Performance Evaluation of Declarative Memory Systems in Soar

John E. Laird, Nate Derbinsky, Jonathan Voigt
University of Michigan
2260 Hayward Street
Ann Arbor, MI 48109-2121
laird@umich.edu, nlderbin@umich.edu, voigtjr@umich.edu

Keywords:
performance evaluation, memory and learning, cognitive architecture

ABSTRACT: *A rarely studied issue with using persistent computational models is whether the underlying computational mechanisms scale as knowledge is accumulated through learning. In this paper we evaluate the declarative memories of Soar: working memory, semantic memory, and episodic memory, using a detailed simulation of a mobile robot running for one hour of real-time. Our results indicate that our implementation is sufficient for tasks of this length. Moreover our system executes orders of magnitudes faster than real-time, with relatively modest storage requirements. We also project the computational resources required for extended operations.*

1. Introduction

There are few computational challenges to model behavior over short time spans when there is minimal prior knowledge or when there is little to no accumulation of knowledge or experience. The challenges arise when we attempt to model behavior over long time spans where an agent builds up internal structures based on its experience. As these structures accumulate, the cost of adding and accessing that knowledge can grow beyond available computational resources (Derbinsky, Laird, & Smith, 2010; Douglass, Ball, & Rogers, 2009). With efficient models, we can test and evaluate them faster, using cheaper systems, and use them in real-time tasks.

The emphasis of our research is on tasks that require human-level reasoning, memory, and learning. We are less concerned with the detailed modeling of human behavior, where the goal is to match human reaction times and error rates. Instead, we are interested in creating models that perform complex tasks, using and acquiring large stores of knowledge across extended time spans, often requiring significant internal processing and planning. TacAir-Soar (Jones et al., 1999) and RWA-Soar (Hill et al., 1997), two systems that modeled U.S. pilot tactical behavior in fixed wing and rotary wing vehicles, have many of these qualities, although they did only limited planning and did not learn from experience.

Over the last five years, we have extended the Soar cognitive architecture with semantic and episodic memory (Laird, 2008). In previous work, we evaluated the performance of those memories, but our evaluations had short comings: either they used artificial tasks (Nuxoll & Laird, 2007), or they focused on using pre-loaded knowledge and not on knowledge that accumulates through experience (Derbinsky & Laird, 2009; Derbinsky et al., 2010). Evaluations of other declarative memories in cognitive architectures have also focused on preloaded structures (Douglass et al., 2009; Douglass & Myers, 2010), whereas evaluations of episodic memory have been restricted to small numbers of preloaded episodes (as in research on case-based reasoning) or small numbers (~250) of episodes (Tecuci & Porter, 2007).

To fill this void, in this paper we evaluate performance within a simulation of a real-world task: a mobile robot exploring, navigating, and moving objects in a small building. In this task, the agent's declarative memories build up incrementally over an hour of real-time execution, and the agent's perception of the environment is based on detailed models of real-world sensory data. One goal is to discover the requirements for semantic and episodic memory in such a task and whether our implementations are sufficient to support real-time behavior over long time scales. A second goal is to discover the relative costs and benefits of the different memory systems for real-world tasks. One justification for adding semantic memory to Soar was the concern that maintaining large number of elements in working memory would significantly degrade performance, independent of its cognitive implausibility (Derbinsky & Laird, 2010). A third goal is to discover interactions between semantic and episodic memory. Soar provides a unique opportunity to pursue these goals.

2. The Soar Cognitive Architecture

Figure 1 shows the structure of Soar. Perception delivers symbolic structures to working memory, which is a symbolic graph. All the long-term memories retrieve information based on the contents of working memory and add, delete, or modify working memory structures.

The knowledge in procedural memory is encoded as rules, which match and fire in parallel to create working memory structures as well as preferences for selecting *operators*. Operators are the locus of decision making and include both primitive actions, such as moving or turning a robot, and abstract actions such as find-a-block, go-to-the-next-room, or go-to-the-storage-room. The decision procedure analyzes the preferences and selects the current operator by adding a structure to working memory.

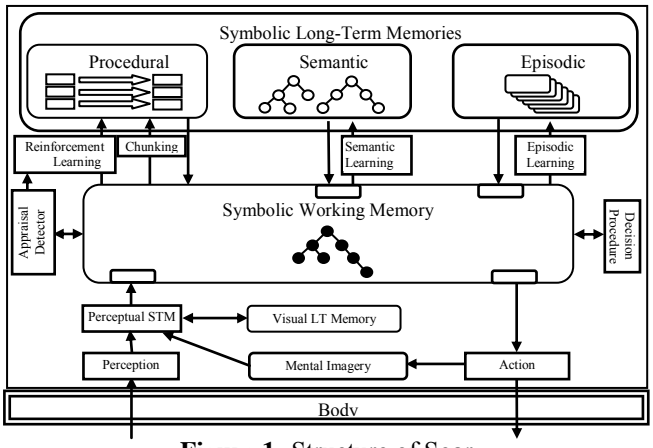

Figure 1: Structure of Soar

Abstract operators are dynamically decomposed into simpler operators until a primitive operator is selected. Once a primitive operator is selected, rules match and apply the operator's actions. For internal operators, such as building up an internal map, changes are made to working memory. For external operators, such as moving the robot, commands are added to the output buffer in working memory, and are sent to the motor system. Retrievals from other long-term memories are initiated by creating cues in those memories' buffers.

The basic cycle is to process input, fire rules that match, select an operator, fire rules to apply the operator, and then process output commands and retrievals from long-term memory. The time to execute this processing cycle determines reactivity. Based on our experience with cognitive modeling, human behavior models, and robotic systems, 50 msec. is sufficient for real-time reactivity. As evident in Figure 1, Soar has additional memories and processing modules; however, they are not evaluated in this paper, and are not discussed further.

3. Mobile Robot Domain

Soar has been used to control both real (Laird & Rosenbloom, 1990; Laird et al., 1991) and simulated vehicles (Hill et al., 1997; Jones et al., 1999). For this evaluation, we use a system where Soar[1] controls a small mobile robot (Figure 2; Laird, 2009). Our evaluation uses a simulation instead of the real robot because of the difficulties in running experiments in the large physical spaces required. Moreover, the simulated robot can run unattended on multiple computers at once. When Soar controls the real robot, it runs on a laptop perched on the robot as shown in Figure 2. The simulation is quite accurate and the Soar rules (and architecture) used in the simulation are *exactly* the same as the rules used to control the real robot.

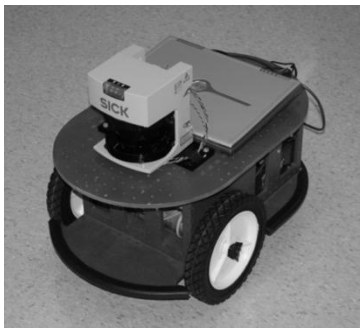

Figure 2: Splinterbot mobile robot.

The robot can move forward and backward, and turn in place. It has a laser-range finder mounted in the front that gives distances to 180 points throughout 180 degrees. Our software condenses those points to 5 regions that are sufficient for it to navigate and avoid obstacles. The robot can sense its own location based on odometry, and it can also sense the room it is in, the location of doors and walls, and different types of objects. These additional sensors are simulated, and when controlling the real robot, it uses a synthesis of real and simulated sensor data. During a run (both real and simulated), there are approximately 150 sensory data elements, with approximately 20 values changing each cycle. The changes can peak at 260 per cycle when the agent moves from room to room because all the data about the current room, walls, and doorways change at once.

Throughout the paper, we refer to the robot, Soar, and rules as an *agent*. We evaluate two agents that differ in their rules. The first is the "simple agent," which serves as a knowledge-lean baseline for comparison. This agent explores randomly, moving from room to room, without maintaining any persistent. It consists of 22 rules.

The second, "complex agent" can perform multiple tasks, including cleaning rooms and patrolling. We focus on room cleaning where it picks up and moves specific types of blocks (such as square green blocks) to a storage room. The complex agent dynamically constructs a map of the rooms, doorways, and their spatial and topological layout, as well as the locations of all blocks as it encounters them

[1] Experiments used Soar 9.3.1, which is available with the simulation environment through sitemaker.umich.edu/soar.

during its explorations. Each room, block, wall, and doorway is represented by 6-9 data elements in the agent's memories. The agent initially has no map information except for the identity of the storage room (but not its location).

When attempting to travel to a distant room, such as when dropping off a block at the storage room, the agent internally simulates moving to neighboring rooms in its map, searching for a path to the storage room. This "simulation" does not use the external simulator, but the agent's map knowledge. The search is a combination of progressive deepening (Newell & Simon, 1972) and A* search (Hart, Nilsson, & Raphael, 1968). The agent also plans when it needs to return to a previously seen block, or when it attempts to move to a room it has detected but not visited. The internal search is performed within the agent, using rules, spread across multiple cycles, made possible by Soar's support for a Universal Weak Method (Laird & Newell, 1983).

The complex agent has 562 rules and is parameterized so that by modifying data elements in working memory, it performs all the variations in behavior described in this paper, including using either working memory or semantic memory to store map information, and using either working memory, semantic memory, or episodic memory to store block location information.

The experiments are performed using a map with 25 rooms connected via 24 doorways in a straight line from north to south. This map eliminates most variations in the agent's behavior that would arise from random decisions in a less structured map. There are a total of 44 blocks spread across the 25 rooms, 23 of which satisfy the agent's criteria for blocks it wants to move to the storage room. These blocks are spread across the rooms so that there is at most one block per room. The agent starts at the north end and the storage room is in the southern-most room. Although the rooms are in a straight line, the agent must plan after it picks up a block to determine whether to go north or south to get to the storage room.

All experiments are run for one hour of elapsed real-time, during which the agent completes approximately 1/3 of the task. During 1 hour, our agents execute ~25,000,000 processing cycles. The data is aggregated every 10 seconds, which is ~70,000 processing cycles. We focus on the following performance metrics.

- Average working memory size. Although Soar's rule matcher is relatively immune to growth effects from the number of rules, it is not immune to growth in the number of working memory elements tested by rules.
- Average msec./processing cycle. This indicates how much time is required for a processing cycle. For these tasks, we expect the average to be low because usually the agent is doing minimal processing as it moves in a straight line toward its next destination (such as a doorway or a block).
- Maximum msec./processing cycle. This indicates the "surge" in computational requirements, which can be orders of magnitude higher than the average. The value of this metric determines real-time reactivity. Our goal is to be below 50 msec.
- Total long-term memory size. This measures how much computer memory is required to hold the structures in semantic and episodic memory. To maintain efficient access, Soar's memories are maintained in the computer's main memory (RAM). 96 GB is a reasonable limit for current workstations.

The data are averaged over 5 runs, except for the maximum msec./processing cycle. For those data, we use a representative run. Because Soar waits for the simulation environment at the end of each cycle, Soar uses only a fraction of the available processing time (~7.5%). All experiments were run on an Intel i7 860@2.8Ghz, with 8 GB of memory. In the following sections, we evaluate working memory, semantic memory, and then episodic memory.

4. Working Memory

As mentioned earlier, working memory in Soar is a graph, with each edge of the graph being a working memory *element* that consists of an *identifier*, an *attribute*, and a *value*. Each element can be added or removed independent of other elements. This is in contrast to ACT-R (Anderson, 2007), which bundles structures together as chunks, which are added, modified, or removed as a single unit. However, we will often refer to all of the working memory elements that share a common identifier as an *object*, which thus corresponds roughly to a chunk, with the object identifier corresponding to a chunk id. Objects in Soar refer to entities in the world (rooms, doors blocks), internal concepts, words, and so on. There is no bound on the number of elements in working memory, and working memory grows and shrinks over the lifetime of an agent as elements are added and removed.

Figure 3 shows the maximum number of working memory elements (WMEs) for the simple and complex agents. As expected, working memory in the simple agent does not grow. The complex agent builds up working memory as it explores, but once it visits all rooms (at ~770 seconds) memory levels off. As evident in the figure, ~ 3,000 elements are needed to represent the map and blocks.

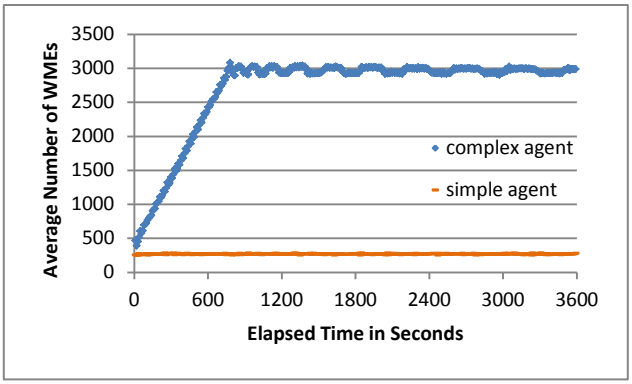

Figure 3: Average size of working memory in the simple vs. complex agents.

Given the large working memory in the complex agent, our expectation is that we will see a significant degradation in performance in the cycle time of the agent as the memory grows. This is because the underlying algorithm for matching rules in Soar, called Rete (Forgy, 1982), is designed to scale well with the number of rules, but not with the size of working memory. As more working memory elements are added, it is often the case that there are more ways a rule can match, and in some cases, that can lead to more than linear growth in the computational processing required to determine which rules match. In the specific task performed by our agents, the number of blocks to be picked up leads to such an increase when the agent must decide which block to pick up next after depositing a block in the storage room.

Figure 4 shows the average CPU time in msec. per processing cycle. Both agents maintain consistent performance throughout the task, showing little, to no degradation over time, with the complex agent being slower on average. Moreover, the average time (<.015 msec./cycle) is orders of magnitude faster than required for real-time performance (50 msec.).

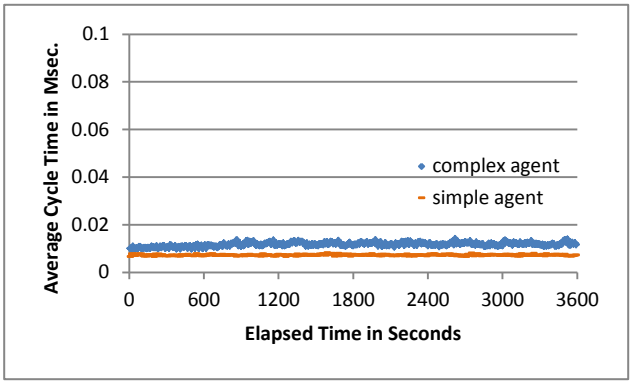

Figure 4: Average processing time per cycle in msec. in the simple vs. complex agents.

To explore the performance of the complex agent in more detail, in Figure 5 we examine the maximum time spent in a processing cycle. The simple agent has a low, stable maximum msec./processing cycle, while the complex agent has significantly more variation. That variation initially increases as the map is built up (0-770 seconds). From that point on, the highest points arise when the agent is choosing which block to pick up next.

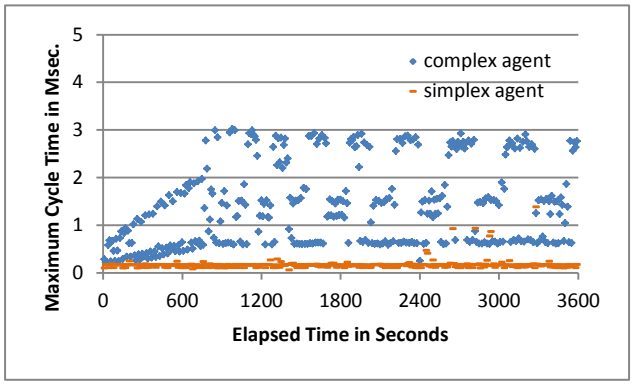

Figure 5: Maximum time per cycle in msec. in the simple vs. complex agents.

The high values are the exception, and even with them, the average cycle time is hundreds of times less at ~.02 msec. Moreover, the variation is minimal when taken within the context of achieving a cycle time of less than 50 msec. Thus, an important result is that maintaining all structures in working memory is adequate for this task.

5. Semantic Memory

In Soar, semantic memory holds long-term, persistent declarative structures that generally correspond to facts about objects or concepts. For this task, that includes the map and the location of the blocks. In Soar, structures in semantic memory are deliberately stored from working memory. Semantic memory can also be pre-loaded with structures from external knowledge bases such as WordNet (Derbinsky et al., 2010). To speed retrievals, our implementation uses an inverted index (similar to what is done in search engines), combined with statistical query optimizations, using SQLite as a backend.

To retrieve structures from semantic memory, a Soar agent creates one of two types of cues in a working memory. In cue-based retrievals, the cue consists of multiple working memory elements that provide a partial description of an object. The semantic memory system finds the object in semantic memory that best matches the cue. The search is biased by recency, and there is no "spreading" of activation as is sometimes used in ACT-R. In identifier retrievals, the object identifier is already in working memory, but its entire substructure is not. For example, when room structures are retrieved, they include

the identifier of each doorway. Using this type of retrieval, the complete representation of doorway objects can be brought into working memory.

For these experiments, the map of the rooms is incrementally added to semantic memory as the agent encounters a new room or block. To minimize working memory, the agent removes representations of distant rooms from working memory. One complexity is that when the agent is performing an internal look-ahead search, it must retrieve room structures from semantic memory as it "imagines" moving through those rooms.

The use of semantic memory to hold rooms and objects can change some aspects of agent behavior. One example is when the agent has just dropped off a block in the storage room and needs to decide which block it should pick up next. When the blocks are all in working memory, a rule can match all blocks and create a pickup operator for each one, and the agent can compute the distance to each block and use that information in choosing an operator. When using semantic memory, the agent cues a retrieval using the characteristics of the object, such as its color and shape, but cannot use distances to the agent, which are not maintained in semantic memory because they are constantly changing. Instead, the agent retrieves the most recent object it has seen.

Figures 6-8 compare an agent that stores the map in working memory (this is the same agent as the complex agent in Figures 3-5) with an agent that stores the map in semantic memory. The total number of objects stored in a run is ~1000, with ~700 cue-based retrievals and ~700 identifier retrievals. The structures in semantic memory take ~.6 MB.

When the map structure is maintained in semantic memory, the average size of working memory is significantly less than when the map structure is maintained exclusively in working memory (Figure 6). However, during internal searches, room structures are retrieved, temporarily boosting the number of elements in working memory.

Figure 7 shows that the average cycle time is higher when using semantic memory, which includes additional costs for storing and retrieving items from semantic memory, as well as removing structures from working memory. However, Figure 8 shows that the maximum msec./processing cycle when structures are stored in semantic memory is less than that of working memory. The range for working memory is broader and higher; however, there is a slight upward trend with semantic memory. Using semantic memory eliminates some

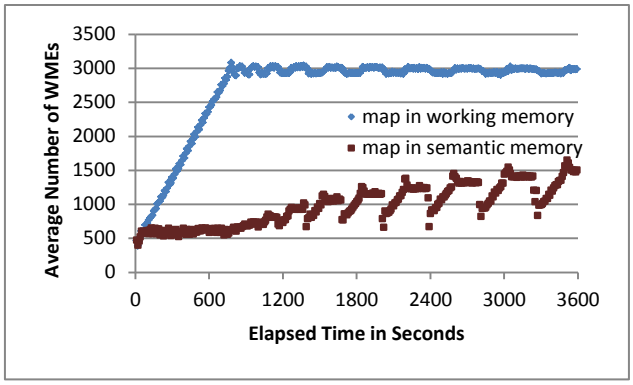

Figure 6: Average size of working memory using semantic vs. working memory in the complex agent.

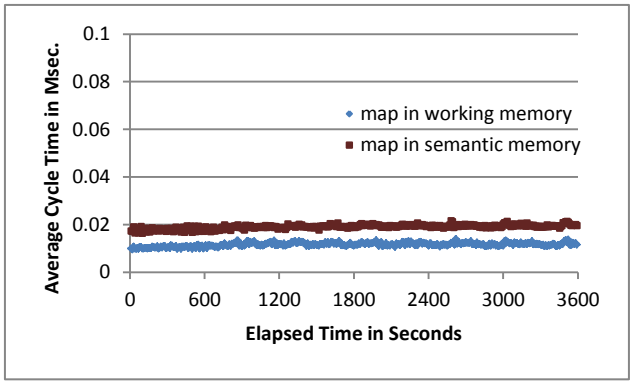

Figure 7: Average cycle time using semantic memory vs. working memory in the complex agent.

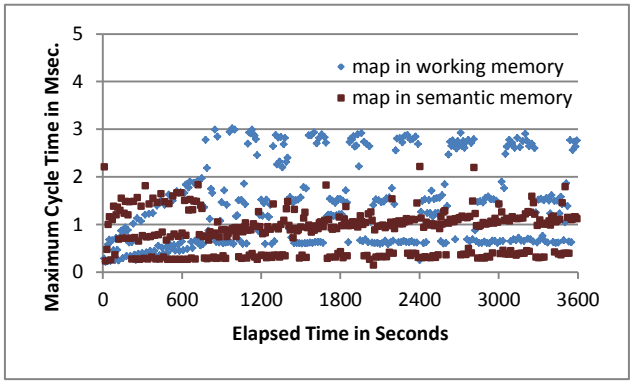

Figure 8: Maximum cycle time using semantic memory vs. working memory in the complex agent.

expensive choices made when all objects are represented in working memory, which leads to a lower maximum. However, it adds the costs for retrievals and removals from working memory, which leads to the higher average. One reason for that the working memory agent is fast on average is that the map is a stable structure throughout the task. The Rete matcher only does work when there are *changes* to working memory, which minimizes the costs of maintaining it in working memory.

6. Episodic Memory

Episodic memory holds a history of the agent's experiences and provides the complete context of an experience. In contrast, semantic memory contains specific facts, independent of context (unless the context is specifically included in the fact, such as "the Watergate break-in occurred on June 17, 1972").

In Soar, an episode is a "snapshot" of working memory. An agent can retrieve an episode by creating a cue, but in contrast to semantic memory, the cue can include multiple objects, and in the extreme can consist of all of working memory. The retrieval is based on finding the most recent episode that "best" matches the cue, and Soar uses a variety of techniques to minimize the time to find the best match (Derbinsky & Laird, 2009).

Although it is possible to record episodes on every processing cycle, we have found that with this and other tasks, it is sufficient to record episodes only when the agent takes an external action. Especially in this task, where the agent is usually just moving forward, most situations are not distinctive nor worth remembering. By restricting it to record when there is an action, the agent records an episode about every 400 msec., which results in the agent storing over 9,000 episodes. In addition, episodes do not include cues or retrievals from semantic and episodic memory in episodes, nor structures created in the look-ahead searches. In this experiment, the episodes include all map and block locations that are in working memory, and other internally created structures, but not raw perceptual structures.

In this task, retrievals from episodic memory provide the location of a previously seen block that the agent needs to pick up. To retrieve a block from memory, the agent creates a query with a description of the object, such as a "green square block," and specifies that it not be one of the objects already in the storage room. The task is organized such that the agent picks up blocks in the reverse order from how they were originally experienced. Thus, we expect the cost of using episodic memory to increase during the task because not only do more and more episodes need to be searched, but more and more episodes must be skipped that contain blocks that have already been moved to the storage area.

In these conditions, the size of working memory mirrors the results in Figure 6. We expect that by maintaining the map in semantic memory, which allows working memory to be smaller, the size of episodes should decrease, which in turn should speed episodic memory retrievals.

Figure 9 shows the average time per processing cycle. The results from the earlier sections without episodic memory are included for comparison and they are the lower two data sets. The top two data sets show episodic memory when the map is in working memory, and above that, when it is semantic memory. We see that there is overhead to using episodic memory, and counter to our expectations, on average, using it with semantic memory is more costly than with working memory.

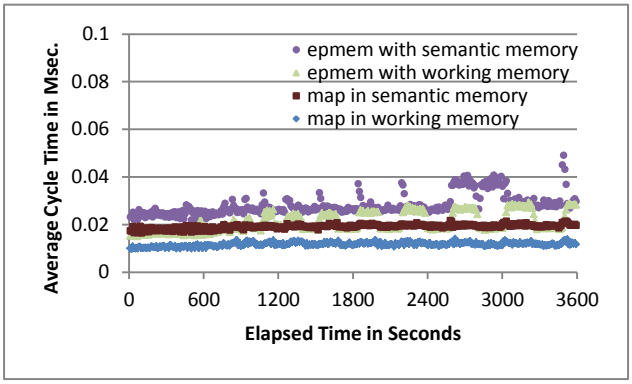

Figure 9: Average cycle time with episodic memory.

Figure 10 shows the maximum cycle times and expands the y axis from previous figures to 50 msec. This shows that using episodic memory comes at a significant cost, with the maximum cycle time being an order of magnitude higher than those in Figure 8.

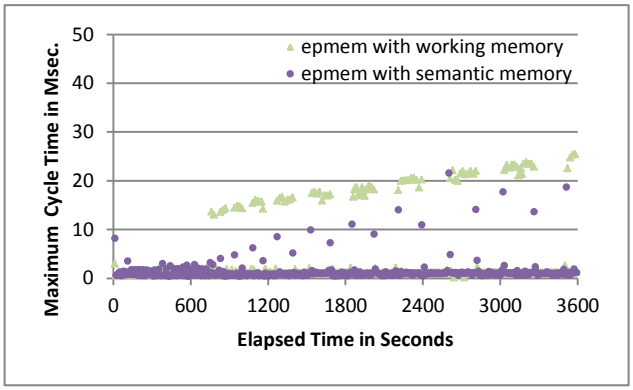

Figure 10: Maximum cycle time with episodic memory.

The high maximum times occur when the agent retrieves an episode. Even though these costs are significant, the maximum cost is still lower than our target of 50 msec. This graph shows that the combination of episodic memory and semantic memory has a lower maximum cycle time than episodic memory and working memory, even though it is worse in the average case. This is because when the complete map is maintained in working memory, every episode contains the complete map, so reconstruction is more expensive.

Finally Figure 11 shows the amount of memory required for the two conditions with episodic memory, which is surprisingly small. Our original expectation was that using semantic memory would *decrease* the amount of storage required because its episodes are smaller; however, episodic memory is optimized so that new structures are only stored when there are changes to working memory. When the map is held in working memory, no additional storage is required for the existing structures of the map in new episodes, only for the changes. In contrast, when the map is stored in semantic memory, subsets of the map are removed from working memory and then later retrieved, which requires additional storage to record those changes.

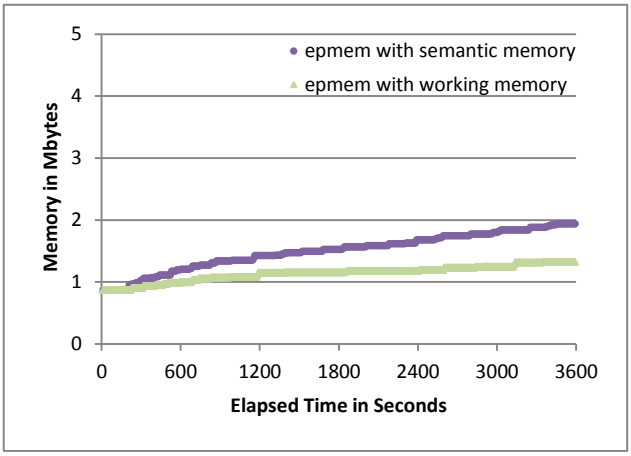

Figure 11: Memory requirements with episodic memory.

7. Discussion

One purpose of this research was to examine the tradeoffs between the different memory systems, especially in terms of the computational cost of using in a task with real-time constraints. For the experiments we have run, keeping all of structures in working memory works surprisingly well. We expected that as the size of working memory grew, the cost of matching rules would become prohibitively expensive.

We also discovered that semantic memory is efficient for storing the number of structures we need in this task, and scales well. There is a performance cost to using it, in terms of average processing time per cycle, but it performs well in terms of maximum processing cost. This bodes well for using semantic memory in the future.

There are additional costs associated with using episodic memory, but in this experiment they do not exceed our threshold of 50 msec. The results of episodic memory also show that in our implementation there are significant costs for large working memories during reconstruction of episodes. Ironically, maintaining a smaller working memory requires more storage for episodic memory because of the increase in the changes to working memory.

Another of the purposes of this research is to evaluate the memory systems in a cognitive architecture on a more realistic task than has been done in the past. One conclusion of this work is that the memory systems we have developed in Soar are sufficient to support real-time behavior for this length of task. On average, Soar is thousands of times faster than it needs to be to achieve real-time performance, and even in the worst cases, it is fast enough. However, this task lasted for one hour, over a relatively limited map, so the question remains as to how these memory systems scale to longer times and larger memory structures.

In previous research (Laird & Derbinsky, 2009), we had attempted to predict what would be the requirements for an agent that learned using episodic memory for a year. Here we build on that analysis, but use our experience with this task to refine our estimates. We observe that for this task the number of episodes that need to be stored is modest (9,000 over one hour), and the size of each episode is relatively small (as evidence by the total memory requirements in Figure 11). Both of these quantities are much smaller than our original estimates.

Extrapolating from Figure 11, where in the worst case 2Mbytes are required per hour, we predict needing 48Mbytes/day so that we can expect to run for ~2,000 days before exceeding 96 Gbytes. These estimates are dependent on the details of our task, and assume linear growth, but provide a ballpark as to what is practical with current algorithms and computer systems.

The time to use semantic and episodic memory is more difficult to evaluate. The time to access semantic memory appears stable; however, this task does not stress semantic memory compared to previous research with WordNet. In this task there was a maximum of 1,000 objects stored in semantic memory compared with 800,000 in WordNet, and in this task, the structures are built up by experience as opposed to loaded from a database, so that the incremental additions to semantic memory in a typical task might also be small. An open question is whether this is the normal use of semantic memory or whether other applications require storing significantly more data.

Our results for episodic memory suggest that the time to retrieve items from episodic memory will exceed 50 msec. after a few hours, although the exact trends are hard to predict, and this task is designed to stress episodic memory retrievals by forcing the agent continually retrieve earlier and earlier episodes from memory.

There are two approaches to deal with the cost of searching episodic memory. The first is to examine alternative algorithms and data structures for episodic memory. That is part of our ongoing research. A second approach is to decouple the processing of semantic and episodic memories from the main processing cycle so that they run asynchronously in separate processing threads and cores. This gives two advantages. First, it increases parallelism, but second, the processing for accessing the memory no longer directly impacts the reactivity of the system. If we look at humans, access to both semantic and episodic memory occurs in parallel with procedural reasoning and at time scales on the order of 500 msec. This change would possibly extend the practicality of using episodic memory to a day for this task.

8. Acknowledgments

This research was supported in part by the Ground Robotics Reliability Center (GRRC) at the University of Michigan, with funding from government contract DoD-DoA W56H2V-04-2-0001 through the US Army Tank Automotive Research, Development, and Engineering Center and the Office of Naval Research under grant number N00014-08-1-0099.
UNCLASSIFIED: Dist. A. Approved for public release.

9. References

Anderson, J. R. (2007). *How Can the Human Mind Occur in the Physical Universe?* Oxford University Press.

Derbinsky, N., and Laird, J. E. (2009). Efficiently Implementing Episodic Memory, *Proceedings of the International Conference on Case-based Reasoning*.

Derbinsky, N., and Laird, J. E. (2010). Extending Soar with Dissociated Symbolic Memories. *Proceedings of the Symposium on Human Memory for Artificial Agents*, 36th AISB.

Derbinsky, N., Laird, J. E., and Smith, B. (2010). Towards Efficiently Supporting Large Symbolic Declarative Memories, *Proceedings of the 10th International Conference on Cognitive Modeling*.

Douglass, S. A., Ball, J., and Rodgers, S. (2009). Large Declarative Memories in ACT-R. *Proceedings of the 9th International Conference on Cognitive Modeling*.

Douglass, S. A., Myers, C. W. (2010). Concurrent Knowledge Activation Calculation in Large Declarative Memories. *Proceedings of the 10th International Conference on Cognitive Modeling*.

Forgy, C. L. (1982). Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. *Artificial Intelligence* 19 (1) 17-37.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths, IEEE *Transactions on Systems Science and Cybernetics SSC4* **4** (2): 100–107.

Hill, R. W., Chen, J., Gratch, J., Rosenbloom, P., and Tambe, M. (1997). Intelligent Agents for the Synthetic Battlefield: A Company of Rotary Wing Aircraft. *Proceedings of AAAI 1997*, 1006-1012.

Jones, R. M., Laird, J. E., Nielsen, P. E., Coulter, K. J., Kenny, P., and Koss, F. V. (1999). Automated Intelligent Agents for Combat Flight Simulation. *AI Magazine*, 20(1), 27-42.

Laird, J. E. (2008). Extending the Soar Cognitive Architecture. *Proceedings of the First Conference on Artificial General Intelligence*.

Laird, J. E., Derbinsky, N. (2009). A Year of Episodic Memory. *Workshop on Grand Challenges for Reasoning from Experiences, IJCAI-2009*.

Laird, J. E., and Newell, A. (1983). A Universal Weak Method: Summary of Results. *Proceedings of IJCAI*.

Laird, J. E., and Rosenbloom, P. S. (1990). Integrating Execution, Planning, and Learning in Soar for External Environments. *Proceedings of AAAI 1990*. 1022-1029.

Nuxoll, A. M., and Laird, J. E. (2007). Extending Cognitive Architecture with Episodic Memory. *Proceedings of AAAI 2007*.

Tecuci, D., Porter, B. (2007). A Generic Memory Module for Events. *Proceedings of the 20th Florida Artificial Intelligence Research Society Conference*.

Author Biographies

JOHN LAIRD is the John L. Tishman Professor of Engineering at the University of Michigan. His research focuses on cognitive architecture, with emphasis on the Soar architecture. He is a Fellow of AAAI, ACM, and the Cognitive Science Society.

NATE DERBINSKY is a Ph.D. candidate in Computer Science and Engineering at the University of Michigan. His research focuses on functionality and efficiency for long-term declarative memories in cognitive architectures.

JONATHAN VOIGT is a research programmer at the University of Michigan. He supports the development of the Soar architecture and associated tools and environments, including robot simulators and interfaces.