In Proceedings of the Sixth International Conference on Cognitive Modeling, 226-230.
Mahwah, NJ: Lawrence Earlbaum.

Comprehensive Working Memory Activation in Soar

Andrew Nuxoll (anuxoll@umich.edu)
University of Michigan, 1101 Beal Ave.
Ann Arbor MI 48109-2110 USA

John E. Laird (laird@umich.edu)
University of Michigan, 1101 Beal Ave.
Ann Arbor MI 48109-2110 USA

Michael R. James (mrjames@umich.edu)
University of Michigan, 1101 Beal Ave.
Ann Arbor MI 48109-2110 USA

Abstract

Memory activation has been modeled in symbolic architectures in the past, but usually at the level of individual chunks or productions in long term memory. Recent research (Chong 2003) has demonstrated activation at the level of individual elements of working memory. In this paper, we present a comprehensive implementation of working memory activation in Soar that takes advantage of the unique characteristic of Soar's working memory structure, namely persistence. We also explore modifications to activation so that the activation of new working memory elements is not a fixed level, but is based on the activation of the working memory elements tested in its creation. We demonstrate our model in terms of how it aids the selection of features relevant to learning.

Introduction

Since its inception, the Soar architecture (Newell 1990) has focused on symbolic reasoning to the exclusion of numeric processing. Even the learning mechanism, chunking, collected conditions and actions for new production rules through an analytic method that did not require any numeric processing. One can think of these earlier versions of Soar as experiments in the sufficiency of purely symbolic reasoning.

In recent years, we have looked to expand the types of knowledge that Soar can learn that has involved moving beyond a single architectural learning mechanism, to incorporate an episodic memory (Nuxoll & Laird, 2004) in Soar. A critical part of episodic memory is determining which stored episode is the best match for the current situation. Using a purely symbolic match is inadequate because of the large number of irrelevant features in the episode. Thus, we needed some way of biasing the match to the most relevant features. One possible bias is the "activation" of the features. Activation of the contents of working memory was pioneered by the ACT family of architectures (Anderson & Labiere, 1998) and was initially implemented in Soar by Ron Chong (2003) to support forgetting in working memory.

In this paper we describe extensions to that original implementation that include a more comprehensive implementation (Chong's implementation supported activation only in a subpart of working memory), a more efficient implementation, and an implementation that takes advantage of the unique structure of Soar's working memory and distinguishes it from ACT-R's activation scheme. Furthermore, we present results of using this activation-based memory to improve the retrieval of episodic memories.

Overview of the Soar Architecture

Soar is a production rule-based cognitive architecture. Like most other architectures of this type, Soar has two types of knowledge: *working memory* (short term, declarative) and *production rules* (long-term, procedural). Working memory consists of a collection of attribute-value pairs. The agent's current state, including both external sensing and internal inferences, is stored in working memory. Production rules consist of actions and conditions. If the conditions of a production match the contents of working memory then that production *fires* its actions which create (or remove) one or more elements in working memory. These changes may cause the agent to take an action in its environment, which can result in additional changes to working memory via its sensory input. These changes to working memory may in turn trigger the firing (or retraction) of additional productions so that this "match-fire" cycle repeats indefinitely.

The Soar architecture has several unique characteristics. We describe here only the characteristics that are relevant to this research:

- Simultaneous Production Firing: Many production systems allow only one production to fire at a time. Soar allows all productions whose conditions match working memory to fire in parallel.
- Operators: To avoid the conflicting behavior that might result from simultaneous production firing, Soar supports a special knowledge structure called an

operator. A Soar production may *propose* an operator by creating it in working memory. Multiple operators may be proposed in a given situation but only one can be *selected*. Selection is controlled by the creation of *preferences*: structures created by production rules that test for proposed operators and details of the current situation. The selection of an operator can trigger additional productions to fire *apply* the operator, making changes to working memory.

- Decision Cycles: Soar operators extend the traditional match-fire cycle of Soar to a three-phase *decision cycle*: propose, select and apply.
- Persistent working memory: Soar distinguishes between two types of persistence in working memory. Working memory elements (WMEs) that are created as part of an application of an operator persist indefinitely; they remain in working memory until they are explicitly removed. These are called *o-supported* WMEs. The remaining WMEs are removed as soon as the *instantiation* of the production that created them ceases to match working memory. (An instantiation is the collection of WMEs that match a production). These are known as *i-supported* WMEs. The advantage of i-support is that it automatically removes WMEs that are no longer relevant to the current situation. For example, an operator is proposed based on specific features of the situation, when one of these features change, the operator is automatically retracted. The same is true for elaborations of the state, such as the calculation that a block is clear based on the fact that no blocks are above it. This is a simple inference that is automatically retracted when a block is placed on top of it, so that it is no longer clear.

Original Implementation of Activation in Soar

The original addition of working memory activation to Soar by Chong (2003) allowed persistent working memory elements (o-supported WMEs) to decay from working memory over time in a manner very similar to the decay of chunks in ACT-R. The implementation reserved a section of Soar's working memory for activated WMEs. The activation level of these WMEs changed as follows:

- WMEs received an initial, fixed activation when they were first created.
- Any time that a WME was tested by a production that fired, it received an activation boost.
- Any time an action would attempt to add an existing WME, the existing WME would receive an activation boost.
- WME activation levels decayed over time using an exponential decay formula identical to that used by Chong:

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

A_i is the activation of a WME at time i. β is a base level constant. t_j is the number of cycles since the WME was referenced for the jth time. d is a learning rate parameter which we set at the same default as in previous research (0.8). ε is a noise component. We set this value to zero for our experiments.

Throughout this paper we will use the term "reference" to refer to an incident that led to an activation boost (i.e., a WME is tested or recreated).

Extensions to Chong's Activation Implementation

Although Chong's work established the possibility of using activation in Soar, several extensions are possible.

- Extend activation to all of working memory, not just a subset.
- Improve the efficiency. Calculating activation values can be computationally expensive. We improved the efficiciency by not updating the activations every cycle, and only calculating the activation values when they were used. Other optimizations included using a bounded approximation of each WME's complete history (an optimization also used by Chong) and pre-calculating commonly used activation values. To evaluate these improvements, we ran ten iterations of a moderately complex task in Soar with three types of memory activation: a) no memory activation (i.e., the original, unmodified Soar) b) a memory activation mechanism without the efficiency enhancements and c) the enhanced memory activation system. Although the unoptimized activation implementation is significantly slower than Soar without activation, the optimized implementation is not (see Figure **1**).

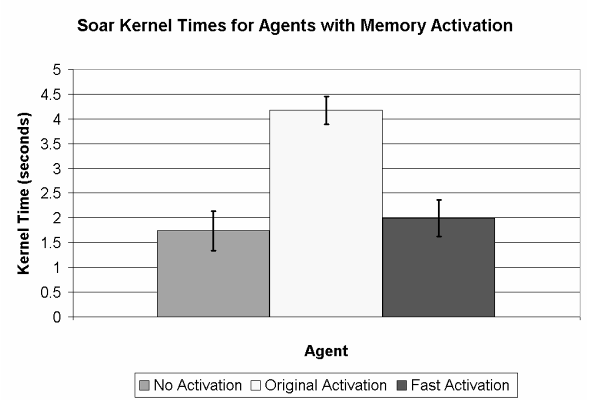

Figure 1: Demonstration of Efficiency Improvements

- Apply activation differentially to i-supported and o-supported working memory elements. This is the major conceptual change and is described in detail below.
- Base the activation of newly created WMEs on the activation of the WMEs matched in the conditions of the rules. This is a departure from both the ACT and Chong approach to activation, where newly created WMEs get a fixed boost. The impact of this change is demonstrated in the results section.

Memory Activation Implementation

Like Chong, our original implementation ignored i-supported WMEs because even without activation, i-supported WMEs are automatically removed when the WMEs they are based on are removed. Thus, the o-supported WMEs are the persistent structures in working memory that the i-supported structures are entailments of. If we correctly maintain the o-supported structures, the i-supported structures will take care of themselves. The activation of an o-supported WME is based on a decay function of all prior references to that WME – each of these references are decayed individually, with the activation being the sum of all of the current values of those references. Therefore, in our implementation, all WMEs have an associated history of prior references (see Figure **2**).

The conceptually simplest approach to maintaining activations is to update the activation values of every WME every cycle. However, the majority of WMEs are not accessed on a given cycle, so that the calculated activations are never used. An alternative approach is to only update the activation when a WME is accessed or when it needs to be removed. But how do we know when a WME must be removed? We can "predict" which cycle a WME should be removed by using the reference history and decay function, and then store that cycle on a "timelist." Then on each cycle we check the timelist to see which elements need to be removed. If a WME is referenced on a cycle, its removal time will be updated on the timelist. As shown in Figure **2**, the timelist is an array indexed by removal time and treated as a large circular queue. Each entry in the array is a future cycle that contains a list of WMEs that will be removed on that cycle if they are not referenced again before that cycle.

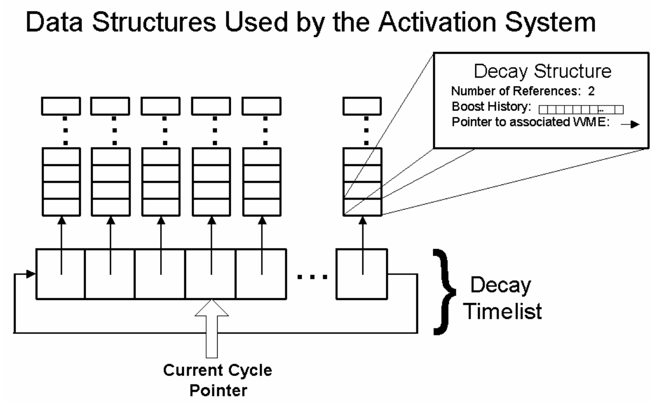

Figure 2: Activation Data Structures

The decay data structure does *not* maintain an actual numerical activation level for the WME. This value is calculated only when the WME is referenced and it is necessary to determine the WME's new position in the decay timelist

The "activation life cycle" of a WME has three stages:

Creation: When a new WME has been created.,a decay data structure is created and attached to the WME. The WME is assigned an initial reference history.

Activation: At the conclusion of each decision cycle all referenced WMEs have their boost histories updated and their position in the decay timelist adjusted appropriately. This step is by far the most computationally intensive. As discussed previously, we were able to reduce the time required by making some efficiency improvements. As a result, the time required for this step is now linear in the number of WMEs that were referenced during the cycle.

Removal: Once all the WMEs in the decay timelist have been adjusted, any WMEs that remain at the current position in the timelist are removed from working memory.

Evaluation of Activation Implementation

Although one use of activation is to decay and possibly remove WMEs, it is difficult to evaluate the direct impact of the activation scheme on behavior. Our initial implementations of activation and decay schemes closely model both ACT-R and Chong's Soar implementation, so there is little to be gained by running any new simulations of decay and removal. Instead, we have chosen to focus on the role activation can play in aiding learning, specifically in selecting relevant features for retrieving past experiences from an episodic memory we are implementing in Soar (Nuxoll & Laird, 2004).

Our initial research on episodic memory is being carried out in a simple interactive domain called Eaters, which consists of a 16x16 gridworld. An agent (i.e., eater) in the world can sense the contents of the cells around it and must make decisions about what direction to move in that world. To aid in its decision, the eater has approximately 30 sensory inputs from the outside world. Most of these inputs consist of the contents of the cells that surround the eater. Cells can contain normal food (●), bonus food (■) or a wall. If an eater has already visited a cell it will be empty. Figure **3** depicts the sensory input of an eater.

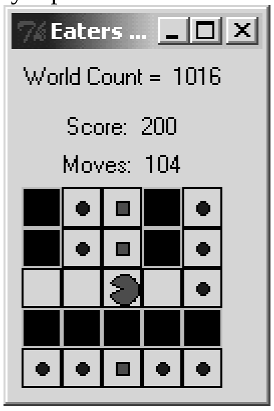

Figure 3: The Eaters World

An important feature of the Eaters world is that different sensory inputs have varying levels of significance. If an eater is considering moving west, the content of the cell just west of it can have a dramatic impact on the results of its decision whereas the content of cells in other directions are less important.

Initial Results

For our experiments, an eater was created that used an episodic memory to record its past situations and recall them to aid in future decisions. The eater would evaluation each action it was considering by attempting to recall a similar situation in its past. The results of that past action were used to evaluate the action the eater was currently considering. As a result, the quality of the eater's action depended heavily upon its ability to select the memory that best matched the current situation and proposed action.

We began initially with an eater that selected the best episodic memory via an unbiased partial match. We then added an activation-bias to the partial matching algorithm. When episodes were recorded, the activation level of the WMEs in the episode were also recorded. Rather than calculate the exact activation level of each WME (a time intensive process), we used its position in the decay timelist as a discretized measure of activation. When a partial match was being performed, WMEs with a higher recorded activation added more weight to the match than those with lower activation. (i.e., The match score was equal to the sum of the activation levels of the matching WMEs.)

We ran both eaters for five iterations of 7500 decision cycles and averaged the results. During a single iteration the eater took more than 2200 actions in the world. (The episodic retrievals and comparisons require multiple decision cycles.) After every 1500 decision cycles the contents of the gridworld would be randomly refilled and rearranged so as to present a variety of situations to the eater rather than a sparse grid.

Figure **4** shows a comparison of these two eaters. The ordinal is a measure of the fraction of retrievals the eater made which were correct (i.e., led to an accurate evaluation of the proposed action). A perfect eater would create a horizontal line across the top of the graph. In this graph, the eater with an activation-biased match is showing about a 30% improvement over the unbiased eater.

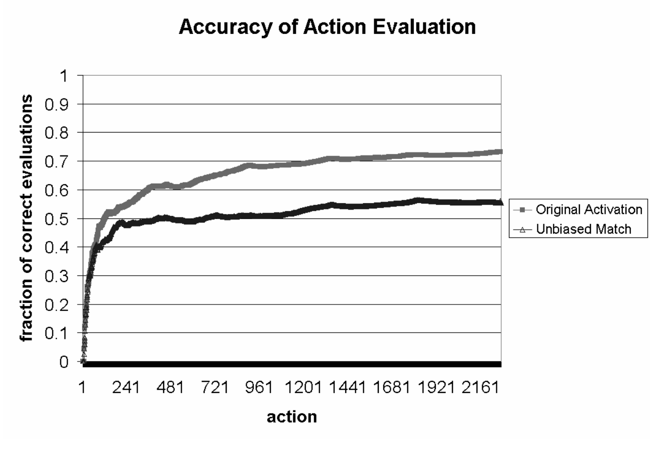

Figure 4: Effectiveness of Activation-Bias

Modifications to the Activation System

During the course of this research, we made two observations about the design of activation system that led us to believe we could improve its ability to predict salient features of an agent's environment. As mentioned above, the original implementation does not activate i-supported WMEs..However, the i-supported WMEs can "block" the activation of the underlying o-supported WMEs responsible for the creation of the i-supported WMES. Consider the case where a production tests an i-supported WME. That WME does not receive any activation boost since it is not activated. However, the o-supported WME(s) that were tested in order to create that WME *also* do not receive a boost. Thus i-supported WMEs are masking o-supported WMEs from activation.

This phenomenon is shown in Figure **5**, where six o-supported WMEs (A-F) are depicted along with five production rule instantiations that create i-supported WMEs (1-5). As mentioned earlier, i-supported WMEs do not directly receive activation, but instead exist "at the pleasure" of the o-supported elements they are based on. Thus, if D is removed, that will cause 4 to be retracted, which in turn will lead to 5 being removed.

In terms of activation, if WME 5 is tested, it will not receive any activation boost (because it is i-supported). Moreover, WMEs E and F (and indirectly C and D) also do not receive any activation boost although they are responsible for maintaining 5. Once they decay, WME 5 will be removed despite the fact that it has been referenced.

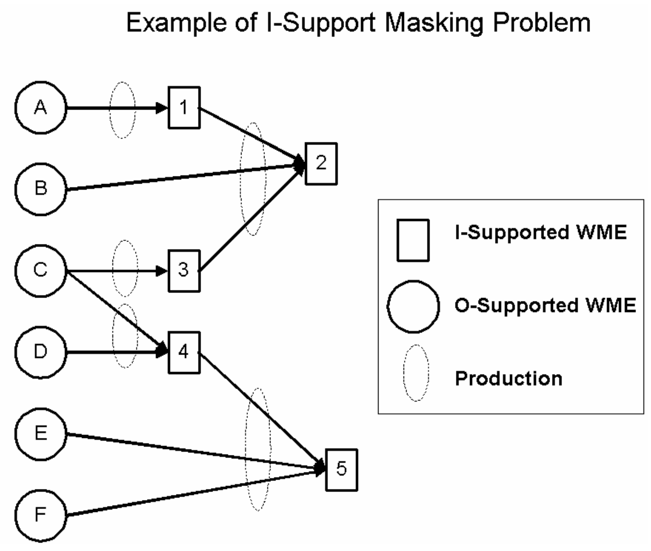

Figure 5: Example of the I-Support Masking Problem

In response to this problem we added a "pay it backward" approach for passing references from i-supported WMEs to o-supported ones. The activation system now calculates the *set of o-support* for any referenced i-supported WME and then boosts the activation of WMEs in the o-support set. In our example, the set of o-support for WME 5 includes

WMEs E and F, which are directly tested by the rule that creates 5, as well as C and D, which are indirectly tested by the WMEs that create 5 (via WME 4).

The second observation we made was in regard to initial activation for newly created WMEs. In the original implementation, these WMEs receive a fixed initial boost equivalent to a single reference. (The same approach is used in ACT-R.) In terms of decay and removal, this may be sufficient because newly created WMEs will usually immediately lead to additional rule firing (and receive a boost in activation) or if they are not relevant to the situation, will not be tested and be removed after a short time. However, when using memory activation as a measure of the importance of features of the agent's state, this flat level of initial activation can be misleading. An agent might test multiple WMEs with high activation and create one new WME that would have much lower activation. Thus, at creation time, when this feature will most likely be very important as a cue for future retrieval, its activation is much lower than the activation of the features that led to its creation. This may be a new manifestation of an old problem. Chong noted that in both his model and some ACT-R models, newly created WMEs/chunks can decay rapidly and never have a chance to participate in reasoning.

To ameliorate this problem, the second modification we implemented was a "pay it forward" approach for setting the activation level of new WMEs. Thus, the activation of a new WME is based on the activation levels of the set of o-support WMEs tested in its conditions, which includes all o-supported WMEs tested in the conditions, as well as o-supported WMEs that were precursors to i-supported WMEs tested in the conditions. Although it is convenient to think of each WME as having activation, the system never directly represents a specific activation value. Instead, it stores a history of all of the activation boosts each WME has received. Thus, we base the activation of a newly created WME on the activation levels of the set of o-support WMES, by averaging the reference histories of those WMEs and assigning that average history to the new WME.

Final Results

The two improvements described in the previous section were incrementally added to the activation system and the experiment was repeated. The results can be seen in Figure **6** along with the results from the previous experiment. Modifying the system so that the o-support sets received boosts in activation raised the accuracy because it was more likely that the correct features (which are o-supported) would contribute to retrieving an episode. Modifying the boosting of newly created WMEs had an even bigger impact because one of the most important features (the proposed direction of movement) is created just before an episode is stored. In the original versions, this feature would have activation that was significantly below others, making it a marginal feature for biasing retrieval. The modification boosted the proposed direction of movement feature – its creation was dependent on highly active features – and made a marked improvement in the agent's ability to retrieve appropriate episodes and select the correct action.

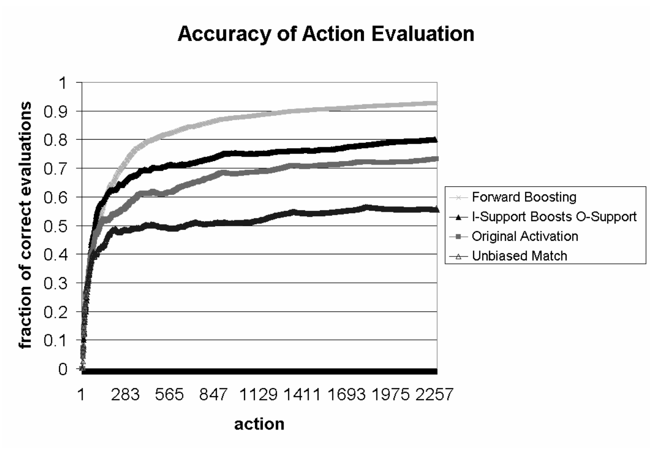

Figure 6: Result of Activation System Modifications

Future Work

We currently see three areas where this work can be expanded. First, this research demonstrates that a comprehensive activation system can be an effective technique for selecting the salient features of an agent's working memory. It would be useful to test the effectiveness of the activation-biased match in other domains and with other learning mechanisms (e.g., reinforcement learning).

Second, the changes we have made to the activation system will likely have an impact on the effectiveness of that system as a memory decay mechanism.

Finally, the use of memory activation as a search control mechanism (i.e., selecting which matching production will be selected to fire) has a long tradition in the ACT-R architecture. What impact might this approach to memory activation have if it was used to influence operator selection in Soar?

References

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

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

Laird, J. E., Newell, A., and Rosenbloom, P. S., (1987) Soar: An architecture for general intelligence. Artificial Intelligence, 33(3), 1-64.

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

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