Effective and Efficient Management of Soar's Working Memory via Base-Level Activation

Nate Derbinsky and John E. Laird

University of Michigan

2260 Hayward St.

Ann Arbor, MI 48109-2121

{nlderbin, laird}@umich.edu

Abstract

This paper documents a functionality-driven exploration of automatic working-memory management in Soar. We first derive and discuss desiderata that arise from the need to embed a mechanism for managing working memory within a general cognitive architecture that is used to develop real-time agents. We provide details of our mechanism, including the decay model and architecture-independent data structures and algorithms that are computationally efficient. Finally, we present empirical results, which demonstrate both that our mechanism performs with little computational overhead and that it helps maintain the reactivity of a Soar agent contending with long-term, autonomous simulated robotic exploration as it reasons using large amounts of acquired information.

Introduction

Long-living, learning agents facing numerous, complex tasks will experience large amounts of information that may be useful to encode and store as internal knowledge. This information may include declarative facts about the world, such as lexical data (Douglass, Ball, and Rodgers 2009), or may relate to the agent's own autobiographical experience (Laird and Derbinsky 2009).

In order to scale to human-level intelligence, cognitive architectures designed to support these types of agents must encode and store experience such that the agent can later retrieve relevant information to reason and make decisions, while remaining reactive to dynamic environments (Laird and Wray 2010). In part to satisfy these requirements, the Soar cognitive architecture (see Figure 1; Laird 2008) has been recently extended with multiple dissociated, symbolic memory systems (Derbinsky and Laird 2010): working memory encodes structures that are directly accessible to agent reasoning, while long-term memory systems support flexible, yet indirect, access to skills, facts, and past experience.

These memory systems have introduced both the computational necessity and the functional opportunity to investigate architectural methods for maintaining a small working memory. The necessity arises because large amounts of knowledge in Soar's working memory can impede agent reactivity as a result of expensive rule matching and episodic reconstruction. However, the advent of semantic memory affords the agent the functionality to efficiently store large amounts of declarative knowledge in a long-term store, retrieving to working memory as necessary for reasoning.

In this paper, we explore methods for automatically managing Soar's working memory. After discussing related work, we derive and discuss desiderata that arise from the need to embed working-memory management within a general cognitive architecture that is used to develop real-time agents. We then provide details of our mechanism, including pertinent structures of Soar; our decay model, which incorporates the base-level activation

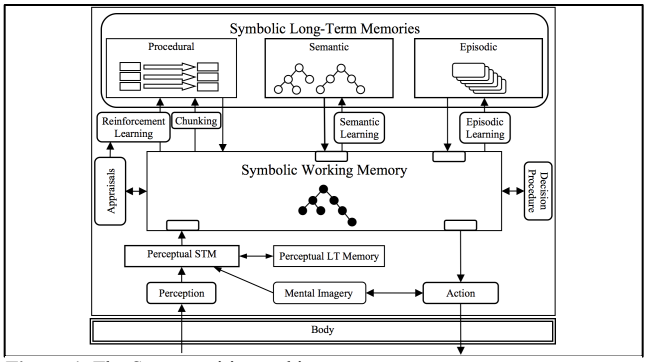

Figure 1. The Soar cognitive architecture.

model (Anderson et al. 2004); and novel system-independent data structures and algorithms for working-memory decay that are computationally efficient. We then we present empirical results, which demonstrate both that our mechanism performs with very little computational overhead, and that it helps maintain the reactivity of a Soar agent contending with long-term, autonomous simulated robotic exploration, even as it reasons using large amounts of acquired information.

Related Work

Previous research in cognitive modeling has investigated models of working-memory decay for the purpose of accounting for human behavior and experimental data. As a prominent example, memory decay has long been a core commitment of the ACT-R theory (Anderson et al. 2004), as it has been shown to account for a class of memory retrieval errors (Anderson, Reder, and Lebiere 1996). Additionally, some modeling work has been done in Soar, specifically investigating particular task-performance effects of forgetting short-term (Chong 2003) and procedural (Chong 2004) knowledge. By contrast, the motivation for and outcome of this work is to investigate and evaluate a specific functional benefit of managing working-memory size.

Prior research supports the potential for cognitive benefits of short-term memory decay, such as in task-switching (Altmann and Gray 2002) and heuristic inference (Schooler and Hertwig 2005). In this paper, we focus on improved agent reactivity.

We extend prior work on working-memory activation in Soar (Nuxoll, Laird, and James 2004). As efficiently implementing base-level decay is a challenging issue in cognitive architecture research (Douglass, Ball, and Rodgers 2009), we contribute algorithms that improve computational efficiency and a focused mechanism evaluation. We also perform the first computational investigation of how multiple memory systems combine within a cognitive architecture to balance sound agent reasoning with the need for a small working memory.

Mechanism Desiderata

These desiderata arise from the need to embed working-memory management within a general cognitive architecture that is used to develop real-time agents.

D1. Task Independence
The mechanism must support agents across a variety of domains and problems.

D2. Minimize Working-Memory Size
The mechanism must maintain a small amount of knowledge directly available to agent reasoning.

D3. Minimize Impact on Agent Reasoning
Many cognitive architectures, including Soar, make a *closed-world assumption* with respect to reasoning over knowledge stored in memory. That is, the memory system is assumed to be a *complete* representation of the agent's current beliefs, and thus if processing over a memory cannot find a knowledge structure, it is assumed not to exist. Consequently, an automatic mechanism for removing knowledge from working memory could have serious implications for the soundness of agent reasoning. It is important that removed knowledge is either unimportant for current processing (such as dated or unrelated information) or recoverable from another source.

D4. Minimize Computational Overhead
The mechanism must not incur substantial computational cost and must scale to large amounts of learned knowledge in dynamic environments.

We claim that a mechanism that satisfies these desiderata serves as a necessary precondition for cognitive systems to engage in complex reasoning while scaling to large amounts of acquired data over long periods of time.

Working-Memory Management in Soar

Soar is a cognitive architecture that has been used for developing intelligent agents and modeling human cognition. Historically, one of Soar's main strengths has been its ability to efficiently represent and bring to bear symbolic knowledge to solve diverse problems using a variety of methods (Laird 2008). We begin with a description of pertinent architectural structures and then convey the design and rationale of our automatic working-memory management mechanism, as related to our mechanism desiderata.

The Soar Cognitive Architecture

Figure 1 shows the structure of Soar. At the center is a symbolic working memory that represents the agent's current state. It is here that perception, goals, retrievals from long-term memory, external action directives, and structures from intermediate reasoning are jointly represented as a connected, directed graph. The primitive representational unit of knowledge in working memory is a symbolic triple (*identifier*, *attribute*, *value*), termed a *working-memory element*, or WME. The first symbol of a WME (*identifier*) must be an existing node in the graph, a constraint that maintains graph connectivity, whereas the second (*attribute*) and third (*value*) symbols may either be terminal constants or non-terminal graph nodes. Multiple WMEs that share the same *identifier* may be termed an "object," and the individual WMEs sharing that *identifier* are termed "augmentations" of that object.

Procedural memory stores the agent's knowledge of when and how to perform actions, both internal, such as querying long-term declarative memories, and external, such as control of robotic actuators. Knowledge in this memory is represented as if-then rules. The conditions of rules test patterns in working memory and the actions of rules add and/or remove working-memory elements. Soar makes use of the Rete algorithm for efficient rule matching (Forgy 1982) and scales to large stores of procedural knowledge (Doorenbos 1995). However, the Rete algorithm is known to scale linearly with the number of elements in working memory, a computational issue that motivates maintaining a relatively small working memory.

Soar incorporates two long-term declarative memories, semantic and episodic (Derbinsky and Laird 2010). Semantic memory stores working-memory objects, independent of overall working-memory connectivity (Derbinsky, Laird, and Smith 2010), and episodic memory incrementally encodes and temporally indexes snapshots of working memory, resulting in an autobiographical history of agent experience (Derbinsky and Laird 2009). Agents retrieve knowledge from one of these memory systems by constructing a symbolic cue in working memory; the intended memory system then interprets the cue, searches its store for the best matching memory, and reconstructs the associated knowledge in working memory. For episodic memory in particular, the time to reconstruct knowledge depends on the size of working memory at the time of encoding, another computational motivation for a concise agent state.

Agent reasoning in Soar consists of a sequence of decisions, where the aim of each decision is to select and apply an *operator* in service of the agent's goal(s). The primitive *decision cycle* consists of the following phases: encode perceptual input; fire rules to elaborate agent state, as well as propose and evaluate operators; select an operator; fire rules that apply the operator; and then process output directives and retrievals from long-term memory. The time to execute the decision cycle, which primarily depends on the speed with which the architecture can match rules and retrieve knowledge from episodic and semantic memories, determines agent reactivity.

There are two levels of persistence for working-memory elements added/removed as the result of rule firing. Rules that fire to apply a selected operator create *operator-supported* structures. These WMEs will persist in working memory until deliberately removed. In contrast, rules that do not condition upon a selected operator create *instantiation-supported* structures. These WMEs only persist as long as the rules that created them match. This distinction is relevant to managing working-memory.

As evident in Figure 1, Soar has additional memories and processing modules; however, they are not pertinent to this paper and are not discussed further.

Working-Memory Management

To design and implement a mechanism that satisfied our desiderata, we built on a previous framework of working-memory activation in Soar (Nuxoll, Laird, and James 2004). The primary activation event for a working-memory element is the firing of a rule that tests or creates that WME. Additionally, when a rule first adds an element to working memory, the activation of the new WME is initialized to reflect the aggregate activation of the set of WMEs responsible for its creation.

Based upon activation history, the activation level of a working-memory element is calculated using a variant of the base-level activation model (Anderson et al. 2004):

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

where n is the number of memory activations, t_j is the time since the jth activation, and d is a free decay parameter. The motivation for base-level activation is to identify those elements that have not been used recently and/or frequently, which is an indication of their importance to reasoning (desideratum D3). For reasons of computational efficiency (D4), we bound activation history size, such that each working-memory element has at most a history of size 10. Our model of activation sources, events, and decay is task-independent, thereby satisfying desideratum D1.

At the end of each decision cycle, Soar removes from working memory each WME that satisfies all of the following requirements, with respect to τ, a static, architectural threshold parameter:

R1. The WME was not encoded directly from perception.
R2. The WME is operator-supported.
R3. The activation level of the WME is less than τ.
R4. The WME is part of an object stored in semantic memory.
R5. The activation levels of all other WMEs that are part of the same object are less than τ.

We adopted requirements R1-R3 from Nuxoll, Laird, and James (2004), whereas R4 and R5 are novel. Requirement R1 distinguishes between the decay of mental representations of perception, and any dynamics that may occur with actual sensors, such as refresh rate, fatigue, noise, or damage. Requirement R2 is a conceptual optimization: since operator-supported structures are persistent, of which instantiation-supported structures are entailments, if we properly manage the former, the latter are handled automatically. This means that if we properly remove operator-supported structures, any instantiation-supported structures that depend upon them will also be removed, and thus our mechanism only manages operator-supported structures. The concept of a fixed lower bound on activation, as defined by R3, was adopted from activation limits in ACT-R (Anderson, Reder, and Lebiere

1996), and dictates that working-memory elements will decay in a task-independent fashion (D1) as their use for reasoning becomes less recent/frequent (D2).

Requirement R4 dictates that our mechanism only removes knowledge from working memory that can be deliberately reconstructed from semantic memory. From the perspective of cognitive modeling, this constraint on decay begins to resemble a working memory that is in part an activated subset of long-term memory (Jonides et al. 2008). From an agent functionality perspective, however, requirement R4 serves to balance the degree of working-memory decay (D2) with support for sound reasoning (D3). Knowledge in Soar's semantic memory is persistent, though may change over time. Depending upon the task and the agent's knowledge management strategies, it is possible that any knowledge our mechanism removes may be recovered via deliberate reconstruction from semantic memory. Additionally, knowledge that is not in semantic memory persists indefinitely to support agent reasoning.

Requirement R5 supplements R4 by providing partial support for the closed-world assumption. R5 dictates that either all object augmentations are removed, or none. This policy leads to an object-oriented representation whereby agent knowledge can distinguish between objects that have been removed, and thus have no augmentations, and those that simply are not augmented with a particular feature or relation. R5 makes an explicit tradeoff between D2 and D3, favoring the ability of an agent to reason soundly over working-memory decay speed. This functionality-driven requirement leads to a decay policy similar to what is in the declarative module of ACT-R, where activation is associated with each chunk and not individual slot values.

Efficient Implementation

We now present and analyze the novel data structures and algorithms that we developed to efficiently support our mechanism. Note that the methods in this section are *not* specific to Soar, so we begin with a problem formulation.

Problem Formulation. Let memory M be a set of elements, $\{m_1, m_2, \ldots\}$. Let each element m_i be defined as a set of pairs (a, k), where k refers to the number of times element m_i was activated at time a. We assume $|m_i| \leq c$: the number of activation events for any element is bounded.

We assume that activation of an element is computed according to the base-level model (Anderson et al. 2004), denoted as $b(m, d, t)$, where m is an element, d is a decay rate parameter, and t is the current time. We define an element m_i as *decayed* at time t with respect to decay rate parameter d and threshold parameter θ if $b(m_i, d, t) < \theta$. Given a static element m_i, we define L_i as the number of time steps required for m_i to decay, relative to time step t:

$$L_i := \inf\{t_d \in \mathbb{N} : b(m_i, d, t + t_d) < \theta\}$$

For example, element $\{(3, 1), (5, 2)\}$ was activated once at time step three and twice at time step five. Assuming decay rate 0.5 and threshold -2, this element has activation about 0.649 at time step 7 and is not decayed (L=489).

During a simulation time step t, the following actions can occur with respect to memory M:

S1. A new element is added to M.
S2. An existing element is removed from M.
S3. An existing element is activated x times.

If simulation action S3 occurs with respect to element m_i, a new pair (t, x) is added to m_i. To maintain boundedness, if $|m_i| > c$, the pair with least a is removed from m_i.

Thus, given a memory M, the *activation-based decay problem*, after each time step in a simulation, is to identify and remove the subset of elements, $D \subseteq M$, that have decayed since the previous time step.

Given this problem definition, a naïve approach is to determine the decay status of each element after every simulation time step. This test will require computation $O(|M|)$, scaling linearly with the average memory size. Furthermore, the computation expended upon each element, m_i, will be linear in the number of time steps where $m_i \in M$, estimated as $O(L_i)$ for a static element m_i.

Our Approach. We draw inspiration from the work of Nuxoll, Laird, and James (2004): rather than checking memory elements for decay status, we "predict" the future time step when the element will decay. Thus, at the end of each simulation time step, we iterate across each element that either (S1) that wasn't previously in the memory or (S3) was newly activated. For each element, we predict the time of future decay (discussed shortly) and add the element to an ordered decay map, where the map key is time step and the value is a set of elements predicted to decay at that time. If the element was already within the map (S3), we remove it from its old location before adding to its new location. All insertions/removals require time logarithmic in the number of distinct decay time steps, which is bounded by the total number of elements ($|M|$). During any time step, the set D is simply those elements in the list indexed by the current time step that are decayed.

To predict element decay, we implement a novel two-phase process. After a new activation (S3), we employ an approximation that is guaranteed to underestimate the true value of L_i. If, at a future time step, we encounter the element in D and it has not decayed, we then determine the correct prediction using a binary parameter search.

The key observation of our novel decay approximation is that there exists a closed-form solution to predict base-level decay if we only consider a single time of activation (i.e. $|m_i|$=1). Thus, we compute decay predictions for each pair of an element independently and sum them to form the approximate prediction. Below we derive the closed form

solution: given a single element pair at time t, we solve for t_p, the future time point of element decay given only a single activation…

$$\ln\left(k \cdot \left[t_p + (t-a)\right]^{-d}\right) = \theta$$

$$\ln(k) - d \cdot \ln\left(t_p + (t-a)\right) = \theta$$

$$t_p = e^{\frac{\theta - \ln(k)}{-d}} - (t-a)$$

For clarity, since k is the number of activations at the *same* time point, we can rewrite the summed terms as a product. Furthermore, we time shift the decay term by the difference between the current time step, t, and that of the element pair, a, thereby predicting L with respect to an element with only a single time of activation.

The time required to compute the approximate prediction of a single pair is constant (and common values can be cached to reduce this effort). The overall approximation prediction is linear in the number of pairs, which, because element size is bounded by c, is $O(1)$. The computation required for binary parameter search of element m_i is $O(\log_2 L_i)$. However, this computation is only necessary if the element has not decayed, or been removed from M, at the predicted time.

Empirical Evaluation

In the previous sections we discussed desiderata for an automatic working-memory management mechanism and described our mechanism in Soar. In this section we evaluate this mechanism according to those desiderata. We make two claims: our mechanism (C1) satisfies our desiderata and (C2) improves agent reactivity within a dynamic environment as compared to an agent without working-memory management.

We begin with an evaluation of the mechanism on synthetic data, focusing on the quality and efficiency of our prediction approach, and then continue to a long-term task that requires an agent to amass and reason about large amounts of learned knowledge.

Focused Mechanism Evaluation

Our synthetic data consists of 50,000 memory elements, each with a randomly generated pair set. The size of each element was randomly selected from between 1 and 10, the number of activations per pair (k) was randomly selected between 1 and 10, and the time of each pair (a) was randomly selected between 1 and 999. We verified that each element had a valid history with respect to time step 1000, meaning that each element would not have decayed before time step 1000. Furthermore, each element contained a pair with at least one access at time point 999, which simulated a fresh activation (S3). For all synthetic experiments we used a decay rate, d, of 0.8 and a decay threshold, θ, of -1.6. Given these constraints, the greatest number of time steps for an element to decay is 3332.

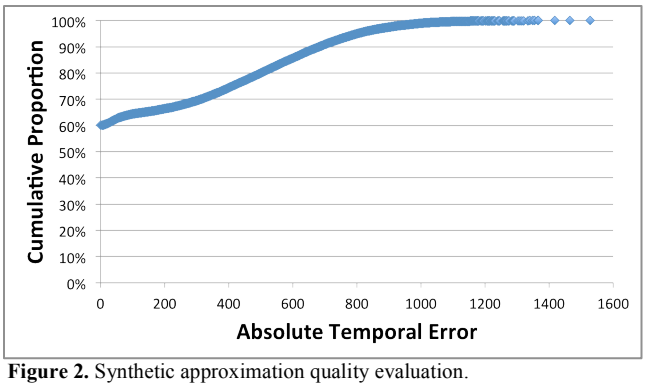

Figure 2. Synthetic approximation quality evaluation.

Our first experiment (see Figure 2) attests to the quality of our novel decay approximation. On the y-axis is the cumulative proportion of the synthetic elements and the x-axis plots absolute temporal error of the approximation, where a value of 0 indicates that the approximation was correct, and non-zero indicates how many time steps the approximation under-predicted. We see that the approximation was correct for over 60% of the elements, but did underestimate over 500 time steps for 20% of the elements and over 1000 time steps for 1% of the elements. Under the constraints of this data set, it is possible for this approximation to underestimate up to 2084 time steps.

The second experiment (see Figure 3) compares aggregate prediction time, in microseconds, between our approximation and exact calculation using binary parameter search. We see an order of magnitude improvement both in the maximum amount of computation time expended across the synthetic elements, as well as average time, though both computations are very fast. These experimental results attest to our progress towards satisfying desideratum D4 in isolation.

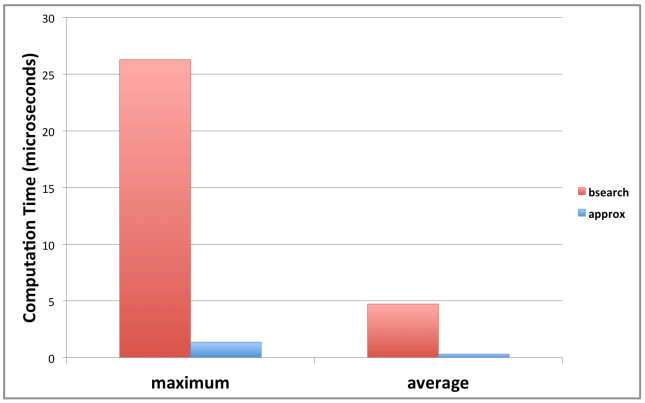

Figure 3. Synthetic prediction approximation efficiency evaluation.

Agent Evaluation

For this evaluation, we extended an existing system where Soar controls a simulated mobile robot (Laird, Derbinsky, and Voigt 2011). Our evaluation uses a simulation instead of a real robot because of the practical difficulties in running numerous long experiments in large physical spaces. However, 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.

The agent can issue output directives to move forward and backward and turn in place. It has a laser-range finder mounted in front that provides distances to 180 points throughout 180 degrees. The middleware between the simulator and Soar condenses those points to 5 regions that are sufficient for the agent to navigate and avoid obstacles. We supplement real sensors with virtual sensors, such that the robot can sense its own location, the room it is in, the location of doors and walls, and different types of objects. In sum, agent perception includes approximately 150 sensory data elements, with approximately 20 changing each decision cycle. The changes peak at 260 per cycle when the robot moves into a room because all the data about the current rooms, walls, and doorways change at once. Based upon prior experience in applying Soar to numerous and varied domains, including action games, large-scale tactical simulation, and interactive mobile phone applications, we contend that this domain is sufficiently dynamic to evaluate claim C2.

The agent's task within this environment is to visit every room on the third floor of the Computer Science and Engineering (CSE) building at the University of Michigan. This task requires the agent to visit over 100 rooms and requires about 1 hour of real time. During its exploration it incrementally builds up an internal map, which when completed, requires over 10,000 working-memory elements to represent and store. We contend that this map constitutes a relatively large amount of learned knowledge (C2), as well as a baseline by which to evaluate claim C1, with respect to desideratum D2. In addition to storing topographic information, the agent must also reason about and plan using the map in order to find efficient paths for moving to distant rooms it has sensed but not visited. We contend that understanding how this reasoning scales computationally as the agent amasses more and more map information provides an indication as to relative agent reactivity, a core component of claim C2.

Evaluation Metrics. In order to evaluate desideratum D2, we measure working-memory size over the course of the task. In order to evaluate desideratum D4 and claim C2, we measure the maximum process time required to complete a primitive Soar decision cycle, a direct measure of agent reactivity, in contrast to average or total time, which can mask computation "surges," over the course of the task and compare this value to 50 msec., a response time we have found as sufficient for real-time reactivity in multiple domains. These metrics are aggregated for each 10 seconds of experimentation. All experiments in this section are run on an Intel i7 2.8GHz CPU with 8GB of memory, running Soar 9.3.1 on the Windows 7 operating system. Soar is open source and freely available to download on all major platforms at [http://sitemaker.umich.edu/soar]. While there is qualitative consistency across our runs, our experiments were not duplicated sufficiently to establish statistical significance, and the variance across runs is small compared to the qualitative results we focus on below.

Experimental Conditions. We make use of the same agent for all of our experiments (Laird, Derbinsky, Voigt 2011). In order to evaluate our claims, however, we modify small amounts of task knowledge, represented in rules, and change architectural parameters, as described below.

The first experimental condition compares alternative approaches to maintaining declarative map information. The baseline agent, which we term A0, maintains all declarative map information in both Soar's working and semantic memories. A slight modification to this baseline, A1, includes hand-coded rules to prune away rooms in working memory that are not required for immediate reasoning or planning. Finally, a second modification of the baseline, A2, makes use of our working-memory management mechanism. Both agents A1 and A2 contain identical task knowledge to reconstruct from semantic memory any WMEs that are needed for immediate reasoning. Comparing the working-memory sizes of agents A1 and A2 allows us to evaluate D2, while comparing decision cycle time of A0 and A2 allows us to evaluate C2. We also experiment with different values of d, the base-level activation decay rate parameter, to understand the tradeoffs involved in a more/less aggressive policy for task-independently pruning working memory, which better informs both claims C1 and C2. We use $\tau = -2$ as our architectural threshold parameter value for all experiments.

The second experimental condition relates to how our mechanism affects agent reactivity (C2) with respect to episodic memory retrievals. Whereas episodic memory is disabled for all experiments above, we make two changes for this series of experiments. First, we set an architectural parameter such that episodic memory automatically encodes episodes each time the agent issues an environmental directive, such as changing direction. Second, we add task knowledge to retrieve episodes related to, but not incorporated in, task reasoning. Specifically, whenever the agent reaches a new doorway, it recalls the last room it visited. While episodic memory is not actively used for the exploration task in this paper, the same agent we are utilizing contends with more complex tasks, including patrolling and building clearing, which do make

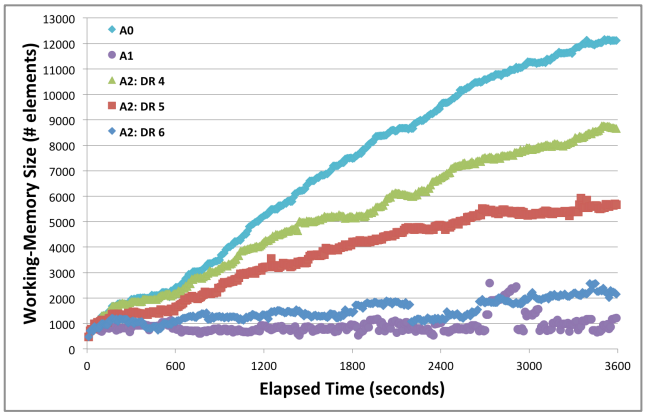

Figure 4. Agent working-memory size comparison.

use of episodic retrievals, so this analysis is important to informing C2. We hypothesized that a more aggressive working-memory management policy would lead to faster episodic reconstructions and thus implemented a parallel experimental setup to those above: E0 implements no working-memory management; E1 uses task knowledge, in rules, to manage working memory; and E2, with various decay rates, uses our automatic mechanism.

Results. The first set of results (see Figure 4) compares working-memory size between conditions A0, A1, and A2. We note first the major difference in working memory size between A0 and A1 at the completion of the simulation, where A1 contains nearly 11,000 fewer working-memory elements, more than 90% less than A0. We also notice that the greater the decay rate parameter for A2 (indicated by "DR"), the smaller the working-memory size, where a value of 0.6 qualitatively tracks A1. This finding suggests that our mechanism, with an appropriate decay rate, does reduce working-memory size comparable to a hand-coded approach in this task (D2).

The second set of results (see Figure 5) compares maximum decision cycle time, measured in msec., between conditions A0, A1, and A2 as the simulation progresses.

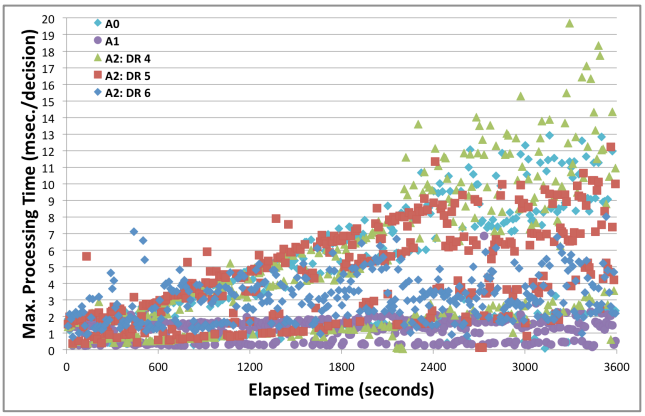

Figure 5. Agent maximum decision cycle time comparison without episodic memory retrievals.

Without any form of working-memory management, processing time for A0 increases linearly, as we would expect from an architecture using a Rete-based algorithm for rule matching, but is still far below the reactivity threshold of 50 msec. after one hour. A1, despite increased rule firing from hand-coded rules, maintains a fairly constant maximum decision time, qualitatively less than A0. A2 results depend upon the decay rate, where lower values (less aggressive management) qualitatively track A0, whereas higher values (more aggressive management) track A1. For instance, decay rate 0.4 exhibits a linear trend and performs worse in absolute computation time, likely because of extra activation-related processing. However, a decay rate of 0.6 exhibits a relatively constant maximum processing time, though it also incurs some extra processing cost, as compared to A1, from activation-related computation. This finding suggests that our mechanism, with an appropriate decay rate, improves agent reactivity, as related to rule matching, compared to an agent without working-memory management (C2). Additionally, our approach does not require task knowledge, which can be burdensome to generate, and arguably more difficult for an agent to learn.

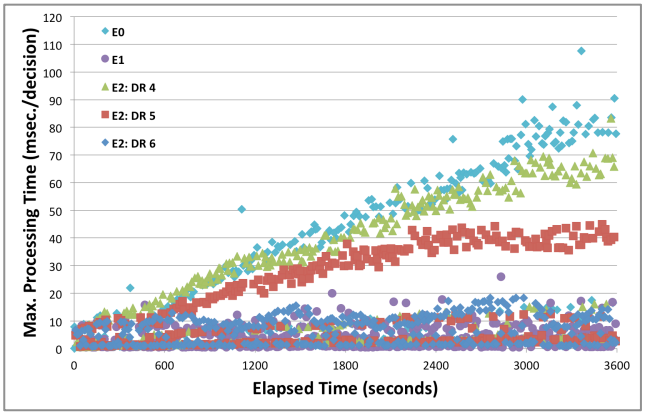

Figure 6. Agent maximum decision time comparison with episodic memory retrievals.

The third set of results (see Figure 6) includes the cost of episodic retrievals and compares maximum decision cycle time, measured in msec., between conditions E0, E1, and E2 as the simulation progresses. As hypothesized, we see a growing difference in time between E0 and E2 as we more aggressively manage working memory (i.e. greater decay rate), demonstrating that episodic reconstruction, which scales with the size of working memory at the time of episodic encoding, benefits from working-memory management (C2). We also see that with a decay rate of 0.6, our automatic mechanism performs comparably to A1: since the costs in this data are dominated by episodic retrievals, as compared to Figure 5, extra activation-related processing is relatively inconsequential. We note that without sufficient working-memory management (E0; E2

with decay rate 0.4), episodic memory retrievals are not tenable for an agent that must reason with this amount of acquired information, as the maximum required processing time exceeds the reactivity threshold of 50msec.

Discussion

In this paper, we demonstrated that an automatic mechanism to manage working-memory elements in Soar serves an important functional purpose: scaling reactivity for agents that must reason about large stores of acquired information. While it always has been possible to write rules to prune Soar's working memory, this approach required task-specific knowledge that was difficult to encode, problematic for the agent to learn, and interrupted deliberate agent processing. In this work, we proposed and presented an evaluation of a novel approach that pushes this functionality into the architecture: our mechanism takes advantage of multiple memory systems in a manner that, empirically, comparably improves agent reactivity without incurring undue computational cost nor loss to reasoning soundness. This is the first work that architecturally exploits activation-based working-memory management, in conjunction with multiple memory systems, in order to support long-lived, learning agents.

There are several important limitations of this paper that should be investigated in future work. First, we have only investigated a single decay model, base-level activation. There is prior analytical and empirical work to support the efficacy of this model in context of cognitive modeling, but further evaluation should consider additional models, especially those that may more directly measure the usefulness of working-memory elements to current agent reasoning. Furthermore, in our implementation of base-level activation, we made an efficiency tradeoff to bound activation history size. It remains to be seen whether this history length is sufficient to support effective working-memory management, especially without any form of longer-term historical tail approximation (Petrov 2006). It is also important to note that we evaluated a single task in a single domain, where base-level activation-based working-memory management was well suited to capture the regularities of spatial locality within a topographic map. Finally, this work should be extended to cognitive systems other than Soar, an exploration for which we have provided efficient data structures and algorithms.

References

Altmann E., Gray, W. 2002. Forgetting to Remember: The Functional Relationship of Decay and Interference. *Psychological Science* 13 (1): 27-33.

Anderson, J. R., Reder, L., Lebiere, C. 1996. Working Memory: Activation Limitations on Retrieval. *Cognitive Psychology* 30 (3): 221-256.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., Qin, Y. 2004. An Integrated Theory of the Mind. *Psychological Review* 111 (4): 1036-1060.

Chong, R. 2003. The Addition of an Activation and Decay Mechanism to the Soar Architecture. In Proc. of the Fifth International Conference on Cognitive Modeling. Bamberg, Germany.

Chong, R. 2004. Architectural Explorations for Modeling Procedural Skill Decay. In Proc. of the Sixth International Conference on Cognitive Modeling. Pittsburgh, PA.

Derbinsky, N. Laird, J. E. 2009. Efficiently Implementing Episodic Memory. In Proc. of the Eighth International Conference on Case-Based Reasoning. Seattle, WA.

Derbinsky, N., Laird, J. E. 2010. Extending Soar with Dissociated Symbolic Memories. In Proc. of the Symposium on Human Memory for Artificial Agents. AISB. Leicester, UK.

Derbinsky, N., Laird, J. E., Smith, B. 2010. Towards Efficiently Supporting Large Symbolic Declarative Memories. In Proc. of the Tenth International Conference on Cognitive Modeling. Phil, PA.

Doorenbos, R. B. 1995. Production Matching for Large Learning Systems. Ph.D. diss., Computer Science Dept., Carnegie Mellon, Pittsburgh, PA.

Douglass, S., Ball, J., Rodgers, S. 2009. Large Declarative Memories in ACT-R. In Proc. of the Ninth International Conference on Cognitive Modeling. Manchester, UK.

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

Jonides, J., Lewis, R. L., Nee, D. E., Lustig, C. A., Berman, M. G., Moore, K. S. 2008. The Mind and Brain of Short-Term Memory. *Annual Review of Psychology* 59: 193-224.

Laird, J. E. 2008. Extending the Soar Cognitive Architecture. In Proc. of the First Conference on Artificial General Intelligence. Memphis, TN.

Laird, J. E., Derbinsky, N. 2009. A Year of Episodic Memory. In Proc. of the Workshop on Grand Challenges for Reasoning from Experiences. IJCAI. Pasadena, CA.

Laird, J. E., Derbinsky, N., Voigt, J. 2009. Performance Evaluation of Declarative memory Systems in Soar. In Proc. of the Twentieth Behavior Representation in Modeling & Simulation Conference. Sundance, UT.

Laird, J. E., Wray III, R. E. 2010. Cognitive Architecture Requirements for Achieving AGI. In Proc. of the Third Conference on Artificial General Intelligence. Lugano, Switz.

Nuxoll, A., Laird, J. E., James, M. 2004. Comprehensive Working Memory Activation in Soar. In Proc. of the Sixth International Conference on Cognitive Modeling. Pittsburgh, PA.

Nuxoll, A., Tecuci, D., Ho, W. C., Wang, N. 2010. Comparing Forgetting Algorithms for Artificial Episodic memory Systems. In Proc. of the Symposium on Human Memory for Artificial Agents. AISB. Leicester, UK.

Petrov, A. 2006. Computationally Efficient Approximation of the Base-Level Learning Equation in ACT-R. In Proc. of the Seventh International Conference on Cognitive Modeling. Trieste, Italy.

Schooler, L., Hertwig, R. How Forgetting Aids Heuristic Inference. *Psychological Review* 112 (3): 610-628.