A Generic Memory Module for Events

Dan G. Tecuci and **Bruce W. Porter**
Department of Computer Sciences
University of Texas at Austin
{tecuci,porter}@cs.utexas.edu

Abstract

Intelligent systems need to store their experience so that it can be reused. A memory for such systems needs to efficiently organize and search previous experience and to retrieve items relevant to the situation at hand. It needs to be content addressable while providing flexible matching. Previous approaches to building such memories suffered from being overly tied to a task or domain.

We propose to separate memory functionality from that of the system as a way to reduce the complexity of the overall system and to allow research to focus on the generic aspects of memory organization and retrieval without the bias of a specific domain and task. We built such a generic memory for events. It employs a representation of generic episodes, uses a multi-layer indexing scheme and provides a generic API to external systems.

We tested this memory module on three different tasks in the logistics domain and showed that it performs as well as serial search in terms of accuracy, while being much more efficient and more scalable.

Introduction

Remembering past experiences is an essential characteristic of any intelligent system. Such experiences enable the system to solve similar problems - by adapting previous solutions - and to avoid unwanted behavior - by detecting potential problems and trying to side-step them.

As the tasks that an intelligent system accomplishes become more and more complex, so does the experience it acquires in the process. A memory for such a system has to be able to cope with this complexity and make available the acquired experience for whatever future use the system may have for if. For example, a planner needs not only to build plans for a given goal, but also to monitor their execution in order to detect and avoid failures and to recover from such failures if they still occur. A memory for such a system should be able to store and retrieve past experience by goals, sequence of actions taken, failures avoided and those encountered.

Although systems that use memory to organize events have been presented before (e.g. (Hammond 1986) used memory to guide the functioning of a planner, and (Kolodner 1984) used memory to organize events by their time and place), they suffered from either being too specific to a task or domain. Recent approaches (Ma & Knight 2003; Nuxoll & Laird 2004; Sànchez-Marrè *et al.* 2005) have recognized the need to build memories that can accommodate more complex and temporally extended experience, in the form of Episodic Memories (Tulving 1983). However they embedded the memory module within the system, making it difficult both to study in isolation and to port to other systems (Vere & Bickmore 1990; Nuxoll & Laird 2004).

The need for *general* tools to aid the development of knowledge-based systems has long been recognized: E-MYCIN (van Melle, Shortliffe, & Buchanan 1984) separates domain specific knowledge (i.e. rules) from the inference mechanisms. We propose to separate the memory functionality from the system and build a *generic* memory module that can be attached to a variety of applications in order to provide episodic memory functionality. Encapsulating the complexity of such a memory into a separate subsystem should reduce the complexity of other parts of the overall system, allowing us to focus on the generic aspects of memory organization and retrieval and its interaction with the external application. Each application will use the retrieved memories differently, depending on their task. We do not propose complete solutions for problem solving in these domains as this would require domain specific knowledge (e.g. for adapting prior experience); rather, the episodic memory will have a supporting role in problem solving.

A Generic Memory Module for Events

General Memory Requirements

Desired qualities of a generic memory module include: *accuracy* (memory should return memories relevant for the situation at hand), *scalability* (memory should be able to accommodate a large number of episodes without a significant decrease in performance), *efficiency* (memory should provide efficient storage and recall), *content addressability* (memory items should be addressable by their content) and *flexible matching* (memory should recall the correct previous episodes even if they only partially match the current context).

From a software application perspective, a generic mem-

ory module for events needs to provide: a *generic representation of events* that can be used with different types of events; a *flexible interface* that allows various types of queries to be formulated and provides feedback to the application on how these queries were matched against memory, and *domain-independent organization and retrieval techniques* that efficiently index events.

Episode Representation

A generic episodic memory needs to have a representation for a generic episode. Episodes are dynamic in nature, changing the state of the world in complex ways. Besides a sequence of actions that make up the episode, the context in which the episode happens as well as its effect on the world are important. We propose that a generic episode have three dimensions: **context**, **contents** and **outcome**. **Context** is the general setting in which an episode happened; for some applications (e.g. planning) this might be the initial state and the goal of the episode (the desired state of the world after the episode is executed). **Contents** is the ordered set of events that make up the episode; in the case of a planner, this would be the plan itself. The **outcome** of an episode is an evaluation of the episode's effect (e.g. if a plan was successful or not, what failures it avoided, etc.).

The idea of indexing episodes based on the different kinds of information encoded by them is not new - Chef (Hammond 1986) indexed plans both by their goals and by their failures. We extend this idea by defining three generic dimensions for episodes and show that retrieval along one or more of these dimensions allows *the same memory structure* to be used for various memory-based tasks. For example a memory of plan goals, their corresponding plans and whether or not they were achieved by a given plan can be used for tasks such as:

planning - devise a plan (i.e. a sequence of actions) to accomplish a given goal. In terms of our representation this corresponds to memory retrieval using episode context (i.e. initial state and goal of a planning problem) and adapting the contents of the retrieved episodes (i.e. their plans).

classification - recognize whether a goal is solvable given a state of the world. This corresponds to retrieval based on episode context and using the outcome of the retrieved episodes (i.e. their success) for classification.

goal recognition - recognize the goal of an agent executing a sequence of actions. This corresponds to retrieval based on episode contents (i.e. observed actions) and adapting the context of retrieved episodes.

The semantics of individual actions (i.e. their applicability conditions and goals they achieve), as well as knowledge about the state of the world is represented using our knowledge base - a library of about 700 general concepts such as `Transport`, `Communicate`, `Enter` and 80 semantic relations like `agent`, `object`, `causes`, `size` (Barker, Porter, & Clark 2001).

The episode is the basic unit of information that memory operates on. The decision as to what constitutes a meaningful episode is domain dependent and left to the external application to make. In general, an episode is a sequence of actions with a common goal, which cannot be inferred from the individual actions taken in isolation.

Memory API

The memory module provides two basic functions: **store** and **retrieve**. **Store** takes a **new Episode** represented as a triple [context, contents, outcome] and stores it in memory, indexing it along all three dimensions; **retrieve** takes a **stimulus** (i.e. a partially specified Episode) and a **dimension** and retrieves the most similar prior episodes along that dimension. Memory retrieval provides also information on how a stimulus matched the retrieved episodes (e.g. shared structure, differences, mappings). This information is intended to be used by the external application that works in connection with the memory module and helps it better utilize the returned episodes for its purpose (e.g. adaptation).

Memory Organization and Retrieval

Episodes are stored in memory unchanged (i.e. no generalization) and are indexed for retrieval. We have adopted a multi-layer indexing scheme similar to MAC/FAC (Forbus, Gentner, & Law 1995), (Börner 1994) and Protos (Porter, Bareiss, & Holte 1990): a *shallow indexing* in which each episode is indexed by all its features taken in isolation and a *deep indexing* in which episodes are linked together by how they differ *structurally* from one another.

During retrieval, shallow indexing will select a set of episodes based on the number of common features between them and the stimulus. Starting from these candidate episodes, a hill-climbing algorithm using semantic-matching will find the episode(s) that best match the external stimulus. A robust memory needs to employ a flexible matching algorithm, such that old situations are still recognized under new trappings. The semantic matcher we use (Yeh, Porter, & Barker 2003) employs taxonomic knowledge, subgraph isomorphism and transformation rules in order to resolve mismatches between two representations.

It is the *organization of memory* given by this indexing mechanism and the *search-based retrieval* that sets our approach apart from those employing a flat memory structure, that is searched serially (e.g. (Nuxoll & Laird 2004; Forbus, Gentner, & Law 1995)).

An important parameter that controls the functioning of the episodic memory module is the number of initial candidate episodes that are explored. Given that all stored episodes might have some - albeit slight - resemblance to a stimulus, a limit on the number of such candidate episodes needs to be imposed. Otherwise, the hill-climb process might explore all stored episodes, failing to scale. This limit can be set by the external applications.

Experimental Evaluation

We empirically evaluated how the proposed memory architecture fares against our goals (being generic in nature and scalable). We chose three different tasks in the logistics domain (Veloso 1994): *planning* (given a goal, find a plan that

would achieve that goal), *classification* (is a given goal solvable?) and *goal-schema recognition* (given the sequence of actions, recognize its goal type).

Episodes for all three tasks had the same representation: a plan goal and initial situation as the context, the corresponding plan that accomplishes that goal for the contents, and whether or not the goal is solvable (i.e. a plan that accomplishes the goal exists) as outcome.

Each of these tasks employed different uses of the memory retrieval mechanism and used the retrieved episodes differently. Planning used retrieval based on context and adapted the contents of the retrieved episodes, recognition used retrieval based on contents and adapted the context of retrieved episodes, while classification employed retrieval based on context and adapted the outcome of the retrieved episodes. Successfully building and using such a multi-functional memory structure for different tasks supports our claim that a generic memory module can be built and that our proposed architecture is a good candidate for that purpose. Besides the performance at the individual tasks, we were also interested in how memory behaves as the number of observed episodes grows.

The Logistics Domain

The logistics domain (Veloso 1994) consists of simple plans involving delivery of packages among various locations. There are two types of locations: post offices and airports, either in the same or in different cities. Within a city, packages are delivered by trucks, whereas between cities airplanes are used. If a vehicle is not available at the pick-up location it has to be moved there from its current location. We restricted the goals to involve deliveries of a single package and three cities, resulting in 11 different goal types.

We have randomly generated a set of 250 pairs of goals and initial situations, so that the distribution of goal types is close to uniform. In order to generate unsolvable problems, we generated minimally solvable problems (i.e. containing the minimal set of instances such that they are solvable), then randomly removed facts from them. Each fact had a 0.2 probability of being removed, independently of other facts being removed, thus generating a rather wide variety of goals and initial situation descriptions. We used the SHOP2 planner (Nau *et al.* 2003) in order to determine whether a [goal, initial situation] pair is solvable and if so, to build a plan that achieves the given goal. The resulting dataset had 129 unsolvable goals and 121 solvable. We used this dataset for all three tasks described above.

Experimental Setup

For each of the three tasks we have built an EM-based system. This required writing a thin interface layer on top of the EM generic memory module. This consisted of functions dealing with the adaptation and evaluation of the retrieved episodes. For all EM systems we limited the number of candidate episodes explored to 5.

We adopted a storage policy similar to (Smyth & Keane 1995) by storing only episodes for which the retrieved memory episode could not accomplish its intended task. This reduced the memory size without a decrease in performance.

For each of the three tasks we performed a 10-fold cross validation, generating learning curves. We measured the performance of each of the systems in terms of *accuracy*, *retrieval cost* (number of explored episodes per task), and the *scalability* of retrieval (number of episodes explored vs. total number of episodes stored). We have compared our approach against a k-nearest neighbor algorithm (denoted here kNN(5)) that performs a serial search through the memory of episodes, and retaining 5 most similar episodes. We chose kNN for two reasons: first, to be able to evaluate the impact of the indexing mechanism employed by our memory module; kNN(5) is an ablation of our EM system: it uses the same semantic match routine to determine similarity between a new episode and an old one and employs the same storage policy. The most significant difference between kNN and EM is that kNN's memory is flat, while EM's memory is multi-layered. Second, serial search is the basic search process employed by memories with a flat organization.

For the **memory-based planning** task we eliminated from the dataset the unsolvable goals, resulting in a total of 121 goal-plan pairs. A retrieved episode is considered correct if its plan can be easily adapted (i.e. using only variable substitutions suggested by the memory retrieval function) to solve the given goal. The plan corresponding to the most similar episode retrieved was adapted. The results of these experiments are presented in Figure 1(a)-1(c).

For the **memory-based classification** task we used all 250 goals, including both solvable and unsolvable goals. A majority vote by the top 5 most similar retrieved episodes was used to determine whether a new goal is solvable or not. The results are presented in Figure 2(a)-2(c).

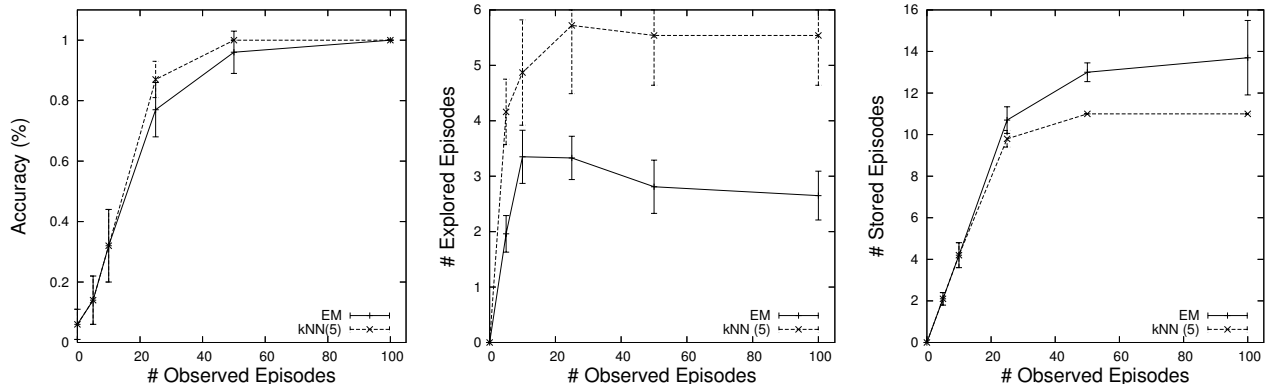
For the **memory-based goal-schema recognition** task, we used only the problems containing non-empty plans, eliminating those that were unsolvable and those for which the goal is already achieved (i.e. trivial plans, in which the package is already at its destination). 109 problems were left. The goal-schema of the most similar plan retrieved was used for classification. The results are presented in Figure 3(a)-3(b). Our goal-schema recognition algorithm works incrementally, generating predictions after each action is seen, therefore we report the number of explored actions (not episodes) per recognition task. There are an average of 8.83 actions per episode in the dataset.

Discussion

For all three tasks EM achieves the same accuracy as kNN(5) after most training episodes have been seen. Even though kNN(5) learns faster, EM is able to catch up in the end.

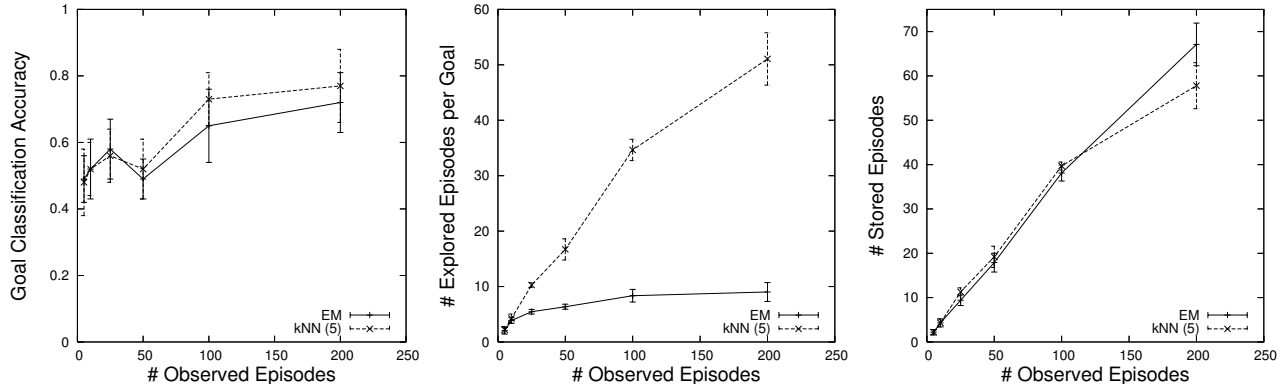
In terms of explored episodes, EM is able to drastically reduce their number compared to kNN(5). As kNN(5) is an EM from which the multi-layered organization of episodes (i.e. indexing mechanism) has been ablated, we attribute the efficient retrieval to this memory organization technique.

In terms of memory size, EM stores slightly more episodes than kNN(5). This is explained by the fact that EM learns slower and in the process stores those episodes for which it did not perform well when they were first seen. Without a memory compaction mechanism, these episodes



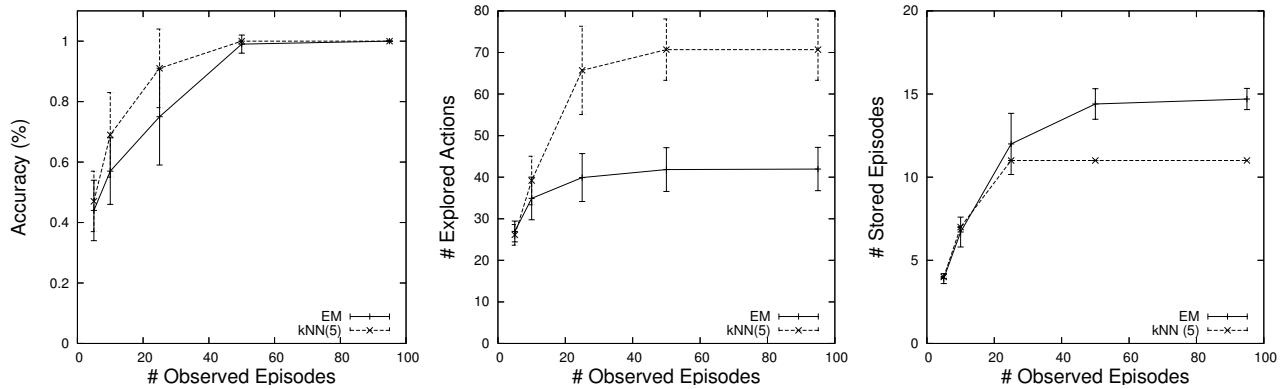
(a) Accuracy. (b) Number of Explored Episodes. (c) Number of Stored Episodes.

Figure 1: Experimental results for the memory-based planning task: accuracy, number of explored episodes and number of stored episodes for EM and kNN(5).



(a) Accuracy. (b) Number of Explored Episodes. (c) Number of Stored Episodes.

Figure 2: Experimental results for the goal classification task: accuracy, number of explored episodes and number of stored episodes for EM and kNN(5).



(a) Accuracy. (b) Number of Explored Actions. (c) Number of Stored Episodes.

Figure 3: Experimental results for the goal-schema recognition task: accuracy, number of explored actions and number of stored episodes for EM and kNN(5).

are left in memory. However, even having stored more episodes than kNN(5), EM examines significantly fewer episodes with respect to the number of stored episodes. This shows that EM's retrieval scheme is scalable. Another argument for this is that even though the number of episodes stored by EM increases, the number of explored episodes per task stays constant (Figure 2(b)) or even decreases (Figure 1(b)).

A limitation of using artificial plan corpora (like the one used here) for goal-schema recognition tasks is that planners usually optimize for some parameter (plan length, cost, etc.), generating the same sequence of actions in a plan every time a similar goal is seen. This is less than desirable since real world plans rarely display this characteristic. We have used the same goal-schema recognition algorithm on real-world data, achieving performance comparable to statistical approaches (Tecuci & Porter 2006). In the future, we plan to apply the same approach on plan corpora specifically generated for recognition tasks (e.g. (Blaylock & Allen 2005)).

Related Work

Early attempts to build a memory for events explored ideas from (Schank 1982) and include Cyrus (Kolodner 1984) which designed a memory for organizing events by their time and place and Chef (Hammond 1986), a case-based planner that indexes its plans by the goals they achieve and the failures they avoid. The Basic Agent (Vere & Bickmore 1990) was the first system to use a separate episodic memory in conjunction with an intelligent system; it employed two such memory modules, one made up of simple recognizers, used for text generation, and the other for guiding the planner's backtracking mechanism.

(Nuxoll & Laird 2004) designed the most advanced cognitive model of an episodic memory, implementing all the functional stages proposed by (Tulving 1983). Its main limitations are the fact that it is embedded in the Soar architecture, the flat episode organization and serial search strategy which results in retrieval time linear in the number of stored episodes. Our approach addresses the same problem of building a *generic* episodic memory, but it does not embed it in a generic cognitive architecture, trying to be architecture-independent. The interaction between the overall system and the memory module needs to go through a well-defined interface. In terms of retrieval, Soar EM uses all the features of an episode during retrieval, while EM will only use the subset corresponding to a specific dimension. An important difference is that retrieval in Soar EM is a serial search through the entire episodic memory, while EM employs hill-climbing through the space of memory structures. Soar EM uses simple feature match algorithm for computing the similarity between the cue and a prior episode, while EM makes use of a semantic matcher.

More recently there has been a lot of interest in the CBR community in incorporating temporal information in the stored cases. (Ram & Santamaria 1997) recorded raw data from prior actions to improve navigation. Although it employs matching during retrieval, cases consist entirely of quantitative data and have a fixed structure. (Ma & Knight 2003) propose the use of a relative temporal knowledge model to support historical CBR. Similarity has two components: non-temporal (based on elemental cases) and temporal (based on the graphical representation of the temporal references).

(Sànchez-Marrè *et al.* 2005) propose Episodic-Based Reasoning (EBR) as an extension to CBR in order to cope with dynamic or continuous temporal domains. Temporal sequences of simple cases form an episode. Different episodes can overlap, thus having simple cases in common. Cases use a flat representation, while Episodes are organized using discrimination trees. Sets of episodes considered to share an important pattern are grouped in meta-episodes (called Episode Bases). Retrieval proceeds in a top-down fashion, by selecting a meta-episode first and then searching inside the corresponding episode-bases. Similarity of episodes based on their set of events is computed by looking at each pair of corresponding events (simple cases), in temporal order. Both our approach and EBR address the same questions: how to represent temporal sequences of events, how to organize and retrieve them efficiently, and how to assess similarity between such episodes. EM does not address the problem of determining the extent of an episode, leaving that to the application that uses the memory module. In EBR an episode consists of a description, a solution-plan and a solution-evaluation, corresponding roughly to our three dimensions (context, contents and outcome). However, retrieval in EBR is only performed on the episode sequence of events, while EM allows retrieval on context and outcome as well. Another important difference is the underlying representation of cases, for which EBR uses an attribute-value representation, while we employ a frame-based representation.

Related approaches also include the application of case-based reasoning technology to experience management (Althoff & Weber 2005), where the intended target is either human (the *experience factory* of (Althoff *et al.* 1998)) or machine (the *lessons learned* of (Weber, Aha, & Becerra-Fernandez 2001)).

Conclusions

Although there is agreement that intelligent systems could benefit from reusing their prior experience, the form that this experience should take is still debated. In this paper we argued that the memory module should be separated from the system that uses it and that it is possible to build a generic memory module for this purpose. Such a memory module is not intended to provide complete solutions for problems, but to have an assisting role.

We presented an architecture for a memory module for events that provides accurate and efficient retrieval, flexible matching, is scalable, and content addressable. It uses a generic representation for events that divides an episode into its context, contents and outcome. Interaction with external applications is done through a simple API composed of a store and retrieve functions. Episodes are indexed using a two-tier process: shallow (looking at features in isolation) and deep (looking at how episodes differ structurally from each other). Generic storage and retrieval mechanisms that exploit this indexing structures were presented. Compared

to similar approaches, our memory module is architecture independent, uses semantic matching for retrieval and generates an indexed memory structure.

We evaluated the proposed memory module on three different tasks in the logistics domains: memory-based planning, memory-based classification and goal-schema recognition. Empirical evaluation showed that the indexing mechanism maintains the same level of performance but significantly improves retrieval efficiency compared to a nearest-neighbor algorithm. The storage and retrieval strategies proved scalable: even though the number of stored episodes grew linearly, the number of explored episodes remained constant or grew at a much slower rate. The fact that the same memory organization and retrieval mechanisms were successfully applied to three different tasks supports our claim that the memory module is generic.

Any long-term memory needs to revisit the items stored during its past execution and decide whether to delete them (i.e to forget) or not. Under our current storage policy the decision to store is made when the episode is first seen and never reversed. We intend to look at forgetting policies in order to make memory structure more efficient.

References

Althoff, K.-D., and Weber, R. 2005. Knowledge Management in Case-Based Reasoning. *The Knowledge Engineering Review* 20:305–310.

Althoff, K.-D.; Birk, A.; von Wangenheim, C. G.; and Tautz, C. 1998. CBR for Experimental Software Engineering. In *Case-Based Reasoning Technology, From Foundations to Applications*, 235–254. London, UK: Springer-Verlag.

Barker, K.; Porter, B.; and Clark, P. 2001. A Library of Generic Concepts for Composing Knowledge Bases. In *K-CAP 2001: Proceedings of the international conference on Knowledge capture*, 14–21. ACM Press.

Blaylock, N., and Allen, J. 2005. Generating Artificial Corpora for Plan Recognition. In Ardissono, L.; Brna, P.; and Mitrovic, A., eds., *Lecture Notes in Artificial Intelligence - User Modeling 2005*, volume 3538. Springer.

Börner, K. 1994. Structural Similarity as Guidance in Case-Based Design. In *EWCBR '93: Selected papers from the First European Workshop on Topics in Case-Based Reasoning*, 197–208. London, UK: Springer-Verlag.

Forbus, K.; Gentner, D.; and Law, K. 1995. MAC/FAC: A Model of Similarity-Based Retrieval. *Cognitive Science* 19(2):141–205.

Hammond, K. J. 1986. CHEF: A Model of Case-Based Planning. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, 267–271.

Kolodner, J. L. 1984. *Retrieval and Organizational Strategies in Conceptual Memory: A Computer Model*. Hillsdale, New Jersey: Lawrence Erlbaum Associates.

Ma, J., and Knight, B. 2003. A Framework for Historical Case-Based Reasoning. In Ashley, K. D., and Bridge, D. G., eds., *ICCBR*, volume 2689 of *Lecture Notes in Computer Science*, 246–260. Springer.

Nau, D.; Au, T.; Ilghami, O.; Kuter, U.; Murdock, J.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN Planning System. *Journal of Artificial Intelligence Research* 20:379–404.

Nuxoll, A., and Laird, J. E. 2004. A Cognitive Model of Episodic Memory Integrated With a General Cognitive Architecture. In *Proceedings of the Sixth International Conference on Cognitive Modeling*, 220–225. Mahwah, NJ: Lawrence Earlbaum.

Porter, B. W.; Bareiss, R.; and Holte, R. C. 1990. Concept Learning and Heuristic Classification in Weak-Theory Domains. *Artifical Intelligence* 45:229–263.

Ram, A., and Santamaria, J. C. 1997. Continuous Case-Based Reasoning. *Artificial Intelligence* 90(1-2):25–77.

Sànchez-Marrè, M.; Cortés, U.; Martínez, M.; Comas, J.; and Rodríguez-Roda, I. 2005. An Approach for Temporal Case-Based Reasoning: Episode-Based Reasoning. In Muñoz-Avila, H., and Ricci, F., eds., *ICCBR*, volume 3620 of *Lecture Notes in Computer Science*, 465–476. Springer.

Schank, R. C. 1982. *Dynamic Memory. A Theory of Reminding and Learning in Computers and People*. Cambridge University Press.

Smyth, B., and Keane, M. T. 1995. Remembering to Forget: A Competence-Preserving Case Deletion Policy for Case-Based Reasoning Systems. In *IJCAI*, 377–383.

Tecuci, D., and Porter, B. 2006. Using an Episodic Memory Module for Pattern Capture and Recognition. In Murray, K., and Harrison, I., eds., *Capturing and Using Patterns for Evidence Detection: Papers from the 2006 Fall Symposium.* Menlo Park, CA: AAAI Press. Technical Report FS-06-02.

Tulving, E. 1983. *Elements of Episodic Memory*. Oxford: Clarendon Press.

van Melle, W.; Shortliffe, E.; and Buchanan, B. 1984. EMYCIN: A Knowledge Engineer's Tool for Constructing Rule-Based Expert Systems. In Buchanan, B., and Shortliffe, E., eds., *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley. chapter 15.

Veloso, M. M. 1994. *Planning and Learning by Analogical Reasoning*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

Vere, S., and Bickmore, T. 1990. A Basic Agent. *Computational Intelligence* 4:41–60.

Weber, R.; Aha, D.; and Becerra-Fernandez, I. 2001. Intelligent Lessons Learned Systems. *International Journal of Expert Systems — Research & Applications* 20(1).

Yeh, P. Z.; Porter, B.; and Barker, K. 2003. Using Transformations to Improve Semantic Matching. In *K-CAP 2003: Proceedings of the Second International Conference on Knowledge Capture*, 180–189.