Soar-EpMem Manual

Version 0.3
16 December 2009

Contributors
Jakub Czyz
Nate Derbinsky
Nicholas Gorski
John Laird
Bob Marinier
Andy Nuxoll
Joseph Xu

Table of Contents

Document History

Version 0.3
Integration with Soar-SMem (long-term identifiers).

Version 0.2
Second release.

Version 0.1
Initial specification.

1. Soar-EpMem Motivation

Soar-EpMem is a task-independent, architectural integration of an artificial episodic memory (EpMem) with Soar. The EpMem mechanism will automatically record episodes as a Soar agent executes. These episodes can later be queried and retrieved in order to improve performance on future tasks.

2. Working Memory Structure

Upon creation of a new state within working memory, the architecture will automatically create a structure in working memory called **epmem**. Within this structure, agents issue requests to Soar-EpMem (see Section 4.1) by populating the **command** identifier with working memory elements (WMEs) and process Soar-EpMem generated WMEs in the **result** identifier (see Section 4.4).

3. Storing Episodes

This section details Soar-EpMem storage of episodes, including new episode triggering, what is stored, interactions with Working Memory Activation (WMA), as well as where and in what format the episodes are stored.

3.1. New Episode Creation

One functional requirement of an artificial episodic memory is that recording new episodes does not require deliberate action/consideration by the agent. Soar-EpMem provides automatic storage of new episodes as dictated by the **phase** and **trigger** parameters. The **phase** parameter sets the phase in the decision cycle (default: end of Output phase) during which Soar-EpMem stores episodes and processes commands. The value of the **trigger** parameter indicates to the architecture the event for automatic storage: adding a new command to the output-link (default) or each decision cycle.

For debugging purposes, the **force** parameter allows the user to manually request that an episode be recorded (or not) during the current decision cycle. Behavior is as follows:

- The value of the **force** parameter is initialized to **off** every decision cycle.
- During the new episode consideration where the **force** parameter has a value other than of **off**, Soar-EpMem follows the forced policy irrespective of the value of the **trigger** parameter.

3.2. Episode Contents

For an artificial episodic memory to be task-independent it must record most, if not all, information available to the agent at the time of episode creation. By default, when Soar-EpMem records a new episode, it stores the entire contents of the top state of Working Memory.

There are classes of WMEs that Soar agents may encounter that provide no benefit in context of EpMem. For instance, the "random" WME on the TankSoar input-link structure provides a different random number on each update; this value is potentially useful to an agent designer, but most likely will not contribute to effective episodic learning. Moreover, excluding WMEs from storage can provide performance benefits (reduced memory consumption and storage/retrieval time). The **exclusions** parameter allows run-time management of a list of attribute names that will be ignored during Soar-EpMem storage.

3.3. Working Memory Activation

During the episodic retrieval process (discussed in detail later), multiple episodes may match an agent's query. Nuxoll has produced data that demonstrates improved retrieval quality when using Working Memory Activation (WMA) of WMEs as a form of feature weighting. Thus, Soar-EpMem supports integration with WMA in Soar. For a theoretical discussion of the Soar implementation of WMA, consider reading *Comprehensive Working Memory Activation in Soar* (Nuxoll, A., Laird, J., James, M., ICCM 2004).

The following sections detail configuration of WMA, including the **wma** command and WMA parameters.

3.3.1. The wma Command

Management of WMA within Soar makes use of the **wma** command. Executing the command with no options will print a table of current parameter information. The **wma** command has getter (**-g** or **--get**) and setter (**-s** or **--set**) options for retrieving/ manipulating parameter values (discussed in the next section).

3.3.2. WMA Parameters

The following table briefly describes the parameters available for manipulation using the **wma** command's **get**/**set** options. Further text below provides more information regarding specific parameters.

Parameter Name	Acceptable Values	Default	Description
activation	on off	off	Turns on/off WMA
decay-rate	[0,1]	0.8	Specifies the speed at which WMEs are decayed
forgetting	on off	off	Turns on/off removal of WMEs with low activation values

activation

If **activation** is **on**, printing working memory elements with the **–i** or **--internal** option will print the current activation value in brackets. For example, after starting Soar:

```
>print --internal <s>
(4: S1 ^epmem E1 [1])
(10: S1 ^io I1 [1])
(3: S1 ^reward-link R1 [1])
(7: S1 ^smem S2 [1])
(2: S1 ^superstate nil [1])
(1: S1 ^type state [1])
```

As shown above, architectural WMEs carry a permanent activation of 1.

decay-rate

The **decay-rate** parameter controls the speed at which WMEs are decayed. A value of 0 will decay WMEs instantly, where 1 will not reduce initial activation level. Note that the value is internally multiplied by -1 (which is reflected upon retrieving the parameter value).

3.4. Soar-EpMem Storage

EpMem currently uses SQLite to facilitate efficient and standardized storage and querying of episodes. The episodic store can be maintained in memory or on disk (per the **database** and **path** parameters). If the store is located on disk, users can use any standard SQLite programs/components to access/query its contents.

4. Retrieving Episodes

This section details the agent interface to Soar-EpMem retrievals, including command protocol, non-cue-based (NCB) retrievals, cue-based (CB) retrievals, and retrieval meta-data.

4.1. Soar-EpMem Commands

An agent issues a command to the Soar-EpMem system by populating appropriate WMEs on the **command** identifier of a state's **epmem** structure. During command processing (per the **phase** parameter), which comes after creating a new episode is considered (and possibly recorded), Soar-EpMem processes each state's EpMem **command** structure. Results, meta-data, and errors are placed on the **result** identifier of that state's **epmem** structure (discussed in Section 4.4).

Only one type of command (which may consist of multiple WMEs) can be issued in a single decision cycle (though multiple states may issue commands). Malformed commands (including attempts at multiple commands) will result in an error.

After a command has been processed, Soar-EpMem will ignore it until some aspect of the **command** structure changes (via addition/removal of WMEs). When this occurs, the **result** structure is cleared and the new command (if one exists) is processed.

4.2. Non-Cue-Based Retrievals

The following sections discuss issuing absolute and relative NCB retrieval commands.

4.2.1. Absolute NCB Retrievals

At time of storage, each episode is attributed a unique temporal id. This id is the current value of **time** statistic (see Section 6) and is provided as the **memory-id** meta-data item of retrieved episodes (see Section 4.4). An absolute NCB retrieval is one that requests an episode by temporal id. An agent issues an absolute NCB retrieval by placing a WME on the **command** structure with name **retrieve** and value equal to the desired temporal id:

```
state.epmem.command.retrieve temporal-id
```

Supplying an invalid value for the **retrieve** command will result in an error.

The temporal id of the first episode in an episodic store will have value 1 and each subsequent episode's temporal id will increase by 1. Thus the desired temporal id may be the mathematical result of operations performed on a known episode's temporal id.

This implementation of Soar-EpMem does not implement any episode dynamics, including forgetting. Thus any integer temporal id greater than 0 and less than the current value of the **time** statistic will be valid. However, if forgetting is implemented in future versions, no such guarantee will be made.

4.2.2. Relative NCB Retrievals

One interesting characteristic of episodic memory is the empirical ability to "play forward" episodes through time. Soar-EpMem implements this functionality through relative NCB retrievals.

The system stores the temporal id of the last successful retrieval (NCB or CB). Agents can indirectly make use of this information by issuing **next** or **previous** commands. Soar-EpMem executes these commands by attempt to retrieve the episode immediately proceeding/preceding the last successful retrieval (respectively). To issue one of these commands, the agent must create a new identifier with the appropriate command name on the **command** structure:

```
state.epmem.command.next <n>
state.epmem.command.previous <p>
```

If no such episode exists then an error is returned.

In this implementation of Soar-EpMem, if the temporal id of the last successfully retrieved episode is known to the agent (as could be the case by accessing result meta-data), these commands are identical to performing an absolute NCB after adding/subtracting 1 to the last temporal id (respectively). However, if an episode dynamic like forgetting is implemented, these relative commands are guaranteed to return the next/previous valid episode (assuming one exists).

4.3. Cue-Based Retrievals

CB retrieval commands are used to search for an episode in the store that "best" matches an agent-supplied cue, while potentially adhering to optional modifiers. A cue is composed of WMEs that partially describe a top state of Working Memory in the retrieved episode. All CB retrieval requests must contain a single **query** cue and, optionally, a single **neg-query** cue. A **query** cue describes structures desired in the retrieved episode, whereas a **neg-query** cue describes non-desired structures. For example, the following Soar production creates a **query** cue consisting of a particular state name and a copy of a current value on the input link structure:

```
sp {sample*query
    (state <s> ^epmem.command <ec>
               ^io.input-link.foo <bar>)
-->
    (<ec> ^query <q>)
    (<q> ^name my-state-name
         ^io.input-link.foo <bar>)
}
```

CB retrievals can be thought of as a nearest-neighbor search. First, all candidate episodes, (defined as episodes containing at least one leaf WME in at least one cue) are identified. Two quantities are calculated for each candidate episode, with respect to the supplied cue(s): the **cardinality** of the match (defined as the number of matching leaf WMEs) and the **activation** of the match (defined as the sum of the WMA decay values of each matching leaf WME). Note that each of these values is negated when applied to a **neg-query**. To

form each candidate episode's match score, these quantities are combined with respect to the **balance** parameter as follows:

```
Match Score = (balance)*(cardinality) + (1 - balance)*(activation)
```

An episode with perfect cardinality is considered a perfect *surface* match and, per the **graph-match** parameter, is subjected to further structural matching. Whereas surface matching efficiently determines if all paths to leaf WMEs *exist* in a candidate episode, graph matching indicates whether or not the cue can be unified with the candidate episode (paying special regard to the structural constraints imposed by shared identifiers). Cue-based matching will return the most recent structural match, or the most recent candidate episode with the largest match score.

A special note should be made with respect to how short- vs. long-term identifiers are interpreted in a cue. Short-term identifiers are processed much as they are in working memory – transient structures. Cue matching will try to find any identifier in an episode (with respect to incoming/outgoing edges) that can apply. Long-term identifiers, however, are treated as constants. Thus, when analyzing the cue, episodic memory will not consider outgoing structures from the long-term identifier, and will only match with the same long-term identifier (in the same context) in an episode.

The CB retrieval process can be further tempered using optional modifiers:

- The **before** command requires that the retrieved episode come relatively before a supplied temporal id:

  ```
  state.epmem.command.before temporal-id
  ```

- The **after** command requires that the retrieved episode come relatively after a supplied temporal id:

  ```
  state.epmem.command.after temporal-id
  ```

- The **prohibit** command requires that the temporal id of the retrieved episode is not equal to a supplied temporal id:

  ```
  state.epmem.command.prohibit temporal-id
  ```

  Multiple **prohibit** command WMEs may be issued as modifiers to a single CB retrieval.

If no episode satisfies the cue(s) and optional modifiers an error is returned.

4.4. Retrieval Meta-Data

The following list details the WMEs Soar-EpMem populates in the **result** identifier of the **epmem** structure wherein a command was issued:

- **retrieved <episode>**
  If Soar-EpMem retrieves an episodic memory, that memory is placed here. This WME is an identifier that is treated as the root of the state that was used to create

the episodic memory. If the **retrieve** command was issued with an invalid temporal id, the value of **retrieved** will be **no-memory**.

- **status**
  This WME provides information about the result of a retrieval command:
  - **success** – the CB retrieval command resulted in a successful match
  - **failure** – the CB retrieval was legitimate but no matching episode was found
  - **bad-cmd** – the command was malformed or more than one command was issued

  If the CB retrieval was well-formed, the WME will have the **status** as the attribute and the value of the identifier of the **query** (and **neg-query**, if applicable).
- **match-score**
  This WME is created whenever an episode is successfully retrieved from a CB retrieval command. The WME value is a decimal indicating the raw match score for that episode with respect to the cue(s).
- **cue-size**
  This WME is created whenever an episode is successfully retrieved from a CB retrieval command. The WME value is an integer indicating the number of leaf WMEs in the cue(s).
- **normalized-match-score**
  This WME is created whenever an episode is successfully retrieved from a CB retrieval command. The WME value is the decimal result of dividing the raw match score by the cue size. It can hypothetically be used as a measure of Soar-EpMem's relative confidence in the retrieval.
- **match-cardinality**
  This WME is created whenever an episode is successfully retrieved from a CB retrieval command. The WME value is an integer indicating the number of leaf WMEs matched in the **query** cue minus those matched in the **neg-query** cue.
- **memory-id**
  This WME is created whenever an episode is successfully retrieved from a CB retrieval command. The WME value is an integer indicating the temporal id of the retrieved episode.
- **present-id**
  This WME is created whenever an episode is successfully retrieved from a CB retrieval command. The WME value is an integer indicating the current temporal id, such as to provide a sense of "now" in EpMem terms. By comparing this value to the **memory-id** value, the agent can gain a sense of the relative time that has passed since the retrieved episode was recorded.
- **graph-match**
  This WME is created whenever an episode is successfully retrieved from a CB retrieval command and the **graph-match** parameter was **on**. The value is an integer with value **1** if graph-match was run and successful and **0** otherwise.
- **mapping**
  This WME is created whenever an episode is successfully retrieved from a CB retrieval command, the **graph-match** parameter was **on**, and structural match was successful on the retrieved episode. This WME provides a mapping between

identifiers in the cue and in the retrieved episode. For each identifier in the cue, there is a **node** WME as a child to the **mapping** WME. The **node** has a **cue** child WME (whose value is an identifier in the cue) and a **retrieved** child WME (whose value is an identifier in the retrieved episode. In a graph match it is possible to have multiple identifier mappings – this map represents the "first" unified mapping.

5. Soar-EpMem Parameters

The following sections discuss how to configure the Soar-EpMem parameters discussed in previous sections.

5.1. Parameter Configuration

Individual configuration parameters are retrieved and manipulated using the **get** and **set** switches of the **epmem** command:

```
epmem [-g|--get] <parameter>
epmem [-s|--set] <parameter> <value>
```

Agents can retrieve and change parameters in the actions of rules using the **cmd** function.

5.2. Parameter Descriptions

All Soar-EpMem parameters are organized below. The *Protected* field is discussed in Section 5.4).

5.2.1. General

Purpose	Enable or disable Soar-EpMem	
Parameter	**learning**	
Values	**off**	Disable Soar-EpMem
	on	Enable Soar-EpMem
Default	**off**	
Protected	no	

5.2.2. Encoding

Purpose	Specifies the phase during which new episode storage is considered and commands are processed	
Parameter	**phase**	
Values	**output**	At the end of output phase
	selection	At the end of selection phase
Default	**output**	
Protected	no	

Purpose	Specifies what triggers new episode creation	
Parameter	**trigger**	
Values	**dc**	Episodes are recorded every decision cycle
	none	Episodes are not automatically recorded
	output	Episodes are recorded decision cycles when there is an addition to the output-link identifier
Default	**output**	
Protected	no	

Purpose	Forces recording/non-recording of an episode during the current decision cycle	
Parameter	**force**	
Values	**ignore**	No episode is recorded this decision cycle
	remember	An episode will be recorded this decision cycle
	off	Episode recording is dependent upon the current trigger
Default	**off**	
Protected	no	

Purpose	Specifies a list of WME attribute names that are ignored during episode creation	
Parameter	**exclusions**	
Values	**<any string>**	If the supplied value does not currently exist within the exclusion list it is added, otherwise it is removed from the list.
Default	**{epmem,smem}**	
Protected	No	

5.2.3. Storage

Purpose	Specifies whether the episodic store will be maintained in memory or on disk	
Parameter	**database**	
Values	**file**	Episodic store is maintained on disk
	memory	Episodic store is maintained in memory
Default	**memory**	
Protected	yes	

Purpose	Specifies where on disk the episodic store will be saved	
Parameter	**path**	
Values	**<empty>**	Soar-EpMem will create a temporary database file on disk during execution (and delete it after use)
	<valid path>	Soar-EpMem will use the specified path for its database file on disk - if the file doesn't exist, it will be created
Default	**<empty>**	
Protected	yes	

Purpose	Specifies the number of episodes between committing episodic store changes to disk.
Parameter	**commit**
Values	Integer, >=**1**
Default	**1**
Protected	no

5.2.4. Retrieval

Purpose	Specifies the degree to which cardinality and WMA are weighted in query processing to calculate a candidate match score	
Parameter	**balance**	
Values	Numeric, **[0,1]**	
Default	**0.5**	
Protected	no	

Purpose	Specifies whether structural matching is to be run after surface match	
Parameter	**graph-match**	
Values	**off**	Cue-based episode retrieval is made based upon surface match alone
	on	Structure match supplements surface match
Default	**on**	
Protected	no	

5.2.5. Performance

Purpose	Specifies the maximum amount of memory used for SQLite cache	
Parameter	**cache**	
Values	**large**	100MB
	medium	20MB
	small	5MB
Default	**large**	
Protected	yes	

Purpose	Specifies architectural focus in data safety vs. epmem performance	
Parameter	**optimization**	
Values	**performance**	Data store on disk is left vulnerable to corruption the case of application/OS/hardware malfunction
	safety	Data store on disk is guaranteed to be consistent
Default	**performance**	
Protected	yes	

Purpose	Declares the level of Soar-EpMem timers that are enabled (like watch levels)	
Parameter	**timers**	
Values	**off**	Timers are disabled
	one	Only total Soar-EpMem time is recorded
	two	High-level timers are enabled (epmem_*)
	three	Low-level operations are timed
Default	**off**	
Protected	no	

5.3. Full Parameter Configuration

Entering simply the **epmem** command (with no switches) will return full parameter configuration information. For example, assuming default configuration, the result of executing **epmem** is as follows:

```
>epmem

EpMem learning: off

Encoding
--------
phase: output
trigger: output
force: off
exclusions: epmem, smem

Storage
-------
database: memory
commit: 1
path:

Retrieval
---------
balance: 0.5
graph-match: on

Performance
-----------
cache: large
optimization: performance
timers: off
```

5.4. Parameter Behavior

Upon attempting to set a Soar-EpMem parameter, the new value is validated. If the value is found to be invalid, the system will use the previous value.

The set of parameters listed above that have a "yes" in the *Protected* field cannot be changed once the Soar-EpMem system has been "initialized." The Soar-EpMem system initializes during recording of the first episode since starting Soar or issuing the **close** switch of the **epmem** command (see Section 10.1.1).

6. Soar-EpMem Statistics

Feedback from the Soar-EpMem system is retrieved using the **stats** switch of the **epmem** command:

```
epmem [-S|--stats] <statistic>
```

If a **statistic** argument is provided, the command returns the value of a specific statistic. The valid statistic arguments are listed below.

Statistic **time**
Description Current episode id (starts at 1 upon store initialization, increases)
Label Time

Statistic **mem_usage**
Description Current SQLite memory usage in bytes
Label Memory Usage

Statistic **mem_high**
Description Greatest SQLite memory usage in bytes since last database initialization
Label Memory Highwater

Statistic **ncb_wmes**
Description Number of WMEs added to Working Memory in the last reconstruction
Label Last Retrieval WMEs

Statistic **qry-pos**
Description Number of leaf WMEs in the **query** cue of the last CB query
Label Last Query Positive

Statistic **qry-neg**
Description Number of leaf WMEs in the **neg-query** cue of the last CB query
Label Last Query Negative

Statistic **qry-ret**
Description Temporal ID of the last retrieved episode
Label Last Query Retrieved

Statistic **qry-card**
Description Cardinality of the last CB query retrieval
Label Last Query Cardinality

Statistic	**qry-lits**
Description	Number of literals in the DNF graph of the last CB query
Label	Last Query Literals

The following additional statistics may be requested for debugging performance issues in the Relational Interval Tree: **rit-offset-1**, **rit-left-root-1**, **rit-right-root-1**, **rit-min-step-1, rit-offset-2**, **rit-left-root-2**, **rit-right-root-2**, **rit-min-step-2**.

Agents can retrieve specific statistics in rule actions using the **cmd** function.

Entering the **epmem --stats** command with no statistic, or an invalid statistic, will return all statistics. A sample execution may look as follows:

```
>epmem --stats
Time: 0
Memory Usage: 0
Memory Highwater: 0
Last Retrieval WMEs: 0
Last Query Positive: 0
Last Query Negative: 0
Last Query Retrieved: 0
Last Query Cardinality: 0
Last Query Literals: 0
```

7. Soar-EpMem Timers

Time spent on Soar-EpMem operations is retrieved using the **timers** switch of the **epmem** command:

```
epmem [-t|--timers] <timer>
```

If a **timer** argument is provided, the command returns the value of a specific timer. The valid statistic arguments are listed below (with their associated level, respecting the **timers** parameter).

Timer	**_total**
Description	Total time spent by Soar-EpMem
Level	one

Timer	**epmem_api**
Description	Time spent validating agent commands
Level	two

Timer	**epmem_hash**
Description	Time spent hashing symbols
Level	two

Timer	**epmem_init**
Description	Time spent initializing the episodic store
Level	two

Timer	**epmem_ncb_retrieval**
Description	Time spent reconstructing episodes
Level	two

Timer	**epmem_next**
Description	Time spent determining the next episode
Level	two

Timer	**epmem_prev**
Description	Time spent determining the previous episode
Level	two

Timer **epmem_query**
Description Time spent performing a cue-based query
Level two

Timer **epmem_storage**
Description Time spent storing new episodes
Level two

Timer **epmem_trigger**
Description Time spent determining if a new episode is to be stored
Level two

Timer **ncb_edge**
Description Time spent collecting edges during reconstruction
Level three

Timer **ncb_edge_rit**
Description Time spent collecting edges from the relational interval tree
Level three

Timer **ncb_node**
Description Time spent collecting nodes during reconstruction
Level three

Timer **ncb_node_rit**
Description Time spent collecting nodes from the relational interval tree
Level three

Timer **query_dnf**
Description Time spent constructing the DNF graph
Level three

Timer **query_graph_match**
Description Time spent performing graph match
Level three

Timer **query_neg_end_ep**
Description Time spent in interval search: negative cue, end point, ranges
Level three

Timer **query_neg_end_now**
Description Time spent in interval search: negative cue, end point, now
Level three

Timer **query_neg_end_point**
Description Time spent in interval search: negative cue, end point, points
Level three

Timer **query_neg_start_ep**
Description Time spent in interval search: negative cue, start point, ranges
Level three

Timer **query_neg_start_now**
Description Time spent in interval search: negative cue, start point, now
Level three

Timer **query_neg_start_point**
Description Time spent in interval search: negative cue, start point, points
Level three

Timer **query_pos_end_ep**
Description Time spent in interval search: positive cue, end point, ranges
Level three

Timer **query_pos_end_now**
Description Time spent in interval search: positive cue, end point, now
Level three

Timer **query_pos_end_point**
Description Time spent in interval search: positive cue, end point, points
Level three

Timer **query_pos_start_ep**
Description Time spent in interval search: positive cue, start point, ranges
Level three

Timer	**query_pos_start_now**
Description	Time spent in interval search: positive cue, start point, now
Level	three

Timer	**query_pos_start_point**
Description	Time spent in interval search: positive cue, start point, points
Level	three

Agents can retrieve specific timer values in rule actions using the **cmd** function. Timer values are re-initialized at the same time points as Soar timers.

Entering the **epmem --timers** command with no timer will return all timers. A sample execution may look as follows:

```
>epmem --timers
_total: 0
epmem_api: 0
epmem_hash: 0
epmem_init: 0
epmem_ncb_retrieval: 0
epmem_next: 0
epmem_prev: 0
epmem_query: 0
epmem_storage: 0
epmem_trigger: 0
ncb_edge: 0
ncb_edge_rit: 0
ncb_node: 0
ncb_node_rit: 0
query_dnf: 0
query_graph_match: 0
query_neg_end_ep: 0
query_neg_end_now: 0
query_neg_end_point: 0
query_neg_start_ep: 0
query_neg_start_now: 0
query_neg_start_point: 0
query_pos_end_ep: 0
query_pos_end_now: 0
query_pos_end_point: 0
query_pos_start_ep: 0
query_pos_start_now: 0
query_pos_start_point: 0
```

8. Trace Information

To view Soar-EpMem debugging information, use the following watch switch:

```
watch [-e|--epmem]
```

This function is not enabled by default or through any watch level. At present, this watch level generates a message when an episode is recorded as well as interval search data during every evaluated candidate episode of a cue-based query.

9. Soar-EpMem Performance

This section discusses performance concerns regarding Soar-EpMem use.

9.1. Sources of Performance Degradation

There are currently two sources of "unbounded" computation: graph matching and cue-based queries. Graph matching is combinatorial in the worst case. Thus, if an episode presents a perfect surface match, but imperfect structural match (i.e. there is no way to unify the cue with the candidate episode), there is the potential for exhaustive search. Each identifier in the cue can be assigned one of any historically consistent identifiers (with respect to the sequence of attributes that leads to the identifier from the root), termed a literal. If the identifier is a multi-valued attribute, there will be more than one candidate literals and this situation can lead to a very expensive search process. Currently there are no heuristics in place to attempt to combat the expensive backtracking. Worst-case performance will be combinatorial in the total number of literals for each cue identifier (with respect to cue-structure).

The cue-based query algorithm begins with the most recent candidate episode and will stop search as soon as a match is found (since this episode must be the most recent). Given this procedure, it is trivial to create a two-WME cue that forces a linear search of the episodic store. Soar-EpMem combats linear scan by only searching candidate episodes, i.e. only those that contain a change in at least one of the cue WMEs. However, a cue that has no match and contains WMEs relevant to all episodes will force inspection of all episodes. Thus, worst-case performance will be linear in the number of episodes.

9.2. Performance Tweaking

When using a database stored to disk, several parameters become crucial to performance. The first is **commit** which controls the number of episodes that occur between writes to disk. If the total number of episodes (or a range) is known ahead of time, setting this value to a greater number will result in greatest performance (due to decreased I/O).

The next parameter is **cache**. Greater settings afford SQLite greater amounts of memory in which to store B-Tree nodes, thus reducing disk I/O for searches. This memory is not pre-allocated, so short/small runs will not automatically make use of this space. Some situations may benefit from smaller cache allocation, to reduce memory allocation calls.

The next parameter is **optimization**. The **safety** parameter setting will use SQLite default settings. If data integrity is of importance, this setting is ideal. The **performance** setting will make use of lesser data consistency guarantees for significantly greater performance. First, writes are no longer synchronous with the OS (*synchronous* pragma), thus Soar-EpMem won't wait for writes to complete before continuing execution. Second, transaction journaling is turned off (*journal_mode* pragma), thus groups of modifications to the episodic store are not atomic (and thus interruptions due to application/os/hardware failure could lead to inconsistent database state). Finally, upon initialization, Soar-EpMem maintains an continuous exclusive lock to the database (*locking_mode* pragma), thus other

applications/agents cannot make simultaneous read/write calls to the database (thereby reducing the need for potentially expensive system calls to secure/release file locks).

Finally, timers are currently very expensive in Soar. The Soar-EpMem timers use Soar timer code. Thus, these should be enabled with caution and understanding of their limitations. First, they *will* affect performance, depending on the level (set via the **timers** parameter). A level of **three**, for instance, times every step of every query in the interval search. Furthermore, because these iterations are relatively cheap (typically a single step in the linked-list of a b-tree), timer values are typically unreliable (depending upon the system, resolution is 1 microsecond more).

10. Soar-EpMem Programmer Reference

The following tables list basic information about each of the Soar-EpMem related commands. It is not intended to substitute for this document, but a quick reference for commonly used commands and options.

10.1. Soar-EpMem

10.1.1. Useful Commands

Command	Description
`epmem`	Summary table of parameter settings
`epmem [-g|--get] <parameter>`	Retrieve a Soar-EpMem parameter value
`epmem [-s|--set] <parameter> <value>`	Set a Soar-EpMem parameter value
`epmem [-S|--stats] <statistic>`	Access Soar-EpMem statistics
`epmem [-t|--timers] <timer>`	Access to Soar-EpMem timers
`epmem [-c|--close]`	Close the current Soar-EpMem database
`watch [-e|--epmem]`	Soar-EpMem debugging trace

10.1.2. Parameters

Parameters noted with a * are *protected*.

General

Parameter Name	Acceptable Values	Default
`learning`	`on` `off`	`on`

Encoding

Parameter Name	Acceptable Values	Default
`phase`	`output` `selection`	`output`

`trigger`	`dc`
	`output`
	`none`
		`output`

`force`	`ignore`
	`remember`
	`off`
		`off`

`exclusions`	`<any string>`	`epmem, smem`

Storage

Parameter Name	Acceptable Values	Default
`database*`	`file` `memory`	`file`
`commit*`	`Integer, >= 1`	`1`
`path*`	`<empty>` `<system path>`	`<empty>`

Retrieval

Parameter Name	Acceptable Values	Default
balance	[0,1]	0.5
graph-match	off on	on

Performance

Parameter Name	Acceptable Values	Default
cache*	large medium small	large
optimization	performance safety	performance
timers	off one two three	off

10.1.3. Retrieval Agent Commands

Absolute NCB Retrieval

```
state.epmem.command.retrieve temporal-id
```

Relative NCB Retrieval

```
state.epmem.command.next
state.epmem.command.previous
```

CB Retrieval

```
state.epmem.command.query <cue>
```

and (optionally)

```
state.epmem.command.neg-query <cue>
```

CB Retrieval Optional Modifiers

```
state.epmem.command.before temporal-id
state.epmem.command.after temporal-id

state.epmem.command.prohibit temporal-id
```

10.1.4. Retrieval Agent Meta-Data

```
state.epmem.result

      ^retrieved <episode>
      ^status << bad-cmd >>
      ^<< success failure >> <query> <neg-query>
      ^match-score double
      ^cue-size integer
      ^normalized-match-score double
      ^match-cardinality integer
      ^memory-id temporal-id
      ^present-id temporal-id
      ^graph-match << 0 1 >>
      ^mapping
            ^node
                  ^cue <id-in-cue>
                  ^retrieved <id-in-retrieval>
```

10.2. Working Memory Activation

10.2.1. Useful Commands

Command	Description
`wma`	Summary table of parameter settings
`wma [-g|--get] <parameter>`	Retrieve a WMA parameter value
`wma [-s|--set] <parameter> <value>`	Set a WMA parameter value

10.2.2. Parameters

Parameter Name	Acceptable Values	Default
`activation`	`on` `off`	`off`
`decay-rate`	`[0,1]`	`0.8`
`forgetting`	`off` `on`	`off`