Available online at www.sciencedirect.com

SciVerse ScienceDirect

Cognitive Systems Research 24 (2013) 104–113

Cognitive Systems RESEARCH

www.elsevier.com/locate/cogsys

Effective and efficient forgetting of learned knowledge in Soar's working and procedural memories

Action editor: David Peebles

Nate Derbinsky [*], John E. Laird

University of Michigan, 2260 Hayward Street, Ann Arbor, MI 48109-2121, USA

Available online 5 January 2013

Abstract

Effective management of learned knowledge is a challenge when modeling human-level behavior within complex, temporally extended tasks. This work evaluates one approach to this problem: forgetting knowledge that is not in active use (as determined by base-level activation) and can likely be reconstructed if it becomes relevant. We apply this model to the working and procedural memories of Soar. When evaluated in simulated, robotic exploration and a competitive, multi-player game, these policies improve model reactivity and scaling while maintaining reasoning competence. To support these policies for real-time modeling, we also present and evaluate a novel algorithm to efficiently forget items from large memory stores while preserving base-level fidelity.

Keywords: Large-scale cognitive modeling; Working memory; Procedural memory; Cognitive architecture; Soar

1. Introduction

Typical cognitive models persist for short periods of time (seconds to a few minutes) and have modest learning requirements. For these models, current cognitive architectures, such as Soar (Laird, 2012) and ACT-R (Anderson et al., 2004), executing on commodity computer systems, are sufficient. However, prior work (e.g. Kennedy & Trafton, 2007) has shown that cognitive models of complex, protracted tasks can accumulate large amounts of knowledge, and that the computational performance of existing architectures degrades as a result.

This issue, where more knowledge can harm problem-solving performance, has been dubbed the *utility* problem, and has been studied in many contexts, such as explanation-based learning (Minton, 1990; Tambe, Newell, & Rosenbloom, 1990), case-based reasoning (Smyth & Keane, 1995; Smyth & Cunningham, 1996), and language learning (Daelemans, van den Bosch, & Zavrel, 1999). Markovitch and Scott (1988) have characterized strategies for dealing with the utility problem in terms of information filters applied at different stages in the problem-solving process. One common strategy that is relevant to cognitive modeling is *selective retention*, or forgetting, of learned knowledge. The benefit of this approach, as opposed to *selective utilization*, is that the agent does not have to expend computational resources at run time to decide whether to utilize knowledge or not, a property that may be crucial for real-time modeling in temporally extended, complex tasks. However, it can be challenging to devise forgetting policies that work well across a variety of problem domains, effectively balancing the task performance of models with reductions in retrieval time and storage requirements of learned knowledge.

In context of this challenge, we present two tasks where effective behavior requires that the model accumulate large amounts of information from the environment, and where over time this learned knowledge overwhelms reasonable computational limits. In response, we present and evaluate novel policies to forget learned knowledge in the working

[*] Corresponding author.
E-mail addresses: nlderbin@umich.edu (N. Derbinsky), laird@umich.edu (J.E. Laird).

and procedural memories of Soar. These policies investigate a common hypothesis: it is rational for the architecture to forget a unit of knowledge when there is a high degree of certainty that it is not of use, as calculated by base-level activation (Anderson et al., 2004), and that it can be reconstructed in the future if it becomes relevant. We demonstrate that these task-independent policies improve model reactivity and scaling, while maintaining problem-solving competence. To support these policies for real-time modeling, we also present and evaluate a novel algorithm to efficiently forget items from large memory stores while preserving fidelity of base-level activation.

2. Related work

Previous cognitive-modeling research has investigated forgetting in order to account 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, & Lebiere, 1996). Similarly, research in Soar investigated task-performance effects of forgetting short-term (Chong, 2003) and procedural (Chong, 2004) knowledge. By contrast, the motivation for this work is to discover the degree to which forgetting can support long-term, real-time modeling in complex tasks.

Prior work has demonstrated that there are potential cognitive benefits to using memory decay, such as in task-switching (Altmann & Gray, 2002) and heuristic inference (Schooler & Hertwig, 2005). In this paper, we focus on improving reactivity and scaling.

We extend prior investigations of long-term symbolic learning in Soar (Kennedy & Trafton, 2007), where the source of learning was internal problem solving. In this paper, the evaluation domains accumulate information from interaction with an external environment.

Prior work has addressed many of the computational challenges associated with retrieving a single memory according to the base-level activation (BLA) model (Petrov, 2006; Derbinsky, Laird, & Smith, 2010; Derbinsky & Laird, 2011). However, efficiently removing items from memory, while preserving BLA fidelity, presents a different challenge. As such, before presenting the empirical evaluation domains, we formally describe this computational problem; present a novel algorithm to forget according to BLA in large memories; and evaluate our approach with synthetic data.

3. The Soar cognitive architecture

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 large amounts of symbolic knowledge to solve diverse problems using a variety of methods (Laird, 2012).

Fig. 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

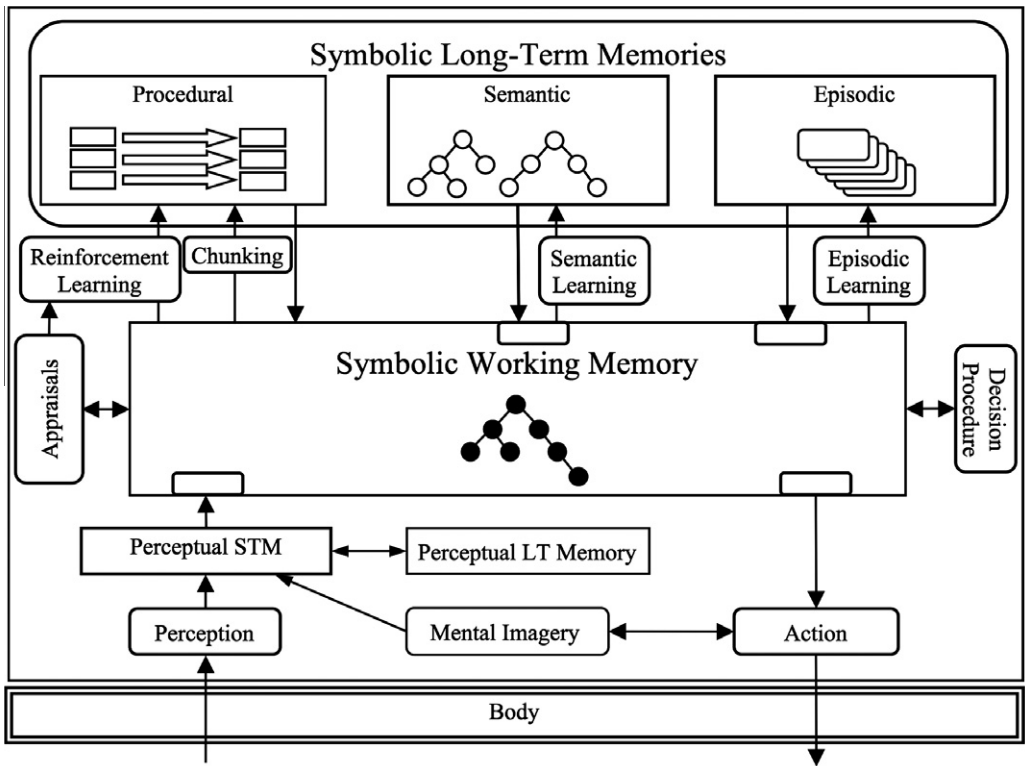

Fig. 1. The Soar cognitive architecture (Laird, 2012).

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, whereas the second (attribute) and third (value) symbols may be either terminal constants or non-terminal graph nodes. Multiple WMEs that share the same identifier are termed an "object," and the set of 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 controlling 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 retrieval time scales to large stores of procedural knowledge (Doorenbos, 1995). However, in the worst case, the Rete algorithm scales linearly with the number of elements in working memory, a computational issue that motivates maintaining a relatively small working memory.

Soar learns procedural knowledge via chunking (Laird, Rosenbloom, & Newell, 1986) and reinforcement-learning (RL; Nason & Laird, 2005) mechanisms. Chunking *creates new* rules: it converts deliberate subgoal processing into reactive rules by compiling over rule-firing traces, a form of explanation-based learning (EBL). If subgoal processing does not interact with the environment, the chunked rule is redundant with existing knowledge and serves to improve performance by reducing deliberate processing. However, memory usage in Soar scales linearly with the number of rules, typically at a rate of 1–5 KB/rule, which provides a motivation for forgetting under-utilized rules.

Reinforcement learning incrementally *tunes existing* rule actions: it updates the expectation of action utility, with respect to a subset of state (represented in rule conditions) and an environmental and/or intrinsic reward signal. A rule that can be updated by the RL mechanism (termed an *RL rule*) must satisfy a few simple criteria related to its actions, and is thus distinguishable from other rules. This distinction is relevant to forgetting rules. When an RL rule that was learned via chunking is updated, that rule is no longer redundant with the knowledge that led to its creation, as it now incorporates information from environmental interaction that was not captured in the original subgoal processing.

Soar incorporates two long-term declarative memories, semantic and episodic (Derbinsky & Laird, 2010). Semantic memory stores working-memory objects, independent of overall working-memory connectivity (Derbinsky et al., 2010), and episodic memory incrementally encodes and temporally indexes snapshots of working memory, resulting in an autobiographical history of agent experience (Derbinsky, Li, & Laird, 2012; Nuxoll & Laird, 2012). 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 if it finds a match, reconstructs the associated knowledge in working memory. For episodic memory, the time to reconstruct knowledge depends on the size of working memory at the time of encoding, another motivation for a concise agent state (Derbinsky & Laird, 2009).

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. Unlike ACT-R, multiple rules may fire in parallel during a single phase. 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. We have found that 50 ms is an acceptable upper bound on this response time across numerous domains, including robotics, video games, and human–computer interaction (HCI) tasks.

There are two types of persistence for working-memory elements added 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 test a selected operator create *instantiation-supported* structures, which persist only as long as the rules that created them match. This distinction is relevant to forgetting WMEs.

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

4. Efficient forgetting via base-level activation

In later sections, we present and evaluate forgetting policies in the working and procedural memories of Soar. Both of these policies use base-level activation (BLA; Anderson et al., 2004) as a heuristic for identifying memories that may not be useful to the agent. In this section, we formally describe the computational problem of forgetting according to the BLA model; present a novel approach that scales efficiently in large memories; and evaluate our approach using synthetic data.

4.1. Problem formulation

Let memory M be a set of elements, $\{m1, m2, \dots\}$. Let each element m_i be defined as a set of pairs (a_{ij}, k_{ij}), where k_{ij} refers to the number of times element m_i was activated at time a_{ij}. We assume $|m_i| \leqslant c$: the number of activation events for any element is bounded. These assumptions are consistent with the ACT-R declarative memory when bounding chunk-history size (Petrov, 2006). This is also consistent with the semantic memory in Soar (Laird, 2012).

We assume that activation of an element m at time t is computed according to the BLA model (Anderson et al., 2004), where d is a fixed decay parameter:

$$B(m,t,d) = \ln\left(\sum_{j=1}^{|m|} k_j \cdot [t - a_j]^{-d}\right)$$

We define an element as *decayed*, with respect to a threshold parameter Θ if $B(m,t,d) < \Theta$. Given a static element m, we define L as the fewest number of time steps required for the element to decay, relative to time step t:

$$L(m,t,d,\Theta) := \inf\{t_d \in \mathbb{N} : B(m, t + t_d, d) < \Theta\}$$

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

During a model 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 y times.

If S3 occurs with respect to element m_i, a new pair (t,y) is added to m_i. To maintain a bounded history size, if $|m_i| > c$, the pair with smallest a (i.e. the oldest) is removed from m_i.

Thus, given a memory M, we define that the *forgetting* problem, at each time step, t, is to identify the subset of elements, $D \subseteq M$, that have decayed since the last time step.

4.2. Efficient approach

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

Our approach draws inspiration from the work of Nuxoll, Laird, and James (2004): rather than checking memory elements for decay status, "predict" the future time step when the element will decay. First, at each time step, examine elements that either (S1) were not previously in the memory or (S3) were activated. The number of items requiring inspection is bounded by the total number of elements ($|M|$), but is likely to be a small subset, assuming few memory elements are created or tested by the model at each time step. For each of these elements, predict the time of future decay (discussed shortly) and add the element to a map, where the map key is the predicted time step and the value is the set of elements predicted to decay at that time. If the element was already within the map (S3), remove it from its old location before adding to its new location. All insertions/removals require time at most logarithmic in the number of distinct decay time steps, which is bounded by the total number of elements ($|M|$). At any time step, the set D is those elements in the set indexed by the current time step that are decayed.

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

We approximate L for an element m as the sum of L for each independent pair $(a,k) \in m$. Here we derive the closed-form calculation: given a single element pair at time t, we solve for t_p, the future time of element decay ...

$$\ln(k \cdot [t_p + (t - a)]^{-d}) = \Theta$$
$$\ln(k) - d \cdot \ln(t_p + (t - a)) = \Theta$$
$$t_p = e^{\frac{\Theta - \ln(k)}{-d}} - (t - a)$$

Since k refers to a single time point, a, we 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.

Computing this approximation for a single pair takes constant time (and common values can be cached). Overall approximation computation is linear in the number of pairs, which is bounded by c, and therefore $\mathcal{O}(1)$. The computation required for binary parameter search of an element is $\mathcal{O}(\log_2 L)$. However, this computation is only necessary if the element has not decayed, or removed from M.

4.3. Synthetic evaluation

In later sections, we empirically evaluate this approach with it embedded within the working and procedural memories of Soar; here we focus on the quality and efficiency of our prediction approach and utilize synthetic data. This synthetic data set comprises 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 $t = 1000$. In addition, each element contained a pair with at least one access at time point 999, which simulated a fresh activation (S3). For this evaluation, we used decay rate $d = 0.8$ and threshold $\Theta = -1.6$. Given these constraints, the largest possible value of L for an element was 3332.

We first evaluated the quality of the decay approximation. In Fig. 2, the y-axis is the cumulative proportion of

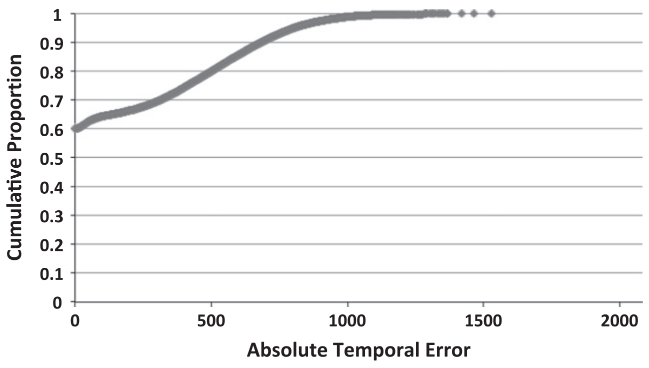

Fig. 2. Evaluation of decay-approximation quality.

the 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 was possible for this approximation to underestimate up to 2084 time steps.

We also compared the prediction time, in microseconds (μs), of the approximation to an exact calculation using binary parameter search. The maximum computation time across the data set was >19× faster for the approximation (1.37 vs. 26.28 μs/element) and the average time was >15× faster (0.31 vs. 4.73 μs/element). We did not compare these results with a naïve algorithm, wherein computation time at each time step depends upon the number of memory elements ($|M|$). This comparison would have required a model of memory size across a variety of tasks and such a model would have been difficult to develop, as prior work (Doorenbos, 1995; Derbinsky et al., 2012) has shown that while the number of memory changes tends to be small across a variety of problem domains, absolute size can vary drastically between tasks.

In summary, this two-phase forgetting approach maintains fidelity to the BLA model (due to the second phase of prediction) and scales to large memories. Results from synthetic data show that the first phase of our approach is a high-quality approximation and is an order of magnitude less costly than the exact calculation in the second phase.

5. Forgetting in Soar's working memory

The core intuition of our working-memory forgetting policy is to remove the augmentations of objects that are not actively in use and that the model can later reconstruct from long-term semantic memory, if they become relevant. As defined earlier, we characterize WME usage via the base-level activation model (BLA; Anderson et al., 2004), which estimates future usefulness of memory based upon prior usage. The primary activation event for a working-memory element is the firing of a rule that tests or creates that WME. In addition, 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. This model of activation sources, events, and decay is task independent.

At the end of each decision cycle, Soar removes from working memory each element 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 augments an object, o, in semantic memory.
R5. The activation levels of all augmentations of o are less than τ.

We adopted requirements R1-R3 from Nuxoll et al. (2004), whereas R4 and R5 are novel. Requirement R1 distinguishes between the decay of 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: as operator-supported WMEs are persistent, while instantiation-supported structures are direct entailments. Thus, if we properly remove operator-supported WMEs, any instantiation-supported structures that depend on them will also be removed. Therefore 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 et al., 1996), and dictates that working-memory elements will decay in a task-independent fashion as their use for reasoning becomes less recent/frequent.

Requirement R4 dictates that our mechanism only removes elements from working memory that augment objects in semantic memory. This requirement serves to balance the degree of working-memory decay with support for sound reasoning. Knowledge in Soar's semantic memory is persistent, though it may change over time. Depending on the task and the model's knowledge-management strategies, it is possible that forgotten working-memory knowledge may be recovered via deliberate reconstruction from semantic memory. Additionally, augmentations of objects that are not in semantic memory can persist indefinitely to support model 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 procedural knowledge can distinguish between objects that have been completely cleared of substructure, and those that simply are not augmented with a particular feature or relation. R5 makes an explicit tradeoff, weighting more heavily model competence at the expense of the speed of working-memory decay. This requirement resembles the declarative module of ACT-R, where activation is associated with each chunk and not individual slot values.

5.1. Empirical evaluation

We extended an existing system where Soar controls a simulated mobile robot (Laird, Derbinsky, & 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 robot's task is to visit every room on the third floor of the Bob & Betty Beyster building at the University of Michigan. For this task, the robot visits over 100 rooms and takes about 1 h of real time. During exploration, it incrementally builds an internal topological map, which, when completed, requires over 10,000 WMEs to represent and store. In addition to storing information, the model reasons about and plans using the map in order to find efficient paths for moving to distant rooms it has sensed but not visited. The model uses episodic memory to recall objects and other task-relevant features during exploration.

In our experiments, we aggregate working-memory size and maximum decision time for each 10 s of elapsed time, all of which is performed on an Intel i7 2.8 GHz CPU, running Soar v9.3.1. Because each experimental run takes 1 h, we did not duplicate our experiments sufficiently to establish statistical significance and the results we present are from individual experimental runs. However, we found qualitative consistency across our runs, such that the variance between runs is small as compared to the trends we focus on below.

We make use of the same model for all experiments, but modify small amounts of procedural knowledge and change architectural parameters, as described here. The baseline model (A0) maintains all declarative map information in Soar's working memory. A modification to this baseline (A1) maintains the declarative map in both working and semantic memories, and additionally includes hand-coded rules to prune away rooms in working memory that are not required for immediate reasoning or planning, as well as to reconstruct these structures from semantic memory when they are needed. The experimental model (A2) also maintains the declarative map in both working and semantic memories, but rather than task-specific rules, it makes use of our task-independent working-memory forgetting policy to prune working-memory structures and task-independent rules to reconstruct knowledge, as needed, from semantic memory. For this experimental condition, we held constant the activation-history size ($c = 10$) and base-level threshold ($\tau = -2$), but explored a set of decay-rate values ($d \in \{0.3, 0.4, 0.5\}$). For more aggressive decay rates ($d \geqslant 0.6$), the model was unable to maintain sufficient declarative-map data in working memory to complete planning in this task.

Fig. 3 compares working-memory size between conditions A0, A1, and A2 over the duration of the experiment. We note first the major difference in working-memory size between A0 and A1 after one hour, when the working memory of A1 contains more than 11,000 fewer elements, more than 90% less than A0. We also find that the greater the decay-rate parameter for A2, the smaller the working-memory size, where a value of 0.5 qualitatively tracks A1. This finding suggests that our policy, with an appropriate decay, keeps working-memory size comparable to that maintained by hand-coded rules.

Fig. 4 compares maximum decision-cycle time in ms, between conditions A0, A1, and A2 as the simulation progresses. The dominant cost reflected by this data is time to reconstruct prior episodes that are retrieved from episodic memory. We see a growing difference in time between A0 and A2 as working memory is more aggressively managed (i.e. greater decay rate), demonstrating that episodic recon-

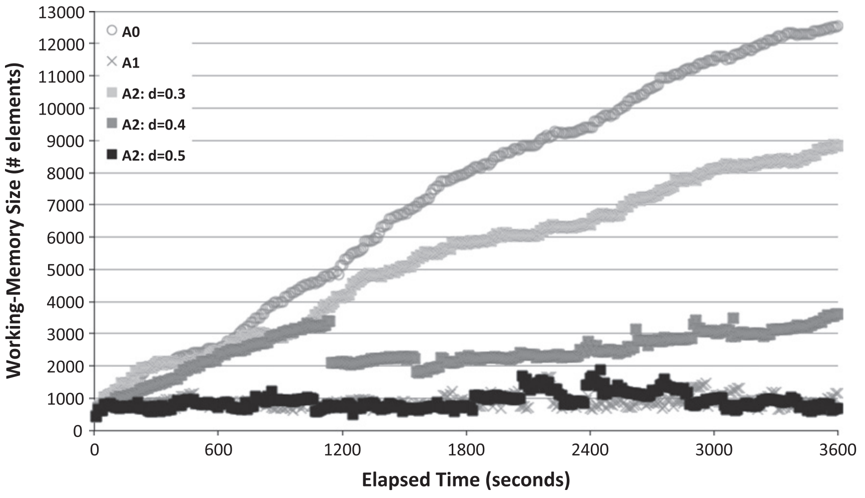

Fig. 3. Model working-memory size comparison.

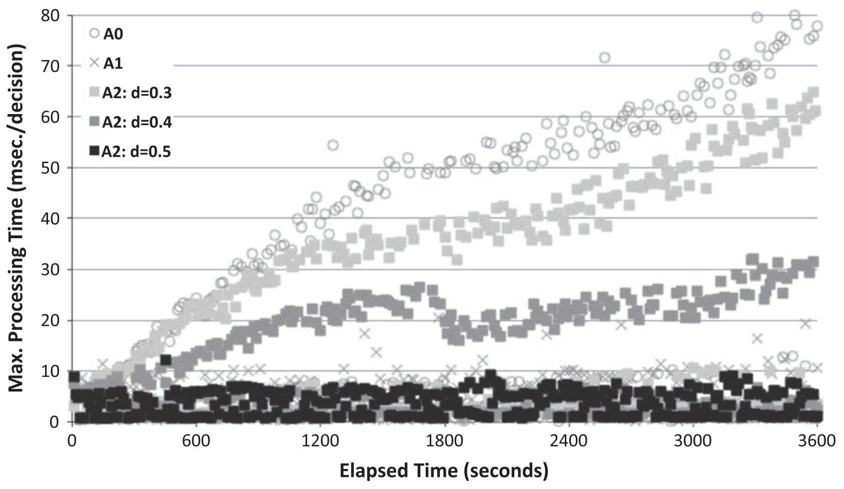

Fig. 4. Model maximum decision time comparison.

struction, which scales with the size of working memory at the time of episodic encoding, benefits from forgetting. We also find that with a decay rate of 0.5, our mechanism performs comparably to A1. We note that without sufficient working-memory management (A0; A2 with decay rate 0.3), episodic-memory retrievals are not tenable for a model that must reason with this amount of acquired information, as the maximum required processing time exceeds the reactivity threshold of 50 ms.

5.2. Discussion

It is possible to write rules that prune Soar's working memory; however, this task-specific knowledge is difficult to encode and learn.

In this work, we presented and evaluated a novel, task-independent approach that utilizes a memory hierarchy to bound working-memory size while maintaining sound reasoning. This approach assumes that the amount of knowledge required for immediate reasoning is small relative to the overall amount of knowledge accumulated by the model. Under this assumption, as demonstrated in the robotic evaluation task, our policy scales even as learned knowledge grows large over long trials.

We note that since Soar's semantic memory can change over time and is independent of working memory, our forgetting policy does admit a class of reasoning errors wherein the contents of semantic memory are changed so as to be inconsistent with decayed WMEs. However, this corruption requires deliberate reasoning in a relatively small time window and has not arisen in our models. While the model completed this task for all conditions reported here, at larger decay rates ($d \geqslant 0.6$) the model thrashed because map information was not held in working memory long enough to complete deep look-ahead planning. Based upon this finding, we expect that if the agent had to perform deeper searches within this task, then the model would thrash with even less aggressive decay rates (e.g. $d = 0.5$), but we do not have data for such circumstances. This line of reasoning suggests that additional research is needed on either adaptive decay-rate settings or approaches to planning, and other forms of temporally extended reasoning, that are robust in the face of memory decay.

6. Forgetting in Soar's procedural memory

The core intuition of our procedural-memory forgetting policy is to remove rules that are not actively used and that the model can later reconstruct via deliberate subgoal reasoning, if the knowledge embedded in them is relevant to a given situation. As with working-memory forgetting, we characterize rule usage via the base-level activation model, where the activation event is the firing of an instantiation of a rule. As with our working-memory forgetting policy, the activation source, event, and decay is task independent: we utilize the base-level activation model to summarize the history of rule firing.

At the end of each decision cycle, Soar removes from procedural memory each rule that satisfies all of the following requirements, with respect to parameter τ:

R1. The rule was learned via chunking.
R2. The rule is not actively firing.
R3. The activation level of the rule is less than τ.
R4. The rule has not been updated by RL.

We adopted R1–R3 from Chong (2004), whereas R4 is novel. Chong was modeling human skill decay, and did not delete rules, so as to not lose each rule's activation history. Instead, decayed rules were prevented from firing, similar to below-utility-threshold rules in ACT-R. R1 is a practical consideration to distinguish learned knowledge from "innate" rules developed by the modeler, which, if modified, would likely break the model. R2 recognizes that matched rules are in active use and thus should not be forgotten. R3 dictates that rules will decay in a task-independent fashion as their use for reasoning becomes less recent/

frequent. We note that for fixed parameters (d and τ) and a single activation, the BLA model is equivalent to the use-gap heuristic of Kennedy and Trafton (2007). However, the time between sequential rule firings ignores firing frequency, which the BLA model incorporates.

Requirement R4 attempts to retain rules that include information that cannot be regenerated in the future. Rules learned by chunking can be regenerated if they have not been updated by RL; however, once they have been updated, they encode expected-utility information, which is not recorded by any other learning mechanism and cannot be regenerated if the rule is removed.

6.1. Empirical evaluation

We extended an existing system (Laird, Derbinsky, & Tinkerhess, 2011) where Soar plays Liar's Dice, a multiplayer game of chance. The rules of the game are numerous and complex, yielding a task that has rampant uncertainty and a large state space (millions-to-billions of relevant states for games of 2–4 players). Prior work has shown that RL allows Soar models to significantly improve performance after playing a few thousand games. However, this involves learning large numbers of RL rules to represent the value function spanning this state space.

The model we use for all experiments learns two classes of rules: RL rules that capture expected action utility; and non-RL rules that capture symbolic game heuristics. Our experimental baseline (B0) does not forget knowledge. The first experimental modification (B1) implements our forgetting policy, but does not enforce requirement R4 and is thereby comparable to prior work (Kennedy & Trafton, 2007; Chong, 2004). The second modification (B2) fully implements our policy. We experiment with a range of representative decay rates (d), including 0.999, where rules not immediately updated by RL are deleted ($c = 10$, $\tau = -2$ for all).

We alternated 1000 2-player games of training then testing, each against a non-learning version of the model. After each testing session, we recorded maximum memory usage (Mac OS v10.7.3; dominated, in this task, by the rules in procedural memory), task performance (% games won), and average decisions/task action. We do not report maximum decision time, as this was below 6 ms. for all conditions (Intel i7 2.8 GHz CPU, Soar v9.3.1). We collected data for all conditions in at least three independent trials of 40,000 games. For conditions that forget knowledge, we were able to gather more data in parallel, due to reduced memory consumption (six trials for $d = 0.35$, seven for remaining).

Fig. 5 presents average memory growth, in megabytes, as the model trains (within each experimental condition, error bars of 1 standard deviation are too small to be consistently visible on this plot and thus variance data is not included in Fig. 5). For all models, the memory growth of games 1–10 K follows a power law ($R^2 \geqslant 0.96$), whereas for 11–40 K, growth is linear ($R^2 \geqslant 0.99$). These plots indicate that memory usage for the baseline (B0) and the slowly decaying model (B2, $d = 0.3$) is much greater, and faster growing, than models that more aggressively decay. It also shows that there is a diminishing benefit from faster decay (e.g. $d = 0.5$ and $d = 0.999$ for B2 are indistinguishable).

Fig. 6 presents average task performance after 1000 games of training, where the error bars represent ± 1 standard deviation. This data shows that given the inherent stochasticity of the task, there is little, if any, difference between the performance of the baseline (B0) and decay levels of B2. However, by comparing B0 and B2 to B1, it is clear that without R4, the model suffers a dramatic loss of task competence. For clarity, the model begins by playing a non-learning copy of itself and learns from experience with each training session. While the B0 and B2 models improve from winning 50% of games to 75–80%, the B1 model improves to below 55%. We conclude that a forgetting policy that only incorporates rule-firing history (e.g. Chong, 2004; Kennedy & Trafton, 2007) will negatively impact performance in tasks that involve informative interaction with an external environment. Our policy incorporates both rule-firing history and rule reconstruction, and thus retains this source of feedback.

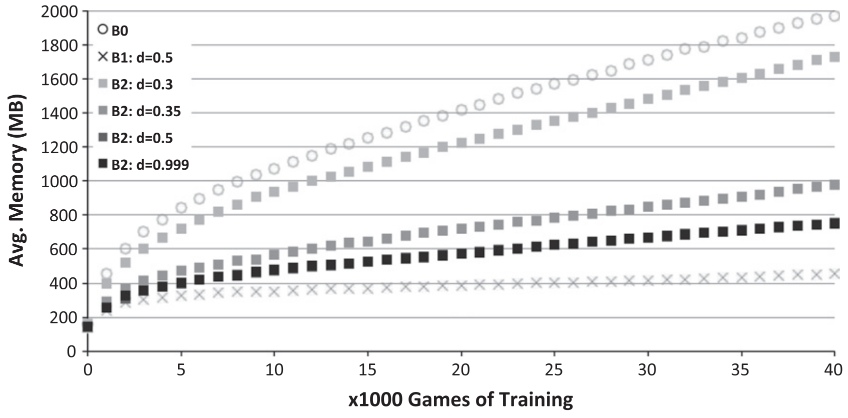

Fig. 5. Avg. memory usage vs. games played.

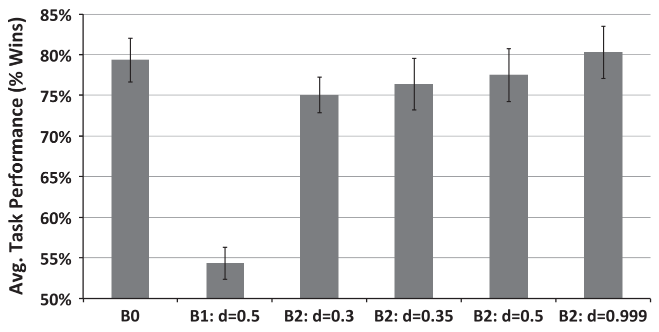

Fig. 6. Avg. task performance ±1 std. dev.

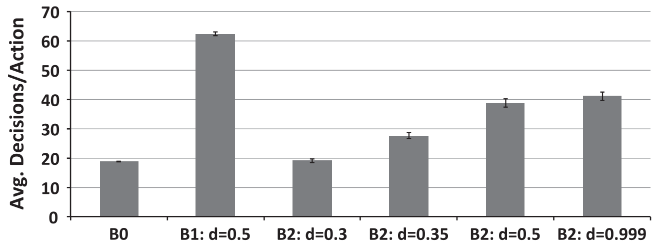

Fig. 7. Avg. decisions/task action ±1 std. dev.

Finally, Fig. 7 presents average number of decisions for the model to take an action in the game after training for 10,000 games. In prior work (e.g. Kennedy & Trafton, 2007), this value was a major performance metric, as it reflected the primary reason for learning new rules. In this work, each decision takes very little time, and so the number of decisions to choose an action is not as crucial to task performance as the correctness of the selected action. However, these data show that there exists a space of decay values (e.g. $d = 0.35$) in which memory usage is relatively low and grows slowly (Fig. 5), task performance is relatively high (Fig. 6), and the model makes decisions relatively quickly (Fig. 7).

6.2. Discussion

This work contributes evidence that we can develop models that improve using RL in tasks with large state spaces. Currently, it is typical to explicitly represent the entire state space, which is not feasible in complex problems. Instead, Soar learns rules to represent only those portions of the space it experiences, and our policy retains only those rules that include feedback from environmental reward. Future work needs to validate this approach in other domains.

7. Concluding remarks

This paper presents and evaluates policies and algorithms for effective and efficient forgetting of learned knowledge in complex environments. While forgetting mechanisms are common in cognitive modeling, this work pursues this line of research for functional reasons: improving computational resource usage while maintaining reasoning competence. We have presented compelling results from applying these policies in two complex, temporally extended tasks, but there is additional work to evaluate these policies, and their parameters, across a wider variety of problem domains.

Acknowledgment

We acknowledge the funding support of the Air Force Office of Scientific Research, contract FA2386-10-1-4127.

References

Altmann, E. M., & Gray, W. D. (2002). Forgetting to remember: The functional relationship of decay and interference. *Psychological Science, 13*, 27–33.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review, 111*, 1036–1060.

Anderson, J. R., Reder, L. M., & Lebiere, C. (1996). Working memory: Activation limits on retrieval. *Cognitive Psychology, 30*, 221–256.

Chong, R. (2003). The addition of an activation and decay mechanism to the Soar architecture. In *Proceedings of the fifth international conference on cognitive modeling* (pp. 45–50). Bamberg, Germany.

Chong, R. (2004). Architectural explorations for modeling procedural skill decay. In *Proceedings of the sixth international conference on cognitive modeling*. Pittsburgh, PA, USA.

Daelemans, W., van den Bosch, A., & Zavrel, J. (1999). Forgetting exceptions is harmful in language learning. *Machine Learning, 34*, 11–41.

Derbinsky, N., & Laird, J. E. (2009). Efficiently implementing episodic memory. In *Proceedings of the 8th international conference on case-based reasoning* (pp. 403–417). Seattle, WA, USA.

Derbinsky, N., & Laird, J. E. (2010). Extending soar with dissociated symbolic memories. In *Proceedings of the 1st symposium on human memory for artificial agents* (pp. 31–37). Leicester, UK.

Derbinsky, N., & Laird, J. E. (2011). A functional analysis of historical memory retrieval bias in the word sense disambiguation task. In *Proceedings of the 25th AAAI conference on artificial intelligence* (pp. 663–668). San Francisco, CA, USA.

Derbinsky, N., Laird, J. E., & Smith, B. (2010). Towards efficiently supporting large symbolic declarative memories. In *Proceedings of the 10th international conference on cognitive modeling* (pp. 49–54). Philadelphia, PA, USA.

Derbinsky, N., Li, J., & Laird, J. E. (2012). A multi-domain evaluation of scaling in a general episodic memory. In *Proceedings of the 26th AAAI conference on artificial intelligence* (pp. 193–199). Toronto, Canada.

Doorenbos, R. B. (1995). *Production matching for large learning systems*. Ph.D. Thesis. Carnegie Mellon University.

Forgy, C. L. (1982). Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence, 19*, 17–37.

Kennedy, W. G., & Trafton, J. G. (2007). Long-term symbolic learning. *Cognitive Systems Research, 8*, 237–247.

Laird, J. E., Derbinsky, N., & Tinkerhess, M. (2011). A case study in integrating probabilistic decision making and learning in a symbolic cognitive architecture: Soar plays dice. In *Papers from the 2011 AAAI fall symposium series: advances in cognitive systems* (pp. 162–169). Arlington, VA, USA.

Laird, J. E., Derbinsky, N., & Voigt, J. (2011). Performance evaluation of declarative memory systems in Soar. In *Proceedings of the 20th behavior representation in modeling and simulation conference* (pp. 33–40). Sundance, UT, USA.

Laird, J. E. (2012). *The Soar Cognitive Architecture*. Cambridge: MIT Press.

Laird, J. E., Rosenbloom, P. S., & Newell, A. (1986). Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning, 1*, 11–46.

Markovitch, S., & Scott, P. D. (1988). The role of forgetting in learning. In *Proceedings of the fifth international conference on machine learning* (pp. 459–465). Ann Arbor, MI, USA.

Minton, S. (1990). Qualitative results concerning the utility of explanation-based learning. *Artificial Intelligence, 42*, 363–391.

Nason, S., & Laird, J. E. (2005). Soar-RL: Integrating reinforcement learning with Soar. *Cognitive Systems Research, 6*, 51–59.

Nuxoll, A. M., Laird, J. E., & James, M. (2004). Comprehensive working memory activation in Soar. In *Proceedings of the sixth international conference on cognitive modeling* (pp. 226–230). Pittsburgh, PA, USA.

Nuxoll, A. M., & Laird, J. E. (2012). Enhancing intelligent agents with episodic memory. *Cognitive Systems Research, 17–18*, 34–48.

Petrov, A. A. (2006). Computationally efficient approximation of the base-level learning equation in ACT-R. In *Proceedings of the seventh international conference on cognitive modeling* (pp. 391–392). Trieste, Italy.

Schooler, L. J., & Hertwig, R. (2005). How forgetting aids heuristic inference. *Psychological Review, 112*, 610–628.

Smyth, B., & Cunningham, P. (1996). The utility problem analysed – A case-based reasoning perspective. In *Proceedings of the third European workshop on case-based reasoning* (pp. 392–399). Lausanne, Switzerland.

Smyth, B., & Keane, M. T. (1995). Remembering to forget: A competence-preserving case deletion policy for case-based reasoning systems. In *Proceedings of the fourteenth international joint conference on artificial intelligence* (pp. 377–383). Montreal, Quebec, Canada.

Tambe, M., Newell, A., & Rosenbloom, P. S. (1990). The problem of expensive chunks and its solution by restricting expressiveness. *Machine Learning, 5*, 299–349.