Adaptive Event Retrieval for Episodic Memory

Colm Flanagan C.FLANAGAN@UNSW.EDU.AU
Claude Sammut C.SAMMUT@UNSW.EDU.AU
School of Computer Science Engineering, University of New South Wales

Abstract

In psychology, it is largely agreed that declarative memory includes two components: episodic and semantic memory. Episodic memory is a collection of an agent's experienced events that have temporally and spatially related information associated with them. Semantic memory is a collection of facts and concepts that may not depend on a particular time and place. Being able to retrieve an episode given a new observation can help an agent reason about its current situation. Thus, finding the best match between the current state and an episode in memory is a necessary function of an episodic memory system. We present a novel approach, based on Ripple-Down Rules (RDR), for matching and retrieving events stored in episodic memory. We evaluate our approach on a set of unique simulated events that may be experienced by a domestic robot on a daily basis. Our results show that on average, with only two observations of a given event, the system can learn a set of matching rules to accurately recall events of that type at a later point in time.

1. Introduction

The concept of episodic memory was first presented by Tulving (1972) when he distinguished between episodic and semantic memories. He conjectured that episodic memories are time lagged and have a context associated with them. All episodic memories have three fundamental components that they share. They all have a time, a location and for them to be created, something had to happen. For the purpose of this paper we will refer to that something as an action. To use the information in episodic memory it is essential to be able to retrieve episodes that match new observations. We use *cue* to refer to a stimulus that invokes a memory. Previous work on episodic memory in cognitive architectures include SOAR (Laird et al. (1987); Derbinsky & Laird (2009); Nuxoll & Laird (2012)), EPIROME (Jockel et al. (2007)) and work by Chang and Tan (2017).

All of these systems use a uniform procedure to match a retrieval cue to stored events. This can lead to incorrect retrievals because each type of event is unique and information that is relevant to one event may not be relevant to another. Relying on a single matching procedure for all types of events means either constraining events to ensure that each type contains the same kind of information or sacrificing accuracy in the retrieval of an event.

We address these issues by introducing a novel approach for retrieving past events, or episodes. The system incrementally learns matching rules that are customised for each type of episode, using Ripple Down Rules (RDR) (Compton & Jansen (1990); Compton et al. (1992); Gaines & Compton (1995)). An RDR is a rule structure that can be updated incrementally as new information is presented. RDRs enable the system to learn different matching conditions for different event types,

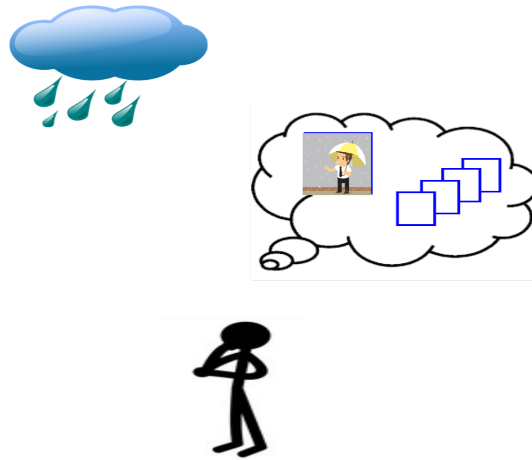

Figure 1: A person sees rain which correctly recalls an experience of getting caught outside.

depending on what is relevant to that event. For example, in one event, a glass falls off a table and breaks and a robot must clean it up. In another event, a friend comes to visit on a Monday evening and the robot makes a cup of tea. Both are events that are likely to occur in a typical home and both events have mostly the same types of information present, a time, a location, an action, etc. However, as should become clear on a second observation of a glass falling, the time and location of that event is irrelevant. A glass can fall off any table, anywhere, at anytime and the consequences are the same, whereas a friend coming to visit on a Monday evening may be a ritual and the time and location of that event are relevant to the recall. If a friend were to visit on a Friday evening, it may trigger a different memory, as the context is different and you no longer wish for the robot to make a cup of tea, but get a beer from the fridge.

Case Based Reasoning (CBR) research has explored the issue of episodic recall extensively. However, there are several problems that previous approaches fail to address. CBR methods, such as Homem et al. (2020) usually rely on a qualitative similarity measure, which is often a simple calculation based on the quantity of common information between two cases. Smyth & Keane (1994) note that instead of simply retrieving the most similar cases a system should retrieve the case that is most easily adaptable to achieve the goal of the current task. They use a look ahead method that models the cost of adapting a particular case to solve the goal before retrieving it. Modelling the cost of adapting a case in memory to the current observation is important, and in a domain where the actions that an agent needs to execute are finite and known, such as a warehouse, this works well. However, in a domestic environment, a robot may have to perform the same kinds of tasks in a variety of contexts for different reasons. Therefore, modelling the cost of adaptation is very difficult and often impossible. This is largely due to the fact that different kinds of events have different information that is relevant. Therefore, an adaptive recall policy is needed. Learning such a policy using an RDR provides this capability and because RDRs are trained incrementally, we do not need to collect a large data set before being able to use the policy.

An RDR update occurs when a new case is not correctly classified by the existing set of rules. This new case becomes a “cornerstone case”, which can be used to explain why a new rule was

created. People often find an explanation clearer when presented with examples. In this case, the explanation describes the differences between cases that caused the system to produce different conclusions. Explanation is important in a domestic robot as people tend to trust a system more if it can explain its decisions (Korpan & Epstein (2018)).

We show that an adaptive matching procedure achieves highly accurate recollection of relevant episodes, with minimal training. We also show how RDRs address many of the issues that we have identified with current episodic recall techniques, such as noisy retrieval cues or nearest neighbour techniques matching incorrect events. We evaluate our approach on ten different types of simulated events.

2. Related Work

Episodic memory in AI often takes its inspiration from cognitive psychology. Episodic memory was first described by Tulving (1972) when he distinguished between episodic and semantic memory. This definition was further extended, in 1983 (Tulving (1983)), when Tulving conjectured that declarative memory was composed of both episodic and semantic memories. Episodic memories are collections of events experienced by a person and are highly contextualised. This means that each memory has a time, a location and an action associated with it.

Wheeler & Ploran (2009) propose a similar definition, however they provide more scientific evidence for the theory by presenting studies showing how people can be episodically, but not semantically impaired and how people who are episodically impaired are significantly less capable at performing cognitive functions.

In AI, much of the work to date using episodic memory has focused on improving an agent's performance on a given task. Liu et al. (2017a,b) propose using episodic memory to improve robot planning under uncertainty. The idea is that the use of previously observed events can reduce aliasing in perceptual data. Botvinick et al. (2019) look at how episodic memory can be used to improve traditional reinforcement learning techniques by showing how it can be used for more efficient sampling. Lin et al. (2018) also address the issue of sample inefficiency in reinforcement learning using episodic memory. Here they use episodic memory to supervise an agent during training, the idea being that episodic control can remember experiences during training that returned a high reward and then replay these experiences during evaluation.

The use of episodic memory has been quite prevalent in dynamic memory networks for question answering, as addressed by Xiong et al. (2016) and Kumar et al. (2016). The episodic memory module passes a memory from one hidden state of a network to the next. The episodic memory system can make multiple passes over the input, gaining a better understanding of the question.

Much of the above work focuses on using episodic memory for neural networks or reinforcement learning and is used in a very different context to how we are using episodic memory in this paper.

In cognitive robotics, Derbinsky & Laird (2009) describe an episodic memory system in the SOAR cognitive architecture (Laird et al. (1987)) and discuss how it can be efficiently implemented. To retrieve events, SOAR finds the nearest neighbour to a retrieval stimulus, referred to as a cue. Nearest neighbour methods are vulnerable to retrieving incorrect episodes, especially when two episodes are semantically similar. While SOAR addresses many of the problems that are associated

with nearest neighbour techniques, for example by providing agents with meta-data to detect sub-perfect matches, it is still not guaranteed to retrieve the correct match. The use of RDRs ensures that if the correct event is in the collection of episodic memories, it will be retrieved and nothing will be retrieved if that event is not present. In the latter case, a new episode is created and stored.

In SOAR, an event is created every time an agent performs an action. While this is a valid approach, events may occur independently of the agent's actions and may occur even when the agent is not present.

Nuxoll & Laird (2012) show how episodic memory can enhance an agent's cognitive capabilities. They use an activation function to determine whether or not an episode should be kept or thrown away. This is an essential functionality, as it prevents an episodic database from getting too large. While we do not address this issue in this paper we do acknowledge the extent to which is can improve performance in retrieving episodic memories and we note that lacking it is a limitation of this work.

Similar to Derbinksy and Laird, they store an episode every time the agent performs an action in the world. This approach is applicable when the world that the agent inhabits is a game and its only job is to play the game. Our agent is a domestic robot, operating in a home environment, with other human and robotic agents. The robot's world model consists of a database of predicates, and events are detected as a change of state, i.e. a change in the database. Thus, a new episode may be recorded independently of an agent's actions, even if the agent did not explicitly observe the event at the time that it occurred, but detected a change later. This could occur if, say the robot left a room and returned to find it different to when it left, inferring that some exogenous action occurred to cause the change. Like SOAR, we store all events generated by an action, but this need not always be necessary. For example, as people we rarely remember subconscious actions that have become automatised through practice.

EPIROME (Jockel et al. (2007)) is a framework for investigating high-level episodic robot memory, in which episodes are divided into perceptual, command and intentional types. While these apply to robots operating in a real world environment, they are "hardcoded" into their framework, whereas our system can handle new event types, because we learn the matching procedures.

Rosenthal & Veloso (2012) describes a system that incorporates something similar to episodic memory. Their robot requests people to help it around an office in a way that limits interruptions. For this, they need a knowledge base of who will be in a particular office at a particular time and who is willing to help. The knowledge base is very episodic in its structure, even though the paper does not explicitly mention episodic memory.

Case Based Reasoning (CBR) attempts to solve problems based on solutions to similar problems that have been previously seen. Kolodner (2014) and Sharma & Sharma (2020) review recent work on CBR. In retrieving cases, many CBR methods use a two-phased approach. This typically involves a simple, inexpensive retrieval to select some candidate matches, followed by a more fine-grained method for ranking the matches. Kendall-Morwick & Leake (2014) compare popular two-phased retrieval methods and note the design considerations necessary for efficient and effective recall. For example, one of the common methods for efficient recall is to have a fixed retrieval window in the phase-1 retrieval so that the phase-2 retrieval time is capped. They conclude that different domains have different demands for retrieval strategies as would be expected.

This paper mainly focuses on accurate event retrieval. Aside from SOAR, or its variants, this has received limited attention in the context of cognitive robots. Lim et al. (2011) use a *compound cue* to retrieve events, which is problematic because it relies on several matches of attributes of two events. In a large enough collection of events, this will almost certainly lead to multiple matches. Shen et al. (2013) propose a method that can retrieve events even with partial or noisy matching cues. This method has been subsequently out performed by Chang & Tan (2017), who propose a method that is based on a generalised self organising neural network known as Adaptive Resonance Theory. The problem however, is that while it performs well on noisy or partial matching cues, there will still be incorrect retrievals.

The use of RDRs removes the problem of noisy or partially matched cues, as will become clear later. We argue that information contained in an event is context dependent and that one type of information that is relevant to one kind of event is irrelevant to another. To attempt to fit one retrieval method to all types of events does not work in a complex environment, such as that of a domestic robot. Therefore, we present this novel approach for retrieving events stored in episodic memory using ripple down rules. We also present our method for representing events. After training a ripple down rule on a given event type just once, we can achieve highly accurate recall.

3. Event Representation

Many of the above mentioned systems represent events as actions of an agent in a particular situation. However, events may also be due to exogenous actions and, may not be observed by the agent at the time that the event occurred. Therefore, our representation of an event is based on a qualitative state change in the world model.

Actions are defined using the PDDL task planning language. The current state of the world is represented by a set of predicates organised within a topological map. Recall that our application domain is that of a domestic robot. A map of the environment is a fundamental component in its planning, reasoning and communication with other agents. An initial geometric map is generated by the robot's simultaneous localisation and mapping (SLAM) system. This is converted into a topological map that includes labels of regions, e.g. the rooms of a house and objects in them. Predicates in the world model include spatial relations between those objects and regions. For example, the agent's position may be represented by an $(agent_at\ ?wp)$ predicate where wp represents the current way point such as kitchen or dining room. All actions have preconditions and effects. If, at one point in time, the preconditions of an action are satisfied and if at some point in the future the effects of an action have been achieved, then it is assumed that that action must have occurred.

Let a predicate, P, represent a belief in the world. Let, A, be an action and let the world at time, t, be represented by Γ_t, where

$$\Gamma_t =< P_1 \wedge, \ldots, \wedge P_n >$$

The following is the PDDL representation of a robot moving from one location to another.

$$
\begin{aligned}
&(: action \quad goto_waypoint \\
&\quad : parameters(?r - robot \ \ ?from \ \ ?to - waypoint) \\
&\quad : precondition((robot_at(?r, \ \ ?from))) \\
&\quad : effect(and \\
&\qquad (not(robot_at(?r, \ \ ?from))) \\
&\qquad (robot_at(?r, \ \ ?to)) \\
&\quad) \\
&)
\end{aligned}
$$

An alternative way of describing any action uses Allen's temporal logic (1990). Let, X, be a state that occurs at time *t*. Y, is a successor state at time *t+1* and, A, is the associated action:

$$\forall X, Y : X \neq Y \wedge \{X \succ Y\} \implies A$$

Or in terms of the world model:

$$(\Gamma_t \neq \Gamma_{t^*}) \wedge (\Gamma_t \succ \Gamma_{t^*}) \implies A$$

This tells us that action A can be deduced if we see state X, followed by state Y provided states X and Y are not equal. Making the closed world assumption, the world model tells us everything that is currently believed to be true in the robot's environment. If one of those beliefs changes, then something must have happened. In addition to the action definition, the episodic memory also stores the matching rules, so we need a more complex structure than just the PDDL action model. This will be described in more detail below, where we explain how episodes are represented as frames (Minsky (1975); McGill et al. (2008)), in a graph database.

Before we go into these in detail, we address some concerns regarding when we choose to remember an episode. As noted above, the system creates an event when there is a change in the world model. However, this can lead to every change triggering a new episode regardless of how inconsequential that change is. There are several ways that this can be dealt with. We briefly describe these methods below, but a detailed discussion is outside the scope of this paper.

The first method is to have a pre-defined domain where all actions that could possibly happen are already defined. This is how the world must be defined for most planners (e.g. Fast Forward (FF) Hoffmann (2001)). This can work, however, it also means that new actions cannot be learnt and must be explicitly added to the domain. The other method is to ignore some events as remembering them has typically proven not to be useful. This involves using some form of attention mechanism and a threshold value, as described by Nuxoll & Laird (2012). If an event does not excite the attention mechanism above the threshold value, then it is not stored.

The latter method is the most practical for a domestic robot. For example, in the case of a robot getting a person a glass of water, the goal is for the person to have the water. However, in executing the task, several other actions may also be performed, such as going to the tap, picking up a glass,

filling the glass etc. To prevent biasing the robot with domain specific knowledge, the robot would need to initially consider each of these actions as a separate event. However, over time, the robot should learn that these additional actions are only intermediate to achieving the final goal and are, in themselves, not episodes.

As a consequence, we define an action hierarchy in which a *compound* action may consist of a sequence of intermediate, lower level actions. We represent the compound action as $\mathcal{A}$ and the intermediate actions as α_n, where $n \in [0, \ldots, t]$, assuming there are *t* intermediate actions in the sequence:

$$\mathcal{A} \leftarrow \alpha_t \succ \alpha_{t-1} \succ \cdots \succ \alpha_1 \succ \alpha_0$$

For ease of explanation we will only use the term *action* to refer to the action that led directly to the goal of the episode and not any of the intermediate actions, if any. Thus, an action in an episode is treated as a single action. As in our previous example of getting a drink for a person, this action can still be described by a PDDL action model, the precondition of which is that the person does not have a drink and the effect is that the person does. It is remembered because it has achieved a goal that was given to it by the human.

Each episode is represented as an instance of a generic frame called *episode* which has slots that contain data that are relevant to the episode. Previous systems, such as Tecuci & Porter (2007), represent generic events as a triple, consisting of a **context, contents** and **outcome**, where *context* is the setting that an episode took place in, *contents* are the set of events that make up an episode and the *outcome* is the episode's effect.

Our representation is similar, except that the *contents* are replaced by the compound action, mentioned above. Thus, each episode must have an *action* slot. Each episode also has a *time* and a *location* slot. Episodes can be linked to other episodes through a *connected event* slot. A connected event is another event or episode with a temporal relation to the event that it is connected to. These temporal relations are expressed as one of Allen's temporal intervals (Allen (1990)).

To summarise a generic episode has the following slots:

- *action* - an instance of generic frame *action*.
- *time* - an instance of the generic frame, *time*.
- *location* - an instance of the generic frame location.
- *connected events* - other events that are connected to this event.
- *other information* - any other data that are present at the time of the event

The main difference between our generic episode representation and that of Tecuci and Porter is that we refer to the contents as being a single action rather than a sequence of actions. Initially, each action is treated as a separate event, and that is how we describe events below. To construct a compound action, the system needs a method of identifying commonly occurring action sequences, but that is outside the scope of this paper.

An action frame consists of a *name*, *parameters*, *preconditions* and *effects*. Preconditions and effects are predicates that are also frames and contain a *name*, *attributes*, *arity* and whether or not the predicate is negated. Much of this stems from the PDDL representation language. A connected event contains another episode and the temporal relation to the connected event. The temporal relation is one of Allen's temporal intervals. If event B were to happen after event A and the two were connected then event B would be the connected event with temporal relation *succeeds*. That is:

$$connected(A, B) \wedge succeeds(B, A)$$

The *other information* slot is unspecified. It contains whatever data presents to the agent at the time the episode was created.

Every generic frame has its own ripple down rule (RDR). The RDR defines what constitutes a valid match for that type of frame. In the following section, we explain the operation of RDRs and why they are well-suited to learning matching procedures. We then explain how they are used in our system.

3.1 Ripple Down Rules

Ripple Down Rules were introduced by Compton & Jansen (1990) as a knowledge acquisition method for knowledge-based systems. RDRs are learned incrementally, guided by a trainer. An initial rule serves as the default:

if $true$ **then** no_match

If a new case causes a rule to fire when it should not, a new exception rule is added with the correct conclusion. That is, the initial rule is *specialised*. Suppose we are training the RDR to learn to recognise fruit. The default is *no match*. If the next training example is a banana, we may add an exception rule:

if $true$ **then** no_match **except**
  if $colour = yellow$ **then** $banana$

Now suppose, an apple is presented. The 'banana' rule fails, so the RDR must be *generalised* by adding an alternative rule:

if $true$ **then** no_match **except**
  if $colour = yellow$ **then** $banana$
  else if $colour = red$ **then** $apple$

The new condition for the new rule is obtained from the differences between an old case for which the rule was correct and the new case that failed. The trainer, which is a person that indicates when a rule has fired incorrectly, has the option of discarding conditions if they are judged to be irrelevant. For example, if the banana is large and the apple small, the system may include size as an attribute to test. Without trainer supervision, further examples will be needed to eliminate size from

consideration. Thus, the trainer's knowledge can help to reduce the number of training examples needed and, consequently, reduce the complexity of the RDR.

Because RDRs store the "cornerstone cases" that caused a new rule to be added, they can provide explanations for their decisions in the form of the conditions satisfied to reach a conclusion and also by presenting example cases.

RDR learning is incremental, which suits our application well. The setting for our experiments is a domestic robot that interacts with the human occupants of a home. We want the robot to be able to retrieve relevant past episodes to assist it in its reasoning about a current situation. If an irrelevant episode is retrieved, RDRs provide a simple method of correcting the matching procedure so that a better match is done in a similar, future, situation.

We already noted that each generic frame has its own unique RDR defining the conditions under which instances of that generic frame match other frames. We also noted that each episode has attributes that are, themselves represented by generic frames: *action, time, location, predicate, connected_event* and *episode*. They too have RDRs that are trained to match them, as explained in Section 3.2.

When comparing an observation of an episode to a previously stored episode, as an example, a rule might state that, if the data in the action slots are the same in both instances then the episodes match. The data in the action slots are themselves instances of generic action frames which have their own RDRs expressing the conditions under which the actions match. We determine if the action slots match by evaluating the action frame RDR, which checks, for example, if the precondition and the effects are the same. This will be explained in more detail in Section 4.

Earlier, we gave two examples events: a glass breaking and a friend visiting. For the glass breaking, the simple rule of checking if the actions match is sufficient. However, that is not the case for when a friend comes to visit as there is other information that is relevant to the event such as what the time is and where it is. As will be explained below, this is where the ability for generic frames to inherit from other generic frames comes in and it is why RDRs are so powerful for event recollection in the context of episodic memory.

3.2 Creating Events

Suppose a new event occurs as the result of some action. The action is represented by an instance of the generic action frame, the time of the event is an instance of the generic time frame etc. Each of these generic frames contains a default RDR that may not be applicable to every event. The reason for defining generic rules that apply to all episode frames, time frames, etc, is so that there is some policy by which we can recall event types that have not had a sufficient number of observations for a unique policy to be trained for that event type. So the first step in constructing a new event frame is to identify the type of generic frame that all of the relevant observed data belong to so that an initial episode instance can be created. As noted, this is an instance of an episode in its most generic sense. We then create a new event type so that an individual retrieval policy can be defined for that event type. This is done for every embedded frame in the episode.

For example, in picking up a cup, there are several individual pieces of information that construct the event, including the action (picking it up), the time, the location, etc. Looking just at the action frame, it inherits from the generic *action* frame. The generic action frame contains the default

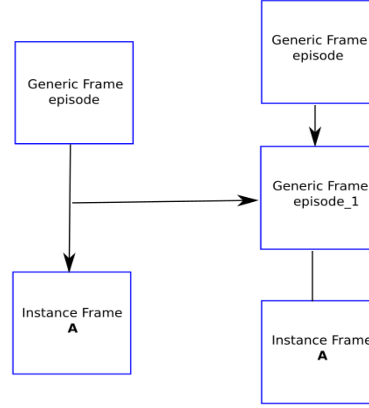

Figure 2: Creating a new event type. Initially frame **A** is an instance frame of type *episode*. After creating a sub-class of this event however, frame **A** is instead an instance of generic frame *episode_1* which inherits from generic frame *episode*. This process is executed for every instance frame in the episode

RDR for all actions, but it may not be applicable to the current action. Empirically we deduced that the best default RDR for the generic action is as follows:

if $true$ **then** no_match **except**
if $precondition = 1 \land effect = 1$ **then** $match$

As noted this may not be applicable to all actions and so the system creates a new subclass of the original *action* frame that initially inherits the default RDR, but then is overridden as the matching procedure for this new type of event is refined. One can think of this process as learning to refine a general concept of an action by constructing specialisations of it that are customised for subsets of different types of events. This process is executed for every data frame that is used in the representation of an episode including the episode frame itself. See figure 2 which shows the process for creating a new event type. The process can be better summarised in algorithm 1. Let the predicate *in(X, Y)* to mean that some frame Y inherits from some generic frame X.

The mechanism by which the RDRs for new generic frames are trained is detailed in Section 4. When a new frame has been fully constructed, it is then written to a database. We use the Mongo NoSQL database to provide a persistent store for the robot.

Algorithm 1 Creating a New Event Type

$$I_e = Instance\ frame\ of\ episode$$
$$G_e = Generic\ frame\ of\ episode$$

1: $in(G_e, I_e) \leftarrow in(G_e, G_{new}) \land in(G_{new}, I_e)$
2: **for** $instance\ I_i\ in\ I_e$ **do**
3: $\quad in(G_i, I_i) \leftarrow in(G_i, G_{new_i}) \land in(G_{new_i}, I_i)$
4: **end for**

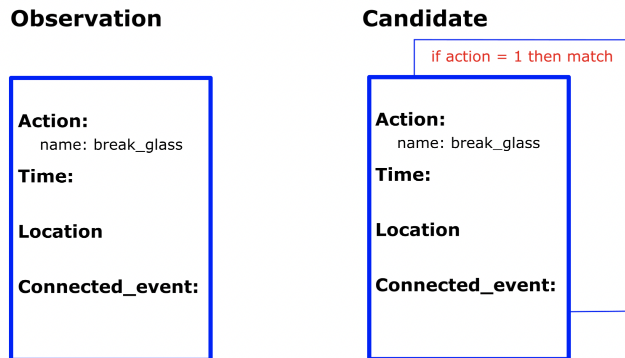

Figure 3: The frame on the left is the observation. The frame on the right is a candidate match. The frame on the right is an instance of a generic episode type that has at least a partially trained RDR. The shadow box represents that generic episode with the RDR in red. The RDR in this case states that if the value of the action of an event that has been observed matches the value of the action in this particular frame then the events do match. In this case that would be true and so the candidate event would be returned as a valid match to the observation

4. Event Retrieval

The database of episodes is split into two separate collections. The first is the episodic collection and the second is the semantic collection. The semantic collection stores non-episodic declarative concepts.

Recalling events is a two phase process and if necessary may be performed on both the semantic and episodic collections. We first perform a *shallow* query where we return events that have at least one or more slot values that match the observation. These events are called candidate matches. The second phase is to evaluate the RDRs for each of the candidate matches to establish if there is an actual match.

When a new event is observed, we first query the semantic collection. The purpose of querying the semantic collection initially is to see if it is possible to establish the event type that the observation is an instance of. The semantic collection stores generalised representations of event types that have been observed more than once. For example, a robot may have observed multiple instances of a person sitting down and turning on the television. So on a subsequent observation of seeing a person sit down the robot queries the semantic collection to see if the observation matches to any of the previously observed events and when a positive match is returned then the objective would be that it anticipates the persons behaviour and turns the television on.

On querying the semantic collection, each candidate match is an instance of a generic event frame that has at least a partially trained RDR specifying the conditions under which that event type matches to an observation. We iterate over each event in the list and evaluate the RDR for the current event.

To evaluate an RDR we compare the values in the slots that are common to the observation and the event we are currently analysing. Events typically have several slots such as time, location, etc. that can have quite complex structures. Therefore to clarify the process we look at the very simple event of breaking a glass, where we are concerned only with the action, see figure 3.

In the example of figure 3 it is clear that the two events do match as the values of the action slots match and are therefore instances of the same event type. However, it is initially not known if the observation is an instance of that event type, and evaluating the RDRs establishes to which

subclass of event the observation belongs. This example simplifies the representation of an episode. In practice, we would determine if the action slots matched by evaluating the RDR for the action frame.

If evaluating the rules for a particular event returns that the observation and the current event do not match, we move to the next candidate match and repeat. If none of the events in the semantic collection are valid matches to the observation, we repeat the entire process on the episodic collection.

If, at this stage we still have no matches then it is assumed that it is a new event that we have observed and the process to create a new event type, as outlined in section 3.2, is executed.

If there is a valid match with one of the event instances in the episodic collection then we are asked if we would like to train the RDR for this new generic event type. Recall that the episodes in the episodic collection are using, as a recall policy, the default RDRs inherited from the generic episode frame, action frame etc. These were deduced empirically and guarantee that a valid observation to an episode will match. However, as the rules are not yet specialised, it is also likely that several invalid observations will also match. Therefore, when we do have a valid match we must specialise the RDR.

Training the RDR involves iterating over each slot in the event instance and comparing the value of this slot to the value of the same slot in the observation. Depending on whether the two slot values match or not we ask if this is relevant to the events matching. If the value of a slot is itself an instance frame then the process is repeated for this instance frame and this is subsequently done recursively for every embedded instance frame within the event. The process is shown in algorithm 2.

On each training iteration performed we are adding at most one new rule to the RDR. The new rule is itself an RDR as RDRs are recursive structures. When adding a rule, we are adding either an exception to a rule already in the RDR or an alternative to a rule in the RDR. Training an RDR is required when a rule incorrectly fired. Either a rule that fired when it should not have, in which case we add an exception to that rule or a rule did not fire when it should have in which case we add an alternative.

Algorithm 2 trainRDR

Input: $InstanceFrames = (I, J)$

```
G_I = Generic from which I inherits
RDR r_new
RDR r_last →Last fired rule in RDR for G_I
whyFired → true if r_last fired when shouldn't
V_{S_x} → value of slot S in instance x
ask → do instances I and J match?
response = wait for response
updateConclusion(r_new, response)
for Slot S in I do
    ask ← is this slot relevant to whether the events match?
    if no then
        continue
    end if
    if V_{S_I} = type(I) then
        I ← V_{S_I}
        J ← V_{S_J}
        trainRDR(I, J)
    else
        updateConditions(r_new, V_{S_I}, V_{S_J})
    end if
end for
updateRule(r_new, r_last, whyFired)
```

Algorithm 3 updateConclusion

Input: $RDR\, rule, String\, response$

```
if response = yes then
    rule.conclusion ← match
else
    rule.conclusion ← no_match
end if
```

Algorithm 4 updateConditions

Input: $RDR\, rule, Slots = (S_1,\ S_2)$

```
Condition c = {slot_name, comparison_value}
if V_{S_1} = V_{S_2} then
    c.comparison_value = true
else
    c.comparison_value = false
end if
c.slot_name = name of S_1
rule.add(c)
```

Algorithm 5 updateRule

Input: $RDR = (new_rule,\ last_rule),\ bool\ whyFired$

```
if whyFired then
    last_rule.exception.add(new_rule)
else
    last_rule.alternative.add(new_rule)
end if
```

As noted by Kendall-Morwick & Leake (2014) one of the main factors that needs to be taken into consideration in event retrieval is the efficiency of the retrieval process. One of the more common approaches taken is to limit the number of candidate cases returned in the phase-1 query. We chose not to constrain the number of candidate matches in the first phase as we do not want to risk filtering out a potentially valid match. Our phase one query is performed on a NoSQL database which is capable of retrieval in logarithmic time and so is very efficient. Our second phase

is evaluated in linear time O(n). RDRs themselves are evaluated in constant time O(1) and we perform this evaluation n times depending on the number of candidate matches.

The matching algorithm could be improved by compiling the RDRs into a matching network, similar to a Rete network (Forgy (1982)). The network would allow the system to fire an RDR when certain data are present. This means that our retrieval algorithm could be much improved in time complexity as we would not need a linear traversal of each event, evaluating the RDRs one at a time. On a small database this would be acceptable however, as the database of episodes grows, a more efficient retrieval will be essential.

5. Evaluation and Results

Our evaluation tests two things: how quickly can an RDR be trained for each event type; and how successfully can instances of that event type be recalled. When referring to how quickly an RDR can be trained we are referring to the number of observations of an event required to train the RDR. We created ten types of events that may be experienced by a domestic robot. Some of the event types, while unique, are connected. For example, cleaning a broken glass is connected to and succeeds the event of the glass falling and breaking. To test how many observations of an event we need to create an RDR that is able to retrieve correct observations, we create a synthetic data set. We manually create events that already have correct RDRs defined for them and use these RDRs to generate new training events. The evaluation then requires the learning system to recreate the RDRs that generated the data.

For each type of event, we synthesise fifty random observations or instances of that event type that we know match the event, i.e. positive examples, and fifty random observations that we know do not match to the event, i.e. negative examples. We refer to the event from which we are synthesising data as the *base event*.

Figure 4 shows an example of how an instance frame is synthesised. An RDR is associated with each generic frame, specifying the matching rule for instances of that generic frame. Slots in a generic frame contain the range of allowable values in the corresponding slot of an instance frame. So if a rule specifies a particular value, a positive example must contain that value. The slot values for negative examples are randomly chosen from the other alternatives. If a slot value is not specified by a rule, that slot may randomly take any allowable value.

The structure of one of the base events is shown in figure 5. In this example, the event is that of a robot getting a beer. For the evaluation, we synthesis 50 positive and 50 negative examples of ten different types of events:

1. A glass falling and breaking;
2. Cleaning a broken glass;
3. Turning on the television;
4. A person sitting on the couch;
5. Getting a beer;

Figure 4: A generic frame for representing the time of an event. Each slot contains a list of possible values that a slot in a time frame instance can have. For example, if the month of the event is June and the RDR for this type of event requires that the month of two events must match, then to create a positive example, the month slot of the time frame is assigned the value, June. To create a negative example, the system randomly chooses any other value.

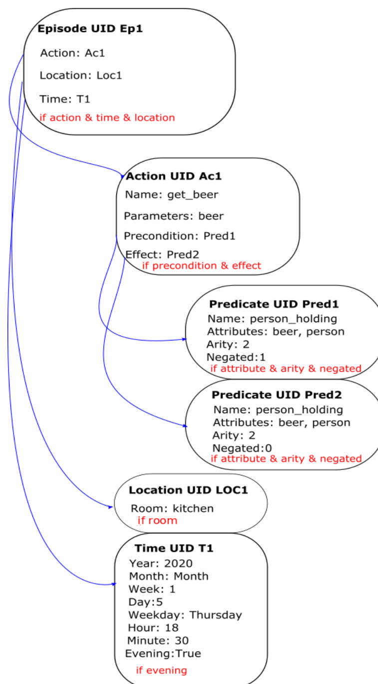

Figure 5: An episode where a robot gets a beer. Slots that refer to other frames are indicated by blue arrows. The RDR matching rules (in red) for each frame are stored in the corresponding generic frame. UID stands for Unique ID. Note that the diagram only shows the compound action that achieves the goal. We omit all the sub-actions need to implement this high-level behaviour. A frame like this would usually be preceded by an event such as, for example, sitting down and switching on the television, which then triggers this behaviour.

6. A person coming home;
7. A person waking up;
8. A person taking a shower;
9. A person entering the kitchen;
10. Making breakfast.

Although getting a beer seems like a very specific type of event, the examples are varied because the times and locations and preceding and succeeding events are generated randomly. So the RDR acquisition must learn which properties are essential and which are not.

RDR training is incremental, which is appropriate for teaching a robot in a home setting, as new training examples occur one at a time. On average, only a few training examples, often just two, are needed to create an RDR that successfully matches new events. If the system creates a rule that is too general or too specific, it is easily corrected when a new case is encountered.

To measure how many training examples are required for an RDR to converge, we randomly choose a positive example and construct the initial RDR. If the RDR correctly classifies all the remaining examples, we stop. If not, like a covering algorithm, we successively choose new random positive and negative examples to extend the RDR. This process is detailed in algorithm 2.

Of the events types that we tested, it was found that only two observations were required to train an RDR that was then able to successfully recall instances of that event type at a later stage. The system successfully matched all events that were expected to match the current event and did not incorrectly recall any of the events that we were not expecting to match.

Many of the methods for event retrieval that we reviewed rely on trying to fit a single match cue to every possible eventuality. Lim et al. (2011) use a method called a *compound cue* to retrieve events that, on a small collection of episodes will work very effectively, but as the sample size grows, it leads to multiple, undesirable matches of events. As each of our event types has its own matching rule, this will not occur, unless the rule requires further training. Another approach by Chang & Tan (2017) uses adaptive resonance theory to handle partial or noisy matching cues. As already explained this is not a problem that RDRs suffer from. The other benefit of using RDRs is that if an event RDR was not trained correctly and an incorrect event was retrieved or a correct event was not retrieved, we are able to dynamically update the rule without the need to rebuild the system.

6. Conclusion and Future Work

We have presented a novel method for event retrieval using ripple down rules for episodic memory. On ten simulated types of events, we are able to achieve perfect recall after a single training instance per event type. There are several extensions that we plan. One is to determine when an observation can be filtered out as an event. Most changes to the state of the world have inconsequential effects. For example, moving a plate from one side of the table to the other. However, with the current model, all changes are considered equal. There are other problems related to connected events. The

first is, at what point do we decide if two events are connected and at what point do we stop? This is not trivial as it means deciding on a metric by which we can decide that two events are relevant to one another aside from just being temporally close to each other.

The efficiency of the matching procedure could be significantly improved if the RDRs can be compiled into something similar to a RETE network (Forgy (1982)). We also do not address the issue of forgetting episodes in this paper. Forgetting certain episodic memories is essential to performance because as the database of episodes increases, a greater demand is placed on the retrieval algorithm. Nuxoll et al. (2010) compare the most common algorithms used for forgetting episodes and conclude that an activation based approach, whereby episodes are selected for removal from memory based on certain criteria relating to the frequency and recency of a particular episode's recall has the best performance.

7. Acknowledgements

We would like to thank Prof. Paul Compton for his very insightful feedback on the research presented above.

References

Allen, J. F. (1990). Maintaining knowledge about temporal intervals. *Readings in qualitative reasoning about physical systems*, *26*, 361–372.

Botvinick, M., Ritter, S., X. Wang, J., Kurth-Nelson, Z., Blundell, C., & Hassabis, D. (2019). Reinforcement learning, fast and slow. *Trends in Cognitive Sciences*.

Chang, P.-H., & Tan, A.-H. (2017). Encoding and recall of spatio-temporal episodic memory in real time. *Proceedings of the 26th International Joint Conference on Artificial Intelligence* (pp. 1490–1496). AAAI Press. From http://dl.acm.org/citation.cfm?id=3172077.3172094.

Compton, P., Edwards, G., Kang, B., Lazarus, L., Malor, R., Preston, P., & Srinivasan, A. (1992). Ripple down rules: Turning knowledge acquisition into knowledge maintenance. *Artificial Intelligence in Medicine*, *4*, 463 – 475. From http://www.sciencedirect.com/science/article/pii/093336579290013F. Representing Knowledge in Medical Decision Support Systems.

Compton, P., & Jansen, R. (1990). Knowledge in context: A strategy for expert system maintenance. *AI '88* (pp. 292–306). Berlin, Heidelberg: Springer Berlin Heidelberg.

Derbinsky, N., & Laird, J. E. (2009). Efficiently implementing episodic memory. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *5650 LNAI*, 403–417.

Forgy, C. L. (1982). Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, *19*, 17 – 37. From http://www.sciencedirect.com/science/article/pii/0004370282900200.

Gaines, B. R., & Compton, P. (1995). Induction of ripple-down rules applied to modeling large databases. *Journal of Intelligent Information Systems*, *5*, 211–228. From https://doi.org/10.1007/

BF00962234.

Hoffmann, J. (2001). Ff: The fast-forward planning system. *AI Magazine*, *22*, 57–62.

Homem, T. P. D., Santos, P. E., Costa], A. H. R., [da Costa Bianchi], R. A., & de Mantaras], R. L. (2020). Qualitative case-based reasoning and learning. *Artificial Intelligence*, *283*, 103258. From http://www.sciencedirect.com/science/article/pii/S0004370218303424.

Jockel, S., Westhoff, D., & Jianwei Zhang (2007). Epirome - a novel framework to investigate high-level episodic robot memory. *2007 IEEE International Conference on Robotics and Biomimetics (ROBIO)* (pp. 1075–1080).

Kendall-Morwick, J., & Leake, D. (2014). *A study of two-phase retrieval for process-oriented case-based reasoning*, (pp. 7–27). Berlin, Heidelberg: Springer Berlin Heidelberg. From https://doi.org/10.1007/978-3-642-38736-4_2.

Kolodner, J. (2014). *Case-based reasoning*. Morgan Kaufmann.

Korpan, R., & Epstein, S. L. (2018). Toward Natural Explanations for a Robot's Navigation Plans.

Kumar, A., et al. (2016). Ask Me Anything: Dynamic Memory Networks for Natural Language Processing. *Proceedings of The 33rd International Conference on Machine Learning, PMLR 48*, *48*, 1378–1387. From http://proceedings.mlr.press/v48/kumar16.html{%}0Ahttp://proceedings.mlr.press/v48/kumar16.pdf.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). SOAR: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1–64. From http://www.sciencedirect.com/science/article/pii/0004370287900506.

Lim, M. Y., Aylett, R., Vargas, P. A., Ho, W. C., & Dias, J. a. (2011). Human-like memory retrieval mechanisms for social companions. *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 3* (pp. 1117–1118). Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems. From http://dl.acm.org/citation.cfm?id=2034396.2034446.

Lin, Z., Zhao, T., Yang, G., & Zhang, L. (2018). Episodic memory deep q-networks. *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18* (pp. 2433–2439). International Joint Conferences on Artificial Intelligence Organization. From https://doi.org/10.24963/ijcai.2018/337.

Liu, D., Cong, M., & Du, Y. (2017a). Episodic Memory-Based Robotic Planning under Uncertainty. *IEEE Transactions on Industrial Electronics*, *64*, 1762–1772.

Liu, D., Cong, M., Du, Y., Zou, Q., & Cui, Y. (2017b). Robotic autonomous behavior selection using episodic memory and attention system. *Industrial Robot: An International Journal*, *44*, 353–362. From http://www.emeraldinsight.com/doi/10.1108/IR-09-2016-0250.

McGill, M., Sammut, C., & Westendorp, J. (2008). FrameScript: A Multi-modal Scripting Language.

Minsky, M. (1975). Minsky's frame system theory. *Proceedings of the 1975 Workshop on Theoretical Issues in Natural Language Processing* (pp. 104–116). Stroudsburg, PA, USA: Association for Computational Linguistics. From https://doi.org/10.3115/980190.980222.

Nuxoll, A., Tecuci, D., Ho, W. C., & Wang, N. (2010). Comparing forgetting algorithms for artificial episodic memory systems. *Proc. of the Symposium on Human Memory for Artificial Agents. AISB* (pp. 14–20).

Nuxoll, A. M., & Laird, J. E. (2012). Enhancing intelligent agents with episodic memory Action editor : Vasant Honavar. *Cognitive Systems Research*, *17-18*, 34–48. From http://dx.doi.org/10.1016/j.cogsys.2011.10.002.

Rosenthal, S., & Veloso, M. (2012). Mobile robot planning to seek help with spatially-situated tasks. *Proceedings of the National Conference on Artificial Intelligence*, *3*.

Sharma, M., & Sharma, C. (2020). A review on diverse applications of case-based reasoning. *Advances in Computing and Intelligent Systems* (pp. 511–517). Singapore: Springer Singapore.

Shen, F., Ouyang, Q., Kasai, W., & Hasegawa, O. (2013). A general associative memory based on self-organizing incremental neural network. *Neurocomputing*, *104*, 57 – 71. From http://www.sciencedirect.com/science/article/pii/S092523121200834X.

Smyth, B., & Keane, M. T. (1994). Retrieving adaptable cases. *Topics in Case-Based Reasoning* (pp. 209–220). Berlin, Heidelberg: Springer Berlin Heidelberg.

Tecuci, D., & Porter, B. (2007). A generic memory module for events. *Proceedings of the Twentieth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2007*, (pp. 152–157).

Tulving, E. (1983). *Elements of episodic memory*. Oxford University Press.

Tulving, E., et al. (1972). Episodic and semantic memory. *Organization of memory*, *1*, 381–403.

Wheeler, M. E., & Ploran, E. J. (2009). Episodic Memory. In L. R. Squire (Ed.), *Encyclopedia of neuroscience*, 1167–1172. Oxford: Academic Press. From http://www.sciencedirect.com/science/article/pii/B9780080450469007609.

Xiong, C., Merity, S., & Socher, R. (2016). Dynamic memory networks for visual and textual question answering.