Efficiently Implementing Episodic Memory

Nate Derbinsky and John E. Laird

University of Michigan
2260 Hayward Street
Ann Arbor, MI 48109-2121
{nlderbin, laird}@umich.edu

Abstract. Endowing an intelligent agent with an episodic memory affords it a multitude of cognitive capabilities. However, providing efficient storage and retrieval in a task-independent episodic memory presents considerable theoretical and practical challenges. We characterize the computational issues bounding an episodic memory. We explore whether even with intractable asymptotic growth, it is possible to develop efficient algorithms and data structures for episodic memory systems that are practical for real-world tasks. We present and evaluate formal and empirical results using Soar-EpMem: a task-independent integration of episodic memory with Soar 9, providing a baseline for graph-based, task-independent episodic memory systems.

1 Introduction

Episodic memory, as first described by Tulving [23], is a long-term, contextualized store of specific events. Episodic memory, what an individual "remembers," is contrasted with semantic memory, a long-term store of isolated, de-contextualized facts that an individual "knows." As an example, a memory of viewing artwork during one's last vacation would be episodic, whereas recalling the name of a famous gallery would likely be semantic (unless, for example, producing this information relied upon a memory of reading a brochure).

Nuxoll and Laird [14] demonstrated that an episodic store can afford an intelligent agent a multitude of cognitive capabilities. For example, an agent that recalls the results of actions taken previously in situations similar to its current state may learn to predict the immediate consequences of future actions (i.e. action modeling). In this paper, we extend this previous work, augmenting and refining the underlying implementation and providing a theoretical and empirical analysis of the algorithms and data structures necessary to support effective and efficient episodic memory.

Episodic memory research is closely related to studies in case-based reasoning (CBR). The goal of CBR is to optimize task performance given a case-base, where each case consists of a problem and its solution [6]. In CBR systems, case structure is typically pre-specified, case-base size is either fixed or grows at a limited rate, and the cases usually do not have any inherent temporal structure. In contrast, an episodic store grows monotonically with experience, accumulating snapshots of an agent's experiences over time. An agent endowed with this

L. McGinty and D.C. Wilson (Eds.): ICCBR 2009, LNAI 5650, pp. 403–417, 2009.

memory can retrieve relevant episodes to facilitate reasoning and learning based upon prior events.

In Section 2, we characterize episodic memory and introduce performance goals for its effective use in computational systems. We then compare our goals for episodic memory with related work. In Section 3, we present a brief overview of Soar-EpMem, the integration of episodic memory with Soar 9 [8], together with the domain we will use for evaluation. In Section 4, we present the details of Soar-EpMem, formally characterize its performance, and empirically evaluate its performance on a complex domain (>2500 features) over one million episodes. Earlier versions of Soar-Epmem [14] emphasized the cognitive capabilities episodic memory affords, while this work describes a completely new, robust, efficient, more complete implementation, together with the formal characterization and empirical evaluation.

2 Characterizing Episodic Memory

In this section, we first enumerate the relevant functional and implementation requirements for episodic memory. We then contextualize episodic memory within related case-based reasoning work. Finally, we analyze an existing task-independent episodic memory system apropos efficient implementation.

2.1 Episodic Memory Constraints

Episodic memory is distinguished from other memory systems by a set of functional requirements [13]. First, episodic memory is *architectural*: it is a functionality that does not change from task-to-task. Second, episodic memory is *automatic*: memories are created without a deliberate decision by the agent. Because memories are experiential, there is some temporal relationship between episodes. Furthermore, in contrast to most case-based reasoning systems, the set of features that can occur in an episodic memory is not pre-specified, but reflects whatever data is available via agent sensing and internal processing. Also, there is not a pre-specified subset of the episode that is used for retrieval nor is there a pre-specified subset that is the result, so that the complete episode is reconstructed during retrieval.

For this work, we adopt two additional restrictions on episodic memory:

1. Stored episodes do not change over time: memories do not decay and are not removed from the episodic store (no forgetting).
2. The goal of retrieval is to find the episode that is the Nearest Neighbor (NN) to the cue based upon qualitative matching (using recency of the episode as a tie-breaking bias).

This specification imposes challenging complexity bounds: the lack of episode dynamics dictates a monotonically increasing episodic store and capturing full agent state over time requires storage at least *linear* in state *changes*.

2.2 Related Case-Based Reasoning Work

Efficient NN algorithms have been studied in CBR for qualitative and quantitative retrieval [11] [21] [24]. The underlying algorithms and data structures supporting these algorithms, however, typically depend upon a relatively small and/or static number of case/cue dimensions, and do not take advantage of the temporal structure inherent to episodic memories.

Considerable work has been expended to explore heuristic methods that exchange reduced competency for increased retrieval efficiency [19], including refined indexing [2] [3], storage reduction [25], and case deletion [17]. Many researchers achieve gains through a two-stage cue matching process that initially considers *surface* similarity, followed by *structural* evaluation [4]. In this work, we take advantage of this approach and apply it to a monotonically growing, task-independent episodic memory.

The requirement of dealing with time-oriented problems has been acknowledged as a significant challenge within the CBR community [1], motivating work on temporal CBR (T-CBR) systems [16], and research on the representation of and reasoning about time-dependent case attributes [5], as well as preliminary approaches to temporal case sequences [12] [18]. However, existing T-CBR work does not deal with accumulating an episodic store, nor does it take advantage of temporal structure for efficient implementations.

2.3 EM: A Generic Memory Module for Events

EM [22] is a generic store to support episodic memory functionality in a variety of systems, including planning, classification, and goal recognition. EM is an external component with an API, wherein host systems must implement a thin interface layer. The term "episode" in EM defines a sequence of actions with a common goal and is represented as a triple: context ("general setting" of the episode), content (ordered set of the events that make up the episode), and outcome (a domain/task-specific evaluation of the result of the episode). Though meaningful in systems like planners, it may be difficult to pre-define action sequences and outcome evaluation functions for long-living agents that must contend with multiple, possibly novel, tasks.

EM queries are partially defined episodes and a single evaluation dimension. EM utilizes a two-stage evaluation scheme, whereby a constant number (5) of potential matches are found, which are then compared using a relatively expensive semantic matcher. While Tecuci and Porter have shown results for learning in short (250 episode), single-task domains, it is unclear whether the underlying algorithms and data structures will scale to agents with many orders of magnitude more episodes.

3 Integration of Episodic Memory in Soar

This section provides an introduction to the integration of episodic memory with Soar 9 [8], followed by an overview of the task we will use for evaluation.

3.1 Soar

Soar is a cognitive architecture that has been used extensively for developing AI applications and cognitive models. One of Soar's main strengths has been its ability to efficiently represent and bring to bear large bodies of symbolic knowledge to solve diverse problems using a variety of methods [10].

Although Soar has many components, the most relevant aspect of its design to episodic memory is that it holds all of its short-term knowledge in its working memory (WM). Working memory contains an agent's dynamic internal state, including perceptual data, situational awareness, current goals and intentions, and motor commands. By recording the contents of working memory, episodic memory can capture an agent's history of experience.

Soar's working memory is implemented as a directed, connected graph of working memory elements (WMEs). Each WME is a triple: *identifier* (a node or vertex), *attribute* (a link or edge), and *value* (a node or terminal value). Working memory is a set: no two WMEs can have the same identifier, attribute, and value.

3.2 Episodic Memory Integration

Figure 1 depicts the high-level integration of episodic memory with Soar. Episodes consist of snapshots of working memory and are automatically *stored* in the episodic store. Episodes are retrieved when an agent deliberately creates a cue, at which point Soar searches the store for candidate episodes, ranks them with respect to the cue (*cue matching*), selects the best match, and then *reconstructs* the episode in working memory (in a special area so that it does not overwrite existing working memory structures, nor is the retrieved episode confused with the agent's current experience).

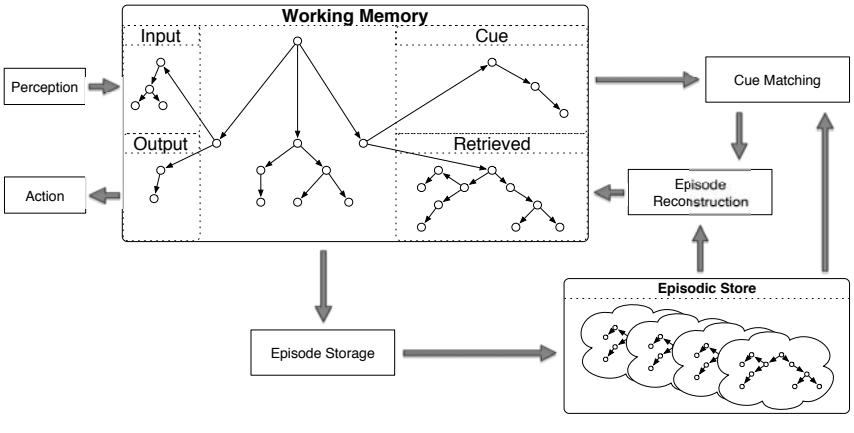

Fig. 1. Soar-EpMem Architecture

Soar's representation of working memory as an arbitrary graph structure has significant implications for the underlying implementation of episodic memory. A simpler representation, such as a vector or propositional representation, would make it possible to develop a simpler and faster implementation of episodic memory, but at significant cost in expressability and generality. The underlying implementation of episodic memory is independent of other details of Soar and should generalize to other architectures with graph-based representations of their dynamic data.

Soar agents interact with real-world, dynamic environments and these agents have real-time constraints on reactivity. Based on our own experience with real-time agents, Soar must execute its primitive cycle in 50-100ms to maintain reactivity in real-world agents. This bound is easily achieved for Soar's non-episodic memory components and puts a limit on how much time can be spent on storage, which can occur every cycle. However, there is more flexibility in retrieval, which can occur in parallel with Soar's other processing and can still be useful if it takes small multiples of Soar's primitive cycle time. Empirical evidence suggests that a retrieval time of approximately 500ms is necessary to be useful in real world applications.

3.3 Evaluation Domain

Our evaluation of Soar-EpMem is based on the TankSoar domain. TankSoar is a pre-existing domain that has been used extensively in evaluating other aspects of Soar and was used in the original episodic memory research in Soar [13]. In TankSoar, each Soar agent controls an individual tank that moves in a discrete 15x15 two-dimensional maze. Agents have only limited sensing of the environment and their actions include turning, moving, firing missiles, raising shields, and controlling radar. A TankSoar agent has access to a rich set of environmental features through its senses, including smell (shortest-path distance to the nearest enemy tank), hearing (the sound of a nearby enemy), path blockage, radar feedback, and incoming missile data.

TankSoar includes an agent named *mapping-bot* that builds up an internal map of the environment as it explores. The *mapping-bot* agent's working memory contains about 2,500 elements. Over 90% of these elements comprise a static map of its environment. A large proportion of the remaining WMEs (usually 70-90%) are related to perception and they typically change within one episode. For the experiments described below, a new episode was stored every time an action was performed in the environment, which is approximately every primitive decision cycle in Soar. The properties of this agent, especially the large working memory and the large number of WMEs changing per episode, make TankSoar an atypically stressful domain for episodic memory experimentation.

The tests were run on an Intel 2.8GHz Core 2 Duo processor and 4GB RAM. The Soar-EpMem episodic store was managed using version 3 of the SQLite in-process relational database engine [20]. The tests described below involved one million *mapping-bot* episodes, averaged over ten trials.

4 Soar-EpMem Structure and Evaluation

In this section, we present Soar-EpMem together with its evaluation. We begin with a description of global data structures that summarize the structures found in working memory. These global structures greatly decrease the amount of storage and processing required for individual episodes. The remaining subsections describe the different phases of episodic memory processing. The first is *storage* of episodic memories, the second is *cue matching* to find a stored episodic memory, and the final is *episodic reconstruction*, which involves finding all of the components of an episode and adding it to Soar's working memory.

The design of Soar-EpMem was motivated by the need to minimize the growth in processing time for all of these operations as the number of episodes increases. However, over long agent lifetimes, *cue matching* has the greatest growth potential and the overall design is meant to minimize the time required for that operation, without significantly impacting required memory or the time required for the other operations.

4.1 Global Episodic Memory Structures

In order to eliminate duplicate representations of working memory elements and speed *cue matching*, a global structure can be maintained that represents all the structures that have existed in working memory. A naïve storage representation would explicitly define an episode as a "bag" of pointers to such a global structure (see Fig. 2, left). Termed an "instance" representation by Nuxoll and Laird [14], such an approach requires time and storage linear in the average number of working memory elements per episode.

Our design for Soar-EpMem takes advantage of the temporal structure of episodes, namely that one episode usually will differ from the previous (and next) episode only in a relatively small number of features. Thus, throughout its design, Soar-EpMem attempts to process *only changes* to working memory instead of the complete episode. As a result, storing an episode involves only noticing which elements have been added and which elements have been removed from working memory, building up ranges of when working memory elements existed. Termed an "interval" representation by Nuxoll and Laird, episodes are implic-

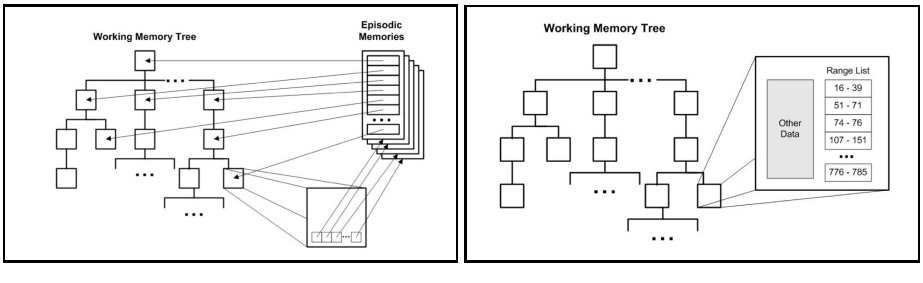

Fig. 2. Global Episodic Memory Structures [13]: Instance (left), Interval (right)

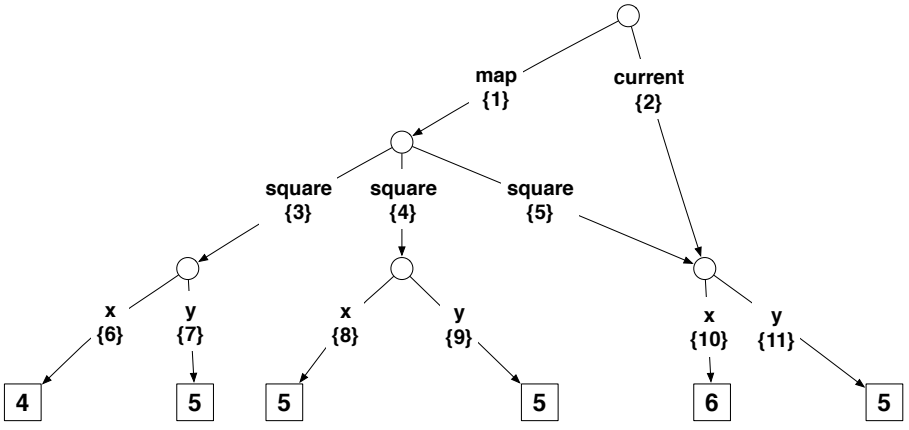

Fig. 3. Working Memory Example

itly represented by associating valid temporal ranges with a global structure (see Fig. 2, right). With this approach, *episode storage* is achieved in time and space linear with respect to the *changes* in the agent's working memory [13].

In Nuxoll's original work [13], the graph structure of working memory was simplified such that every attribute of an object was unique and that the overall structure was a tree. Fig. 3 shows a simple working memory that violates both of these assumptions. There are multiple squares ({3}, {4}, {5}) in the map ({1}), and a "square" ({5}) and "current" ({2}) edge share a common descendant. Under the assumption that working memory was a tree, Nuxoll's system built up a global structure (termed a *Working Memory Tree*) of all unique structures that had ever occurred in working memory. Using a tree instead of a graph for modeling working memory greatly simplified Nuxoll's implementation and was sufficient for him to explore applications of episodic memory; however these simplifications make it impossible to correctly search and reconstruct episodes based upon relational working memory structures, which is necessary for many real-world applications. To correct this deficiency, Soar-EpMem implements a new global structure, the *Working Memory Graph* (WMG), which captures all information necessary for a faithful episode representation.

4.2 Episode Storage

The Working Memory Graph captures all distinct edges that have occurred in Soar's working memory. By associating valid temporal intervals (see Fig. 2, right) with the unique ids of each edge (such as {1}, {2}, etc.), we implicitly define structure for individual episodes. Episode *storage* is the process of efficiently recording the start/end of these intervals.

In Soar-EpMem, interval ranges are started by executing the following algorithm for every element of working memory:

1. if element already points to the WMG, ignore
2. else:
   (a) if a corresponding edge does not exist in the WMG, add it
   (b) point the element to the corresponding WMG edge
   (c) start a new interval at the pointed WMG edge

By ignoring elements with existing pointers (step 1), we process only *new* working memory elements. Later, following an element's removal from working memory, Soar-EpMem records the end of the corresponding edge interval. Thus our implementation only stores element *changes*.

Using this approach, Soar-EpMem *storage* time across one million episodes of *mapping-bot* remained approximately constant, requiring 1.48-2.68 ms/episode. Total episodic store size ranged from 625-1620 MB, averaging between 0.64 and 1.66 KB/episode (respectively).

4.3 Cue Matching

Episodic cues take the form of an acyclic graph, partially specifying a subset of an episode. For example, an agent can create a cue of a position in the map to recall what it sensed the last time it was in that position.

Episode retrieval proceeds as follows:

1. 2-Phase Nearest Neighbor Cue Matching
   (a) Identify candidate episodes based upon *surface* cue analysis.
   (b) Perform *structural* cue analysis on *perfect surface* matches.
2. Select and reconstruct the best matching episode (if it exists).

Candidate episodes are defined as containing at least one cue leaf element. Our *surface* evaluation function returns the "balanced" sum of match cardinality (number of matching leaf nodes) and feature weighting [15].

Candidate episodes with perfect match cardinality are considered *perfect surface matches* and are submitted to a second phase of *structural* graph-match. The graph-match algorithm implements standard CSP backtracking. The best matching episode is either the most recent *structural* match or the highest scoring *surface* match.

Soar-EpMem efficiently iterates over candidate episodes by implementing Nuxoll and Laird's [13] interval searching algorithm. The major insight of this algorithm is that a candidate match score changes only at the endpoints of episode element intervals. For example, consider Fig. 4. The dashed vertical line at episode 6 indicates the time point of maximum coverage (i.e. maximum match cardinality). If we begin from the most recent time point (time point 12) and consider each candidate episode, we will examine 7 episodes before arriving at this peak. However, after scoring episode 8, there will be no change in coverage until time point 6: considering episodes between endpoints is redundant. Thus

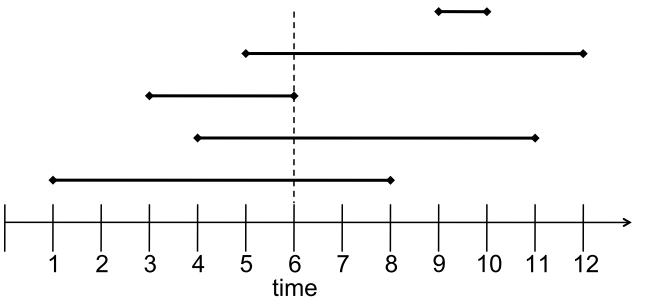

Fig. 4. Interval Search Example

if we just walk interval endpoints, incrementally updating match score, we can achieve significant computational savings. By *only processing changes*, candidate episode iteration achieves time linear in the relevant element *changes* (as opposed to linear in the number of episodes).

As an optimization, Soar-EpMem will cease interval search upon reaching a *structural* match. In Fig. 4, however, there is no perfect *surface* match (since no episode is covered by all ranges). Thus, if we assume uniform feature weighting, interval search will return episode 6 after processing all 10 endpoints.

Soar-EpMem efficiently implements Nuxoll and Laird's interval search algorithm by maintaining B+-tree indexes of all temporal interval endpoints (one for interval start, one for interval end), keyed on episode nodes. Walking a node's endpoints in descending order of time entails finding the most recent time of interest (log time with respect to the number of endpoints), and walking the leaf nodes in order (constant time per endpoint). To process a multi-node cue, we maintain parallel B+-tree leaf pointers for each cue node. All pointers within a B+-tree are stored in priority queues (keyed on endpoint value). We then efficiently walk endpoints as described above.

Because Soar's working memory is implemented as a graph, cue leaf nodes do not uniquely identify an edge in the Working Memory Graph. For example, consider the cue in Fig. 5, left (relating to working memory as illustrated in Fig. 3). The internal cue node representing a "square" can be satisfied by any of the three working memory squares ($\{3\}$, $\{4\}$, $\{5\}$). Thus, the problem of satisfying a leaf node is tantamount to satisfying the sequence of nodes that leads to (and includes) the leaf node. To continue the example, we can formally express the satisfaction of the two cue leaf nodes with the following monotonic, disjunctive normal form (DNF) boolean statements (respecting ids in Fig. 3):

$$
\begin{aligned}
sat(x = 4) := {} & (\{1\} \bigwedge \{3\} \bigwedge \{6\}) \\
sat(y = 5) := {} & (\{1\} \bigwedge \{3\} \bigwedge \{7\}) \bigvee \\
& (\{1\} \bigwedge \{4\} \bigwedge \{9\}) \bigvee \\
& (\{1\} \bigwedge \{5\} \bigwedge \{11\})
\end{aligned}
$$

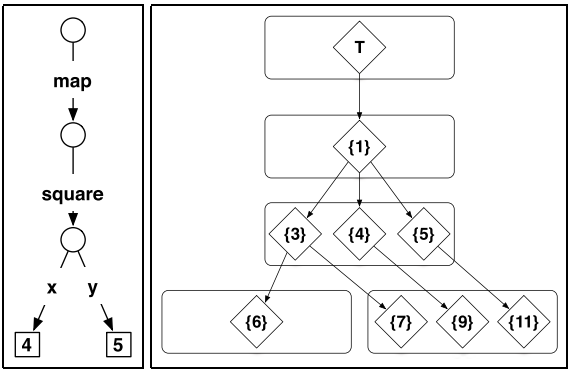

Fig. 5. Retrieval Example: Cue (left), Corresponding DNF Graph (right)

Thus, the problem of efficiently implementing interval search with a Working Memory Graph is analogous to efficiently tracking satisfaction of a set of DNF boolean equations. Soar-EpMem solves this problem by *only processing changes* to a corresponding *DNF graph* (depicted in Fig. 5, right).

The interval search process is initialized while processing the cue. During a breadth-first search, we simultaneously create the priority queue of B+-tree pointers (as discussed above) and the DNF graph. For clarity, a clause in a DNF statement is represented as a root-to-leaf path in the DNF graph. Each literal in the DNF graph (depicted as a diamond) has an associated count value, initialized to 0, representing local satisfaction. A literal count is incremented if its associated cue element node is "active" (i.e. our current state of interval search is within the start/end endpoints of the node) OR its parent has a counter value of 2. The root literal (shown with a special id of "T") is initialized with count 2 (thus initializing the count of all direct children to 1). If at least one leaf literal is satisfied (i.e. has count value 2), the clause must be satisfied. We additionally maintain a global match score, initialized to 0.

At each endpoint in the interval search algorithm, exactly one literal is activated or de-activated (depending on whether we encounter an end/start). If the literal is part of an internal cue node, this change may entail recursive propagation to child literals. If during propagation we alter clause satisfaction, we modify the global match score. Thus, we extend endpoint iteration to track *only changes* in boolean satisfaction of the DNF graph and, by extension, modifications of candidate match score.

To compare Soar-EpMem *cue matching* performance with theoretical bounds, we developed the following model to reflect the effects of operational algorithms and data structures:

$$\textit{Cue Match} = \textit{DNF} + \textit{Interval Search} + \textit{Graph Match}$$
$$\textit{DNF} = (X_1)(\log_2\lceil\ U * R\ \rceil)(L)$$
$$\textit{Interval Search} = (X_2)(1/T)(\textit{Distance})(\Delta)$$

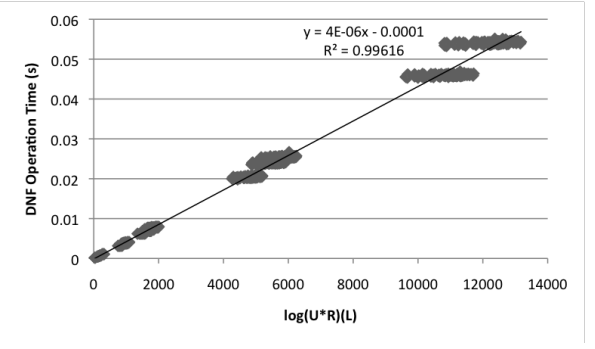

Fig. 6. *Cue Matching* DNF Regression Data

The constants in the equations (X_1, X_2) reflect linear scaling factors for a given computer. To derive these values for our experimental setup, we performed 100 isolated executions of primitive operations (*DNF* and *Interval Search*) on data collected from 10 trials of *mapping-bot* data at 10 time points (100K, 200K, ... 1M). We collected the necessary episode statistics (described below) and performed linear regressions to fit data points for 15 different queries. Low performance timers (resolution was 1μs) caused most model noise.

The *Cue Match* operation comprises *DNF* construction and *Interval Search*. The former is linearly dependent upon the logarithmic growth of the average number, U, of historically unique internal and leaf nodes multiplied by R, the total number of stored intervals, as well as linearly dependent upon L, the number of literals associated with the cue nodes. In our tests (see Fig. 6), we found X_1 to be 4.33μs (R^2=0.996). In experiments with *mapping-bot* to one million episodes, results depended greatly on the cue. For all cues that did not reference "squares" on the agent's internal map, *DNF* operation time was constant and below 8 ms. Cues containing references to map "squares" (and thus referring to over 250 underlying structures) brought this upper bound to 55.1 ms.

The *Interval Search* operation is expressed as a proportion of relevant cue node intervals. T represents the total number of episodes recorded. *Distance* represents the temporal difference between the current episode and the best match. Δ represents the total number of intervals relevant to the cue. Intuitively, the farther back in time we must search for an episode, the more intervals we must examine. This ratio could be re-written as the product the minimal relative co-occurrence probability of the cue nodes, and the total number of changes experienced to date by these cue nodes. In our tests (see Fig. 7), the X_2 constant was 1.29μs (R^2=0.989). We found absolute operation times depended greatly on the supplied cue. For cues that did not compel distant searches, *Interval Search* was constant with an upper bound of 2.5 ms. With cues crafted to force a linear scan of the episodic store, time increased linearly to a maximum of 1.03 seconds over one million episodes.

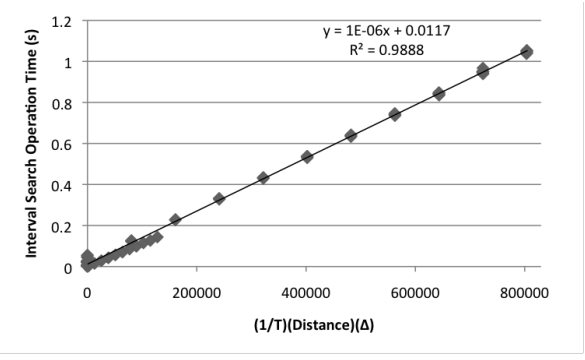

Fig. 7. *Cue Matching* Interval Search Regression Data

Since the linear factors, L and Δ, grow proportionally to *changes* in agent working memory, the first phase of Soar-EpMem *cue matching* achieves the lower bound of growing *linearly* with agent *changes*.

The *Graph Match* operation, however, is much more difficult to characterize. CSP backtracking depends upon cue breadth, depth, structure (such as shared internal cue nodes), and corresponding candidate episodes, but can be combinatorial in the worst case (though our two-phase matching policy attempts to minimize this cost). We have not extensively evaluated this component, but we expect a studied application of heuristic search will effectively constrain graph-match in the average case.

4.4 Episode Reconstruction

Given the temporal id of an episode, *reconstruction* in Soar-EpMem can be summarized by the following two steps:

1. Collect contributing episode elements
2. Add elements to working memory

Given the episode elements, adding WMEs to Soar is a straight forward process that takes advantage of Soar's existing architectural primitives and thus we focus on the former step.

Collecting episode elements in an "interval" representation (see Fig. 2, right) is tantamount to an interval intersection query: collect all elements that started before and ended after time t. To facilitate efficient *episode reconstruction*, Soar-EpMem maintains a Relational Interval Tree (RI-tree) [7]. An RI-tree is a mapping of the interval tree data structure onto Relational Database Management System (RDBMS) B+-tree indexes and SQL queries. As with a standard interval tree, intersection queries execute in logarithmic time with respect to the number of stored intervals.

Excluding system-specific step 2 above, we developed the following model for *episode reconstruction* performance in Soar-EpMem:

$$Reconstruction = RI\text{-}tree + Collect$$
$$RI\text{-}tree = (X_3)(log_2 R)$$
$$Collect = (X_4)(M)(1 + log_2 U)$$

To validate our model we performed 100 isolated executions of primitive operations (*RI-tree* and *Collect*) on the same data collected for *cue-matching* (10 trials, *mapping-bot*, 10 time points from 100K to 1M episodes). We collected the necessary statistics (described below) for 50 episodes selected randomly (5 per 10,000 episodes through the first 100,000 episodes of exection) and performed linear regressions to fit data points.

Total time for *episode reconstruction* is the sum of two operations: *RI-tree* and *Collect*. *RI-tree* refers to the process of extracting pertinent intervals from the Relational Interval Tree. The logarithmic dependent variable, R, refers to the total number of ranges in the RI-tree structure. In our experiments, the X_3 constant was 2.55μs (over 70% R^2). After one million episodes, we recorded the upper bound of *RI-tree* operation time as 0.1 ms.

The *Collect* operation refers to cross-referencing pertinent episode intervals with structural information in the Working Memory Graph. This process depends upon the average number, U, of historically unique internal and leaf nodes, as well as the number of elements, M, comprising the episode to be reconstructed. With *mapping-bot* we regressed an X_4 value of 1.6μs. Because episode size does not vary greatly in *mapping-bot* (2500-2600 elements, typically), the dominating linear factor, M, highlighted noise in the experimental data and thus R^2 was 73%. After one million episodes, we recorded an upper bound of 22.55 ms for the *collection* operation with episodes ranging from 2521-2631 elements.

If we assume a constant or slowly growing average episode size, the M factor can be considered a constant and thus *Reconstruction* becomes the linear sum of logarithmic components R and U. Both R and U increases result from *changes* in agent working memory. Thus, under these assumptions, Soar-EpMem *episode reconstruction* achieves the lower bound of growing *linearly* with agent *changes*.

5 Conclusion

In this paper, we presented an implementation of a graph-based, task-independent episodic memory and characterized the associated computational challenges. We provided formal models of the costs associated with the different phases of episodic processing and provided empirical results over one million episodes.

In this work we applied efficient data structures and algorithms to limit episodic computation, while still guaranteeing a best-match retrieval. Thus, we consider that a typical cue is one for which retrieval does not require a linear scan of the episodic store. Table 1 summarizes typical empirical results for *mapping-bot* over one million episodes. *Storage* time remains nearly constant, and well below the bound of 50ms, with linear growth in storage. *Cue matching* time is within desired bounds, but suffers linear growth in the atypical case (with a

Table 1. Soar-EpMem Typical Operation Costs: *mapping-bot*

episodes	*storage*	*cue matching*	*reconstruction*	total
1,000,000	2.68ms, 1620MB	57.6ms	22.65ms	82.93ms

maximum observed cost of 1.03 sec.). Episodic *reconstruction* time is dominated by episode size but falls within desired bounds for over 2500 features. Although these results achieve our initial performance goals, there is still much to be done:

- **Additional Empirical Evaluation.** We need to expand to additional domains and establish a set of test cases for episodic memory systems. TankSoar is a stressful test, useful for initial exploration and evaluation, but it is artificial and we need a collection of tasks that use episodic memory in a variety of ecologically valid ways.
- **Extended Evaluations.** We also need to explore much longer runs. One million episodes corresponds to approximately fourteen hours of real time. Our goal is to have agents that exist continually for 1 year (42-420 million episodes [9]).
- **Cue Match Bounding.** Although storage and reconstruction are relatively well behaved, the cost of cue matching can be extremely variable depending on the complexity of the cue and the structures in episodic memory. We need to explore alternative or even heuristic graph-matching schemes that provide tighter bounds.

References

1. Combi, C., Shahar, Y.: Temporal Reasoning and Temporal Data Maintenance in Medicine: Issues and Challenges. Computers in Biology and Medicine 27(5), 353–368 (1997)
2. Daengdej, J., Lukose, D., Tsui, E., Beinat, P., Prophet, L.: Dynamically Creating Indices for Two Million Cases: A Real World Problem. In: Smith, I., Faltings, B. (eds.) EWCBR 1996. LNCS, vol. 1168, pp. 105–119. Springer, Heidelberg (1996)
3. Fox, S., Leake, D.: Using Introspective Reasoning to Refine Indexing. In: Proceedings of the 14th International Joint Conference on Artificial Intelligence, pp. 391–399. Morgan Kaufmann, San Francisco (1995)
4. Forbus, K., Gentner, D., Law, K.: MAC/FAC: A Model of Similarity-based Retrieval. Cognitive Science 19(2), 141–205 (1995)
5. Jære, M., Aamodt, A., Skalle, P.: Representing temporal knowledge for case-based prediction. In: Craw, S., Preece, A.D. (eds.) ECCBR 2002. LNCS, vol. 2416, pp. 174–234. Springer, Heidelberg (2002)
6. Kolodner, J.: An Introduction to Case-Based Reasoning. Artificial Intelligence Review 6(1), 3–34 (1992)
7. Kriegel, H., Pötke, M., Seidl, T.: Managing Intervals Efficiently in Object-Relational Databases. In: Proceedings of the 26th International Conference on Very Large Databases, pp. 407–418. Morgan Kaufmann, San Francisco (2000)

8. Laird, J.E.: Extending the Soar Cognitive Architecture. In: Proceedings of the 1st Conference on Artificial General Intelligence, pp. 224–235. IOS Press, Amsterdam (2008)
9. Laird, J.E., Derbinsky, N.: A Year of Episodic Memory. In: Workshop on Grand Challenges for Reasoning from Experiences, 21st International Joint Conference on Artificial Intelligence (2009)
10. Laird, J.E., Rosenbloom, P.: The Evolution of the Soar Cognitive Architecture. Mind Matters: A Tribute to Allen Newell, pp. 1–50. Lawrence Erlbaum Associates, Inc., Mahwah (1996)
11. Lenz, M., Burkhard, H.: Case Retrieval Nets: Basic Ideas and Extensions. In: Görz, G., Hölldobler, S. (eds.) KI 1996. LNCS, vol. 1137, pp. 227–239. Springer, Heidelberg (1996)
12. Ma, J., Knight, B.: A Framework for Historical Case-Based Reasoning. In: Ashley, K., Bridge, D. (eds.) ICCBR 2003. LNCS, vol. 2689, pp. 246–260. Springer, Heidelberg (2003)
13. Nuxoll, A.: Enhancing Intelligent Agents with Episodic Memory. PhD Dissertation, University of Michigan, Ann Arbor (2007)
14. Nuxoll, A., Laird, J.E.: Extending Cognitive Architecture with Episodic Memory. In: Proceedings of the 22nd AAAI Conference on Artificial Intelligence, pp. 1560–1564. AAAI Press, Vancouver (2007)
15. Nuxoll, A., Laird, J.E., James, M.: Comprehensive Working Memory Activation in Soar. In: Proceedings of the 6th International Conference on Cognitive Modeling, pp. 226–230. Lawrence Erlbaum Associates, Inc., Mahwah (2004)
16. Patterson, D., Galushka, M., Rooney, N.: An Effective Indexing and Retrieval Approach for Temporal Cases. In: Proceedings of the 17th International Florida Artificial Intelligence Research Society Conference, pp. 190–195. AAAI Press, Vancouver (2004)
17. Patterson, D., Rooney, N., Galushka, M.: Efficient Retrieval for Case-Based Reasoning. In: Proceedings of the 16th International Florida Artificial Intelligence Research Society Conference, pp. 144–149. AAAI Press, Vancouver (2003)
18. Sánchez-Marré, M., Cortés, U., Martínez, M., Comas, J., Rodríguez-Roda, I.: An approach for temporal case-based reasoning: Episode-based reasoning. In: Muñoz-Ávila, H., Ricci, F. (eds.) ICCBR 2005. LNCS, vol. 3620, pp. 465–476. Springer, Heidelberg (2005)
19. Smyth, B., Cunningham, P.: The Utility Problem Analysed: A Case-Based Reasoning Perspective. In: Smith, I., Faltings, B. (eds.) EWCBR 1996. LNCS, vol. 1168, pp. 392–399. Springer, Heidelberg (1996)
20. SQLite, `http://www.sqlite.org`
21. Stottler, R., Henke, A., King, J.: Rapid Retrieval Algorithms for Case-Based Reasoning. In: Proceedings of the 11th International Joint Conference on Artificial Intelligence, pp. 233–237. Morgan Kaufmann, San Francisco (1989)
22. Tecuci, D., Porter, B.: A Generic Memory Module for Events. In: Proceedings of the 20th International Florida Artificial Intelligence Research Society Conference, pp. 152–157. AAAI Press, Vancouver (2007)
23. Tulving, E.: Elements of Episodic Memory. Clarendon Press, Oxford (1983)
24. Wess, S., Althoff, K., Derwand, G.: Using k-d Trees to Improve the Retrieval Step in Case-Based Reasoning. In: Wess, S., Althoff, K., Richter, M. (eds.) EWCBR 1993, vol. 837, pp. 167–181. Springer, Heidelberg (1994)
25. Wilson, D., Martinez, T.: Reduction Techniques for Instance-Based Learning Algorithms. Machine Learning 38(3), 257–286 (2000)