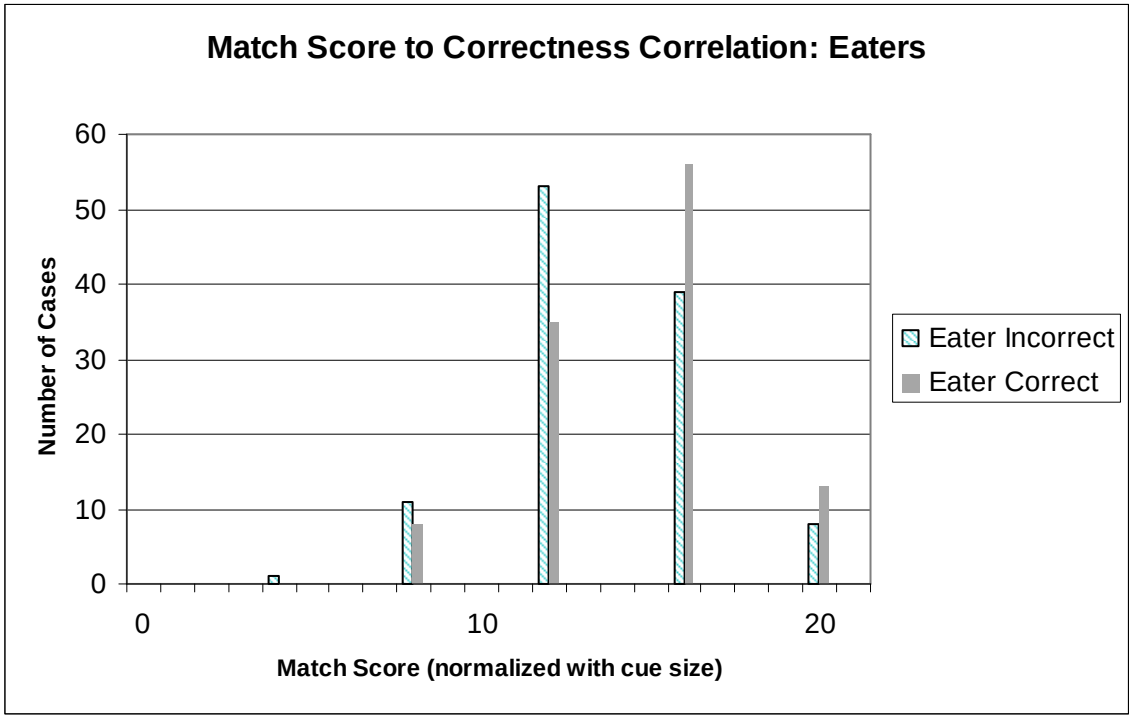

Figure 8-6: Correlation of Match Score to Correct Decisions (Eaters)

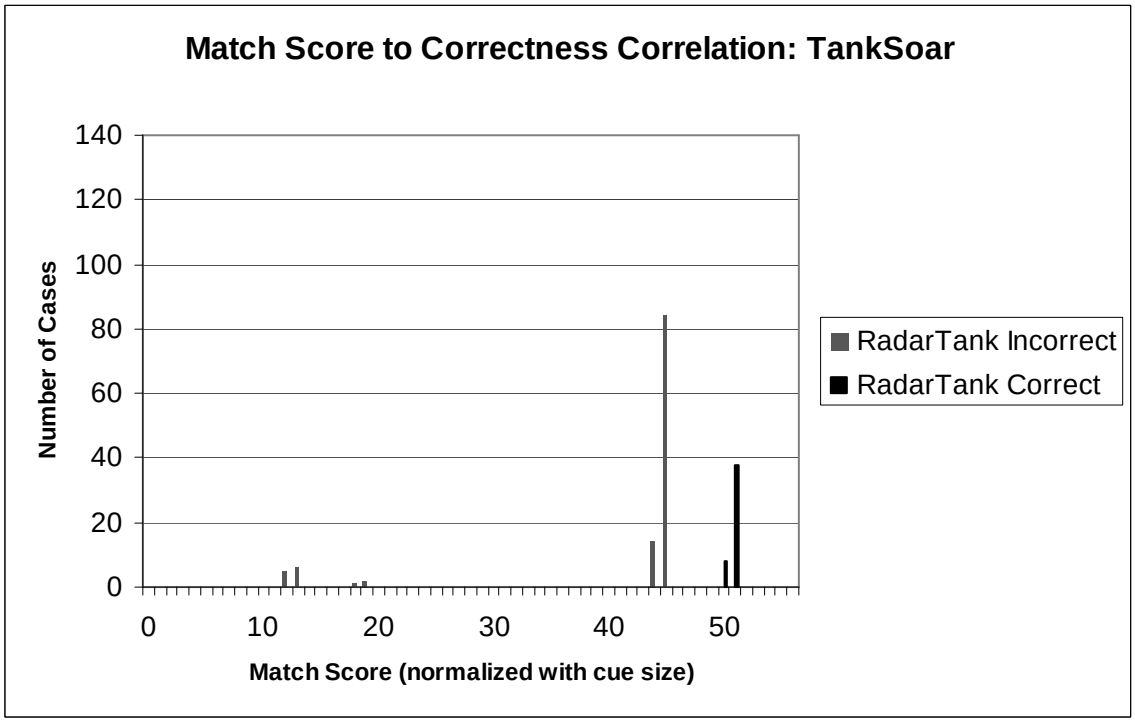

Figure 8-7: Correlation of Match Score to Correct Decisions (TankSoar)

Two things are readily apparent upon viewing this graph. First, scaling the match scores from the two domains based upon cue size did not normalize them. The adjusted match scores from the Eaters agent were significantly smaller than those from the TankSoar agent. Second, there is a correlation between match score and correctness. This is particularly true for the TankSoar agent.

Based upon these observations we concluded that it is possible to use the match score as a measure of confidence. However, we did not have a clear method for making this measure task independent.

Despite the lack of task independence, we proceeded with an experiment using the episodic memory system's match score as a measure of confidence for the TankSoar agent. Specifically, we modified the episodic memory system so that it could turn the chunking mechanism on or off depending upon its confidence in the most recent retrieval. We set the threshold using a hardcoded (task-dependent) value for the raw match score. The thresholds were set based upon the data from the figures above. Given the high correlation between confidence and correctness, we hypothesized the agent would only learn new chunks when the episodic retrieval was likely yield correct behavior. As a result, we anticipated improved behavior from the agent vs. an agent with chunking turned on all the time.

Figure 8-8 depicts the average performance of the agent at difference confidence thresholds. The y-axis measures the fraction of correct evaluations made by the agent out of a fixed number of such evaluations. The agent with confidence threshold 0 is chunking all the time. The control agent was an action modeling agent for TankSoar that did not use chunking at all. Instead, it used task-specific information to manually verify that a retrieved episode exactly matched the cue and was usable by the agent.

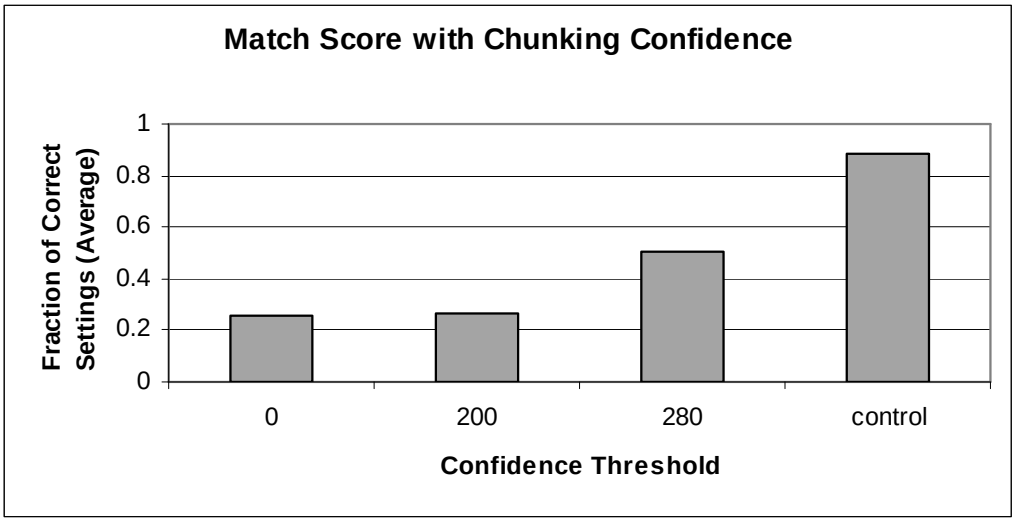

Figure 8-8: Impact of using a Hard-Coded Confidence Threshold for Chunking

As the results show, we did perceive improvement in performance compared to the original chunking agent but the performance was still far below that of an agent that self-verified. In other words, despite the task-specific threshold, the agent was still producing some incorrect learning.

Upon examining the data, we discovered that the correctness correlation actually shifted as a result of turning on the learning mechanism. Specifically, the changed created by chunking altered the activation levels. This, in turn, altered the match scores of retrieved memories that, in turn, altered the strict correlation between match score and the correctness of agent behavior. In short, we discovered a circular dependency between these four factors (depicted in Figure 8-9).

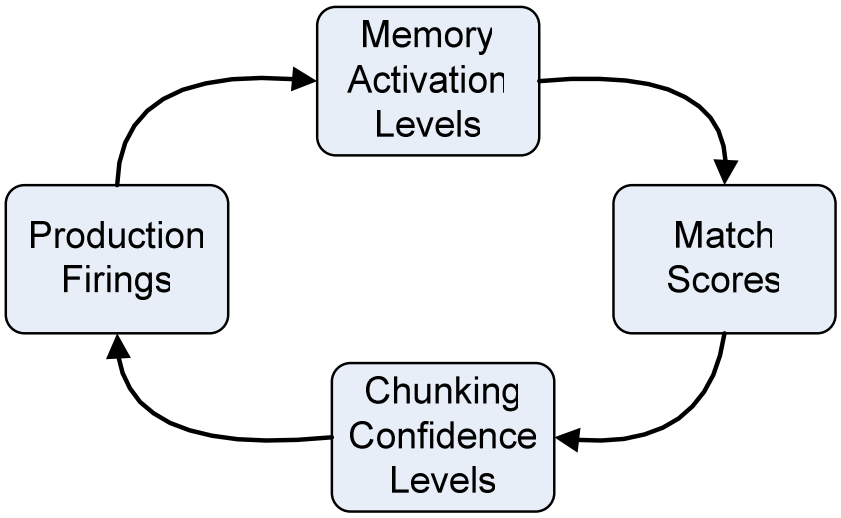

Figure 8-9: Circular Dependency

Based on these experiments we concluded that match score was a fair predictor of the usefulness of a retrieved episode. As a result, a learning algorithm that is using the results of episodic memory retrievals can gain a boost from having access to the match scores. However, harnessing these values may be difficult. Not only did we see evidence that the range of match scores is task dependent but we also observed that the relationship between match scores and chunking with confidence is dynamic and interdependent.

8.4 Virtual Sensors

When an agent originally senses something, it may be irrelevant to its current task. Then, at some future point, that past sensing may become important. An agent with episodic memory can retrieve details of its past sensing. This capability is useful in environments with large bodies of data that are irrelevant to the current task, but may be relevant to future tasks. For example, a traffic controller agent may be asked by authorities if a green car being driven by tall, balding man passed by within the last hour. While the agent normally has no need to record information about the drivers of cars, it may help stop a criminal by remembering this critical information.

To demonstrate this cognitive capability in the TankSoar domain, we chose the task of locating the battery used to recharge the agent's energy supply. When the tank's

energy supply runs low, the agent uses its episodic memory to construct a map or path to the battery in its working memory. The high level algorithm is as follows:

1. If the agent can only move forward or backward, it continues to move forward until this changes. Then it proceeds to step 2.
2. If the agent already has knowledge of a path from this position to the battery, then it follows that path and returns to step 1. For this agent a "path" is a location (x,y coordinate) combined with a direction (north, south, east or west). For our implementation, these were stored in the agent's working memory. The agent follows this step until it occupies a square that does not contain a path. Then it proceeds to step 3.
3. The agent attempts to retrieve an episodic memory of seeing the battery from this position. The cue for this retrieval consists of the agent's current location and the perceptual elements that represent seeing the battery on radar. If the retrieval is unsuccessful, the agent proceeds to step 4. If the retrieval is successful, the agent records a path in working memory originating in this position and moving directly toward the battery. It then resets to step 1.
4. For each path the agent knows of, it attempts to retrieve an episodic memory of seeing a location from this position from which it has already recorded a path to the battery. If any retrieval is successful, then creates a new path in working memory that directs the agent from this position toward the origin of the existing path and resets to step 1.
5. If step 4 fails, the agent moves in a random direction (search) and resets to step 1.

For this experiment, we used the baseline implementation with working memory activation being used to weight the features of episodes during memory selection. The agent also began with a set of episodic memories that had been gathered in previous, exploratory movements in the maze. Over time, an agent using this algorithm constructs a set of paths that can direct it from any position in the maze to the battery (see Figure 8-10 for an example).

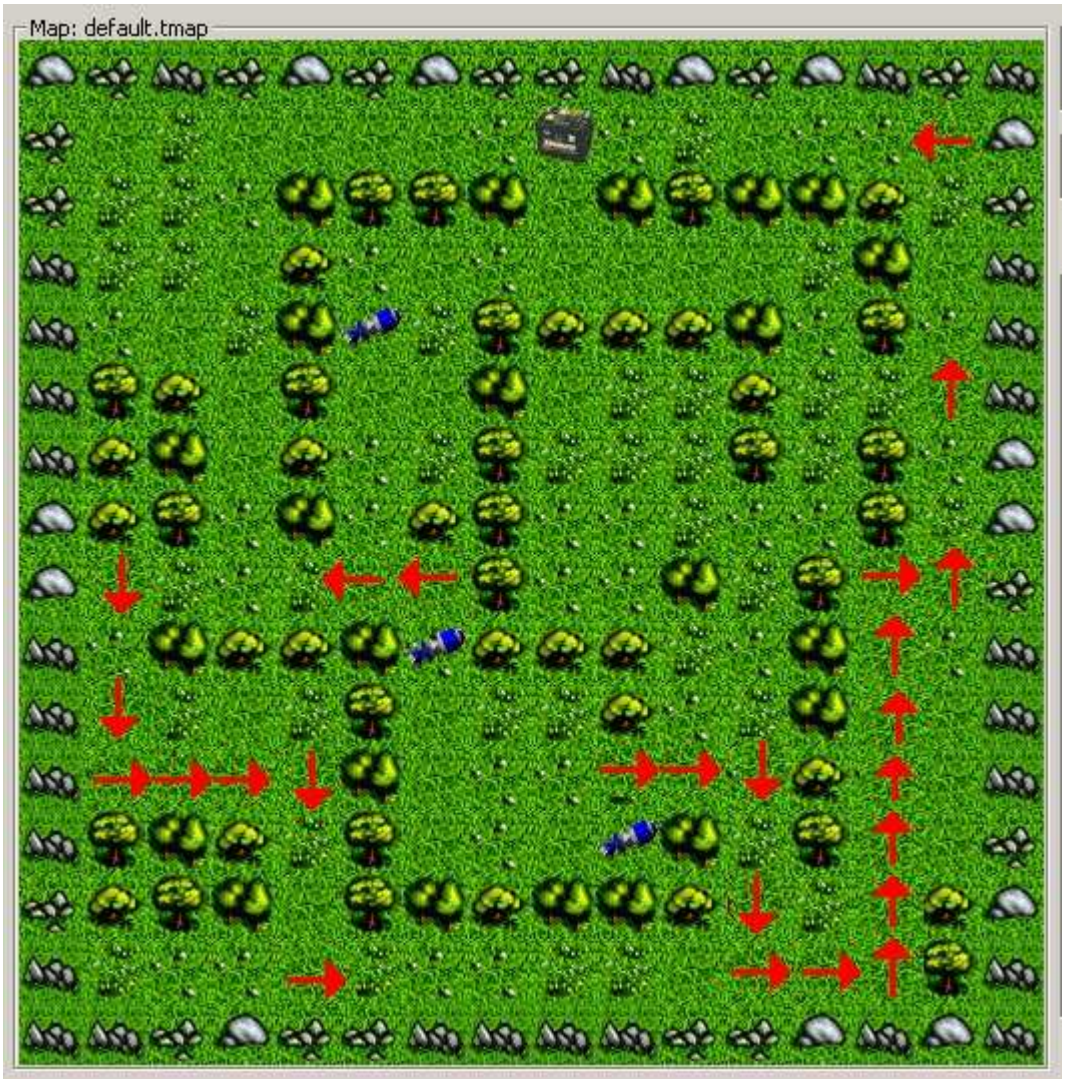

Figure 8-10: A Graphical Depiction of a Set of Paths Learned by the TankSoar Agent

Figure 8-11 depicts the results from this experiment. The y-axis (which has a logarithmic scale) measures the number of moves required to reach the battery. (Thus, a lower score is better.) The x-axis represents subsequent searches over time. For the first search, the agent has only a few episodic memories. As the agent gains more paths and more episodic memories the time required to find the battery diminishes. Specifically, this data show that it is an order of magnitude faster than a random search.

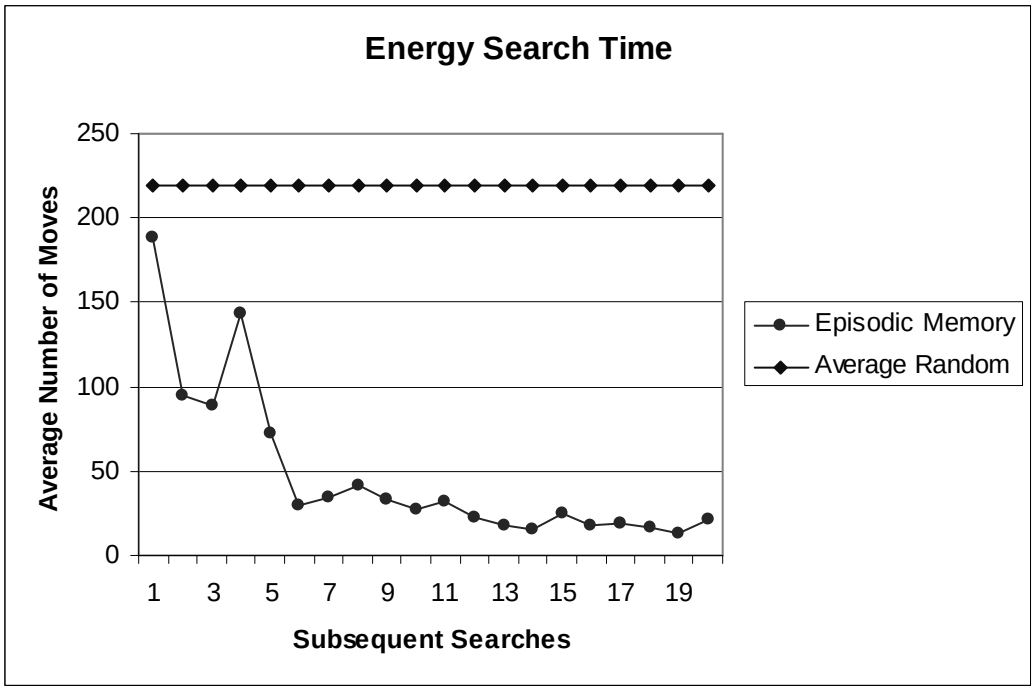

Figure 8-11: Virtual Sensors Results

8.5 Recording Previous Successes and Failures

Action modeling allows an agent to predict the immediate outcome of an action. However, in many tasks, the long term outcome of an action is more critical to success. In such situations, the cognitive capability of remembering the long term success or failure that followed a particular action in a particular situation can lead to better behavior on the part of the agent.

The TankSoar environment is well suited for demonstrating this cognitive capability (see section 6.2.2 for an overview of TankSoar). The environment is fairly complex with multiple different sensors and many unique combinations of actions that can be taken simultaneously. Within this environment, the agent must balance conflicting goals of survival, resource management and scoring hits on other agents. Most importantly, the outcome of a particular action is usually not immediate and often dependent upon future actions. For example, missiles fired by TankSoar agents take time to travel and their effect may be unknown for several time steps. Most poignantly, the decision to attack with low health and few missiles will almost certainly lead to disaster

in the long term. As a result, success in the TankSoar domain requires the ability to predict the long term outcomes associated with an action.

To demonstrate this cognitive capability, we started with an existing hand-coded agent for the TankSoar domain. This agent has four major subgoals:

- **Attack** – This subgoal is selected when the agent sees a nearby enemy and has sufficient resources (i.e., missiles, health and energy) and knowledge (i.e., the enemy is visible) to attack. For this agent, "sufficient" is a hard-coded set of criteria.
- **Chase** – This subgoal is selected when the agent detects (but can not see) a nearby enemy and has sufficient resources to attack.
- **Retreat** – This subgoal is selected when the agent detects a nearby enemy and does not have sufficient resources to attack
- **Wander** – This subgoal is selected when the agent does not detect a nearby enemy.

We modified this default agent in two ways. First, to reduce complexity, focus the agent on effective tactics, and shorten simulations we prevented the agent from turning on its shields.

Second, we introduced a 10% chance for aberrant behavior rather than normal behavior in all subgoals except Wander. This second modification was designed to reduce the incidents of mutually repetitive behavior between two competing agents. For example, two agents might be diagonally adjacent to each other and simultaneously decide to move sideways so as to be directly adjacent to its opponent. However, since both agents are moving sideways they end up diagonally adjacent to each other again.

This modified agent was this control agent for all experiments with this task in the TankSoar environment. The control agent has a total of 53 Soar productions. In all experiments, the episodic memory agent was pitted against this agent in a one-on-one match.

To demonstrate the target cognitive capability we created an episodic memory agent by further modifying the control agent. We removed the control agent's logic for selecting actions in the Attack subgoal. We replaced this logic with results from a

subgoal for evaluating actions based upon episodic memory retrievals. This algorithm has the following steps (refer to Figure 8-12):

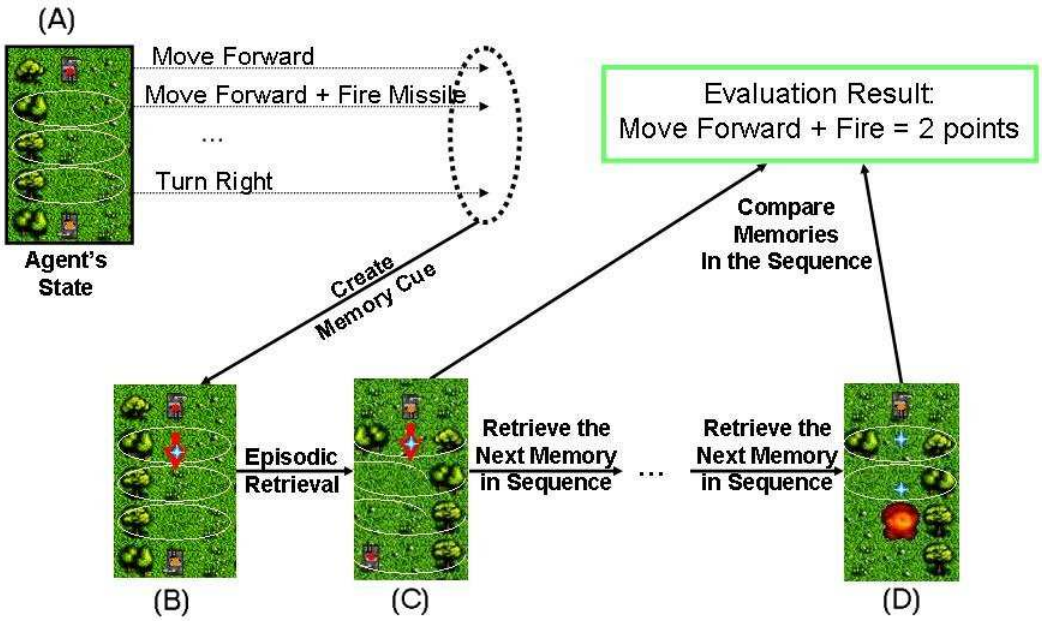

Figure 8-12: An Episodic Memory TankSoar Agent

1. The agent first determines what actions it is able to take based upon its current state. Certain actions may be prevented by the presence of the obstacles or a lack of resources. In the figure, the box labeled (A) represents the agent's current state and the arrows emerging from the figure represent a list of possible actions.
2. The agent does not have sufficient procedural knowledge to select which action is best. Therefore, it uses episodic memory to evaluate each possible action as follows:
   a. First, the agent creates an episodic memory cue that contains heuristically-selected portions of the agent's current state plus the action to be evaluated. In other words, the agent is trying to retrieve an episode wherein it took the same action in a similar situation. To emphasize the importance of

this specific action, the agent creates a negative cue that biases against memories where a different action is being taken. In the figure, an example of the memory cue is represented by the box labeled (B).

b. The agent allows the episodic memory system to retrieve the episodic memory that best matches the given cue. In the figure, an example retrieval is labeled (C). As you can see, this retrieved episode is not an exact match to the cue but merely the closest match that the episodic memory system can identify. Therefore, decisions based upon this retrieval may or may not be best.

c. The agent records the score that it had in that current episode. Then, through repeated uses of the "next" command, the agent retrieves the sequence of subsequent memories (in temporal order) that occurred until one of the following events occurred in the most recently retrieved episode:

   i. one of the two agents is destroyed (indicated by a significant change in score)

   ii. the end of the game

   iii. a sequence of ten memories had been retrieved

d. The agent uses the overall change in score between the first and last retrieved episode as a quantitative evaluation of the action being considered. Inconclusive outcomes result in a default (slightly positive) evaluation in order to encourage exploration.

3. Once all the possible actions have been evaluated, the agent selects the action with the highest evaluation. Ties are resolved randomly. (There is also a 10% chance that the agent will take a random action in order to put it on equal footing with the control agent.)

Figure 8-13 shows the episodic memory agent's performance against the control agent. The x-axis represents successive games. The agent retained its episodic memories from game to game and so had a larger episodic store at each successive game. The y-axis measures the agent's margin of victory (i.e., the control agent's score subtracted from the episodic memory agent's score). Thus, a negative margin of victory indicates a loss. The results depicted here are the average of ten repetitions of the same experiment.

Each episode contained approximately 120 WMEs. The error bars in the figure represent the range of 95% confidence.

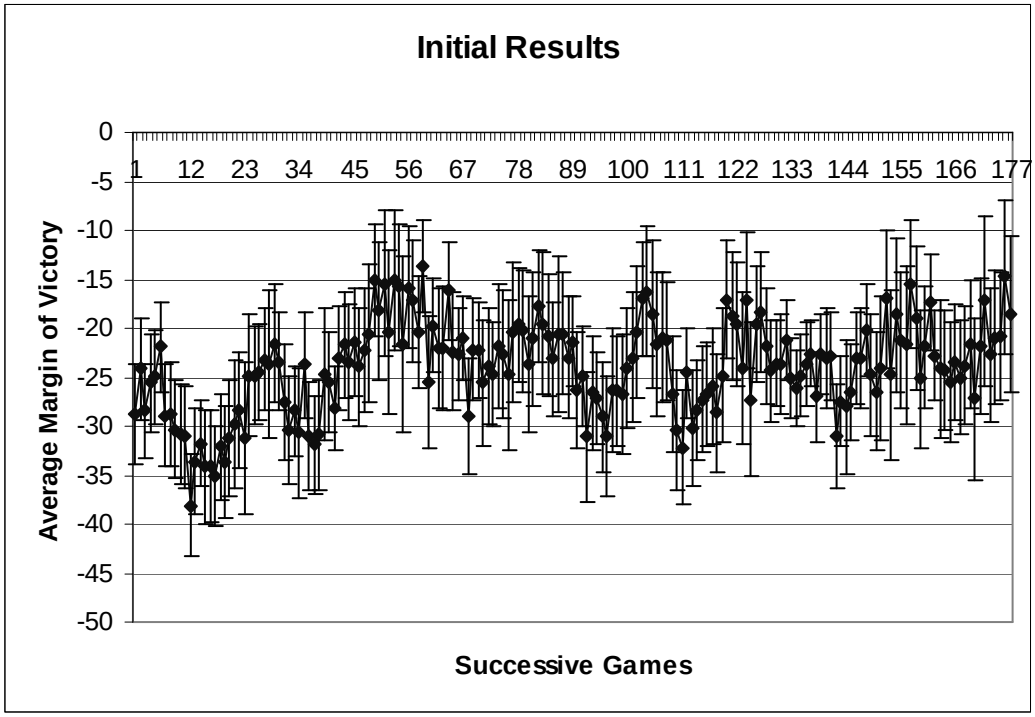

Figure 8-13: Initial Results from Learning from Past Success and Failure

As the graph shows, overall performance versus the control agent was poor. Closer examination of the agent's behavior indicated that the agent was often incorrectly associating particular actions with particular outcomes. In retrospect, this is not surprising given the delayed rewards inherent in the TankSoar domain. We hypothesized that this failure was because all actions leading up to a particular outcome were given equal credit for that outcome. To correct for this, we modified the agent to use a discount factor such that more recent actions would receive more credit for a particular outcome.

We performed the same experiment again with this new agent. The results are shown Figure 8-14. This new agent's performance is much improved with an average margin of victory defeating the control agent.

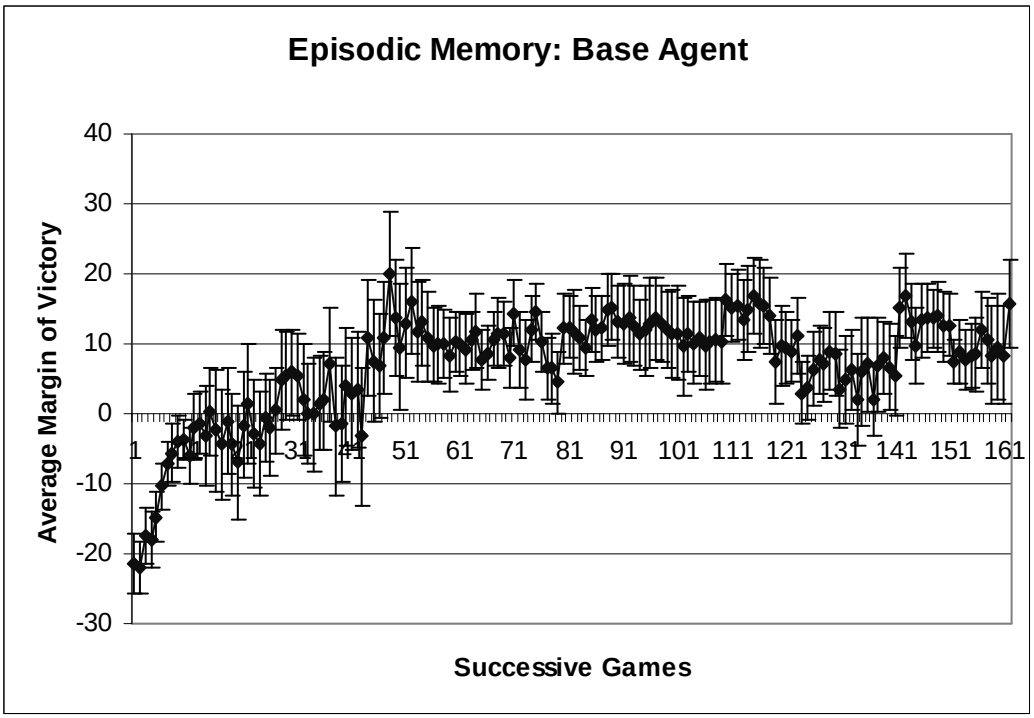

Figure 8-14: Subsequent Results with Learning from Past Success and Failure

Upon further analysis of the agent's behavior, we identified three learned tactics that had a significant impact on its performance. First, the agent learned to dodge hits from short-range enemy missile attacks before they occurred. Second, the agent learned to back away from an enemy while firing its missiles. This delayed the impact of enemy missiles and opened up future opportunities to subsequently dodge. Finally, the agent learned to move out of the enemy's sight (thus leaving the Attack subgoal) when it was in a tactically unfavorable situation.

8.5.1 Removing Heuristic Cue Selection

One aspect of this episodic memory TankSoar agent is its use of a small set or heuristically-selected features of its state to construct a cue. This approach was used in an attempt to simplify the learning problem faced by the episodic memory agent. The agent's state in the TankSoar environment was approximately 2.5 times larger than in Eaters.

Once we had a successful agent in hand, we temporarily removed the heuristic cue content selection and replaced with the use of the agent's entire state. (The to-be-evaluated action continued to be added as normal and the negative cue was left in place.)

We repeated the experiment with the same control agent. The result is shown in Figure 8-15. As the graph indicates, the agent performed much worse without the benefit of this heuristic cue content. However, there is some indication that the agent's behavior does improve gradually over time. Figure 7-11 (in section 7.4.1.6) shows how the performance can be improved by adjusting the cardinality vs. activation ratio to strongly favor working memory activation for feature weighting during the match. However, the overall result is the same: performance is significantly worse without the heuristically selected cue.

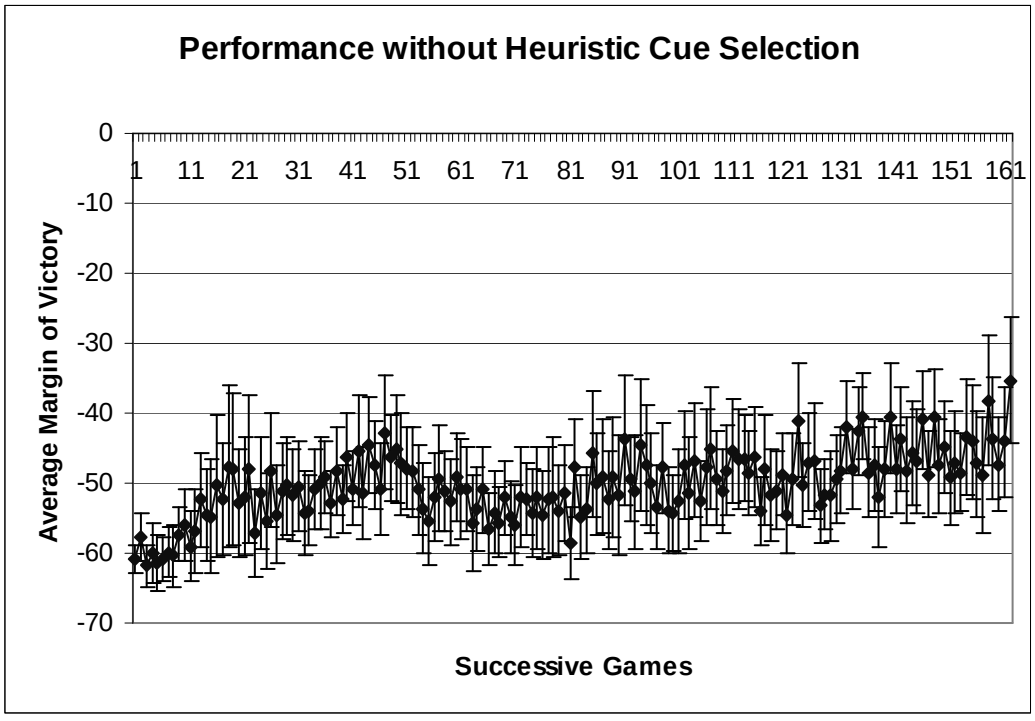

Figure 8-15: Impact of Removing the Heuristic Cue

It remains to be seen whether this agent would eventually learn to outperform the control agent. Extending the experiment to a much longer duration is infeasible with the current architecture due to limited memory resources of this equipment. However, it is

possible that a forgetting mechanism could be used in the future to overcome this limitation (see section 9.2.4.2).

8.5.2 Learning-Based Control Agent

In order to ground the episodic memory agent's performance we performed a second performance comparison with a control agent whose rules were derived by using a learning algorithm rather than the hand-coded decisions made by this original control agent. The work of Chia and Williams (2003) describes a TankSoar agent that uses naïve Bayesian Learning to learn rules for selecting between the Attack, Chase, Retreat and Wander subgoals. As part of their presentation, they describe their learned rules in sufficient detail that were we able to modify this control agent to use the same learned behavior. We then performed the same experiment with this new control agent. This episodic memory agent performed even more effectively than against this original control. The results are shown in Figure 8-16.

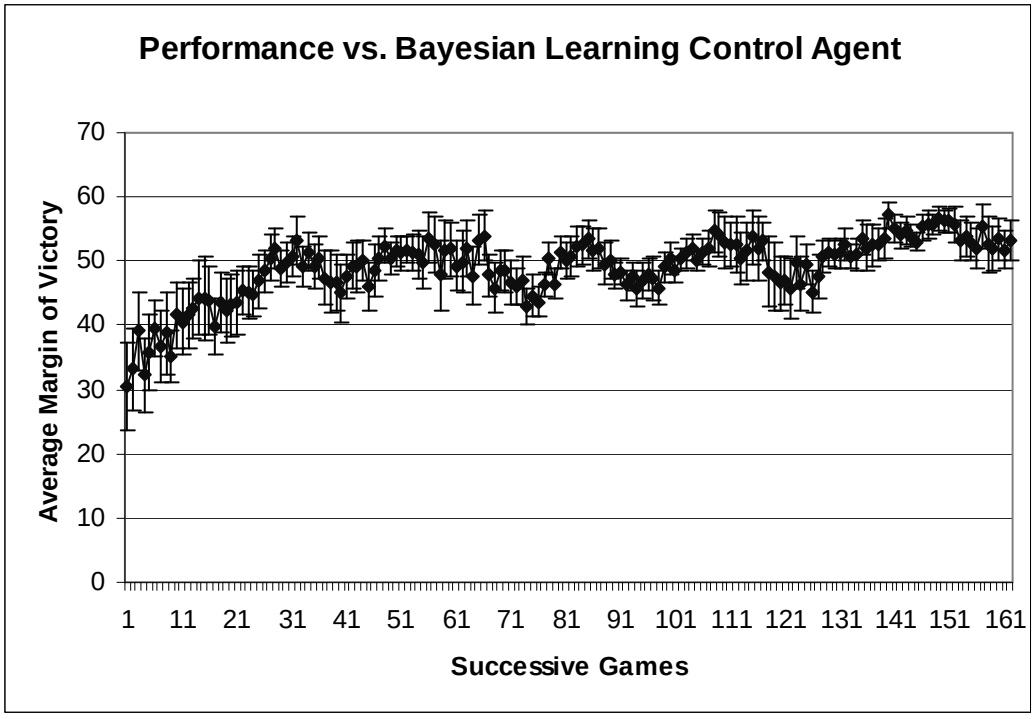

Figure 8-16: Comparison with a Learning-Based Control Agent (Naive Bayesian)

8.5.3 A Reinforcement Learning TankSoar Agent

The TankSoar task is essentially a temporal difference learning task (Sutton & Barto 1998) and thus well suited to reinforcement learning. Since the Soar architecture contains a reinforcement learning module, a reinforcement learning agent can be created. Figure 8-17 shows the performance of such an agent versus the same control agent. The agent's state is defined as the same heuristically selected features[7] that were used to create the cue for the episodic memory agent.

As before, the x-axis represents successive games. The agent retained the knowledge that it learned from game to game. The y-axis measures the agent's margin of victory (i.e., the control agent's score subtracted from the reinforcement learning agent's score). Thus, a negative margin of victory indicates a loss. The results depicted here are the average of thirty repetitions of the same experiment. The error bars in the figure represent the range of 95% confidence.

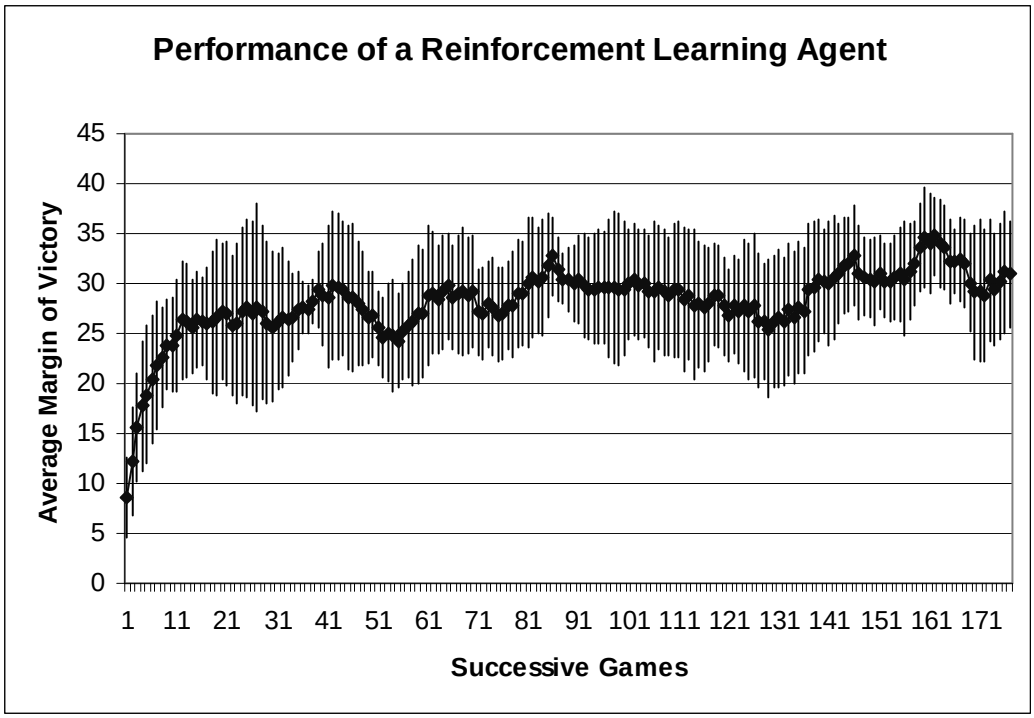

Figure 8-17: Performance of a Reinforcement Learning Agent in the TankSoar Task

[7] The set is not identical as one of the features was omitted since it was not always present.

By comparing this graph to Figure 8-14, it's evident that the reinforcement learning agent outperforms the episodic memory agent by a significant margin. Given reinforcement learning's general suitability for this task and the commitment its knowledge makes to the task, this provides a benchmark for work with this domain.

The episodic memory agent’s more modest performance is mostly due to the one-shot approach to its learning. Success in the task requires a series of correction actions where any error in that sequence hurts overall performance and one improbable event can immediately skew the agent’s performance toward bad behavior in the short to medium term.

Integrating episodic learning and reinforcement learning in order to capitalize on their complementary strengths is a good direction for future research (see section 9.2.1.2).

Chapter 9

Discussion

The preceding chapters of this dissertation present a broad exploration of the challenges and benefits of providing an episodic memory to an intelligent agent. We began by describing and defining in detail what episodic memory is and describing a design space of episodic memories. We then provided a detailed discussion of the cognitive capabilities we expect to be granted to the agent by virtue of possessing an episodic memory. The bulk of this work involved constructing and refining a task-independent and architectural implementation of an episodic memory and using it to demonstrate some of the expected cognitive capabilities.

This chapter reviews the major contributions of this work and also presents the major avenues we perceive for future research.

9.1 Contributions of this Research

The major contributions of this research include the following:

- **Episodic Memory Framework** – This research provides a comprehensive definition of the design space of episodic memory systems. By defining this space, we provide a taxonomy for comparing disparate episodic memory systems

to each other. In addition, the framework guides future research by suggesting "spaces" in this design space that merit investigation.

- **Cognitive Capabilities** – This research defines a list of cognitive capabilities that episodic memory could possibly support in a general agent. By investigating a subset of these capabilities and demonstrating the possibility of supporting them, we strengthen the thesis that a single memory system can, indeed, provide them. The existence of this list also provides a clear objective for future research.
- **Algorithms for Effective Selection** – Any effective episodic memory system must have an effective matching algorithm. Section 7.4.1 presents an overview of existing research in partial matching and this research's position in that field and this relative position in that research (particularly within the context of nearest neighbor matching). Two of the algorithms we use include the approach of using working memory activation as a feature weighting mechanism for memory selection. As far as we are able to discern, this approach is unique.
- **Effective Query Interface** – This research has presented the concept that the effectiveness of a task-independent episodic memory is dependent upon the quality of the interface with the intelligent agent. We perform experiments that demonstrate the value of allowing the agent access to a flexible cueing interface and giving the agent meta-information about the results of an episodic memory query.

  No research that we are aware of presents a focus on the impact of the agent-architecture interface on their mutual performance. In section 9.2.5.4 we present ways in which the existing interface can be expanded.
- **Episodic Memory Performance** – Due to the comprehensive nature of episodic memory there is significant potential for overextending the agent's use of computing resources (i.e., storage space and processing time). This research has gathered data about the impact of an episodic memory system on storage and processing time and how the resource requirements change depending up the size of the episodic store and the matching algorithm being used (see section 7.4.2).

9.2 Future Work

Given the unusual breadth of this research, there are many directions that it could be taken. This section describes four major avenues of research as well as some interesting options within each area.

9.2.1 Integration with other Learning Systems

As we have shown episodic memory can create synergy when it works in cooperation with other learning mechanisms, particularly semantic memory and reinforcement learning. This relationship is illustrated with the chart in Figure 9-1.

	Number of Tasks	**Number of Available Types of Inputs**	**Size of Output**
Episodic Learning	many	many	large
Semantic Learning	many	few	small
Reinforcement Learning	few	many	small

Figure 9-1 Comparison of Episodic Memory and other Learning Systems

Episodic and semantic memory systems are strongest when the information they gather is needed in many contexts. In contrast, a reinforcement learning agent has more difficulty transferring knowledge from task to task. Conversely, the "one-shot learning" approach of episodic and semantic memory systems hinder them in dynamic environments.

Episodic memory cues (and states used by reinforcement learning algorithms) can contain many features. Semantic information is generally requested (and doled out) at a much smaller bandwidth.

One direction for this research is to investigate the potential synergy of these systems in more detail.

9.2.1.1 Semantic Memory

As described in the introduction, semantic memory is knowledge of facts which have been extracted from experience. In humans, this roughly corresponds to what we "know"

as compared to what we "remember." An example of this from this research is the TankSoar agent that used its episodic memory to build a map of directions to the battery.

A deeper investigation of the relationship between semantic and episodic memory seems particularly appropriate. An initial semantic memory system has already been developed for Soar (Wang and Laird 2006) and would likely facilitate this research. Key questions that should be answered include:

- What criteria determine when a portion of an episodic memory becomes a semantic memory?
- How is semantic information abstracted from episodic memories?
- How does semantic memory increase the efficacy of episodic memory cueing and retrieval?
- How does semantic memory increase the efficiency of episodic memory cueing and retrieval?

9.2.1.2 Reinforcement Learning

In section 8.2, we presented an agent that used episodic memory to demonstrate the cognitive capability we call retroactive learning. While we used a simple semantic storage mechanism to learn, reinforcement learning relies upon learning from a series of experiences. As a result, a reinforcement learning algorithm can use episodic memory to train (or retrain) itself to perform correctly in the future using episodic memories of past events. Given that Soar already contains a reinforcement learning mechanism, there is ample opportunity for this line of research.

Key questions to be addressed include:

- Given experiences performing another task, how much knowledge can be transferred to a new task in the same environment?
- Can episodic memory be used to speed up the reinforcement learning module's adjustment to changes in its environment that affect its current task?

9.2.2 Demonstrating More Cognitive Capabilities

This research presents demonstrations of five cognitive capabilities: virtual sensing, action modeling, recording previous success/failures, retroactive learning and boosting other learning mechanisms. However, of this original list this leaves six cognitive capabilities unexplored:

- **Noticing Significant Input** – detecting what is important about a given situation by its relative familiarity
- **Detecting Repetition** – realizing when you're repeating the same series of actions and altering your behavior as a result
- **Environment Modeling –** using past experience to predict how the environment will change
- **Sense of Identity –** understanding ones behavior in relation to other agents
- **Managing Long Term Goals –** keeping track of a plan and what steps in that plan have been accomplished so far
- **Reanalysis Given New Knowledge** – relearning from experience upon receiving new knowledge
- **Explaining Behavior** – explaining your past actions to others for mutual benefit

Investigating each of these would not only complete this original work but would almost certainly provide additional insights into creating an effective episodic memory system.

9.2.3 More Complex Environments and Tasks

Most of the environments and tasks presented in this research were necessarily simple with a graduation to the more significant task of combat in the TankSoar environment. As environments and tasks get more and more complex, they are more difficult to approach. Using more complex environments would test the limits of episodic memory and provide insights on the limits of learning as well as tradeoffs in implementation. Particular dimensions that could be expanded include:

- **Feature Rich Environment** – The more feature rich that an environment is, the larger episodic memories have the potential to be. Larger memories, in turn,

require more resources from the architecture. A feature rich environment also makes it more difficult for the episodic memory system to select which features are important to the agent's current task.

- **Multi-Step Task** – If succeeding at a task requires that the agent take multiple steps in a particular order, the episodic memory system will be required to accurately track long sequences of behavior and, perhaps, distinguish more important parts of those sequence from others.
- **Partially Observable Environment** – An environment which contains many disparate states that appear identical to agent sensing will require the agent's episodic memory to provide past context to agent's current actions.
- **Changing Task** – If the agent's task changes regularly, long term learning algorithms suffer. However, a retroactive learning approach that uses episodic memory (and particularly the temporal information stored in episodic memories) may be more successful. In particular, we envision an agent that reviews events leading to a failure and mentally replays them selecting different actions and using episodic memory to guess the results of these alternative actions. As a result, the agent can be better prepared for future similar situations.

9.2.4 Improving Performance

As was discussed in section 2.3, the resource usage (storage and processing time) of an episodic memory system is a critical consideration. As a result, an available avenue for research is to investigate ways to reduce the resource usage without a large sacrifice in the efficacy of the system. This section discusses some ways in which that could be accomplished.

9.2.4.1 Engineering

To date, most of our efforts to improve the performance of this episodic memory system have focused on improving performance without changing behavior. Specifically, we have made changes to the algorithms and data structures used to store and manipulate the episodic memories. The potential exists to make still more improvements in this manner.

9.2.4.2 Forgetting

A forgetting mechanism is the most straightforward approach to creating a memory store with a constrained size. Previous work on long term learning (Kennedy and De Jong 2003) indicates that in addition to reduced storage an agent's performance can actually improve if low-utility information is removed from its long term memory. Any implementation of this approach requires a task-independent method of deciding which memories are important and which are not. Some possible approaches include the following:

- **Remove the oldest memories first** – This approach guarantees that as many memories as possible are in the episodic store with the least amount of processing. However, it carries the distinct risk that important episodes will be lost.
- **Remove the least activated memory** – A long term activation level on each memory could be used to measure the frequency and recency with which a memory has been retrieved. The least activated memory seems least likely to be needed in the future. This approach bears the most similarity to that used by Kennedy and De Jong.
- **Remove the most redundant memory** – This method would involve locating the two memories that are most similar and removing one of them (probably the oldest or least activated). Ram & Santamaría (1997) used an approach similar to this in their continuous case-based reasoning work though they did not report on its effectiveness vs. a non-forgetting agent.
- **Memory Decay** – In this approach, the system maintains an activation level on individual features of episodic memories. Features with low activation are removed to make room for new memories.

9.2.4.3 Adjusting the Frequency of Recording

Rather than forget existing episodic memories, performance can be improved by recording less memories to begin with. Some possible approaches include the following:

- **Fixed Frequency** – The simplest approach is to simply reduce the overall frequency with which memories are recorded to a fixed value. This is a simple

method, though it stands the strongest chance of missing important events in the agent's history.

- **Significant Change in State** – By monitoring the content of the agent's state, the system can attempt to measure the boundaries between significant events by looking for major changes. This is the method used by this pilot implementation which recorded a new episode whenever the top N most activated features of the state changed. There are alterative ways to measure when the agent's state is "significantly different," including a threshold match between the current state and the last recorded episodic memory and a count of the number of changes to the agent's state since the last recording.
- **High Overall Activation** – The overall activation level of the features of the agent's state can also be used as a relative measure of the agent's arousal or activity level. The more highly activated working memory elements that there are, the more active the agent is. Hypothetically, the most important events will occur during periods of high activity. By recording more frequently during high activity and less frequently during low activity, the memory store might be reduced without missing important episodes.
- **Familiarity Based** – In this method, the agent measures the familiarity of a situation by comparing it to the closest matching memory in the episodic store. If the closest match is "strong", no memory is recorded since the new memory will be similar to one already in the episodic store. The computational overhead required for this approach may be prohibitive.

9.2.4.4 Anytime Algorithm

By implementing key subroutines as anytime algorithms, the amount of time spent performing these operations could be limited to a fixed maximum. Matching routines, in particular, are well suited to an anytime implementation. However, as the episodic store continued to grow, this approach would become less and less effective.

9.2.4.5 Two Stage Match

This current system always retrieves the best partial match. As an alternative, it may be that merely retrieving an excellent (but not necessarily best) match is sufficient. A familiar approach to improving performance at the sacrifice of perfection is to use a two stage match (Forbus, et al. 1994, Cercone, et al. 1997). In the first stage, a small subset of the database is retrieved using a very efficient, but approximate algorithm. This subset is likely (but not guaranteed) to contain the best possible result. In the second stage, a less efficient but accurate algorithm is used to select the best member of the subset.

9.2.5 Advanced Features

The final avenue for future research is to improve the functionality of this episodic memory system. Throughout this research we encountered significant features that were filed away for future work. Often this was because they were too expensive (in terms of implementation time or resource usage) or unnecessary for the work we were doing. With a firm episodic memory system established it may be time to revisit these shelved issues.

9.2.5.1 Agent State as a Secondary Cue

For agents who create very small cues, the idea of using the agent's entire state as a second, lower-priority cue is compelling because it could be used to bias the retrieval toward matching memories that occurred in similar circumstances to the agent's current state. A secondary cue allows for the possibility of spontaneous recognition of a state which may be essential for demonstrating the cognitive capability of detecting repetition. Since a secondary cue is not task-specific in nature, it could be generated and handled automatically by the episodic memory system.

The cost of the feature in system resources could be high. The time required to select a best matching episodic memory is usually dependent upon the size of the cue. Therefore, adding a large, secondary cue would challenge the processing time limits required by the system.

9.2.5.2 Recursive Retrieval

An agent that uses its episodic memory to make decisions may benefit from being able to retrieve episodic memories of retrieving episodic memories. This ability has the potential to allow the agent to keep track of how helpful a particular memory has been to a given task. For example, an agent navigating a complicated series of hallways may use its episodic memory to determine that its best action is to turn right. However, if that same agent can also remember choosing to turn right at that intersection multiple times in the past without reaching its goal, that provides a hint that the robot may be going in circles.

Any implementation of recursive retrieval must avoid the danger of infinite recursion. It must also be able to distinguish a "root" memory from a "meta-memory." We foresee three major approaches to recursive retrieval:

1. A retrieved episodic memory contains the episodic memories which were retrieved at the time of recording. These episodic memories may, in turn, contain further episodic memories, and so on, up to a prescribed limit.
2. Given a particular retrieved episode, the agent can perform a second specialized recursive retrieval using the original episode as a cue. The agent can continue to perform this specialized retrieval indefinitely.
3. A retrieved episode contains particular highlights from the entire chain of remembered episodes thus condensing the entire chain into a single memory.

9.2.5.3 Variable Episode Extent

Most of the agents we used in this research relied upon the ability to retrieve a sequence of episodes in order to determine the short or long term consequences of their actions. In effect, these agents were retrieving meta-episodes consisting of multiple instances in its history. For example, you might remember the instant of making a particular play in a ping-pong match. By placing that episodic memory into the context of a meta-memory which describes "that time I won the ping-pong tournament" provides a context that can improve the agent's sense of the importance or relevance of an episodic memory.

More specifically, there are several potential benefits to the system supporting the concept of meta-episodes directly:

- A longer term meta-episode provides a context for an individual episodic memory. This allows the agent to better cope with situations where the immediate result of an action is poor but the long term result was good (or vice versa).
- If the agent's state is being used as a secondary cue (see 9.2.5.1 above) then that cue could be expanded to include a meta-episode. This would further bias retrieval toward more relevant memories.
- If multiple episodes can be combined into a single meta-episode, memory usage might improve as redundancies among constituent episodes could be condensed.
- Similarly, a meta-episode provides a high level index into the episodic store which could, in turn, reduce the processing time required to select a match for a given cue.

The cost of these benefits is the additional complexity they will add to the episodic memory system. There is also no guarantee that the resources spent in creating and retaining information about meta-episodes will outweigh the performance benefits of having them.

9.2.5.4 Improved Agent and Episodic Memory Interface

In section 7.4.3, we introduced some research on the benefits of improving the communication between the agent and the episodic memory system. The breadth of this communication could be improved still further. Below are two areas for improvement, each of which has already received attention in the lazy learning (Wettschereck 1997) and/or the case-based reasoning (Kolodner 1993) communities.

- **Soft Quantities** – It is advantageous for the agent to have access to more relative entries for the numeric values in a cue. Specifically, the agent may gain from the ability to specify that the numeric value for a given attribute must be greater-than or less-than a given value. The agent may also gain from an architecture that measures the relative difference between the requested value and the actual value in a cue. For example, an agent that creates a memory cue where the radar setting is 13 is likely to get much more value from a partially matching memory with a

radar setting of 12 rather than a setting of 2. The current implementation makes no distinction between these two mismatches.

- **Agent Feedback** – If the agent has a method for providing feedback to the episodic memory system about the utility of a retrieved memory, then the episodic memory system has the opportunity to improve future retrievals. This feedback could take many forms including the following:
    - identification of specifically unfavorable mismatches between a cue and a retrieved memory
    - a general reward based upon the helpfulness of the memory
    - sharing with the episodic memory system the agent's perception of success or failure in its current task.

9.3 Conclusion

Past exploration into the challenges and benefits of implementing an episodic memory for an intelligent agent has focused on individual benefits or challenges of that memory. In addition, previous research has often been implemented in a task specific manner. This dissertation presents this work to achieve a task-independent, architectural episodic memory for general intelligence. By constructing that episodic memory system, we have identified and cataloged the challenges faced by any researcher who attempts to implement such an episodic memory.

Other results presented herein present significant evidence that episodic memory provides several cognitive capabilities that are essential to general intelligence. While any individual cognitive capability could be implemented separately, each individual implementation will contain functionality that is redundant with other implementations and consistent with the capabilities of an episodic memory system. Given these benefits, it seems eminently practical to continue to investigate the role of episodic memory in creating intelligent agents.

Appendix A

Properties of Human Memory

While modeling human episodic memory is not a goal of this research, there are many potential insights hidden in the observations of human episodic memory. This appendix provides a brief list of established properties of human long term memory. Unless noted otherwise, these properties are drawn from Anderson (2000) and Tulving & Craik (2000).

- **Includes Interference Effects** – Interference is ability of a memory to prevent or delay the retrieval of similar memories. This interference can occur either proactively or retroactively. Also, if multiple memories share the same cue the time to retrieve a particular memory is extended ("the fan effect.")
- **Demonstrates Ecphory** – The term "ecphory" was coined by Endel Tulving (Tulving 1983) to describe the fact that memory is constructive. The act of retrieving a memory in the current situation can alter the original memory to include features of the current situation. In effect, episodes can leak over into each other over time. Tulving conjectured that ecphory is the reason that brainwashing techniques are effective on humans.
- **Comprehensive Cue** – Episodic memory storage and retrieval can be influenced by several other factors:
    - Emotional state can be part of an episode and the current emotional state can influence the memory that is retrieved.

    - Physiological state can also influence the effectiveness of episodic storage and retrieval.
    - A phenomenon known as encoding specificity describes the fact that humans are more likely to recall episodes that took place in an environment similar to current one.
- **Demonstrates Priming Effects** – Priming is an increase in the probability that a particular episode will be retrieved. Common examples of the phenomenon include:
    - Frequency and recency of retrieval of an episode increase the probability that an episode will be retrieved again.
    - Repetition of multiple episodes similar to one particular episode, primes that single for future retrieval.
    - Retrieval of an episode increases the probability that similar episodes will be retrieved.
    - Retrieval of semantic facts that are contained in an episode increase the probability of the retrieval of that episode.
- **Forgetting** – Even eidetic memory is not really perfect. Human forgetting bears these properties:
    - The more that an agent processes the events of an episode as they are happening (and being recorded) the longer it will be remembered. Similarly, the features of an episode that were processed by the agent are the last parts to be forgotten.
    - Consciously intending to remember an episode does not improve the likelihood of retrieval.
    - Forgetting follows a power law. Most of the memory content is lost in the short term.

Bibliography

Altmann, E. M. & John, B.E. (1999). Episodic indexing: A model of memory for attention events. Cognitive Science, 23, pp. 117-156.

Anderson, J. R. & Lebiere C. (1998). The Atomic Components of Thought. Mahwah, NJ: Lawrence Erlbaum Associates.

Anderson, J. R. (2000). *Cognitive psychology and its implications*, Fifth Edition, W. H. Freeman and Company, New York.

Atkeson, C., Moore, A. & Schaal, S. (1997). Locally weighted learning. Artificial Intelligence Review, 11(1-5), pp. 11-73.

Baddeley, A., Conway, M. & Aggleton, J. (2002). *Episodic Memory: New directions in research*. Oxford University Press, Oxford.

Brom, C., Pešková, K. & Lukavský, J. (2007). Modelling human-like RPG agents with a full episodic memory. Technical Report No. 2007/4 of the Department of Software and Computer Science Education, Charles University in Prague, Czech Republic (2007).

Burkhard, W.A. (1979). Partial-match hash coding: benefits of redundancy. ACM Transactions on Database Systems 4 (2), pp. 228-239.

Cain, T., Pazzani, M.J. & Silverstein, G. (1991). Using domain knowledge to influence similarity judgement. In *Proceedings of the Case-Based Reasoning Workshop* (pp.1919-202). Washington, DC: Morgan Kaufmann.

Cercone, N., An, A. & Chan, C. (1999). Rule-Induction and Case-Based Reasoning: Hybrid Architectures Appear Advantageous, IEEE Transactions on Knowledge and Data Engineering, 11 (1), pp.166-174.

Chia, C.W. & Williams, K.E. (2003). A Modified Naive Bayes Approach for Autonomous Learning in an Intelligent CGF. In *Behavior Representation in Modeling and Simulation Conference*.

Chong, R. (2003). The addition of an activation and decay mechanism to the Soar architecture. Proceedings of the 5th International Conference on Cognitive Modeling. Bamberg, Germany.

Conway, M. A. (2001). Sensory-perceptual episodic memory and its context: autobiographical memory. In Episodic Memory, ed. A. Baddeley, M. Conway, and J. Aggleton. Oxford University Press.

Dasrathy, B.V., (1990). *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*, IEEE Computer Society Press.

Forbus, K., Gentner, D. & Law, K. (1994). MAC/FAC: A Model of Similarity-based Retrieval. In Cognitive Science, 19, pp.141-205.

Goodman, M. (1993). Projective visualization: Acting from experience. In Proceedings of the Eleventh National Conference on Artificial Intelligence (pp. 54-59). San Jose, CA: AAAI Press.

Grzymala-Busse, J.W. & Wang, C.P.B. (1996). Classification and rule induction based on rough sets, IEEE International Conference on Fuzzy Systems, 2, pp.744-747.

Ho, W.C., Dautenhahn, K. & Nehaniv, C.L. (2003). Comparing different control architectures for autobiographic agents in static virtual environments. Intelligent Virtual Agents, Springer Lecture Notes in Artificial Intelligence, 2792, pp. 182-191.

Kennedy, W.G. & De Jong, K.A. (2003). Characteristics of Long-Term Learning in Soar and its Application to the Utility Problem, In Proceedings of the Twentieth International Conference on Machine Learning (pp 337-344). Washington, DC: AAAI Press.

Kieras, D. & Meyer, D.E. (1997). An overview of the EPIC architecture for cognition and performance with application to human-computer interaction. Human-Computer Interaction, 12, pp. 391-438.

Kolodner, J. (1993). Case-Based Reasoning, Morgan Kaufmann Publishers, San Mateo, CA.

Marr, D. (1971). Simple Memory: A theory for the archicortex. *Philosophical Transactions of the Royal Society of London* B 262, pp. 23-81.

Meek, C., & Birmingham, W. P. (2004). A comprehensive trainable error model for sung music queries. Journal of AI Research (JAIR), 22, pp. 57-91.

Moll, M. & Miikkulainen, R. (1997). Convergence-Zone Episodic Memory: Analysis and Simulations. Neural Networks, 10, pp. 1017-1036.

Newell, A. (1990). *Unified Theories of Cognition.* Harvard University Press, Cambridge, Mass.

Nolan, C. (Director) (2000). Memento [Film]. Newmarket Films. http://us.imdb.com/title/tt0209144/

Nuxoll, A. & Laird, J. (2004). A Cognitive Model of Episodic Memory Integrated With a General Cognitive Architecture. Proceedings of the International Conference on Cognitive Modeling.

Nuxoll, A., Laird, J., James, M. (2004). Comprehensive Working Memory Activation in Soar. Proceedings of the International Conference on Cognitive Modeling.

Pardo, B., Birmingham, W. & Shifrin, J. (2004). Name that Tune: A Pilot Study in Finding a Melody from a Sung Query. Journal of the American Society for Information Science and Technology, vol. 55 (4), pp. 283-300.

Ram, A. & Santamaría, J.C. (1997). Continuous Case-Based Reasoning. Artificial Intelligence, 90(1-2), pp. 25-77.

Rhodes, B.J. (1997). The Wearable Remembrance Agent: A System for Augmented Memory. Personal Technologies.

Schank, R. (1999). Dynamic Memory Revisited. Cambridge University Press.

Sheppard, J. W., & Salzberg, S. L. (1997). A teaching strategy for memory-based control. Artificial Intelligence Review, 11(1-5), pp. 343-370.

Stone, P. (2007). Learning and Multiagent Reasoning for Autonomous Agents. Proceedings of the 20th International Joint Conference on Artificial Intelligence.

Sutton, R. S. & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Tian, Y., McEachin, R.C., Santos, C., States, D.J. & Patel, J.M. (2007). SAGA: A Subgraph Matching Tool for Biological Graphs, Bioinformatics, 23(2), pp. 232-239.

Tulving, E. (1972). Episodic and semantic memory. In E. Tulving and W. Donaldson (Eds.), *Organization of Memory* (pp. 381-403). New York: Academic Press.

Tulving, E. (1983). *Elements of Episodic Memory*. Oxford: Clarendon Press.

Tulving, E. and Craik, F.I.M., editors (2000). *The Oxford Handbook of Memory*. Oxford: Oxford University Press.

Tulving, E. (2002). Episodic Memory: From mind to brain. *Annual Review of Psychology*, 53, pp. 1-25.

Vere, S. and Bickmore, T. (1990). A Basic Agent. *Computational Intelligence*. 6, pp. 41-60.

Wang, Y., and Laird, J.E. (2006). Integrating Semantic Memory into a Cognitive Architecture. University of Michigan Center for Cognitive Architecture Technical Report #CCA-TR-2006-02, Ann Arbor, MI.

Wettschereck, D., Aha, D.W., & Mohri, T. (1997). A review and comparative evaluation of feature weighting methods for lazy learning algorithms. Artificial Intelligence Review, 11, pp. 273-314.