The Dissertation Committee for Dan Gabriel Tecuci

certifies that this is the approved version of the following dissertation:

# A Generic Memory Module for Events

Committee:

Bruce W. Porter, Supervisor

Raymond J. Mooney

Benjamin J. Kuipers

Kenneth J. Barker

Bradley C. Love

# A Generic Memory Module for Events

by

**Dan Gabriel Tecuci, B.S., M.S.**

**Dissertation**

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

**Doctor of Philosophy**

**The University of Texas at Austin**

August 2007

# Acknowledgments

First and foremost, I would like to thank my advisor, Bruce Porter, for his friendship, advice and guidance during the difficult process of writing a dissertation. I benefited greatly from his generosity with his time, his patience, and his willingness to help every step of the way while allowing me the freedom to find my own way as a researcher. His ability to see through my often confused ideas the core of something valuable helped me immensely. He truly was the best advisor I could have asked for.

I would also like to thank the members of my committee, Ray Mooney, Ben Kuipers, Ken Barker and Brad Love, for their helpful suggestions, comments and stimulating discussions. Ray's Machine Learning class, one of my first in grad school, trained me as an experimentalist. I used his code in the experiments reported in this thesis. Working with Ben on modeling human way-finding taught me about long term research goals and how simple things like visualizing data properly can go a long way. From Brad I learned that studying human memory, even when you are not trying to model it, can be a great source of inspiration for AI. Over the years, I had the opportunity to work closely with Ken on various projects. He has been a mentor and a great friend. This dissertation and my research career in general benefited greatly from his advice.

I am fortunate to have collaborated with accomplished researchers like Peter Clark, Vinay Chaudry, and Brian Stankiewicz from whom I had a lot to learn.

I really enjoyed working in the Knowledge Systems Research Group with great people including Bill Jarold, Art Souther, Geoff King, Saurabh Baji, Bhalchandra Agashe, Sourabh Patwardhan, Steve Wilder, and Charlie Benton.

Many thanks to my officemates over the years James Fan, Peter Yeh, Jason Chaw, Michael Glass, and Doo Soon Kim. I will miss our stimulating discussions and the atmosphere of Taylor 4.115A. Special thanks for Peter Yeh for providing the matcher used in the implementation of the generic memory module and valuable advice on graph matching. Jason Chaw provided technical support for the Basic Problem Solver and helpful suggestions on some parts of this dissertation.

I would like to thank Gloria Ramirez, Katherine Utz and the rest of the staff of the Department of Computer Sciences for all their help over the years. Besides sharing advice on life, fashion and politics, Stacy Miller made sure my paycheck was always on time. The people at the International Office helped me seamlessly navigate through the bureaucracy associated with studying in a foreign country.

I would also like to thank all my teachers over the years and especially Mihai Stan, Ştefan Smarandache, Ilie Grigore, Adina Florea and Irina Atanasiu who inspired and entertained my thirst for knowledge.

My going away across the world to grad school was not easy for my family. However, they offered me their unconditional support and encouragement every step of the way. For that I am grateful. Without them I would not be who I am today. I just wish all of them could still be here and share this moment with me. I am especially indebted to my aunt and uncle, Sanda and Gheorghe Tecuci, for their continued support. My uncle's advice regarding various career choices has been invaluable.

My friends in Austin made my stay here so much more enjoyable.

I would like to thank my wife, Viorica Alexandra Teodorescu, for so many things, but most of all, for giving all this a purpose.

The research presented here was funded by SRI, DARPA, Vulcan, and The University of Texas at Austin. I gratefully acknowledge their support.

DAN GABRIEL TECUCI

*The University of Texas at Austin*

*August 2007*

# A Generic Memory Module for Events

Publication No. ____________________

Dan Gabriel Tecuci, Ph.D.
The University of Texas at Austin, 2007

Supervisor: Bruce W. Porter

The ability to remember past experiences enables a system to improve its performance as well as its competence. For example, a system might be able solve problems faster by adapting previous solutions. Additional tasks, such as avoiding unwanted behavior by detecting potential problems, monitoring long-term goals by remembering what subgoals have been achieved, and reflection on past actions, become feasible.

As the tasks that an intelligent system accomplishes become more and more complex, so does the experience it acquires in the process. Such experience has a temporal extent and is expressed in terms of concepts and relations with deep semantics associated to them. Memory systems should be able to deal with the temporal aspect of experience, exploit this semantic knowledge for storage and retrieval and do so in a scalable fashion.

However, relying just on experience will not achieve a broad coverage, as it needs to be used in conjunction with other reasoning mechanisms. That is why we need the ability to *add episodic memory* functionality to intelligent systems.

Today's knowledge-based systems are complex software applications and the ability to develop them in a modular fashion, using generic, reusable components is essential.

We propose to separate the episodic memory from the system that uses it and to build a *generic, reusable memory module* that can be *attached* to a variety of applications in order to provide this functionality. Its goal is to provide *accurate, scalable, efficient* and *content-addressable* access to prior episodes. Having such a reusable memory module should allow *research to focus on the generic aspects of memory representation, organization and retrieval* and its interaction with the external application and it should also *reduce the complexity* of the overall system.

In this dissertation we propose a set of general requirements that any memory module should provide regarding memory encoding, storage and retrieval. We present an implementation that satisfies these requirements and evaluate it on three different tasks: plan synthesis, plan recognition and Physics problem solving. The memory module proved easily adaptable to these tasks, providing fast, accurate and scalable retrieval.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

Past experience is an important source of knowledge that enables a system to improve its performance as well as its competence. For example, a system might be able solve problems faster by adapting previous solutions. Additional tasks become feasible, such as avoiding unwanted behavior by detecting potential problems, monitoring long-term goals by remembering what subgoals have been achieved, and reflection.

Learning at the time of experiencing might not have been possible (e.g. due to time constraints) or desirable (e.g. the observed event is deemed uninteresting), but may become so later (e.g. more time is available and reflection can be performed, or a pattern emerges in the light of new events). A learner that tries too eagerly to learn from experience and then discards it makes the strong assumption that it can learn all that is possible from that experience at that time. This assumption might be true for systems accomplishing simple tasks and for those that are short-lived, but a complex intelligent system with a longer life-expectancy needs to store and reuse its past experience.

A lot of today's intelligent systems do not use past experience in their functioning. Take, for example, Project Halo [57] whose goal is to develop tools that would enable subject matter experts to encode their knowledge (i.e. concepts, relations, procedures) in domains like chemistry, physics, and biology. The resulting knowledge bases are intended to answer AP-level questions in those domains.

Let us consider the following question (Figure 1.1) from the Physics domain.

```
An object starts from rest
and reaches a speed of 28 m/s in 2 s.
What distance does it cover?
```

Figure 1.1: An example of question in Physics.

A memory-less Halo system would systematically search the knowledge base of models (i.e. sets of equations describing motion) and apply their rules until an answer is found or a certain threshold has been exceeded. When a new question (see Figure 1.2) has to be answered, the system will discard the results of its problem-solving process for the previous question and redo the search process.

```
An object is moving at a speed of 1 m/s.
Over the next 5 m, its speed reaches 2 m/s.
What is the duration of the move?
```

Figure 1.2: Another example of question in Physics.

In contrast, a memory-enhanced Halo system, would store its previous problem-solving experience and use it to guide search by suggesting models that have been previously applied to similar questions. Thus, more appropriate models are tried first and the system is able to improve both its performance at the task (by solving problems faster), as well as its competence (by spending more time examining more relevant models).

## 1.1 The Problem of Storing Experience

For a memory system to be useful it needs to provide *fast access* to the *appropriate set of experiences* that are relevant to the situation at hand, and to ignore irrelevant ones.

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 it. 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 using a *broad range of criteria*: the goals that were achieved, the sequence of actions that were taken, the failures that were avoided and those encountered. As the life-expectancy of intelligent systems increases, a memory system needs to be able to deal with such *complex experience* in a *scalable* way.

## 1.2 The Goals of this Dissertation

Today's knowledge-based systems are complex software applications and the ability to develop them in a modular fashion, using generic, reusable components is essential. The need for *generic* tools to aid the development of knowledge-based systems has long been recognized: E-MYCIN [120] separates domain specific knowledge (i.e. rules) from the inference mechanisms.

We propose to separate the *episodic memory functionality* from the system that uses it and to build a *generic, reusable memory module* that can be attached to a variety of applications in order to provide this functionality. Its goal is to provide *accurate, scalable, efficient* and *content-addressable* access to prior episodes. Having such a reusable memory module should allow *research to focus on the generic aspects of memory representation, organization and retrieval* and its interaction with the external application and it should also *reduce the complexity* of the overall system.

In this dissertation we address the following questions:

- Can a *generic memory module* be built? What should be its characteristics?
- Does there exist a *representation for generic events*, such that it can be used with

different types of events and queries?

- Can such a memory provide a *flexible interface* that allows various types of queries to be formulated?
- Can we devise *domain-independent organization and retrieval techniques* that efficiently index events?

The proposed episodic memory module is not intended to provide complete problem solving solutions as this would require domain specific knowledge (e.g. for adapting prior experience); rather, the episodic memory will have a supporting role.

## 1.3 Summary of Contributions of this Dissertation

In this dissertation we proposed to *separate the episodic memory functionality* from the application itself and build a *generic episodic memory module for events* that can add such a functionality to a broad range of intelligent systems.

We presented a *set requirements* that any memory module should follow including internal requirements (in terms of memory encoding, storage and retrieval) and external ones (in terms of the interface to external applications).

We proposed a *generic representation for events*, an *organization scheme* for such representations and two *retrieval algorithms* - one for the case when the stimulus is presented all at once and one for the case when it is presented sequentially.

We *implemented a memory module* that satisfies these requirements using the proposed representation and retrieval mechanisms.

We *evaluated the implementation* of the proposed memory module on three different tasks: planning, plan recognition, and Physics problem solving. The memory module was easily customized to perform these tasks.

The results of the our empirical evaluation show that memory is efficient, accurate and scalable. The proposed indexing mechanism significantly increased performance over systematic search, while preserving competence (for planning and Physics problem solving). This increase is considerably larger for search spaces that grow exponentially, as memory is able to prune more alternatives. Incremental episodic-based goal schema recognition achieved similar precision, higher recall and higher convergence when compared to a statistical approach. For parameter recognition, the episodic-based approach provided higher recall, but lower precision and convergence than the chosen statistical approach.

## 1.4 Organization of the Dissertation

The remainder of this dissertation is organized as follows: Chapter 2 surveys previous approaches dedicated to the study or implementation of episodic memories in intelligent systems, ranging from psychology to cognitive modelling.

Chapter 3 presents the set of requirements for a generic memory module for events, including episodic encoding, storage, retrieval and the interface with the external application.

Chapter 4 presents our implementation of such a memory module. We present a generic representation of events, a multi-layer indexing scheme and retrieval algorithms for such representations, as well as a flexible programming interface.

Chapter 5 presents the application of the memory module to the tasks of memory-based planning and goal classification in the Logistics domain and the results of an empirical evaluation.

Chapter 6 presents an application to the domain of plan recognition in the Linux and Monroe domains as well as the results of an empirical evaluation.

Chapter 7 presents the application of the memory module to the task of problem solving. The generic memory module was used to enhance a problem solver in the domain of question answering of AP-level Physics questions. Results of the empirical evaluation of this application are reported.

# Chapter 2

# Related Work

The modern study of memory (started by Ebbinghaus [36]) has been approached in different contexts and with different goals by philosophers, psychologists and researchers in Artificial Intelligence and Cognitive Science. We will look at findings from these domains as inspiration.

Our goal is to implement a generic memory for events that can be used by intelligent systems in a large number of applications. Although we seek inspiration from the human episodic memory, we do not attempt to build a model of it, but merely to replicate some of its functionality such that it can be used by an intelligent system.

We will look first at the division of human memory into semantic, episodic and procedural memories by characterizing the similarities and differences between them. As we are interested in building a memory module for events, we will take a closer look at the human episodic memory and its general functions as described by Tulving [119].

We will then survey approaches to building episodic memories from the both a cognitive and from an engineering perspective. We look at models of each of the three activities of episodic memory and then at how memory has been used in intelligent systems.

## 2.1 Human Episodic Memory

Human memory can be divided mainly into *episodic*, *semantic*, *procedural* and *working memories* [1].

Human Episodic Memory is a functionally distinct subsystem of human memory that is concerned with remembering specific sequences of events [119]. This ability of humans to rapidly acquire **episodic memories** has been the focus of considerable research in psychology and neuroscience, and there is a broad consensus that this form of memory is distinct both in its functional properties and in its neural basis from other forms of memories involving common sense knowledge, perceptual-motor skills, priming, and simple classical conditioning [107][2].

### 2.1.1 Episodic vs. Semantic Memory

**Episodic Memory** refers to a memory that maintains a record of 'events' pertaining to a person's ongoing perceptions, experiences, decisions and actions [119, 109]. Episodic memory is concerned with unique, concrete, personal experience dated in the rememberer's past (e.g. our last trip to the mall), while **semantic memory** [39] refers to a person's abstract, timeless knowledge of the world that he/she shares with others (e.g. the color of the sky). Episodic and semantic memory subsystems are thought to be functionally distinct but closely interacting memory systems [119].

#### Similarities

The episodic and semantic memory subsystems have a number of similarities: they deal with *knowledge acquisition* mostly through senses, this knowledge is *retained* in a passive and automatic way, requiring no effort on the part of the subject, and they both *use this*

[1]Shastri [109] mentions also prospective (memories of intentions) and utile memory (stores memories of utilities associated with situations).

[2]For a review of relevant experimental findings see [113, 33, 103].

*knowledge*, retrieval being triggered by stimuli, subjects are not being aware of it, but only of its results.

Both episodic and semantic memories are thought to be propositional in nature - they can be contemplated introspectively, can be communicated to others in some symbolic form and questions about their veridicality can be asked. These characteristics contrast with those of the **procedural memory** [127], a memory subsystem concerned with the acquisition and utilization of procedures and skills, which is **non-propositional**.

### Differences

Although they have a number of similarities, episodic and semantic memories differ significantly in other respects. These differences can be categorized along three dimensions: *the kind of information* they handle, their *operation* and their *applications* [119].

In terms of the kind of information handled, episodic memory stores specific events and situations, whose source is in the sensory systems, temporally organized and self-centered. On the other hand, semantic memory stores statistical summaries and abstractions acquired from several experiences through comprehension, organized conceptually and generally agreed upon.

Episodic memories are acquired rapidly (one-shot learning), have limited inferential capabilities, are more dependent on the context in which they were encountered, and are more vulnerable (i.e. can be easily lost, modified or changed). Access to episodic memories depends more on the mental state of the person (whether or not he/she is in 'retrieval mode') and tends to change the stored information. Semantic memories are acquired gradually through incremental learning, have rich inferential capabilities, and are less vulnerable. Access to semantic memories is more automatic and does not change the stored knowledge that much. There is evidence that episodic memory develops after the semantic memory, suggesting that the former needs more advanced capabilities [119].

Although not widely agreed upon[3], the interdependence of episodic and semantic memory has not been successfully argued against either. Tulving [118] cites evidence for the 'important role that the semantic system plays in the storage and retrieval of episodic memory information'. He notes, however, that the two systems *could* operate independently of one another, although not necessarily as efficiently [119].

In our current research, we will adopt the view expressed by Shastri [109], stating that episodic memory is built in terms of semantic memory, its elements (event types, their roles and parameters and their fillers) being represented in the semantic memory. Episodic and semantic memory work together seamlessly, which constitutes the basis for generating predictions, building explanations and making decisions.

### 2.1.2 Characteristics of Human Episodic Memory

The characteristics and general functions of human Episodic Memory will serve as a starting point and inspiration in our investigation of the design of a generic episodic memory for an intelligent system.

Tulving [119] lists the following characteristics of human episodic memory:

**autobiographical** - a person remembers episodes from his/her own perspective.

**autonoetic** - remembering entails the conscious re-experience of past memories, but the retrieved memories are distinguished from the perception of the person's current state.

**temporally annotated** - the person has a sense of the time when an episode has occurred.

**imperfect** - the memory is incomplete and can have errors.

**activated** - exposure frequency and recency affect the speed and probability of recall.

**primed** - recall occurs more quickly when it is primed by repetition or by recall of related information or similar state.

[3]For a comprehensive discussion of various pros and cons of this view see [119]

**forgetting** - memory performance declines with time or intervening events; this behavior is well fit by a power function (the power law of forgetting) [128, 129, 130]

### 2.1.3 Functions of Human Episodic Memory

A memory system is widely believed to have three high-level activities: encoding, storage and retrieval. Each of these activities achieves a particular set of functions [119].

#### Encoding

Encoding [78, 119] is the process that converts a perceived event into a memory representation. The characteristics of the representation of such an event depend not only on the event itself but also on how this event is encoded. The experiencer is not aware of this process.

**Activation** is the process of determining **when** a new episode needs to be recorded. As observations are continuously made, an agent must have the ability to segment them into episodes and decide when one needs to be stored.

**Salient feature selection** is concerned with deciding **what information** will be stored. According to Tulving [119] only a salient part of an episode is recorded in memory. This might play a role in the fact that memory is not perfect, but it is not clear where the imperfections originate: during encoding, storage, retrieval or some combination of them.

**Cue selection** is the process of deciding **what features** of the state will cue the memory. These are the features that index the specific episode.

From a computational point of view, another aspect of encoding is the particular **representation of episodes** and their **organization in memory**.

#### Storage

Storage deals with the maintenance of the encoded episodes over time.

**Storage medium** addresses the question of **how** the encoded episode is **maintained** in the memory store. There are diverging views of how human episodic memories are stored. One asserts that episodes are stored in the hippo-campus and later migrate to the cerebral cortex [113, 76], while others consider this to be unmotivated on computational and biological grounds [81, 82, 108]. The latter proposes that an event's episodic memory trace persists in hippo-campus for as long as the event is remembered as a specific episode.

**Forgetting** means preventing recall of episodes (or portions of episodes). Empirical studies show that human memory performance declines with time or intervening events. This decay is well described by a power function - called the 'power-law of forgetting' [11, 128, 129, 130]. Tulving [119] presents a theory under which newer memories prevent the recall of older ones.

### Retrieval

Retrieval is the process by which encoded episodes become available again to the agent, being **triggered** by the agent's state. There are two kinds of retrieval: automatic - not under the control of the agent - and voluntary - triggered by the agent.

Retrieval is based on *cues* - especially salient or significant parts of the retrieval information. **Cue construction** is the process of constructing the data used to retrieve a memory. Depending on the type of retrieval (automatic or voluntary) a probe is constructed either by the episodic memory system or by the agent.

**Matching** is the process of searching for episodic memories that are similar to given a retrieval cue. It is dependent on the way the memory is stored and organized.

**Recall** means retrieving the memory from storage and placing it into working memory. After this process completes, the memory becomes available for the agent to use. This process is affected by the rememberer's current environment and state [119]. This process is constructive [109], akin to mental simulation; upon its completion the event's role-fillers and parameter-value bindings are reinstated in the semantic, perceptual and sensory-motor representations.

**Recollective experience** allows the act of remembering to affect future recall. Remembering an episode is a new episode itself, a sort of ‘meta-memory’, which may interfere with the original memory.

The functions of the human episodic memory outlined here constitute a general framework with respect to which all episodic memory models (or their implementations) can be compared.

Each computational model of episodic memory - be it intended to model human episodic memory capabilities or not - has to address all these functions by providing specific algorithms that would implement them.

In the following sections we evaluate models of episodic memory based on how they address the three main functions outlined above: encoding, storage and retrieval. Given the knowledge representation framework we work in, we are mainly interested in symbolic approaches as opposed to connectionist ones.

## 2.2 Models of Episodic Encoding

### 2.2.1 Dynamic Memory

One of the first and most influential computational models of human memory was proposed by Schank [104]. It addresses mainly the encoding and retrieval aspects of episodic memory, identifying different kinds of memory retrievals and proposing memory structures that support them. The idea of ‘experience as expertise’ promoted by Schank provided inspiration for a whole field in Artificial Intelligence - Case-based reasoning [2].

**Reminding** - the process that manifests itself by the availability/recollection of previous memories upon encountering certain stimuli - is a crucial aspect of human memory. Schank asserts that it is the root of human understanding and learning processes. He proposes the study of this phenomenon in order to uncover the structures in memory, his hypothesis being that every time reminding occurs, some memory structure is revealed.

While processing information, humans make assumptions about what will happen next; whenever such an assumption fails, it is recorded. Next time the same assumption fails, the previous failure will be remembered. Schank calls this *reminding based upon event expectations*. There is also *goal-based reminding* - that occurs while trying to understand/predict other peoples' goals - and *plan-based reminding* - when plans are tracked and their quality in achieving the agents' goals is assessed.

In order to encode these assumptions (called *expectations*) Schank proposes various structures in memory at different levels of abstraction.

*Scripts* are sequences of actions that take place in one physical setting (e.g. a doctor's waiting room).

*Scenes* group actions with the same goal that happen at the same time (for example: ordering at a restaurant). Specific memories are stored here, indexed by how they differ from the general actions in the scene.

*Memory Organization Packets* (MOPs) are ordered set of scenes directed toward the achievement of some goal, not inferable from the individual scenes. The MOP's processing function is to provide relevant memory structures (expectations) necessary to understand the input (e.g. correct sequence of scenes). Schank identifies three types of MOPs: personal, societal and physical.

*Thematic Organization Packets* (TOPs) contain abstract, domain independent knowledge organized in terms of goals, plans and themes. TOPs organize collections of memories with the same goals and conditions.

TOPs processing functions include: retrieving the memory of a story that illustrates a point, coming up with adages at an appropriate point, recognizing an old story in new trappings, noticing co-occurrence of seemingly disparate events and drawing conclusions, recognizing something based on partial information and drawing conclusions, transferring knowledge from one situation to another, predicting the outcome of newly encountered situations.

All these structures act as organizers for their respective substructures [65]. MOPs organize scenes and index instances that provide expectations about setting, characters and sequences of scenes. Scenes index scripts and instances that provide expectations about how the scene will unravel in different circumstances. Scripts index cases that provide expectations about variations in a script. Cases can be specializations of MOPs or scenes; they are the instances that show how the specifics of the situation vary from what is expected in a significant way.

Schank does not present specific retrieval algorithms that use the memory structures presented here. In the following section we will review a few systems [64, 50, 68] based on his ideas that propose retrieval algorithms for memory structures like MOPs and TOPs. Acquisition and forgetting of episodes is also not considered.

Another aspect not fully specified in the Dynamic Memory approach is the organization of such a memory. How are MOPs and TOPs organized such that retrieval is scalable when the number of items grows?

Another important idea put forth by Schank is that retrieval and storage of memory items is in fact the same process. This means that the same kind of memory processing that happens when we remember something goes on when we store a memory. Thus, memory items are not stored in their 'raw' form, in isolation of other memory items, but are processed and linked to other structures as a result.

To implement a system based on the ideas of dynamic memory, an adequate knowledge representation formalism and an organization scheme are needed. We will adopt Schank's view that remembering is being caused by expectation failure while processing new information and will propose a retrieval algorithm based on this. Building and updating memory structures will have to be incorporated into the retrieval algorithm.

### 2.2.2 Episodic Encoding in Soar

Soar is a computational cognitive architecture based on Newell's 'Unified Theories of Cognition' [85]. An episodic memory for Soar is described in [87], [88] and [89]. It proposes solutions for all the functional stages proposed by Tulving: encoding, storage, retrieval and usage of the retrieved episodes. The goal of this work is both to create episodic memories for AI agents as well as to model human episodic memory. The model is architectural, in the sense that episodic learning will be part of the underlying cognitive architecture and the episodes will be available to all cognitive tasks. However, what is stored and retrieved will be determined by the task at hand.

The authors identify as main challenges in building such an episodic memory system the selection of cues based on which an episode is retrieved and the constraint that the continually increasing number of acquired episodes imposes on retrieval time.

Episodes are based on the contents of Soar's working memory and are stored as rules, having as right-hand side the contents of the working memory. All features are potentially used for retrieval. Storage tries to minimize memory size by reusing previously seen instances. Episodes are stored each time an action is performed. Notably, there is no organization among stored episodes.

## 2.3 Models of Episodic Retrieval

### 2.3.1 Feature Indexing

Indexing is the traditional solution to the problem of organizing items in categories. The more features of an item are indexed, the more likely it is to be retrieved given one of its features. At the same time, the space and computational effort required by the indexing and retrieval processes grows with the number of indexed features. A common solution to this problem is building manually a set of indexing features. This allows making use of domain knowledge but limits the applicability of the approach. Automatically generating indices tries to alleviate this problem.

**Chef**

Chef [50, 51, 52] is a case-based planner in the domain of Szechwan cooking. A case-based planner differs from a planner that uses libraries of goals and plans, in that it uses episodic memories of past experiences. These memories are used in multiple ways: successful plans are modified to create new plans, failed plans are used to warn the planner of possible problems and offer guidance in dealing with them.

In order to be able to store and retrieve relevant plans, Chef uses a discrimination network to index successful plans by the goals they achieve and the problems they avoid, and to index failed plans by the features that would predict them. Chef also explains why previous plans succeeded or failed so that it can retrieve them when appropriate.

Chef addresses the problem of interacting plan steps that generate unwanted behavior. It tries to anticipate such problems when planning for a goal, then searches for plans that avoid the anticipated problems and satisfy as many of the initial goals as possible. After the plan is built, it is run in a simulator. Successful plans are placed in memory indexed by the goals they achieve. Failed plans are repaired by building a causal description of their failure(s), and using this description to find repair strategies. After being repaired, the plan is stored in memory, indexed by the goals it satisfies and the problems it now avoids. In order for the planner to be able to anticipate problems before they arise, the planner has to determine which features are responsible for a failure and to remember them.

The explanation of a plan's failure is used to search a set of strategies for fixing such failures. These strategies are organized in Thematic Organization Packets, each TOP being indexed by the description of a particular type of planning problem and each organizing a set of strategies that deal with that type of problem. Chef knows about five categories of failures totaling about twenty failure types, all due to interactions between plan steps (e.g. Side-Effect:Disabled-Condition:Concurrent - the interaction between two concurrent plans causes a failure because a side-effect of one violates a precondition of the other). Each such TOP organizes a set of strategies designed to repair the failure. TOPs are stored in a discrimination network, indexed by the features of the explanations they correspond to.

### Cyrus

Cyrus [64] is a computer program able to organize and retrieve a large number of events. This work addresses the following questions: what are the processes for retrieving events from memory, what memory organization do the retrieval processes imply, what are the processes for adding new events to memory and how does memory change as a result of adding new events.

Kolodner identifies some desirable characteristics of a long-term memory: retrieval should not slow down significantly as new events are stored, and retrieval from any category must be able to happen without enumeration. Indexing must be controlled so that memory does not grow exponentially. Memory should keep track of the similarities among events in a category, while indexing the differences between them.

In Cyrus, retrieval is a process of specifying and elaborating contexts for search. Memory organization is based on categories (E-MOPs) for events. Each E-MOP contains generalized information about its episodes and tree-like structures that index sub-MOPs and episodes. Indexing is two-tiered, the first tier being the type of feature, the second the values for that feature.

Search through memory is directed only to categories whose events are relevant by employing retrieval strategies that expand query components, inferring relevant paths through the memory structures. Retrieval starts with index selection (specification of path to follow), the same indices being selected for both storage and retrieval. The process is recursive and ends either with a retrieved event or when there are no more paths to follow. If there are multiple paths to the target event, the shortest one is found. This traversal is a breadth-first search which implements parallel traversal of all appropriate indices.

Retrieval of events with features that are both indexed and unique is accomplished through traversal. It is more difficult to retrieve a target event with *unindexed features* or one that *does not specify a unique combination of features*. The traversal process as described above could continue if *plausible indices* are computed. This is the process of *elaboration* that is given the target concept and the E-MOP that the target concept fits into, then tries to better specify target concept features. For this purpose Cyrus uses *instantiation strategies* like inferring participants to a diplomatic meeting by retrieving representatives of specific organizations, members of known groups, representatives of known countries - in general those entities most likely to have been participants.

Context construction is the process of selecting a category to start searching from. Cyrus uses *component-to-context instantiation strategies* that employ information in the query to infer plausible E-MOPs[4].

Elaboration is not successful when not enough information is available. In such cases Cyrus uses the fact that events occur in the contexts of other events and refers to them. Thus, finding a related event and searching its context might help. *Context to context instantiation strategies* are use to construct alternate contexts for search. For this process to work, E-MOPs must specify the types of events they are related to and the relation between them. This retrieval process trades speed for space. It is faster than enumeration but needs more memory. It's also less accurate than enumeration.

Indices should be discriminative and have predictive power. Organizing events according to their differences from the norm does not burden the memory with unnecessary redundancy and allows retrieval when the query contains the difference. Predictive features tend to co-occur with others. These predictions are used during elaboration. Predictive power of indices must be tracked as memory grows. When computing indices, Cyrus disregards E-MOP norms and features known to be non-predictive.

Generalization is the process that builds the E-MOP's norms and constrains indexing and creation of new E-MOPs, preventing combinatorial explosion of indices. The generalization process in Cyrus assumes the presence of a domain theory. Recovery from bad

[4]This is similar to feature to category remindings in Protos.

generalizations is more complicated as generalizations are norms and no events are indexed by them.

### Retrieval in Protos

Protos [95, 12] is an exemplar-based approach to knowledge acquisition for classification tasks in weak domain theories. It attempts to reason with exemplars in a knowledge-based rather than statistical fashion. Classification is performed by retrieving the most similar prior case and predicting the new example to be of the same class.

Retrieval of the most similar prior exemplar is done in two steps: first, a set of exemplars are selected based on their featural similarity with the new example. Features are weighted by so-called *reminding weights*. The second step (called *knowledge-based pattern matching*) is designed to improve these matches using domain knowledge. Two features are considered equivalent if it can be shown through inference that they imply a common feature.

The use of background knowledge in judging the similarity between a new situation and a memory item is an important characteristic of Protos. Other systems like Cyrus or Chef use only surface features in their similarity assessment. In our work, we want to store and retrieve rich knowledge structures which will require a flexible knowledge-based pattern matching similar to that in Protos.

### Episode-Based Reasoning

Sànchez-Marrè et al. [102] 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.

EBR addresses questions relevant to building an episodic memory module: how to represent temporal sequences of events, how to organize and retrieve them efficiently, and how to assess similarity between such episodes.

Related approaches also include the application of case-based reasoning technology to experience management [7], where the intended target is either human (the *experience factory* [6]) or machine (the *lessons learned* approach [125]).

### 2.3.2 Structural Indexing

The recent emergence of massive data collections consisting of complex structures (i.e. labeled graphs) in domains like bioinformatics [15, 91] and chem-informatics [90] requires database systems support for their efficient retrieval.

There are inherent limitations in existing database infrastructure that make graph-based search techniques infeasible. The indices built on the labels of edges or vertices of such graphs are usually not selective enough to distinguish among such complex structures [132].

New research has focused on addressing these limitations and proposed methods that fall into four categories according to whether they require matching full structures or substructures and they kind of match they perform (exact or approximate):

**full-structure search** - finds structures that are exactly the same as the query structure [16];

**substructure search** - finds structures that either contain the exact query graph (e.g. searching for chemicals that contain a certain substructure) by or are exactly contained by it (e.g. pattern recognition) [106, 114, 131];

**full structure similarity search** - finds structures that are *similar* to the query graph [93, 98, 126];

**substructure similarity search** - finds graphs that nearly match the given query by automatically relaxing it up to a certain threshold; a filtering algorithm that can reduce the number of candidate answers without performing graph-matching is proposed [132].

Given that it addresses the problem of scalable retrieval of labeled graphs using approximate matching (called *similarity search* in the database literature), the work of Yan et al. [132] seems the most relevant to our goal. We will summarize it below.

**Substructure similarity search**

The proposed filtering algorithm (called Grafil), transforms the structure-based similarity measure into a feature-based measure and uses it to remove candidates whose similarity to the query is below a threshold.

Given a database of graphs, a set of indexing features is computed by selecting the most discriminative and frequent subgraphs [131]. These features are then used to index the database of graphs, creating a *feature-graph matrix index* that stores the number of features that appear in each graph.

Given a set of features, the filtering algorithm calculates the maximum number of features that might be missing from a target graph if the query is relaxed by deleting or relabeling one edge. All graphs that differ from the query by more than this are filtered.

Feature sets used in filtering are computed by clustering together features with similar filtering power (defined as the frequency of the feature in the target graph compared to that in the query graph). The resulted feature sets are applied sequentially by the filtering algorithm.

Experimental results show that Grafil with feature clustering filters significantly more of the database compared to just filtering using all features or using just using edge-based filtering. However, this is true when the number of labels is relatively small (e.g around 5); when this number grows (e.g. over 20), edge-based filtering performs nearly as well.

An important aspect of substructure similarity search differs is that it is agnostic with respect to the semantics of the graph labels. These labels act as hard constraints on what edges or vertices match. For example, in matching two chemical structures, a node representing a Carbon atom can only match another Carbon atom. Even though Grafil does not perform matching per se, it is biased toward purely structural matching: it removes certain candidates based on the absence of (structural) features. There are two kinds of solutions to this problem: either allow the testing for the presence of a feature in a graph to use label semantics or transform (i.e. relax) the given query using label semantics and use the same test for feature presence.

Modifying the test for the presence of a feature in a graph to take into account semantics means that a feature like [Carbon, bond, Carbon] will (imperfectly) match a graph containing [Carbon, bond, Non-Metal]. Allowing such an imperfect match will affect the relaxation ratio score, which will now have to take into account not only number of common features, but also how well they match. Testing for feature presence will be more computationally expensive. Also, an investigation is needed to ensure that the modified algorithm does not filter useful candidates.

Relaxing the query using label semantics will change a query feature like [Carbon, bond, Carbon] into [Carbon, bond, Non-Metal]. This will generate an exponential number of relaxed feature queries, affecting the efficiency of the algorithm.

### 2.3.3 Analogical Retrieval

Analogical retrieval is relevant to our inquiry into building an episodic memory module because it addresses the retrieval problem: how to retrieve relevant items from a memory in a scalable fashion?

**Analogy** is the cognitive process of transferring information from a particular subject called the *base* (or source) to another particular subject referred to as the *target*.

Making an analogy requires an *abstract mapping* to be established between two cases or domains based on their common structure (e.g. common systems of relations). This may require re-representation of one (or both) of the domains in terms of the other one (or in terms of a third domain) [63].

Analogy is an inference or an argument from a particular to another particular, as opposed to deduction, induction, and abduction, where at least one of the premises or the conclusion is general. A classical example of analogy is Niels Bohr's model of the atom's structure that parallels the structure of the solar system.

Analogy plays a significant role in problem solving, decision making, perception, memory, creativity, emotion, explanation and communication. It lies behind basic tasks such as the identification of places, objects and people, for example, in face perception and facial recognition systems [44]. It has been argued that analogy is 'the core of cognition' [54].

Analogy-making involves at least the following sub-processes [63]: *representation-building*, *retrieval* of a base for the analogy, *mapping* this base onto the target, *transferring* of unmapped elements from the base to the target, thereby making inferences, *evaluating* the validity and applicability of these inferences, and *learning from the experience*, which includes generalizing from specific cases and, possibly, developing general mental schemas. There are, at present, no models that incorporate all these sub-processes, although individual models focus on one or, in some cases, several of these sub-processes.

Although computational models of analogy-making differ from our approach in that they try to model human analogy making, they address some of the same problems one faces when trying to build a memory module for an intelligent system: the retrieval of complex structures (i.e. memory items) in a scalable manner. We will examine from this perspective some of the most important analogy models.

### The Structure-Mapping Engine

Arguably the most influential model of analogy-making is the Structure-Mapping Engine (SME) [43]. It describes the implicit interpretation rules of analogy and claims that analogy is characterized by the mapping of relations between objects, rather than by attributes of objects, and that those relations mapped are dominated by higher-order relations that belong to the mapping. These rules have the property that they depend only on syntactic properties of the knowledge representation and not on its contents.

### MAC/FAC

MAC/FAC [40] is a model of similarity-based retrieval that tries to capture the psychological phenomena observed in human analogical reminding. It tries to account for the fact that people are extremely good at judging similarity and analogy when given two items to compare, that superficial remindings are much more frequent than structural ones, and that people often experience and use purely structural analogical remindings.

The retrieval model proposed by MAC/FAC consists of two stages, hence the name - ‘many are called, but few are chosen’. The first stage (called MAC) uses a computationally cheap process to select the potential candidates from all the memory items. The selection is based on an estimate of the similarity between the input and memory items, computed as a dot product between their respective vector representations. The second stage (called FAC) uses the Structure Mapping Engine to compute the similarity between such two items based on their structural similarity.

Having two stages in the retrieval process is a known solution to preserve scalability [13]. MAC/FAC uses ‘numerosity’ - the number of local matches (e.g. number of relations of type `implies`) between a probe and a memory item as a computationally cheap way of assessing their similarity. The surface similarity between the probe and all memory items is computed in this manner. The MAC selector filters all these match results so that items that are not within 10% of the best match are removed. The FAC stage takes the selected set of memory items and computes their structural similarity with the given probe. The complexity of the FAC stage is $O(N^2)$, where $N$ is the number of items in the base or target.

MAC/FAC is able to account for several patterns of access exhibited by human subjects [40]. More importantly, it provides a scalable retrieval mechanism for a memory of complex structures.

Some of MAC/FAC's shortcomings include the fact that memory items are not connected to one another. To be useful as an episodic memory retrieval strategy, a scheme like MAC/FAC would need to be coupled with a mechanism that learns from prior analogies, be they good or bad.

#### Other Models

There are a number of other symbolic models of analogy-making[5]. *Derivational-analogy* [22] proposes that the analogy be drawn not with the final solution of the old problem, but with its *derivation*. That is, the important piece of experience in the prior case is not in its final solution, but in how it was reached. This approach was further developed by Veloso [122].

### 2.3.4 Spreading Activation

Spreading activation [8] is a method for searching semantic networks by labeling a set of source concepts with weights or 'activations' and then iteratively propagating or 'spreading' that activation out to other concepts linked to the source concepts or their children. Most often these 'weights' are real values that decay as activation propagates through the network. When the weights are discrete this process is often referred to as marker passing.

[5]For a comparison see [49].

### Retrieval in SOAR-EM

Retrieval in Soar-EM is deliberate, with the cue being provided by the system that uses the retrieved episodes. The system uses spreading activation to retrieve a set of potential candidates from memory. The cue is then compared serially with all candidate memory items and the best-matching episode is retrieved.

Two retrieval algorithms and episodic memory structures have been presented: *instance-based* which stores individual episodes and *interval-based* which groups episodes based on ranges for different values in their description.

The complexity of the instance-based retrieval is $O(nm)$ where $n$ is the size of the retrieval cue and $m$ the number of episodes that share a common feature with the cue (the memory size in the worst case). Interval-based retrieval complexity is $O(n^2l)$ where $n$ is the size of the cue and $l$ is the average number of intervals in each node of the working tree. For both these retrieval algorithms, the authors report a linear increase in retrieval time with memory size, with the interval-based one being 15% faster and requiring only 25% of physical memory compared to the instance-based.

### Context Sensitive Asynchronous Memory

The context-sensitive asynchronous memory [58] addresses the problem of retrieving useful answers from large knowledge bases given under-specified questions. The goal is to obtain this information without knowing how to ask the right questions when exhaustive search is not feasible.

It provides a model of memory retrieval that exploits feedback from the task and environment to guide memory search by interleaving memory retrieval and problem solving.

Memory is a semantic network and retrieval is done through spreading activation similar to the declarative portion of the ACT architecture [9]. The context-sensitive asynchronous memory approach builds upon this foundation, but differs in the following respects: relations in the semantic network are also nodes, the spreading activation process is context sensitive, in that the activation of relation nodes alters the propagation of activation; it maintains a set of active retrieval requests which it constantly and incrementally attempts to satisfy. These features enable the system to focus its search effort on those parts of the knowledge base likely to be relevant while interleaving search with other processes, and allowing updates to the search with new information obtained through reasoning.

This approach has been applied to pure memory retrieval, planning, story understanding, and information retrieval.

## 2.4 Models of Episodic Storage

### 2.4.1 Models of Forgetting

Forgetting - preventing recall of episodes or portions of episodes - is an important aspect of memory maintenance. Forgetting can be attributed to the loss of the actual memory item or to the inability to recall it.

#### Cognitive Models of Forgetting

Empirical studies show that the decline in human memory performance with time or intervening events can be accurately described by a power function [11, 128, 129, 130].

The decay of memory performance can be attributed to two factors: decay of unused information or interference of new and old information. Tulving [119] argues that newer memories prevent the recall of older ones.

#### Pragmatic Models of Forgetting

The mechanism of forgetting has been employed in case-based reasoning systems in order to deal with an increasing number of stored cases.

Kibler and Aha [61] propose two techniques to reduce the number of exemplars stored by an instance-based classifier without seriously affecting the accuracy of a system. The **growth** algorithm stores new instances only if they were incorrectly classified. The **shrink** algorithm stores all instances as exemplars in the first phase and then tests to see whether each instance, in turn, can be correctly classified. Those that can are removed.

Smyth and Keane [110] show that while traditional deletion policies can manage effectively the growth of the case-base from a performance standpoint, they may lead to competence degradation in many CBR systems. The proposed solution uses a model of case competence that guides the acquisition and deletion of cases. Cases are divided into *pivotal cases* that give the system its basic competence and *auxiliary cases* that only contribute to performance.

The *footprint deletion* strategy orders the cases by their utility and chooses first auxiliary cases to delete. If pivotal cases are to be deleted, those that can be solved by the most number of other cases are chosen first, thus minimizing impact on competence. This policy only takes into account competence, disregarding performance.

The *footprint-utility deletion* strategy is intended to deal with the performance aspect. It takes into account the utility of a case when considering it for deletion.

Episodes are more complex than simple cases in that they are intended to encapsulate much more knowledge and be relevant in various situations that cannot be anticipated at storage time. In this case, the decision of whether to store a new episode is more complicated.

### 2.4.2 Memory Reorganization

Besides forgetting (i.e. removing items from memory), an agent can also reorganize its memory during the course of its functioning. Such a reorganization can be performed periodically (either based on time or on the number of instances organized by a memory structure like in [64]) or be triggered by failure. In this case, the system recognizes failure as an opportunity to learn.

Protos [12] falls into this latter category. Upon failure (i.e. making a wrong classification) it does two things: first, it decreases the reminding weights responsible for the wrong prediction. This is common in most learning systems. The second and more interesting thing it does is the creation of *difference links* - links between the failed and correct exemplars, labeled with the differences between the two. These links are used during classification to suggest other exemplars for consideration. Effectively, Protos is creating another indexing structure that links exemplars. This indexing structure is *localized* - it is accessible only from the linked episodes - and *knowledge rich* - encoding the full set of differences between the failed and correct episodes. Traversal of such a difference link is done only when all features of a new case match those in the link. For more complex representation formalisms (e.g. conceptual graphs), new algorithms for synthesizing such difference links and deciding when to traverse them will need to be devised.

## 2.5 Episodic Memory Uses in Intelligent Systems

Arguably, episodic memory capability is a central part of human intelligence and any system that tries to reproduce human-level intelligence or implement super-human intelligence would need to have such capabilities. This section reviews work on the uses of episodic memory in intelligent systems and its potential benefits.

Nuxoll and Laird [89] put forth a comprehensive list of *cognitive capabilities* that an episode memory might support. They include:

**sensing** : determining the *familiarity of a situation* (including noticing novel situations and detecting repeated sensing of the same situation) and *recalling useful information* from a previous situation (they call it virtual sensing).

**reasoning** : *action modelling* (learning the immediate effects of actions), *environment modeling* (learning how the world changes independent of the agent's actions), *pre-*

*dicting the success/failure of actions*, *managing long term goals*.

**learning** : *retroactive learning* (e.g. when no longer under time constraints), *reanalyzing knowledge* (in the light of new information), and *explaining behavior* (e.g. by episodic replay)

Mueller [80] proposes that study of **daydreaming** - the spontaneous recall of past or future (i.e. imagined) experiences. The author postulates the important roles daydreaming plays in human cognition including: *plan preparation* in anticipation of future situations, *plan rehearsal* by imagining situations and how a plan would unfold in those situations, *learning from failures and successes* by recalling past experiences, *support for creativity* e.g. by generating fanciful possibilities that can lead to the discovery of novel solutions to a problem.

In the proposed computational theory of daydreaming stored episodes are used to reduce the need for search in the future and to increase knowledge accessibility.

### 2.5.1 The Basic Agent

One of the few episodic memories integrated into an intelligent agent is described in [124, 123]. The Basic Agent is an attempt to create a conscious, mind-like AI artifact functioning in a simulated dynamic environment. The agent uses planning and replanning, reasoning, action execution, limited natural language understanding and generation, symbolic perception, episodic memory and reflection.

It employs two episodic memories: one for managing the dialog with the user and one for guiding the planner subsystem. Memories alone do not do anything; a reflection process is necessary to process episodic memory and perform abstraction, forgetting, loop recognition, etc.

The episodic memory inside the natural language module contains a set of *realizers* - daemons that recognize concepts at a higher level of abstraction than raw episodes (e.g. ‘reach’ = change of location where the current location is that of a known object; ‘return’

= change of location to a previously known one; ‘pass’ = change of location where the reference object lies between the origin and destination). There is also a *repetition recognizer* that picks up repeated actions. General world knowledge is stored in the same module with episodic memory, since the mechanisms used for both are similar. The stored episodes are rather primitive and are only used to generate descriptions of the agent’s actions.

The second episodic memory is used to guide the functioning of the planner. In case there is an impasse in planning, the system begins selective backtracking, using the episodic memory. All planning events are recorded here, allowing the control to retreat to a prior decision point to resolve the problem.

The basic agent was implemented as a simulated robot submarine operating in a two-dimensional world about which it has only partial knowledge. It responds to natural language commands using a vocabulary of about 800 common English words by invoking its temporal task planner to synthesize a plan, which is then executed. The agent can form and retain compound future plans, and re-plan in response to new information or new commands. No evaluation of its performance of these tasks was carried.

### 2.5.2 Soar EM

The Soar system was extended to incorporate an episodic memory [87]. Retrieved episodes are used to guide action selection by providing an evaluation of alternative actions. This was tested on a simple grid-world [87] in which the agent can move in four directions, each cell having an associated value. The purpose of the system is to maximize the score, computed as the sum of the values of cells traveled. The agent does not know how its actions move it through the world or their associated value. The system uses episodes as knowledge to help the agent travel in this world. An agent with an unbiased match function (i.e. number of features matched) quickly achieved better than random performance but did not improve beyond that as the number of episodes increased. The authors blame the irrelevant features used as cues. Using an activation-biased matching function [88] showed a significant improvement over the unbiased match.

In a more recent implementation [89], the Soar episodic memory is used to extend the cognitive capabilities of the agent along several dimensions. Experiments in a domain called TankSoar (similar to the grid world in the previous implementation) was presented. Nuxoll and Laird show that an episodic memory contributes to the improvement in performance in:

- *action modeling*: the agent learned to predict the immediate outcome of its actions;
- *virtual sensing*: the agent can recall details from a past situation that might be relevant to the current situation (e.g. where an energy source was encountered);
- *learning from experience*: the agent learned tactics in the given domain, outperforming a hand-coded agent. It learned to predict the enemy's actions as well as valuable attack tactics (e.g. back away after firing).

The retrieval time in Soar EM grows linearly with memory size. To address the scalability issue posed by this the authors determined the maximum number of episodes that the system can effectively process and restricted the use of episodic memory to only those problems that meet that limit. This is similar to the 'scaling up by dumbing down' approach [79].

Extensions to the system to more complex tasks (like repetition detection, goal tracking) and domains will require reducing the size of memory in order to accommodate episode complexity. Future experimental evaluations will need to investigate how this will affect performance on the given task.

Computing similarity between episodes encoding complex knowledge will have to be addressed. The semantics of instances appearing in an episode will need to be considered during this process.

### 2.5.3 ISAC

Dodd [35] developed a memory system for a cognitive robot (ISAC), including both short-term (e.g. storing perceptual information) and long-term memories (storing information obtained from past experience). The long-term memory system is divided into procedural, semantic and episodic memories.

The Episodic Memory is intended to provide the robot with the ability to learn from past task performances. An Episode corresponds to the time period over which the goal of the robot does not change. Episodes are represented using the agent's semantic memory. Retrieval is based on 'the rational approach to memory design' of Anderson [10] which tries to balance the benefits of retrieving a memory unit against the costs of this operation. The probability that a previous episode is relevant is based both on the current context as well as on its history (i.e. emotional salience, age). The retrieval algorithm is linear in the number of stored episodes, just like Soar-EM. Memory decay is incorporated into this framework and a memory item is removed when its history value decays past a certain threshold. Episodes have a flat structure and are stored in a traditional database.

While the authors present some small scale experiments showing how each memory module accomplishes a few individual tasks, a more extended evaluation showing whether having such memory modules improves the performance of the robot on a given set of tasks is needed.

### 2.5.4 Temporal Sensorial Information

#### The Wearable Remembrance Agent

Rhodes [99] developed a wearable agent that continuously stores the wearer's context received from various sensors and reminds the wearer of potentially relevant information based on his/hers current context. For example, while attending a conference the context might include the name of the speaker, the location of the talk, names of persons sitting nearby; the suggestions in this context are papers related to the one currently being presented, when the wearer last met the speaker, etc.

Potential benefits of such an agent include: availability of useful or supporting information relevant to the current task, contextualization of the current task in a broader framework, retrieval of information that leads to the discovery of useful information.

### Temporal Case-Based Reasoning

More recently there has been a lot of interest in the CBR community in incorporating temporal information in the stored cases.

Ram and Santamaria [96] recorded raw data from prior actions to improve navigation. Cases consist entirely of quantitative data and do not have clear boundaries. The stored cases are similar to episodes in that they have a temporal extent, the agent having to determine when and what to store. Memory size is kept fixed by merging the most similar two cases when needed. A matching process can use multiple such contiguous cases. The system improved its navigation performance by using the stored cases in an action modeling approach.

Ma and Knight [75] propose the a framework for historical case-based reasoning (HCBR). They argue in many CBR systems including prediction, explanation, planning, etc. the history of cases, rather then distinct episodes, are important. Their approach is based on a general temporal theory that allows both time instances and intervals as primitives elements. The framework uses *fluents* (propositions whose truth values depend on time), *elemental cases* (collections of fluents), and *case histories* (sequences of cases).

Similarity for case histories is defined as having two components: non-temporal (based on elemental cases) and temporal (based on the graphical representation of the temporal references using traditional graph similarity measurements).

## 2.6 Episodic Memory vs. Case-Based Reasoning

Case-based reasoning [2] is the process of solving new problems based on the solutions of similar past problems. Computational approaches to episodic memory and case-based reasoning systems share not only similar goals, but also a common ancestry: Schank's model of dynamic memory [104] was the basis for the earliest CBR systems (e.g. Kolodner's CYRUS [65] and Lebowitz's IPP [69]).

However, Episodic Memory and Case-Based Reasoning differ in significant ways [67].

The most important difference between them is that, usually, episodic memories are *part of a larger system* (e.g. Soar EM is built on top of the Soar cognitive architecture), whereas most CBR applications are *standalone applications*. Episodic memory research focuses on building models or storage and retrieval and leaves adaptation of retrieved episodes to other components of the overall system. CBR systems address the adaptation problem as both performance and competence at the given task depend on this.

Episodic memory retrieval and adaptation is *used in conjunction with other reasoning mechanisms* (e.g. Soar EM uses the chunking mechanism in Soar to store and retrieve episodes), whereas case reuse and adaptation is *the primary reasoning mechanism*.

Episodes are *multifunctional* in that they can be *used for a variety of tasks* like improving performance, improving competence, generating explanations, learning, etc. Cases have usually *more restricted, domain-dependent uses*, like action selection, problem solving, etc. The architectural constraints imposed by these differences will be explored in the next chapter.

## 2.7 Chapter Summary

In this chapter we looked at the basic characteristics of human episodic memory, how it differs from the semantic memory and what its main functions are. We also examined different models of episodic encoding, retrieval and storage and how different systems use episodic memories in their functioning. Each of the examined systems exhibits some desirable characteristic(s) of a memory system; however, no single system has them all.

Even though a number of intelligent systems use memory in their functioning only a few employ an episodic memory per se. Those that do [64, 50, 124, 88, 35] embed the memory module inside the system that uses it, making it hard to port to other systems and to study its effect to the system's overall task.

Unlike these approaches, we propose to build a memory module that is *separated* from the host system. This should both allow research to focus on memory organization issues in isolation of the system using the memory module and simplify the overall design of the system. To achieve this separation, we need to devise an interface through which memory will interact with the system and a generic representation of events.

The set of tasks an intelligent systems accomplishes grows in size and *complexity* every day. A memory module for such an intelligent system needs to be able to deal with the complex experience acquired in the process and use it for different reasoning tasks, some of them unanticipated at the moment they were stored. Simple vector representation schemes are not able to do this, but structured representations need to be employed. Flexible storage and retrieval mechanisms for these representations need to be devised. Indexing schemes that capture similarities in context need to be developed.

The life expectancy of intelligent systems grows as well. This adds another dimension to the task of building a memory module: *scalability*. Scalable retrieval mechanisms are needed. Serially searching large memory structures is not feasible.

Detailed evaluations of the benefits and costs associated with using an episodic memory in a system in different domains and for different tasks need to be carried out.

Chapter 3

# A Generic Memory Module for Events

## 3.1 Motivation

A lot of today's intelligent systems do not use past experience in their functioning. Take, for example, Project Halo [57] whose goal is to develop tools that would enable subject matter experts to convey their knowledge (i.e. concepts, relations, procedures) in various domains to an intelligent system. These systems are intended to answer AP-level questions in those respective domains (e.g. chemistry, physics, biology).

A student preparing to take the AP test on a particular subject would also learn domain concepts, relations and procedures. In addition, he/she would also solve some AP-style questions and use those learned concepts in context. Subsequently, the student will be able to recall these experiences and use them for various purposes like:

**goal tracking** - assessing whether they have covered all the material in the syllabus,

**solution adaptation** - solving similar problems by adapting previous solutions,

**failure avoidance** - avoiding getting stuck in solving a problem using the wrong concepts by noticing similarities between the current situation and a previous failure.

Being able to store and reuse past experience can enable an intelligent system to accomplish these tasks as well. However, relying just on experience will not achieve a broad coverage in terms of types of tasks: experience needs to be used in conjunction with other reasoning mechanisms. That is why we need the ability to add episodic memory functionality to intelligent systems.

## 3.2 The Need for a Generic Memory Module

Today's knowledge-based systems are complex software applications and the ability to develop them in a modular fashion, using generic, reusable components is essential. This requirement is even more stringent given the growing complexity of the tasks they achieve and the increase in their expected life-span.

This need for *generic modules* in the development of knowledge-based systems has long been recognized. For example, E-MYCIN [120] employed a reusable inference engine and separated domain specific knowledge (i.e. rules) from it.

We propose to separate the *episodic memory functionality* from the system using it and build a *generic, reusable memory module* that can be attached to a variety of applications in order to provide this functionality [115, 117].

The development of such a generic, reusable memory module will allow *easy portability* to different systems and applications. It could enable systems that were not designed to rely on a memory system to benefit from it, requiring only minimal changes. Separating memory functionality from the system that uses it should also *reduce the overall complexity* of the system since it will not have to be concerned with this any more.

Having such a reusable memory module should allow *research to focus on the generic aspects of memory representation, organization and retrieval* and its interaction with the external application.

Separating the memory module from the application requires several things: the interaction between memory and the application has to be channeled through a *well-defined and flexible interface* and memory has to be able to *represent, organize and retrieve* a wide variety of events in a wide variety of contexts.

An external application using such a generic memory module will use the retrieved memories differently, depending on their task. It is important to note that we do not propose a generic problem-solving solution for such tasks as this would require domain specific knowledge (e.g. for adapting prior experience); rather, the episodic memory module will have a supporting role in problem solving, providing access to prior experience that might be useful in the current context.

## 3.3 General Memory Requirements

We separate the memory requirements into two categories: internal, that any memory should satisfy, and external, related to their interaction with external applications.

In this section we look at general memory requirements using Tulving's list of memory functions [119] (see also Section 2.1.3). Next section will examine application-level memory requirements.

### 3.3.1 Encoding Requirements

A generic memory module will have to represent a wide variety of episodes. These episodes will need to be organized in memory structures such that they can be retrieved when needed. Therefore, a generic memory module needs to provide:

**a generic representation of events** that can be used with different types of events (e.g. different temporal extent, different granularity of representation, incorporating qualitative as well as quantitative knowledge, causality, temporal as well as spatial information), in different domains (e.g. planning, problem solving)

**a domain independent organization** of these events that supports flexible retrieval; given that such a generic memory needs to be able to retrieve episodes based on variety of queries, unanticipated at storage time, the memory organization needs to allow any piece of knowledge encoded in an episode to be a retrieval cue.

### 3.3.2 Storage Requirements

Memory should be able to store a large number of episodes, acquired during its functioning, and do so efficiently. Removing episodes to maintain memory size manageable can be employed, but the costs of doing so need to be taken into account.

**efficiency** - memory should provide efficient storage; one way to operationalize this requirement is to require a constant (or almost constant) storage time. In this case the system might need to revisit the stored memories in order to better integrate them into the overall memory structure.

**scalability** - memory should be able to accommodate a large number of episodes without a significant decrease in performance. As the intelligent system matures, it will acquire more experience that needs to be stored. The importance of scalability grows with the life expectancy of the system.

**competence preservation** - any forgetting strategy used should preserve the competence of the system within some specified bounds. Forgetting past memories is one way to maintain a certain memory size, but it has to take into account the cost in terms of competence lost through forgetting.

### 3.3.3 Retrieval Requirements

The retrieval algorithm of a generic memory module for events should have the following characteristics:

**accuracy** - memory should return experiences relevant to the situation at hand. For example, given a planning problem, memory should return similar planning problems from memory.

**efficiency** - memory should provide fast recall of stored items. Efficiency is concerned with the retrieval time, in isolation of memory size.

**scalability** - the number of stored episodes should not directly affect retrieval efficiency. This requirement concerns the increase of retrieval time with memory size. The importance of scalability grows with the life expectancy of the system.

**content addressability** - memory items should be addressable by their content. This requirement is meant to allow the external application to formulate flexible queries.

**flexible matching** - memory should recall the correct previous episodes even if they only partially match the current context.

## 3.4 Application Requirements for a Memory Module

Given that such a generic memory module is intended to be implemented as a stand-alone application, it needs to provide a clean but flexible programming interface (API) that external applications can use.

When dealing with complex knowledge structures, the results of querying memory are as important as an explanation of *why* they were retrieved (e.g. how the query was judged to be similar to the retrieved memory items). This similarity might not be obvious from the retrieved items alone.

### 3.4.1 A Flexible Interface

The interface implemented by the memory module needs to be flexible so as to allow various types of queries to be formulated. For example, a surveillance agent might query memory with a sequence of actions and request prior plans that are similar to the observed sequence of actions. In contrast, a planner might use the description of a planning problem in order to retrieve a prior planning episode.

Given our goal of building a memory that can be attached to a variety of applications solving different tasks, it is important not to restrict the types of queries allowed by this interface.

### 3.4.2 Explaining the Retrieved Results

The match between a query and a memory item is complex: e.g. it might be partial, it might use semantic information or transformation rules to resolve mismatches between the query and a memory item. Therefore, besides the actual query result, memory needs to provide feedback to the application on *how such a query matched* a memory item.

This feedback might include what part of the query matched the memory item, what part of it did not, what part of the memory item matched the query, what part did not, and the set of correspondences (i.e. mappings) between the query and the memory item.

Such knowledge could be used by the external application in accomplishing its task. For example, a planner that calls memory with the description of a new problem could use the matched part of the query to assess the effort required to adapt the retrieved plan and the set of correspondences as a guidance for the adaptation process.

## 3.5 An Analysis of SOAR-EM

Given that Soar-EM [88] (see also Section 2.3.4) is the most similar attempt to build a generic episodic memory module, we will examine how it fares with respect to the set of requirements presented here.

In terms of **encoding**, Soar-EM uses a generic episode representation (i.e. rules) and a domain independent organization of episodes (i.e. a list of either episodes or intervals).

Episode **storage** is efficient: for the *instance-based* representation it only requires linking to the elements in the working memory tree, an operation independent of memory size, while for the *interval-based* storage it requires a search through memory to find the appropriate intervals to store this episode; this operation is dependent on memory size. The two organization schemes provide a trade-off between memory size and scalability: instance-based storage is efficient but not scalable, while interval-based storage reduces the memory size at the expense of increasing storage time. There is no forgetting strategy - rather, the authors adopt the reverse strategy, by limiting the complexity and size of problems tackled to only those that can be solved given the maximum memory size that can currently be stored.

**Retrieval** seems accurate (using memory improves performance on three different tasks), but in the worst case is linear in the number of stored memory items. Memory items are content-addressable, but the matching algorithm does not take into account semantic information and cannot handle mismatches between the cue and memory items.

## 3.6 Chapter Summary

In this chapter we argued that intelligent systems need to use memory in order to be able to track their long-term goals, avoid failures and solve problems by adapting previous solutions. Adding episodic memory functionality to such systems should be done in a modular, reusable way.

We propose separating the episodic memory functionality from the system, and implementing it as a generic memory module that can be attached to various applications. Benefits of such a separation include increased portability, a reduction in the complexity of the overall system as well as allowing research to focus on studying memory organization and retrieval in isolation of a specific system.

We presented a set of requirements that any memory module should try to follow. General memory requirements include providing a generic *encoding* and organization for events; efficient, scalable and competence preserving *storage*; accurate, efficient, scalable and content-addressable retrieval. At the application level, such a memory module should provide a flexible API to external systems.

The next chapter will present our proposed implementation of such a generic episodic memory module.

# Chapter 4

# An Implementation of a Generic Memory Module for Events

In this chapter we present our proposed implementation of a generic memory module. We look at the implementation choices made for each of the three episodic memory functions (encoding, storage, and retrieval) as well as the programming interface provided to external applications.

## 4.1 Episodic Encoding

### 4.1.1 Episode Determination

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

### 4.1.2 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 an 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**.

#### Episode Dimensions

**Context** is the general setting in which an episode happened. In a planning application the context 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).

The idea of indexing episodes based on the different kinds of information encoded by them is not new: e.g. Chef [50] indexed plans both by their goals and by their failures and Episode-Based Reasoning [102] encodes a problem description, a solution-plan and a solution-evaluation in an episode, similar to our three dimensions.

The separation of an episode into these dimensions is left to the application using memory. We therefore assume the memory module receives an already partitioned episode for storage or retrieval.

#### Knowledge Formalism

The knowledge formalism used to represent the episodes is the Component Library (referred to henceforth as CLib) [14]. It is an upper ontology of composable concepts, consisting of about 700 general concepts such as Transport, Communicate, Enter,etc. These concepts are related to one another using 80 binary semantic relations such as agent, object, causes, size [14]. A knowledge base in this formalism can be thought of as a set of triples, where each triple consists of two frame instances connected by a relation.

We make the assumption that an application using the proposed memory module is sharing an ontology with the memory module. This can be achieved by the external application adopting the upper ontology provided by CLib and extending it with domain specific concepts and relations. This does not conflict with our goals of building a generic memory module as there is nothing in the current implementation of memory that prevents it from working with a different ontology, as long as it provides inheritance and can be translated into a triple format.

**Episodes as Sets of Triples**

An Episode is represented in our knowledge formalism as a collection of instantiated frames linked by relations. Usually there does not exist a reified concept corresponding to an entire episode.

Another way to look at an episode is to view it as a conceptual graph [111], where nodes are instances of frames, and edges connecting them are relations. The three dimensions institute a partition on this graph.

The semantics of frames and relations is defined by the CLib. In this way a direct connection between Episodic Memory and Semantic Memory (i.e. CLib) is established.

For example Figure 4.1.2 depicts graphically describes a planning problem (called *Package-Deliver1)[1] in the Logistics domain [122] involving the delivery of a package (called *Package5) from a post-office (called *Post-Office6) to another one (called *Post-Office2). Both post-offices are located in the same city (called *City7),

[1]A note on representation: we use a triple notation with the following properties: each triple has the form (instance-1 relation instance-2); a star in front of a name means the name refers to an instance of the class with that name; in order to distinguish between different instances they have been numbered.

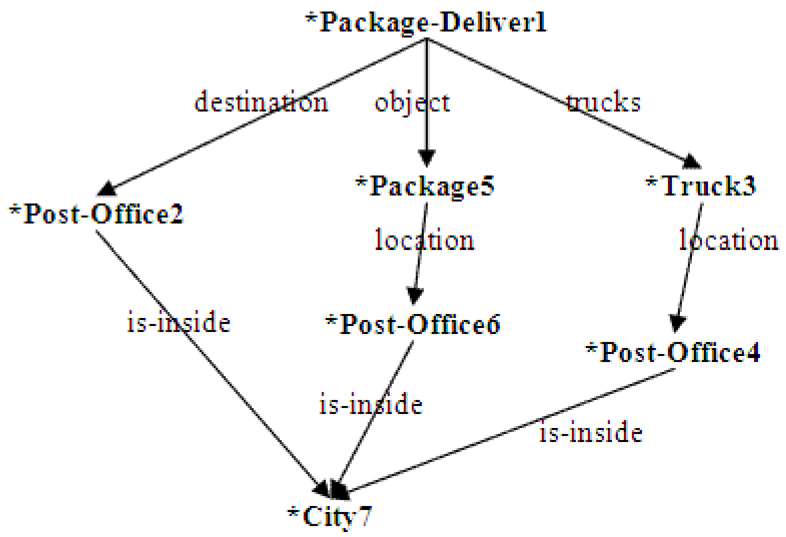


Figure 4.1: Graphical representation of a planning problem description from the Logistics domain

which contains another post-office `*Post-Office4`. Figure 4.2 shows the same planning problem represented as a set of triples.

**Benefits of Episode Representation using Different Dimensions**

We propose using these 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 the results 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).

```
(*Package-Deliver1 destination *Post-Office2)
(*Package-Deliver1 trucks *Truck3)
(*Truck3 location *Post-Office4)
(*Package-Deliver1 object *Package5)
(*Post-Office6 is-inside *City7)
(*Package5 location *Post-Office6)
(*Post-Office2 is-inside *City7)
(*Post-Office4 is-inside *City7)
```

Figure 4.2: The same planning problem from the Logistics domain represented as triples

```
(Package-Deliver destination Post-Office)
(Package-Deliver trucks Truck)
(Truck location Post-Office)
(Package-Deliver object Package)
(Post-Office is-inside City)
(Package location Post-Office)
```

Figure 4.3: The set of remindings for the above planning problem

**classification** - determine 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.

**episode recognition** - recognize a prior episode (or one similar to a prior episode) being executed. This corresponds to retrieval based on episode contents (i.e. observed actions) and adapting the context of retrieved episodes.

The semantics of individual actions (e.g. their applicability conditions and goals they achieve), as well as knowledge about the state of the world is represented using the concepts and relations in the CLib.

## 4.2 Storage

### 4.2.1 Memory Indexing

Episodes are stored in memory unchanged, with no generalization, and are indexed for retrieval. We have adopted a multi-layer indexing scheme similar to mechanisms found in systems such as: MAC/FAC [40], Börner [21] and Protos [95]:

**a shallow indexing** step in which each episode is indexed by all its *feature types* taken in isolation. The shallow indexing is meant to quickly select a set of episodes that might be relevant for the current query. The most promising ones will be inspected closer to determine their similarity with the given query.

**a deep indexing** step in which episodes are linked together by how they differ *structurally* from one another. It provides access to additional episodes that were relevant in similar situations in the past.

#### Shallow Indexing

The shallow indexing scheme indexes an episode separately on each of its three dimensions: context, contents and outcome. For each such dimension, a set of *generalized triples* called **remindings** is computed by generalizing the two nodes in each triple to their respective types. Duplicate generalized triples are ignored. Figure 4.3 list the remindings corresponding to the planning problem in Figure 4.2.

Triple generalization discards the structural information contained in an episode by replacing instances with their respective types. This structural information will be considered during the second step of the retrieval process, the semantic matching phase. Triple generalization is fast (the type info is usually contained in the episode representation) and allows for fast comparison between two generalized sets of triples.

We define the **reminding-weight** as a measure of how discriminative the reminding is. This weight is computed as the inverse of the number of episodes that are indexed by the given reminding:

$$\text{reminding-weight(rem)} = 1 \,/\, |\text{reminded-episodes(rem)}|$$

The shallow indexing step is similar to the MAC stage of MAC/FAC retrieval model [40]. MAC computes a *content vector* for each memory item by computing how many relations of each type appear in that memory item. There are however important differences: the approach we propose takes into account *the types of arguments* for those relations and *disregards their number*, while MAC considers *only relation types* and their number, disregarding the types of their arguments.

Both these approaches seem to fit their purpose: MAC is intended to retrieve analogical matches, where the set of relations is important while their arguments might be very different (e.g. planet and electron in the classical analogy of the atom as the Solar system); our shallow indexing scheme is intended to select the most similar prior episodes (which prefers matching types for relation arguments), not necessarily intricate analogies.

A possible extension to this indexing scheme is to generalize a triple beyond just the types of its two instances, by looking at a fixed number of their superclasses and subclasses. This has the advantage of matching potentially more diverse triples in the input, at the expense of increased memory size and retrieval time. More specifically the number of such generalized triples will be proportional to the square of the number of considered superclasses/subclasses, while retrieval time will increase due to the increase in number of triples used to query memory and the number of indices in memory. If we generalize a triple to all its superclasses/subclasses, this algorithm becomes the MAC stage without the counting of relations. We chose not to implement this extension due to its negative effects on performance.

We have, however, implemented a less selective indexing scheme (and more generous in selecting potentially useful episodes) that only looks at the types of the instances in each triple, disregarding the relation. This was implemented as a fall-back for the case when the original reminding scheme does not return any episodes. The intuition is that similar episodes are more likely to share an instance of a certain type than two instances of different types. If such an instance is shared, the relations involving it are likely to be similar as well.

Remindings are stored in a hash-table and matching is exact.

Indexing based on remindings is quite similar to the edge-based indexing in substructure similarity search [132] (see also Section 2.3.2). Edge-based indexing can be viewed as a degenerate case of feature-based indexing using a filter with single edge features. Yan et al. [132] show that when the number of graph labels (i.e. nodes and edges) grows, edge-based performs nearly as well as the filtering approach based on feature indexing. Given that our knowledge base contains hundreds of concepts and relations (see Section 4.1.2) and considering that all of them could potentially be labels, we expect this to be the case for episodic memory applications.

The advantages of using feature-based filtering are most prominent when the number of labels is small and the number of features is large.

### Deep Indexing

Shallow indexing provides fast access to individual episodes given a set of triples. Deep indexing is intended to link episodes to other episodes by *how they differ structurally* from one another. We call such links **difference links** as they are similar to those in Protos [95]. Such links will only be followed if the current memory cue has the same difference when compared to one of the linked episodes.

Creating difference links is a way to store important structural differences between episodes in memory. Such qualitative differences are computed during memory retrieval and stored when feedback from the external application suggests that a retrieved episode was not appropriate for the task. For example, if `*Episode1` was retrieved by memory when queried with stimulus `S`, and the external application deems `*Episode1` not appropriate (e.g. by using it for some task that results in a failure), and `S` in to be stored as part of

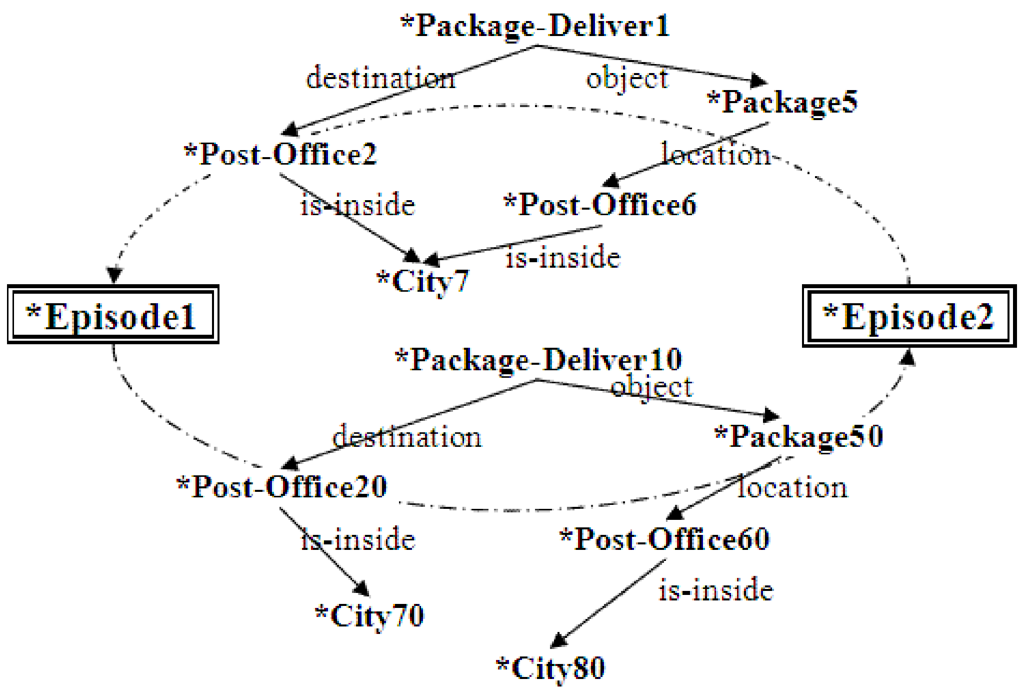


Figure 4.4: Difference links connecting two episodes. *Episode1 represents a Package-Delivery where the original city of the Package is the same as the delivery city. *Episode2 represents a Package-Delivery in which the package has to be delivered to a location in a different city than that of origin.

⋆Episode2, the differences between S and ⋆Episode1 computed during retrieval are installed as differences between ⋆Episode1 and ⋆Episode2.

Figure 4.4 presents an example of difference links. ⋆Episode1 represents a Package-Delivery where the original city of the Package is the same as the delivery city. ⋆Episode2 represents a Package-Delivery in which the package has to be delivered to a location in a different city than that of origin.

### 4.2.2 Forgetting

Restricting the number of stored episodes is an effective way to achieve scalability for a long-term memory. Various techniques have been proposed for case deletion that address competence and performance preservation (see Section 2.4.1).

There are task-specific techniques that measure the performance on a given task in order to decide whether to keep a memory item or not. Because they are task specific, we decided to let the application using the generic memory module implement them. Therefore, in our current implementation, both the decision on *when to store* a new episode and *when to delete* an old one are left to the external application.

Even though the current memory implementation does not implement forgetting, there is some built-in support for task-independent memory management in the form of redundancy detection. Time-based decay and episodic salience are natural extensions to our memory module and we plan to add them and investigate their influence in the future.

## 4.3 Matching

A robust memory needs to employ a flexible matching algorithm, such that old situations are still recognized under new trappings. For this purpose we use a flexible semantic matcher [134] that can handle a broad range of misalignments between the source and target concepts.

### 4.3.1 A Flexible Semantic Matcher

We build on Yeh's work on flexible semantic matching [134]. His semantic matcher takes in two representations (equivalent to conceptual graphs [111]) and uses *taxonomic knowledge* and *transformation rules* to find the largest connected subgraph in one representation that is isomorphic to a subgraph of the other.

The taxonomic knowledge includes both concepts and relations and is expressed as components in the CLib. Domain knowledge can be represented by extending the CLib and will be thus employed seamlessly by the matcher.

Yeh's matcher employs a set of about 200 transformation rules to shift the representations in order to improve the match. These transformations might enable other subgraphs to match isomorphically, which in turn might enable more transformation rules to apply, and so on until the match improves no further.

Transformation rules are instances of the *transfers through pattern* [70] which has the following form:

$$C_1 \xrightarrow{r_1} C_2 \xrightarrow{r_2} C_3 \Rightarrow C_1 \xrightarrow{r_1} C_3$$

where $C_i$ is a concept and $r_j$ is a relation. Example rules include[2]:

**part descension** - acting on a whole means also acting on its parts

$$Event_1 \xrightarrow{object} Entity_2 \xrightarrow{has-part} Entity_3 \Rightarrow Event_1 \xrightarrow{object} Entity_3$$

**transitivity of `has-part` relation**

$$Entity_1 \xrightarrow{has-part} Entity_2 \xrightarrow{has-part} Entity_3 \Rightarrow Event_1 \xrightarrow{has-part} Entity_3$$

Transformation rules are intended to breach the representational gap that might exists between base and a target concepts. This library of transformation rules is based on the Component Library [14] upper ontology and has been used to improve matching in the domains of battle space planning [136], office equipment purchasing [137], and word-sense disambiguation and semantic role labeling [135].

[2]For a complete set of transformation rules see [134]

### 4.3.2 Semantic Matching Uses in Memory Retrieval

The memory module uses the semantic matcher to assess the semantic similarity between two graphs (expressed as a *numeric score*) and also to determine the *qualitative similarities* (i.e. the common subgraphs) and *qualitative differences* between them (i.e. the part of the graph other than the common subgraph).

#### Match Score

There are multiple possibilities for computing the similarity score between concepts, depending on which concept is taken as a reference:

$$semsim(C_1, C_2) = \frac{1}{|C_1|} * \sum_{t_i \in C_1 \sim C_2} score(t_i)$$

measures how well $C_1$ matches $C_2$ with respect to $C_1$,
where $C_1 \sim C_2$ represents the isomorphic mapping from $C_1$ to $C_2$,
$t_i$ represents the isomorphic relation between a vertex in the $C_1$ graph and its corresponding counterpart in $C_2$;
$score(t_i)$ measures how well the two vertices match and is a number between 0 and 1 provided by the matcher.

This similarity measure is not commutative as

$$semsim(C_1, C_2) \neq semsim(C_2, C_1)$$

A commutative similarity metric can be defined as:

$$sim(C_1, C_2) = semsim(C_1, C_2) * semsim(C_2, C_1)$$

### Qualitative Similarities and Differences

The similarity score between the two concepts gives only a *quantitative* measure of their similarity. Often times a *qualitative* characterization is needed. For example, when assessing the similarities between the goals of two plans, we are interested in what the *structural differences* between them are. These differences can be used to index memory and retrieve more relevant episodes.

Based on the common isomorphic subgraph returned by the matcher when matching $C_1$ and $C_2$, we compute:

- $C_1$-matched = the set of vertices in $C_1$ that have corresponding vertices in $C_2$
- $C_1$-unmatched = the set of vertices in $C_1$ that don't have corresponding vertices in $C_2$
- $C_2$-matched = the set of vertices in $C_2$ that have corresponding vertices in $C_1$
- $C_2$-unmatched = the set of vertices in $C_2$ that don't have corresponding vertices in $C_1$

### Application Control

A memory module that can be attached to multiple applications has to provide those applications as much control over its functioning as possible. We tried to do that by making the similarity function a parameter to the memory module.

External applications can provide their own such function, as long as it is monotonically non-decreasing. That is:

$$\forall\ M_1 \subseteq M_2, \text{sim}(M_1) \geq \text{sim}(M_2)$$

where $M_i$ is a match result and sim is a similarity function:

$$\text{sim}(M)\text{: } \mathbf{M} \rightarrow \mathbf{R}_+$$

defined on the set of all possible match results **M** with non-negative real values.

This requirement comes from the bias that is built into the memory retrieval algorithm. The algorithm tries to maximize the size of the matching set of triples $M$.

This allows the external application to customize it according to its needs (e.g. using a domain dependent feature weighting scheme).

Additionally, any semantic matcher that works on conceptual graphs (e.g. the Structure-Mapping Engine [43]) can be used in connection with the our episodic memory module.

## 4.4 Retrieval

### 4.4.1 Retrieval Algorithm

During retrieval, shallow indexing will select a set of episodes based on the number of common features between each one and the stimulus (see Algorithm 1 - shallow-index-retrieve). Retrieved episodes are sorted in descending order of their **hit-count**. This is the sum of the weights of all remindings that indexed a particular episode.

$$\text{hit-count}(E_j) = \sum_{E_j \in reminded-episodes(r_i)} \text{reminding-weight}(r_i)$$

Starting from a subset of these candidate episodes, a hill-climbing algorithm (see Algorithm 2 [3]) using semantic-matching will find the set of episodes that best match the external stimulus.

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. [88, 40]).

[3]A note on the pseudocode notation of these algorithms: anything after a double slash until the end of the line is a comment.

**Algorithm 1** Algorithm for shallow-index-retrieve(stimulus, dimension)

```
reminded-episodes ← [] // initialize result
remindings ← generalize-triples(stimulus, dimension) // generate remindings
for all rem ∈ remindings do
    new-rem-eps ← collect-reminded-episodes(rem)
    reminded-episodes ← reminded-episodes ∪ new-rem-eps
end for
return sort-reminded-episodes(reminded-episodes) // order episodes by hit-count
```

**Algorithm 2** Algorithm for retrieve(stimulus, dimension)

```
// do shallow retrieval to generate candidates
all-reminded-eps ← shallow-index-retrieve(stimulus, dimension)
// restrict their number to *MAX-REMINDINGS*
open ← first-n (*MAX-REMINDINGS*, all-reminded-episodes)
result ← [] // initialize result
current-best ← [] // initialize current-best
best-match-result ← [] // initialize best match
// while there are candidate to examine
while open ≠ [] do
    current-episode ← pop(open) // get the next candidate
    // match it against stimulus on dimension
    match-result ← graph-match(stimulus, current-episode, dimension)
    // a better match has been found
    if match-result is better then best-match-result then
        current-best ← current-episode // capture new best match
        best-match-result ← match-result
        // retrieve the difference links of current-episode that match match-result
        matching-diff-links ← matching-diff-links(match-result, current-episode)
        // there are matching difference links
        if matching-diff-links ≠ [] then
            // add linked episodes to open
            open ← open ∪ linked-episodes(matching-diff-links)
        end if
    end if
    result ← result ∪ match-result // record current match in result
end while
// return up to *MAX-RETRIEVED* episodes
return first-n(*MAX-RETRIEVED*, sort-result(result))
```

### 4.4.2 Retrieval Complexity

An important parameter that controls the functioning of the episodic memory module is the number of initial candidate episodes that are explored (*MAX-REMINDINGS* in Algorithm 2). 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[4]. Otherwise, the hill-climb process might explore all stored episodes, failing to scale up. This limit is a parameter of the memory module. In all experiments reported in this thesis we have used 5 as the value for the maximum remindings explored.

One way to optimize the shallow-index-retrieve algorithm (Algorithm 1) is to generate *only* the remindings that will actually be explored (the best *MAX-REMINDINGS*), removing the need to sort *all* reminded episodes by their reminding weights.

The average case complexity of the retrieval algorithm is $O(Nd)$ where $N$ is the maximum number of remindings explored (denoted by *MAX-REMINDINGS* in Algorithm 2) and $d$ is the average number of episodes connected by difference links. In the worst case an episode is be linked to all other episodes in memory, making the the complexity linear in the number of stored episodes, the same as serial search. However, this situation cannot arise because of the way difference links are created: an episode is linked, at most, to all retrieved episodes. The number of retrieved episodes is bound by the *MAX-RETRIEVED* parameter. Therefore, the worst-case complexity is $O(NM)$ where $M$ is the maximum number of episodes retrieved by memory (denoted by *MAX-RETRIEVED* in Algorithm 2). In practice, an episode is usually linked to fewer episodes than that.

Retrieval complexity is directly influenced by episode size and matching complexity. However, these are external to the memory itself. It is important that the memory module reduce the number of such matches performed by using filtering techniques like shallow-indexing.

[4]MAC stage of MAC/FAC reduces the number of memory items to only those that scored within 10% of the best matching score

## 4.5 Incremental Retrieval

Humans are good at dealing with continuous streams of stimuli and employing expectations to focus attention and guide recognition. The question we address here is: can we devise such an algorithm for an episodic memory?

This idea has been put forth before [105, 104] and has been applied in areas like dialogue processing [47, 73] and plan recognition [105]. The sequential structure of events helps constrain the type of expectations a system might form to just the next event(s) (its type and possibly its description).

To be able to take advantage of this, a memory should have the ability to [105]:

**form hypotheses** based on a set of initial observations and background knowledge;

**build expectations** about next actions based on current hypotheses;

**recognize** if expectations were met when new observations become available;

**refine and revise a set of hypotheses** when expectations are not met; this includes dropping hypotheses that do not conform to the observed stimuli and building new ones that do.

### 4.5.1 Incremental Retrieval Algorithm

We have implemented an incremental version of the retrieval algorithm for cases when the stimuli are presented incrementally. Examples of such situations are the plan recognition task, where the agent observes the actions in some order, evidence gathering tasks, where pieces of evidence become known one by one, and dialogue processing.

At first glance, the fact that data is presented incrementally appears to increase retrieval time due to the need to query memory with the presentation of each new stimulus. However, incremental data reduces the size of each query, so individual memory retrieval should be faster. The results of these individual retrievals should be combined (e.g. by checking that each instance in the base is either not mapped or mapped to a single instance in the target).

The incremental retrieval algorithm (see Algorithm 3) is intended to work with sequences of actions that are presented one by one. It functions as follows: after a new stimulus (i.e. action) is observed, the current set of candidate episodes is revised so that they account for the last seen stimulus. This is done by trying to match the stimulus against current episodes or by retrieving additional episodes, if necessary. New episodes are retrieved using the shallow indexing mechanism and are 'synchronized' with the previously encountered stimuli, if necessary. There are cases when an episode does not become relevant until several stimuli have been observed. The synchronization process tries to match these episodes with all previous stimuli so that, when matching against current stimulus, they are not at a disadvantage compared to the rest. All current episodes are then semantically matched to the new stimulus and, based on the result, they are re-ranked according to their similarity to the plan observed so far.

Mismatches between an observed action and the action of a prior episode are allowed: memory treats both these actions as possibly noisy actions. That is, either the observed stimulus is noise, or the action we are trying to match against is. In contrast, Episode-Based Reasoning [102] discards an episode from its hypotheses list as soon as there is a mismatch between its actions and the observed input. Such an approach deals poorly with noise commonly present in plan recognition datasets.

Similarity between a sequence of stimuli (i.e. actions) $S_i$ and the sequence of actions in an episode $A_j$ is computed as:

$$sim(\{S_i\},\{A_j\}) = seqsim(\{S_i\},\{A_j\}) * seqsim(\{A_j\},\{S_i\})$$

$$seqsim(\{S_i\},\{A_j\}) = \frac{1}{|\{S_i\}|} * \sum_{\forall S_k \sim A_l} semsim(S_k, A_l)$$

**Algorithm 3** Algorithm for incremental-retrieve(stimulus, dimension)

```
// initialize candidate-episodes
candidate-episodes $\leftarrow$ []
// initialize stimuli-history
prior-stimuli $\leftarrow$ []
**while** there are stimuli left **do**
   // get current stimulus
   current-stimulus $\leftarrow$ get-current-stimulus()
   // generate remindings based on current-stimulus
   reminded-episodes $\leftarrow$ shallow-index-retrieve(current-stimulus, dimension)
   // for all reminded episodes
   **for all** episode $\in$ reminded-episodes **do**
      // if it is not in candidate-episodes yet
      **if** episode $\notin$ candidate-episodes **then**
         // compare it to stimuli seen so far
         synchronize-candidate(episode, prior-stimuli)
      **end if**
   **end for**
   // for all candidates
   **for all** candidate $\in$ candidate-episodes **do**
      // compare candidate to current-stimulus
      candidate-match $\leftarrow$ match-stimulus-to-candidate(current-stimulus, candidate)
      // if it matches
      **if** candidate-match $\neq$ [] **then**
         // record its match in candidate-episodes
         candidate-episodes $\leftarrow$ update-candidate-matched(candidate-match)
      **else**
         // record the fact that it did not match
         candidate-episodes $\leftarrow$ update-candidate-unmatched(candidate-match)
      **end if**
   **end for**
   // re-rank candidate-episodes
   candidate-episodes $\leftarrow$ sort-matched-episodes(candidate-episodes)
   // record current-stimulus
   prior-stimuli $\leftarrow$ prior-stimuli $\cup$ current-stimulus
**end while**
// sort candidate episodes in decreasing order of their matching scores
result $\leftarrow$ sort-matched-episodes(candidate-episodes)
// return up to *MAX-RETRIEVED* episodes
**return** first-n (*MAX-RETRIEVED*, result)
```

where $semsim$ is defined as

$$semsim(C_1, C_2) = \frac{1}{|C_1|} \sum_{t_i \in C_1 \sim C_2} score(t_i)$$

in Section 4.3.2.

This similarity measure is commutative.

### 4.5.2 Incremental Retrieval Complexity

The complexity of the incremental retrieval algorithm in the best case is linear in the number of observed actions ($s$) and the maximum number of remindings explored for a new stimulus ($N$). This happens if, at every step, the algorithm chooses to explore only the correct episodes, whose events line up perfectly with the observed actions.

The worst case happens if, at every step, memory explores the maximum number of episodes allowed ($N$) and each of them requiring synchronization with all previously observed actions ($s - 1$ after seeing $s$ stimuli). In this case the complexity is:

$$\textstyle\sum_{i=1}^{i=s} iNs = Nsm(m-1)/2$$

which, assuming the average number of observed actions (m) is the same as the average number of events in an episode, is $O(Ns^3)$

However, this situation rarely arises in practice as the likelihood of seeing a completely new sequence of actions consisting of entirely new actions decreases rapidly as memory matures. In the early stages of building the memory, sequences of actions are very dissimilar to one another and memory tends to do more exploration. However, a mitigating factor is that the number of explored episodes is bound by the number of episodes stored in memory. Therefore, in the early development of memory, many of the stored episodes are explored, but there are few of them. Later, as memory matures, few of the (now many) episodes are explored.

As opposed to statistical approaches (such as [17]), the complexity is not a function of the number of goal schemas, but only of the number of observed actions.

## 4.6 Memory Interface

The memory module provides two basic functions: **store** and **retrieve**.

### 4.6.1 The Store Function

**Store** takes a *new episode* represented as a triple [context, contents, outcome] and stores it in memory, creating remindings along the three dimensions.

An optional parameter to the **store** function is the result of a call to the **retrieve** function with one of the *new episode*'s dimensions. Using the differences computed by this, memory creates difference links between *new episode* and the retrieved episodes. Difference links are a way to cache the result of the matching between episodes.

### 4.6.2 The Retrieve Function

**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 a retrieved episodes. This information is intended to be used by the external application to help it make better use of the returned episodes.

The retrieval function returns a list of most similar episodes, each item in the list containing:

**episode id** - an identifier for the retrieved episode;

**score** - the match score between the given stimulus and the retrieved episode on the given dimension;

**stimulus-matched** - the set of triples in stimulus that matched triples in episode;

**stimulus-unmatched** - the set of triples in stimulus that did not match any triples in episode;

**episode-matched** - the set of triples in episode that matched triples in stimulus;

**episode-unmatched** - the set of triples in episode that did not match any triples in stimulus;

**mappings** - a set of mappings from instances in stimulus to those in episode;

**triple mappings** - a set of mappings from triples in stimulus to corresponding triples in episode; note that due to the use of transformations these mappings might not be one-to-one.

## 4.7 Chapter Summary

In this chapter we presented our implementation of the generic episodic memory module that tries to satisfies all the requirements put forth in Chapter 3.

We have presented an episode representation intended to capture the dynamic aspect of episodes, the *context* in which they take place, the set of actions they are composed of (their *contents*) and the effect they have on the state of the world (their *outcome*). We argue that such a representation is suitable to generate a memory structure that can be used for various tasks by simply using different retrieval dimensions.

Our proposed organization scheme involves a two layer indexing scheme, with a first stage (*shallow indexing*) that only looks at generalized features and disregards structural information, and a second stage (*deep indexing*) that links episodes by how they differ structurally. The retrieval process takes advantage of these index structures by selecting a subset of the episodes most likely to be structurally similar to the current situation (using shallow indexing) and hill-climbing from that set using semantic match to more throughly assess their similarity. An incremental retrieval algorithm that works on sequences of events as well as a similarity measure for sequences of events were proposed.

During the design and implementation process of the generic memory module, we tried to leave task specific decisions to the application. For example, the scoring function can be provided by the application so that it can take advantage of a domain-dependant weighting scheme.

In the following chapters we will evaluate this implementation in terms of its performance and competence at several tasks.

# Chapter 5

# Memory-based Planning

## 5.1 The Planning Problem

The classical *planning problem* is defined as follows: given the description of an *initial state* of a world, a *goal state*, and the description of *a set of actions* that can be performed, find a *sequence of actions* that change the initial state into the goal state [133].

This problem has been proven theoretically and experimentally intractable [66, 23, 133], so many methods have been proposed to reduce its computational cost.

*Hierarchical planning* [86, 101] tries to order goals and actions based on their importance. Planning proceeds by building an abstract plan that satisfies the more important goals, which is then specialized at a lower level of abstraction until an executable plan is obtained. The size of the search space is reduced by ordering the goals and actions.

*Skeletal planning* [42] relies on instantiation and adaptation of skeletal plans (i.e. sequences of generalized planning steps). It emphasizes the role of representing expert knowledge as 'chunks' (e.g. skeletal plans), which are retrieved and adapted when needed. Efficiency is gained by using previous skeletal plans, without having to build them from scratch.

*Memory-based planning* [52] (also known as case-based planning) solves new planning problems by remembering similar past experiences and reusing plans that succeeded or repairing those that failed[1]. Stored cases are specific instances of prior planning problems along with their solutions.

## 5.2 Memory-Based Planning

In this chapter we will look at how our memory module can be applied to memory based planning. We will evaluate our memory-based planner on a dataset from the Logistics domain.

We will not contribute to the debate as to whether memory-based planning is more efficient then planning from scratch. The claim that reusing plans (or plan subparts) improves efficiency by avoiding unnecessary repetition ([45, 53], etc.) is controversial, both on theoretical and empirical grounds [84]: the worst-case complexity of plan reuse is at least the same as that of plan generation and efficiency gains are strongly dependent on the particular domain and on how similar new and old problems are.

We are however interested in some of the problems addressed by memory-based planning [112]: *plan memory representation* and *plan retrieval*. Being designed to address the representation and organization of sequences of events, our memory module is directly applicable to memory-based planning and tasks related to it like assessing the solvability of planning problems, plan recognition, failure detection, etc.

## 5.3 Applying the Memory Module to Planning

We empirically evaluated how the proposed memory module performs on two tasks in the Logistics domain [122]: *planning* and *classification* of planning problems into solvable and unsolvable.

[1]For an extensive survey of case-based planning see [112]

The goal of this experiment is to evaluate the influence of indexing on retrieval accuracy and speed and to investigate memory scalability.

Episodes for two 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.

These two tasks used the memory retrieval mechanism and the retrieved episodes in different ways. Planning used retrieval based on context and adapted the contents of the 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. We measured the number of stored episodes as well as the number of explored episodes during retrieval.

### 5.3.1 The Logistics Domain

The logistics domain [122] 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.

This domain has been extensively used in the planning literature and has become one of the benchmarks of the planning competitions (e.g. [1]). An example of a planning problem from the Logistics domain is presented in Figure 5.1.

| | |
|---|---|
| Initial: | (*Package-Deliver1 trucks *Truck3) |
| | (*Truck3 location *Post-Office4) |
| | (*Package-Deliver1 object *Package5) |
| | (*Post-Office6 is-inside *City7) |
| | (*Package5 location *Post-Office6) |
| | (*Post-Office4 is-inside *City7) |
| Goal: | (*Package-Deliver1 destination *Post-Office2) |
| | (*Post-Office2 is-inside *City7) |
| Plan: | (*Drive8 object *Truck3) |
| | (*Drive8 destination *Post-Office6) |
| | (*Load-Truck9 object *Package5) |
| | (*Drive-Truck10 destination *Post-Office2) |
| | (*Unload-Truck9 object *Package5) |

Figure 5.1: An example of a planning problem in the Logistics domain.

### 5.3.2 Dataset

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 [83] 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 the tasks described above.

### 5.3.3 Domain Knowledge

Knowledge about the actions, states and objects in the Logistics domain was encoded as an extension to our knowledge base (i.e. CLib). This domain knowledge is necessary in order to be able to judge the similarity of planning problems and actions in the given domain. A total of 7 objects, 11 actions and 3 relations have been defined. Figure 5.2 shows a part of

| **Place** | **Physical-Object** | **Move** |
|---|---|---|
| *Location* | Truck | **Move-Into** |
| Post-Office | Airplane | Load-Airplane |
| Airport | Package | Load-Truck |
| City | | **Move-Out-Of** |
| | | Unload-Airplane |
| | | Unload-Truck |
| | | Drive-Truck |
| | | Fly-Airplane |

Figure 5.2: A part of the ontology for the Logistics domain. Concepts in bold are predefined in the CLib, those in italic are intermediate levels of the ontology, while those in cursive correspond to domain objects or operators.

the taxonomy of objects and actions for the Logistics domain and their relation with CLib concepts.

### 5.3.4 Experimental Setup

Building a memory based system for these tasks required writing a thin interface layer on top of the EM generic memory module. This consisted of functions dealing with the adaptation 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 [110] 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 both 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 retains the

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[2]. 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 (e.g. SoarEM).

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 Figures 5.3(a), 5.4(a), and 5.5(a).

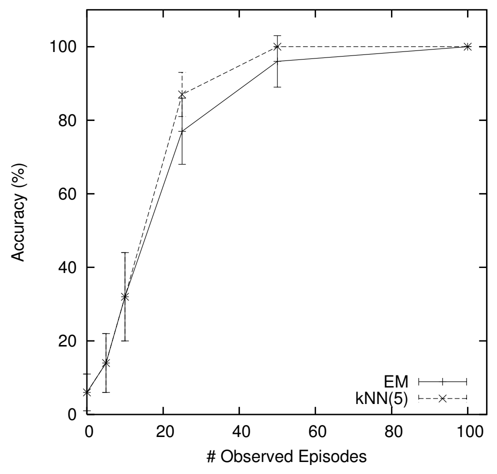
For the **memory-based classification** task we used all 250 goals, including both solvable and unsolvable goals. The adaptation of retrieved episodes consists in taking the majority vote of the top 5 most similar retrieved episodes in order to determine whether a new goal is solvable or not. The results are presented in Figure 5.3(b), 5.4(b), and 5.5(b).

### 5.3.5 Discussion

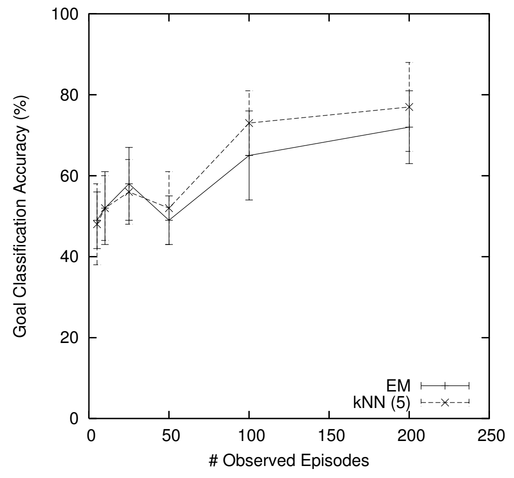
For both tasks EM achieves the same accuracy as kNN(5) after most training episodes have been seen. Even though kNN(5) learns faster, EM is able to catch up in the end.

In terms of explored episodes, EM is able to drastically reduce their number compared to kNN(5). The difference is statistically significant at the 0.05 level for a two-tailed t-test, after training has completed. 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.

[2]The resulting kNN(5) implementation is similar to RIBL [37].

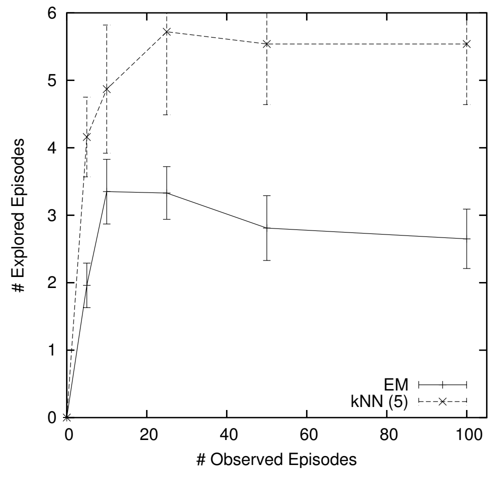


(a) Memory-based planning.

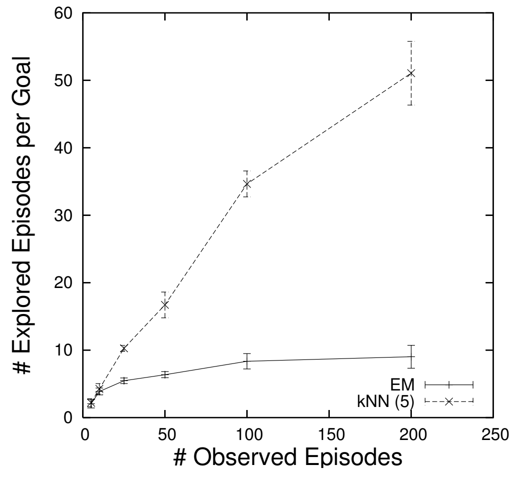


(b) Goal classification.

Figure 5.3: Accuracy results for the memory-based planning and goal classification tasks for EM and kNN(5). Error-bars represent the standard deviation.

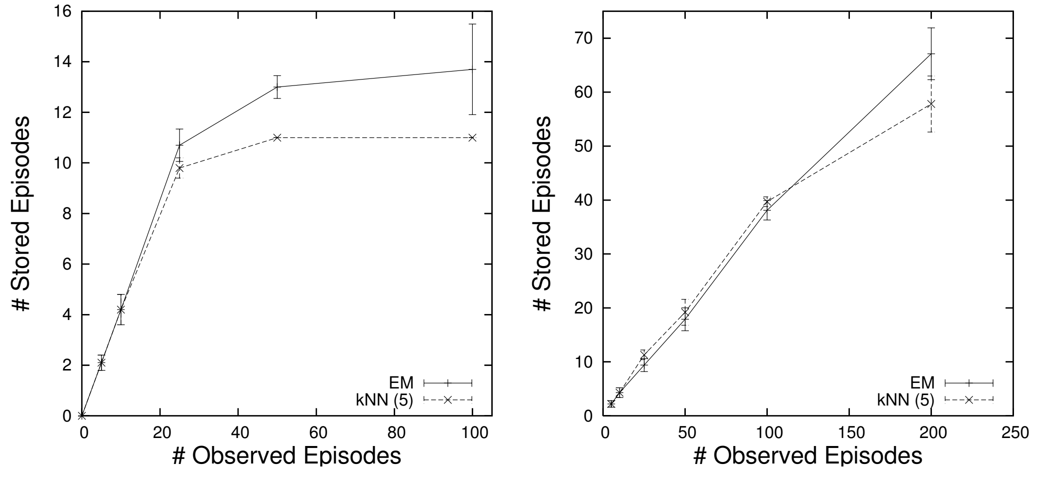


(a) Memory-based planning.



(b) Goal classification.

Figure 5.4: Number of explored episodes for the memory based planning and goal classification tasks for EM and kNN(5). Error-bars represent the standard deviation.



(a) Goal classification. (b) Memory-based planning.

Figure 5.5: Number of stored episodes for the memory based planning and goal classification tasks for EM and kNN(5). Error-bars represent the standard deviation.

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 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 5.4).

## 5.4 Chapter Summary

This chapter presented an application of the generic episodic memory module to the problem of memory-based planning and classification. Developing this application required representing domain specific knowledge (i.e. plan operators) and a thin interface layer that adapted the retrieved episodes for the use of the respective task.

We evaluated the proposed memory module on two different tasks in the logistics domain: memory-based planning and memory-based classification. 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.

# Chapter 6

# Episodic-Based Plan Recognition

We applied the **episodic memory module** to the problem of plan recognition. Episodic memory lends itself easily to this task: it organizes *temporally ordered events*, these events are *dynamic*, changing the state of the world, and they are *observed incrementally*. Storage and recognition of past events are the basic processes of an episodic memory.

We use the generic episodic memory module to perform incremental plan recognition and to build the plan library. Unlike other case-based plan recognizers our approach does not require complete knowledge of the planning domain or the ability to record intermediate planning states. The memory structure that it builds is multi-functional and can be used for other tasks such as plan generation or classification.

## 6.1 The Plan Recognition Problem

Plan recognition is the problem of ascribing goals, intentions and future actions to an actor based on the actor's observed actions [105].

A growing class of AI applications relies on recognizing complex ongoing events: language understanding and response generation [4, 92], user interfaces [46], help systems [77], and collaborative problem solving [72].

The use of plan recognition in an application has several benefits [55]:

**improved performance** - e.g. reducing the wait time for resources by anticipating their use given the user's observed actions,

**higher reliability** - by detecting errors faster and correcting them, and

**reduction of user workload** - e.g. by plan completion.

Plan recognition algorithms should generate *incremental predictions*, preferably after each action is observed, thus offering *early predictions* and they should be *fast* at this task (e.g. faster than it takes for the observed agent to execute the next action so that the recognizer system can take counter-action).

Plan recognition approaches can be characterized along several dimensions: the *type of recognition method* used, the type of *relation between the agent executing the plan and the observer*, and whether the *plan library is fixed* (i.e. pre-built) or *can be extended* during recognition.

According to the recognition method used, plan recognition approaches can be classified as: *deductive* [59], *abductive* [5], *probabilistic* [24], and *case-based* [60].

Based on the relation between the agent executing the plan and the recognizer, the plan recognition problems can be classified as *intended* (in which the agent cooperates with the recognizer), or *keyhole* [3] (where there is no cooperation).

The need for domain-independent plan recognition systems has long been recognized. Huwer et al. [56] identifies several requirements for such plan recognition algorithms: *efficiency*, *robustness*, use of *domain-independent representation* of actions and plans, ability to *detect and incorporate new plans* into the plan library, ways of *limiting the search* through the plan library, and ability to *deal with noise* (i.e. erroneous actions). Adding domain-specific knowledge to a plan recognizer might improve performance, but has to be balanced against the recognizer's purpose of being domain-independent.

## 6.2 Episode-Based Plan Recognition

We built a memory-based planner for keyhole plan recognition that uses our generic episodic memory module and its recognition mechanism to store and retrieve plans. It addresses all the requirements listed above: the action and plan representation are generic in nature, plan memory (i.e. the plan library) is grown at the same time as recognition is attempted, and memory retrieval limits the search to the most similar prior plans through indexing. The memory module employs a flexible matching algorithm for sequences of actions that can effectively deal with noise.

### 6.2.1 Case-Based Plan Recognition

Related approaches include the case-based plan recognition of Cox and Kerkez [34], which is based on state indexing. In their approach, after an action is observed, the recognizer uses the resulting state to perform retrieval on the plan library. To deal with the exponential growth of the state space, the observed actions are used to compute an *abstract planning state*, which is then used as an index into plan memory.

Unlike traditional plan recognizers [59, 94], both this approach and ours can deal with incomplete plan libraries, growing these libraries as recognition proceeds, effectively learning from observations.

The approach of Cox and Kerkez [34] has several applicability requirements: that the observer is able to record intermediate planning states, that it has a complete model of the planning domain including consequences of actions, and that is able to detect whether an action completed successfully. In contrast, our approach does not require complete domain knowledge and its matching algorithm can take advantage of as much or as little domain knowledge is available (in the form of ontologies of actions, objects, states, etc.).

Another important difference compared to case-based reasoning approaches in general is that an episodic-based approach builds a *multi-functional plan library* that can be used for tasks other than plan recognition. For example, such a memory can also generate plans for a given goal, it can classify goals into solvable or not, or assess user proficiency at a task by observing their actions when executing the task.

### 6.2.2 Episode Representation for Memory-Based Plan Recognition

For the plan recognition task an episode's context is the instantiated goal of the plan being executed. An instantiated goal is composed of a goal type (e.g. `remove-file`) and its particular parameters (e.g. `file.exe`) The contents of the episode is the sequence of actions observed by the recognizing agent. The outcome is the observed outcome of the plan and, while not used directly in plan recognition, could be useful for tasks like predicting the outcome of an ongoing plan (e.g. failure detection).

## 6.3 Experimental Evaluation

We evaluated out approach on a plan recognition task on two corpora: the Linux Plan Corpus [17] and the Monroe Plan Corpus [18][1].

We decomposed the task of plan recognition into:

**goal schema recognition** - the recognition of the *type* of goal being attempted by the agent,

**goal parameter recognition** - the recognition of the *parameters* instantiating the goal type being attempted,

**instantiated goal recognition** - the combination of the two, where both the type of goal and its parameters are recognized.

This allowed us to both gain more insight into the functioning of our episodic-based recognizer and to do a head-to-head comparison against a statistical approach on the same corpora [19].

[1]We would like to thank Nate Blaylock for providing access to these plan recognition corpora.

This experiment is intended to test how memory accuracy at the task of plan recognition compares to that of a classical approach, whether episodic-based recognition can handle noise and variability in the data, and whether incremental retrieval is scalable.

### 6.3.1 The Plan Corpora

#### The Linux Corpus

The Linux plan corpus [17] was gathered from human Linux users from the University of Rochester, Department of Computer Science. It is similar to Lesh [71], but is an order of magnitude larger in size.

Users were given a goal like 'find a file with 'exe' extension' and were instructed to achieve it using simple Linux commands (e.g. without using pipes, or programs such as `awk`, etc.). All user commands along with their results were recorded. For each goal, users were also asked to assess whether they accomplished it. The users judged a total of 457 sessions to be successful, involving 19 goal schemas and 48 action schemas (i.e. Linux commands) (see Table 6.1). These sessions constitute the Linux dataset. Because some users were not able to judge this correctly, there are still a number of failed sessions and, therefore, data is noisy.

#### The Monroe Corpus

The Monroe corpus [18] consists of stochastically generated plans in the domain of emergency response. The plans have been randomly generated by allowing a planner to make nondeterministic decisions and therefore generating a diverse set of plans (in terms or ordering of their actions) for the goal. It contains 5000 plans with an average of 9.5 actions per plan, a total of 10 goal schemas and 30 action schemas (see Table 6.1).

Stochastically generating plan corpora has an advantage over the approach that uses plans generated by a deterministic planner: the planner usually optimizes for some parameter (plan length, cost, etc.), generating the *same sequence of actions* in a plan every time

|  | Linux | Monroe |
|---|---|---|
| Plan sessions | 457 | 5000 |
| Goal schemas | 19 | 10 |
| Action schemas | 48 | 30 |
| Avg. Actions/Plan | 6.1 | 9.5 |

Table 6.1: The Linux and Monroe plan corpora description

a similar goal is seen. This is less than desirable since real world plans rarely display this characteristic.

We chose these two corpora because they have been commonly used in the plan recognition community (e.g. [17]).

### 6.3.2 Background Knowledge

To capture the domain knowledge for the Linux and Monroe datasets, we extended the CLib by adding an ontology of domain actions and their parameters. Action pre and post-conditions were not encoded.

The encoding process is straight-forward: usually, for each action in the specified domain (i.e. plan operator) we created a corresponding class in CLib (e.g. `Find-File`) and tried to find the most appropriate superclass for it based on the action semantics (e.g. `Action`). We tried to group together similar operators whenever possible. This lead to the creation of intermediate levels in the ontology (e.g. `Find-Resource`). Classes belonging to intermediate levels were not instantiated in plans. For each operator, we encoded its parameter types as a set of objects in the ontology (e.g. `File` for `Find-File`) and their relationship to the goal-schema (e.g. x is the object of find-file in ‘find-file(x)’, which is encoded as `(Find-File object X)`).

```
Intangible-Entity | Action
  Linux-Resource  |   Shell-Command
    Port          |     Linux-Shell-Command
    Host          |       Remote-Connect
    File          |         Rsh
    Directory     |         Rlogin
    User          |       Create-Resource
    Linux-Device  |         Create-File
    Host          |           Archive-File
...               |             Zip
Message           | ...
  Resource-Specifier|       Find-Resource
    Port-Name     |         Find-File
    Host-Name     |           Find-By-Name
                  |           Find-By-Size
```

Figure 6.1: A part of the ontology for the Linux domain. Indentation is proportional to depth in the ontology. Concepts in bold are pre-defined in the CLib, those in italic are intermediate levels of the ontology, while those in cursive correspond to domain objects or operators.

### Linux Domain Knowledge

For the Linux domain, given the number of tasks a single command can achieve based on its parameters, we created separate classes based on the main type of its parameters. For example, there are two types of `Find` operators `Find-By-Name`, `Find-By-Size`. Figure 6.2 contains a part of the Linux domain knowledge.

### Monroe Domain Knowledge

For the Monroe domain the same encoding methodology was followed. However, fewer classes were encoded given that there are fewer domain action and goal schemas. Fewer intermediate levels have been added to the ontology as actions in this domain deal mostly with concepts such as `Transfer` and `Move`, two CLib clusters that are well populated.

```
Location              | Action
  Town                |   Treat
  Garbage-Dump        |     Treat-In-Hospital
    Mall              | ...
...                   | Replenish
Vehicle               |   Fill-In
  Truck               |   Pump-Gas-Into
    Tow-Truck         |
    Tree-Truck        | ...
...                   | Attach
Person                |   Hook-Up
  Driver              |   Hook-To-Tow-Truck
    Bus-Driver        |
    Truck-Driver      |
```

Figure 6.2: A part of the ontology for the Monroe domain. Indentation is proportional to depth in the ontology. Concepts in bold are pre-defined in the CLib, those in italic are intermediate levels of the ontology, while those in cursive correspond to domain objects or operators.

### 6.3.3 Experimental Setup

We evaluated the recognition performance on three tasks: goal-schema recognition, parameter recognition and instantiated goal-schema recognition, using memory to retrieve the most similar episodes given a sequence of actions and making a prediction based on the retrieved episodes.

#### Making Predictions

For the tasks of goal schema recognition and instantiated goal recognition the top three predictions were generated. For parameter recognition the top two predictions were generated.

For the *goal schema recognition* task a weighted majority vote using similarity scores of the retrieved episodes was used to determine the predicted goal schema.

For each prior plan retrieved, memory provides the associated set of mappings from the parameters of the observed actions to those of the stored plan. For the *goal parameter* *recognition* these mappings were used to map the parameters of the most similar episodes retrieved back to those of the actions observed. If not all parameters of the goal were mapped, those of the prior episode were predicted instead.

For the *instantiated goal recognition* the goal schema and parameters (using the mapping scheme presented above) of the most similar episode were predicted.

The set of test plan sessions was presented to memory one action at a time, predictions being generated after each action from the top four most similar prior episodes retrieved by memory. We limited the maximum number of explored remindings generated after each action is observed to five for both domains. All observed plans have been stored in memory, no storage policy being implemented.

## Measurements

We measured the accuracy of the recognizer in terms of **Precision (P)**, **Recall (R)** and **F-measure (F)**.

**Precision** is the number of correct predictions divided by the total number of predictions (i.e. the number of times the recognizer chooses to make a prediction).

**P** = # correct-predictions / # total-predictions

**Recall** is the number of correct predictions divided by the number of predictions opportunities (i.e. the number of observed actions).

**R** = # correct-predictions / # prediction-opportunities

**F-Measure** is the harmonic mean of P and R.

**F** = 2PR/(P+R)

These three measures represent the *overall accuracy* of the recognizer as they include predictions made after each *new* observed action.

A measure of how many plans were *eventually* recognized is denoted by **convergence (Conv)**, which is the number of correct predictions after the last plan action was observed. A recognition session is said to have converged if its last prediction was correct.

**Conv** = # correct-last-predictions / # plans

An important characteristic of incremental recognizers is how soon after a plan starts they start making the *same correct prediction*. This is measured by the **convergence point (CP)**. It was measured both in terms of number of observed actions as well as in terms of percentage with respect to the average number of actions of converged sessions.

**CP** = $i \Leftrightarrow \forall j \geq i \ P_j$ is correct

where $P_j$ is the prediction after seeing action $j$.

Besides its performance as a plan recognizer, we are also interested in how the memory mechanism performs in terms of *efficiency of retrieval*. We measured the retrieval effort for each prediction, both in terms of number of actions matched, as well as a percentage of the total number of stored episodes (whether the approach *scales*).

We performed a 10-fold cross-validation on each of the two corpora by dividing the set of plan sessions into 10 equal-sized subsets, and using 9 of them for training and the 10th for testing.

### The Statistical Approach

We compared our approach (referred to as *Episodic-Based*) to that of Blaylock and Allen [19] (referred to as *Statistical*) on the the three plan recognition tasks on the Linux and Monroe corpora.

The statistical recognizer treats goal recognition as a classification task, trying to find the most likely instantiated goal given the observed actions. Two simplifying assumptions are made: that goal parameters are independent of one another and that a goal schema is independent from an action's parameter values. An implicit assumption is that the observed actions are carried out in pursuing a single goal. The authors note that it is unclear how to extend this approach to deal with recognition of goals pursued simultaneously.

The goal schema recognizer uses a bigram approximation to compute the goal schema from the sequence of observed actions. The parameter recognizer takes the action sequence and goal schema as its input and estimates the probability of all action parameters seen so far as being the values of the goal parameters. For this it uses a tractable subset of Dempster-Schafer Theory. Each parameter is predicted independently

We did not reimplement their recognizer, and therefore only compared our results against those reported in their papers. Those results do not contain standard deviation, so we only report those for the episodic-based approach.

### 6.3.4 Experimental Results

#### Goal Schema Recognition

Experimental results for the goal schema recognition task using the episodic-based approach are reported in Table 6.2 for the Linux domain and Table 6.3 for the Monroe domain. The comparison between the episodic-based and the statistical approach is presented in Figure 6.3.

The precision, recall and F-measure are the same because the episodic-based approach makes predictions after each action (e.g. the number of prediction opportunities is the same as the number of predictions made.) This is in contrast to Blaylock and Allen [19] where predictions are made only if confidence is above a certain threshold.

Compared to the statistical approach, EM converges on more sessions for the Linux domain (see Figure 6.3(a)) and on a similar number for the Monroe domain (Figure 6.3(b)). Precision is slightly lower on both domains (see Figure 6.3(a) and 6.3(b)).[2] However, recall is much higher for the episodic-based approach on both domains. An increase in precision at

[2]Although probably not significantly different. [19] does not report variance for their data.

| | Linux | | |
|---|---|---|---|
| N-best | 1 | 2 | 3 |
| P, R, F (%) | 39.05 | 59.55 | 65.58 |
| Conv (%) | 50.14 | 73.76 | 78.95 |
| CP/AvgLen | 2.7/4.30 | 2.37/4.14 | 2.28/4.48 |
| CP/AvgLen (%) | 62.79 | 57.24 | 50.89 |

Table 6.2: Experimental results for the episodic-based approach on the goal schema recognition task in the Linux domain.

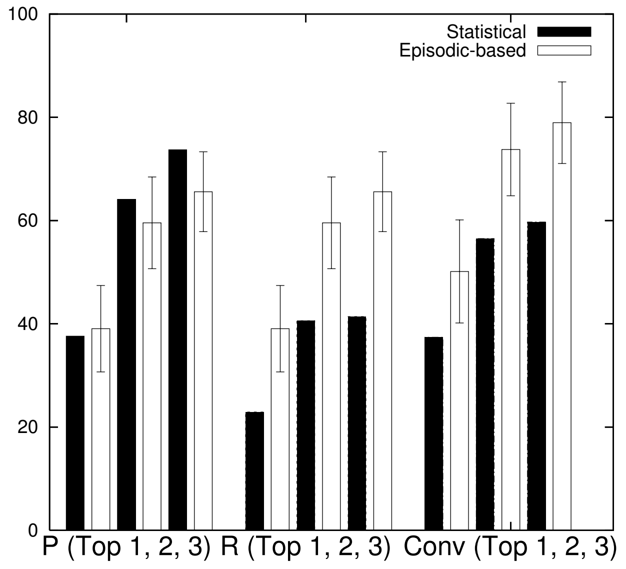
| | Monroe | | |
|---|---|---|---|
| N-best | 1 | 2 | 3 |
| P, R, F (%) | 81.53 | 85.94 | 88.19 |
| Conv (%) | 97.82 | 99.24 | 99.60 |
| CP/AvgLen | 3.04/9.44 | 2.56/9.49 | 2.29/9.49 |
| CP/AvgLen (%) | 32.2 | 26.97 | 24.13 |

Table 6.3: Experimental results for the episodic-based approach on the goal schema recognition task in the Monroe domain.

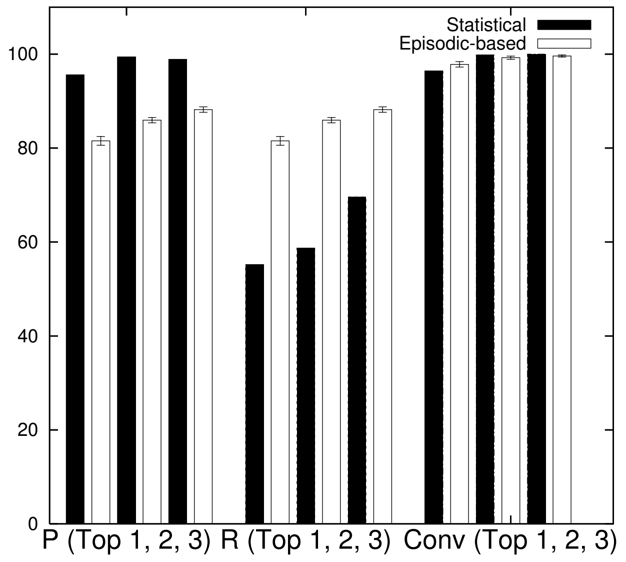
the expense of recall is expected given that the statistical recognizer only makes predictions when a certain confidence threshold was achieved.

Although we compute the similarity between the observed plan and some prior plans and use it to rank predictions we chose not to implement confidence thresholds, thus generating predictions at each recognition step. The variability of the plans in our corpora makes the similarity measure not a good candidate for the confidence thresholds.

In terms of convergence point, EM converges with approximately the same speed as the statistical approach (after seeing 63%, 57% and 51% of actions in sessions that converged, compared to 59%, 55% and 57%), but the length of a converged session is lower (4.30, 4.14, 4.48 compared to 5.9, 7.2 and 7.2). This might be due to the fact that the statistical approach only makes predictions when above a certainty level, for which it needs to see more actions.



(a) Linux Domain



(b) Monroe Domain

Figure 6.3: Comparison between episodic-based and a statistical approach on the goal schema recognition task in the Linux and Monroe domains. Error-bars represent the standard deviation.

**Goal Parameter Recognition**

Experimental results for the goal parameter recognition task are reported in Table 6.4 for the Linux domain and Table 6.5 for the Monroe domain. The comparison with the statistical approach is presented in Figure 6.4.

Here precision differs from recall, as there are potentially multiple parameters per goal schema, and their recognition is independent of one another.

Comparing the episodic-based approach to the statistical one, the same pattern can be observed: the episodic-based approach achieves lower precision (see Figure 6.4(a) and 6.4(b)) and lower convergence, but higher recall.

The goal parameter recognition proved to be a more challenging task for memory, especially in the Monroe domain, as showed by the performance difference when compared to the statistical approach. Both approaches can only predict a goal schema parameter after it appeared as an action parameter. However, [19] reports that the correct goal parameters appear in *any* observed actions only in 82.1% of the sessions for the Linux domain and in 79.4% of them in the Monroe domain. Of this upper-bound, memory achieves 62.1%, 75.5% for the top two predictions in the Linux domain. For Monroe similar levels of performance are achieved: 52.1%, 61.5% for the best two predictions.

These levels are lower that those of the statistical approach and we think the blame lies with the combination between the incremental retrieval algorithm and the matcher used. When matching individual actions of a plan, the retrieval algorithm has to enforce consistency in parameter mappings. However, the matcher we use cannot take advice on how to match two structures, effectively starting from scratch. This might generate inconsistent mappings for parameters already mapped in a previous step. These inconsistent matches are discarded and matching of the two actions is not attempted again (as it would yield the same result). The fact that converged sessions for this task are shorter seems to support this argument, as the likelihood of discarding potentially useful actions due to inconsistent matches grows with the number of observed actions.

|  | Linux |  |
|---|---|---|
| N-best | 1 | 2 |
| P (%) | 52.50 | 60.78 |
| R (%) | 49.33 | 58.26 |
| F (%) | 50.86 | 59.49 |
| Conv (%) | 50.99 | 62.00 |
| CP/AvgLen | 1.96/3.54 | 1.83/3.80 |
| CP/AvgLen (%) | 55.36 | 48.15 |

Table 6.4: Experimental results for the episodic-based approach on the goal parameter recognition task in the Linux domain.

|  | Monroe |  |
|---|---|---|
| N-best | 1 | 2 |
| P (%) | 40.98 | 47.85 |
| R (%) | 40.34 | 47.66 |
| F (%) | 40.65 | 47.78 |
| Conv (%) | 41.41 | 53.33 |
| CP/AvgLen | 3.03/6.07 | 3.01/6.72 |
| CP/AvgLen (%) | 49.94 | 47.79 |

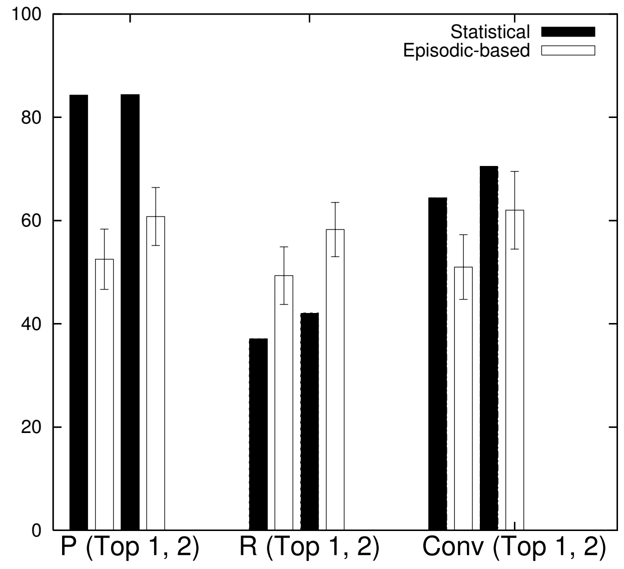
Table 6.5: Experimental results for the episodic-based approach on the goal parameter recognition task in the Monroe domain.

However, to confirm this hypothesis an extension to the matcher is needed so that it either accepts some initial mappings or returns all possible matches from which the one that agrees with the set of prior mappings can be selected.

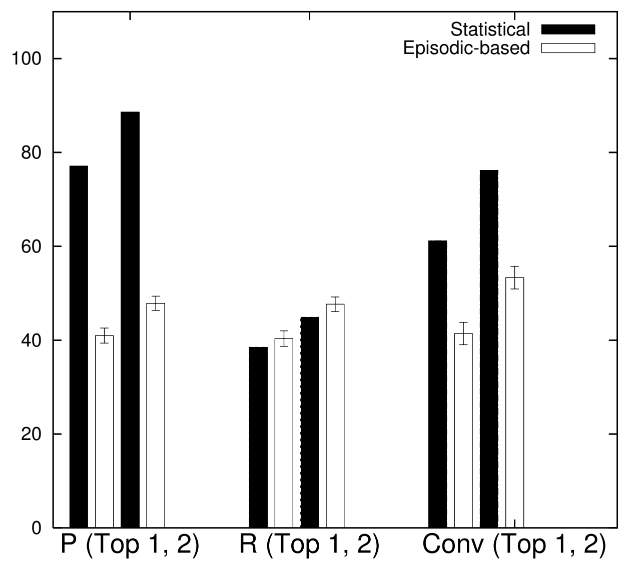
An adaptation strategy for parameter prediction would also likely improve these results.

### Instantiated Goal Recognition

Experimental results for the instantiated goal recognition for the episodic-based approach are reported in Table 6.6 for the Linux domain and Table 6.7 for the Monroe domain. Figure 6.5 presents the comparison between the episodic-based approach and the statistical



(a) Linux Domain



(b) Monroe Domain

Figure 6.4: Comparison between episodic-based and a statistical approach on the goal parameter recognition task in the Linux and Monroe domains. Error-bars represent the standard deviation.

|  | Linux |  |  |
|---|---|---|---|
| N-best | 1 | 2 | 3 |
| P, R, F (%) | 26.23 | 41.68 | 46.6 |
| Conv (%) | 33.65 | 51.87 | 56.82 |
| CP/AvgLen | 2.5/3.70 | 2.25/3.65 | 2.20/3.91 |
| CP/AvgLen (%) | 67.56 | 61.64 | 56.27 |

Table 6.6: Experimental results for the episodic-based approach on the instantiated goal recognition task in the Linux domain.

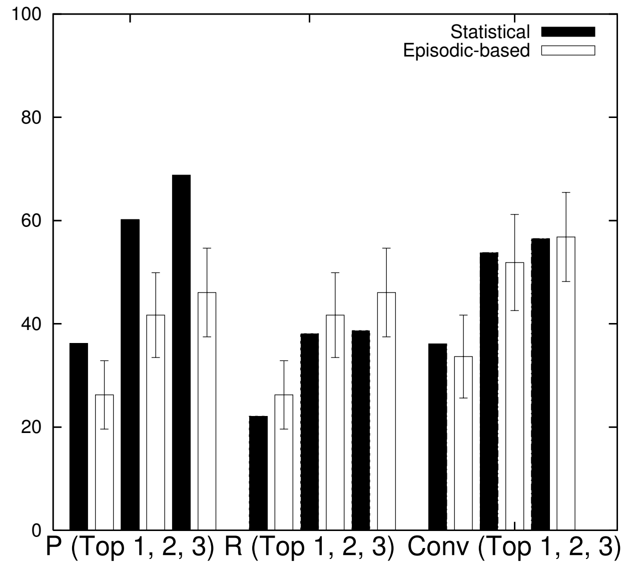
|  | Monroe |  |  |
|---|---|---|---|
| N-best | 1 | 2 | 3 |
| P, R, F (%) | 28.54 | 32.23 | 34.66 |
| Conv (%) | 41.39 | 48.81 | 54.15 |
| CP/AvgLen | 3.46/6.07 | 3.78/6.50 | 3.97/6.77 |
| CP/AvgLen (%) | 57 | 58.15 | 58.64 |

Table 6.7: Experimental results for the episodic-based approach on the instantiated goal recognition task in the Monroe domain.

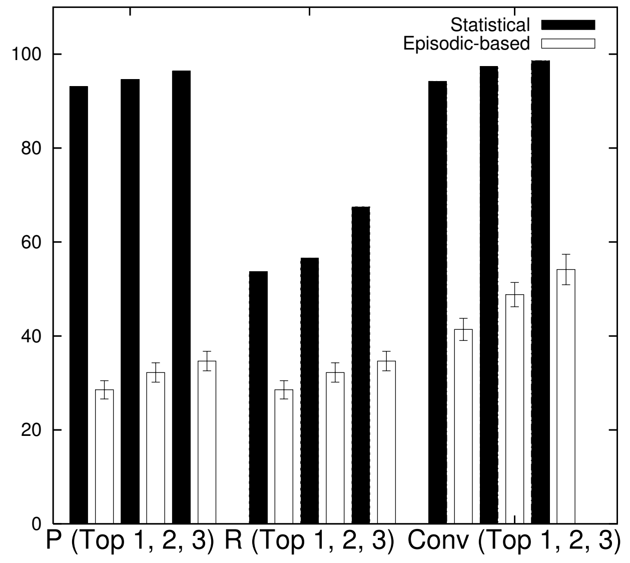
approach. In this case also, precision, recall and F-measure are the same for a particular N-best prediction, as the number of prediction opportunities is the same as that of predictions being made.

The results are different for the two domains: in the Linux domain (see Figure 6.5(a)), the episodic-based approach achieves lower precision then the statistical approach, but higher recall and similar convergence; in the Monroe domain however, its performance on all of the three tasks is worse when compared to the statistical approach.

As a successful recognition of the instantiated goal implies both correctly predicting the goal schema and all its parameters, the performance on this task cannot exceed the minimum performance on the two individual tasks. The problem with the goal parameter recognition performance explained above is to blame for the poor performance on the instantiated goal recognition task as well.

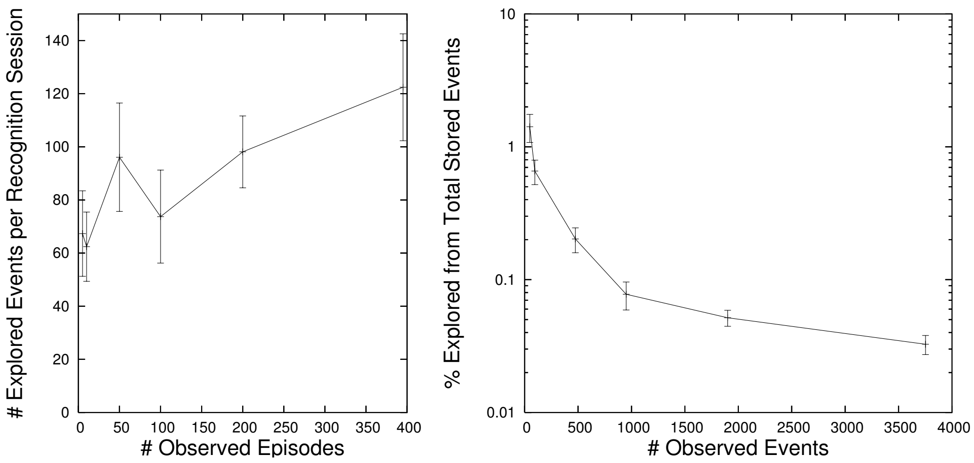


(a) Linux domain



(b) Monroe domain

Figure 6.5: Comparison between episodic-based and a statistical approach on the instantiated plan recognition task in the Linux and Monroe domain. Error-bars represent the standard deviation.



(a) Number of total explored events (b) Explored events as percentage of total stored

Figure 6.6: Number of explored actions per recognition session in the Linux domain. Error-bars represent the standard deviation.

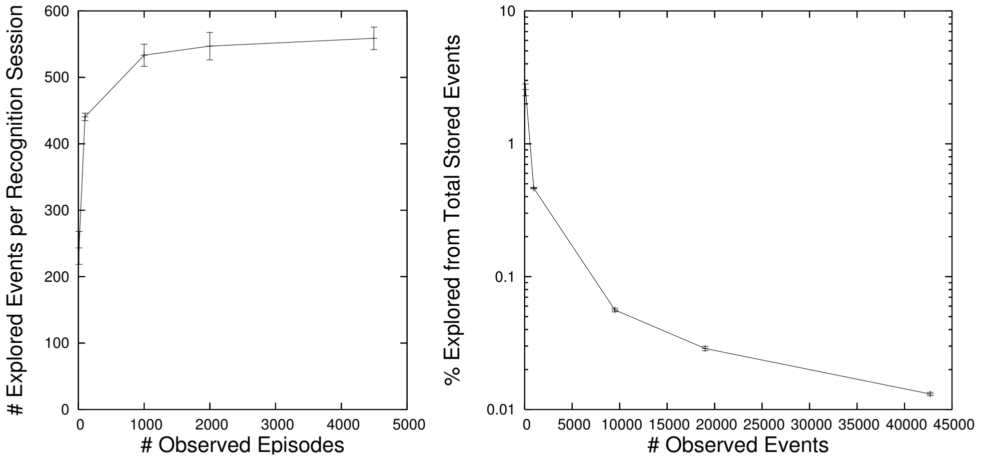
## Memory Performance

Figures 6.6(a) and 6.7(a) plot the number of explored actions per recognition session versus the number of total episodes observed. The number of observed episodes is also the number of stored episodes since all episodes are stored in memory.

The absolute number of explored events grows fast as memory develops, but at a much slower pace after memory has matured. Please note that we measure the number of events (not of episodes) matched, since the retrieval is incremental.

Figures 6.6(b) and 6.7(b) plot the percentage of stored episodes that have been explored in one recognition session.

Although the worst-case complexity of the retrieval algorithm is $O(Ns^3)$ where N is the number of remindings explored and s is the average number of actions per observed episode (see Section 4.5.1), in practice, only a fraction of that is explored. After all training data has been observed in the Linux domain, memory explores about 120 events, which represents under 11% of the number of episodes predicted by the theoretical worst case.



(a) Number of total explored events (b) Explored events as percentage of total stored

Figure 6.7: Number of explored actions per recognition session in the Monroe domain. Error-bars represent the standard deviation.

For the Linux domain $N = 5$ and $s = 6.1$.

The corresponding numbers for the Monroe domain are $N = 5$ and $s = 9.5$. The total number of explored events after all episodes have been seen is 558 episode, about 13% of the worst case prediction.

## 6.4 Discussion and Future Work

### 6.4.1 Summary of Results

The incremental episodic-based goal schema recognition achieved comparable precision, higher recall and higher convergence than the statistical approach of Blaylock and Allen [19]. For parameter recognition and instantiated goal-schema recognition in the Linux domain, the episodic-based approach achieved about the same recall and convergence levels, but lower precision. For the goal-schema recognition in the Monroe domain recall was about the same as for the statistical approach, but precision and convergence was significantly lower. Precision, recall and convergence were significantly lower for the instantiated goal recognition task in the Monroe domain.

In terms of memory performance, memory explored about 11% of the number of events predicted by the worst-case scenario for the Linux domain and 13% for the Monroe domain. The number of explored represents under .01% of the total stored events after about a quarter of the dataset is observed.

### 6.4.2 Limitations of Current Approach

The performance of the current episodic-based plan recognizer is limited by its adaptation strategy. We adopted simple adaptation strategies compared to the statistical recognizer. The task that is least affected by this is the goal schema recognition task, where the episodic-based recognizer performs the same or better than the statistical one. The goal parameter recognition task is the most affected by the lack of an adaptation strategy, resulting in lower performance compared to the statistical approach.

To improve the performance of the episodic recognizer two extensions are needed: an extension to the matcher so that it can make use of prior mappings and find only those matches that are consistent with them and a better adaptation strategy.

One issue that we did not address here is the sensitivity of memory retrieval performance to noise. Even though the Linux plan corpus is noisy we don't have good measure of how much noise there is (e.g. how many sessions are misclassified as successes at accomplishing their goal; how many unnecessary actions were taken by users given a goal). We would also like to study how memory performance degrades when noise is introduced. This would also show the degree of flexibility of our recognition algorithm.

### 6.4.3 Lessons Learned

The evaluation of the episodic-based approach to plan recognition brought to light some improvements, and extensions one can make to the generic memory module implementation, as well as new applications.

One such improvement involves the indexing of events. Our current approach indexes only individual plan events, and not their relative ordering, relying on the retrieval algorithm to match them in the order they were observed. We think that indexing plan actions based also on their ordering might be beneficial as a way to limit the search space.

Another possible improvement is the addition of a measure of *salience* to each episode, a reinforcement or penalty for an episode based on the correctness of its prior predictions. Having such a mechanism seems useful when dealing with noisy data.

The current approach builds implicit expectations in the form of candidate episodes that match the events observed so far. A way to improve retrieval speed is for memory to actively look for how these candidate episodes differ from each other and test for the presence/absence of those differences in the new stimuli.

Expectations built by a generic episodic memory could be used to focus processing on confirming/disproving a smaller set of candidate hypotheses, by giving the application the choice of specifying the ordering function. For example, in a domain like crime prevention, one might want to test first the hypotheses that have the worst outcome, so that preventive measures could be taken as quickly as possible.

Incremental recognition seems a promising way to manage the complexity of matching knowledge structures. Plans as sequences of actions suggest a straight-forward way of dividing such structures into small, coherent pieces. It would be interesting to extend this idea to other kinds of knowledge structures by using some sort of attention focussing mechanism that can serve up small chunks of knowledge to the incremental recognizer.

Complex plans happen over longer periods of time and consist of many low-level events. They are unlikely to be recognized just by looking at these individual events. A good recognizer needs to be able to recognize subgoals and use them in the subsequent recognition process.

In complex domains it is likely that an agent will carry on multiple plans at the same time, interleaving their actions. A recognizer will have to be able to deal with these different plans unfolding at the same time and still perform recognition on such data. Unlike statistical approaches that might require major changes to accomplish this, the proposed memory based plan recognition lends itself easily to this task. Episodic-based plan recognition is already able to entertain multiple hypotheses at the same time. We think that the only required change consists in adjusting the similarity measure so as to reward episodes that match actions not matched by other episodes, actions most likely part of a different ongoing plan.

## 6.5 Chapter Summary

In this chapter we have presented an episodic-based approach to plan recognition that uses the generic memory module for events to store and retrieve plans.

We evaluated it against a statistical approach on three tasks in two domains and found that it achieves similar if not better performance in all but one of the task-domain combinations. We think that a better adaptation strategy might improve these results. The incremental retrieval algorithm proved efficient and scalable.

Unlike statistical approaches that train different recognizers for the task of goal schema and parameter recognition, ours captures this knowledge at once.

Due to the generic nature of the memory module, its organization and its retrieval algorithm, this approach should be easily portable to new domains. The memory structure built by the plan recognizer as well as its retrieval algorithm are multi-functional and can be used for other purposes, like plan generation or classification (e.g. of goals as solvable or not, of plans as likely to succeed or fail).

# Chapter 7

# Memory-Based Problem Solving

In this chapter we present an application of the episodic memory module to problem solving. Past experience is not commonly used by intelligent systems in conjunction with other reasoning mechanisms.

Take for example Project Halo [57, 41] which attempts to develop tools that will allow scientists without expertise in knowledge engineering to formulate, debug, extend, validate and query knowledge bases. These knowledge bases are intended to answer novel AP-level questions and provide domain appropriate explanations of how those answers were derived. The subject matter experts will develop these knowledge bases by teaching a system domain concepts, relations and methods for solving problems.

Answering such questions implies finding appropriate models in the knowledge base that can provide meaningful answers and explanations of how they were obtained. A selection process is usually necessary to select the set of appropriate models that solve a question. By incorporating prior experience, an intelligent system will be able to make more informed decisions while searching for an appropriate model. In this way a system can improve its performance (by speeding up the search process) and even its competence (by avoiding mistakes made answering similar questions in the past).

We have attached the memory module to a problem solver [25] and tested its performance in the context questions answering of Project Halo. The memory module significantly improved the performance of the problem solver.

## 7.1 The Problem Solving Problem

The problem of answering complex questions posed in natural language requires more than producing a valid, logical interpretation of the user's question. It requires finding an appropriate model that can derive the desired answer. This is referred to as *the problem-solving problem*.

For example, in Newtonian physics one may have learned models (a system of objects, relationships, and equations) about moving objects, or two-object collision, or movement with friction. Then, given a new problem (e.g. a description of a specific object moving in a specific way), the challenge is to map this problem onto the appropriate model (or set of models) that contains the necessary knowledge to answer the question. An example of such a question in Physics is presented in Figure 7.1.

```
An object starts from rest
and reaches a speed of 28 m/s in 2 s.
What distance does it cover?
```

Figure 7.1: An example of question in Physics.

Besides the question itself, problems in scientific domains tend to have an elaborate **scenario** - a description of the specific objects involved and relations among them [27]. Information contained in the scenario is crucial in *identifying the right set of models* to use in answering that specific question.

Only in a few cases the question will provide exactly the right information so that the appropriate model(s) can be applied through standard deductive reasoning (e.g. inheritance). In general, a search process is needed to find potentially applicable models.

The size of the set of models to be searched (i.e. knowledge base) and the complexity of individual models (e.g. number of axioms) might render an uninformed search process either unfeasible or incomplete. Another complication comes from the fact that finding whether a model is applicable to a given scenario might be an expensive process. Memory can help a problem solver alleviate both these problem by offering fast, accurate and content-based access to previous problem solving episodes that are used to guide the search. In this way, more problem solving time is spent on models more likely to be relevant, potentially increasing both performance and competence.

## 7.2 The Basic Problem Solver

The Basic Problem Solver (BPS) [25, 26] is a state-space search process that is given *a logical representation of a question* and *a set of models*. It uses semantic matching [134] to choose the most appropriate models for the current question. Selected models are *applied* to the current problem description (called **scenario**), generating a new problem description by expanding the previous one with new facts. The process continues until an answer for the given question is found.

To illustrate how this works, consider the previous question in Figure 7.1, expressed in simplified English [28]:

```
An object moves.
The initial speed of the object is 0 m/s.
The final speed of the object is 28 m/s.
The duration of the move is 2 s.
What is the distance of the move?
```

Figure 7.2: An example of question in simplified English.

The formulation is translated into a logical form [28] which is passed on to BPS. The initial scenario contains the logical form (an instantiation of the `Move` concept with the initial and final speed values for its `object`). In this form, the question cannot be answered, as the concept `Move` does not contain any equations relating queried variable (`distance`) to those known (`initial-speed`, `final-speed` and `duration`).

To overcome this problem, BPS systematically explores the space of models looking for an adequate model to apply to the current scenario. Model application is expensive, so the problem solver is selective in the models it applies. The semantic match score between a model and a scenario is used to guide model selection. However, as semantically matching against *all models* in the KB is infeasible, BPS *heuristically* guides this search so that the models that best match the scenario are considered first. This heuristic is based solely on the contents of models and scenarios, not on past applications of those models (i.e. problem solving episodes).

For the example in Figure 7.2, the model `Move-with-constant-acceleration` is eventually found; it contains the appropriate equations and is able to answer the question.

## 7.3 Episodic-Based Problem Solving

We have applied the episodic memory module to the problem of model selection in problem solving, replacing BPS's concept selection heuristic. The advantage of using a memory based concept selection strategy comes from the fact that it uses models that have solved similar scenarios in the past, and not the contents of a model, which might not be very similar to a scenario.

Such a memory-based approach needs appropriate training data. Memory will be presented with successful problem-solving episodes, each consisting of a scenario (the episode context) and the model that was applied to solve it (the episode contents).

The problem solver will call memory every time a model is needed to expand the current scenario. Given a scenario, memory will retrieve the most similar prior episodes based on their context (i.e. scenario), adapt their contents (i.e. set of models) and present them to the Basic Problem Solver.

## 7.4 Experimental Evaluation

We evaluated a version of the Basic-Problem Solver enhanced with the episodic memory module against the original system on two datasets collected during the course of Project Halo [57].

In this experiment we test whether memory can provide the same level of problem solving accuracy as search, while improving problem solving time. We also measure memory overhead and how this grows with memory size.

### 7.4.1 Dataset

The dataset consists of questions in the Physics domain formulated by Subject Matter Experts (SMEs) in the context of one of Project Halo's evaluations. All questions involved problem-solving (i.e. required finding and applying a model in order to be solved); there were no questions that asked for only for descriptions of concepts (e.g. 'What is circular motion?').

These questions were formulated in simplified English by SMEs and translated into logical forms. Question interpretation is outside the scope of this dissertation[1]. To separate it from question answering, we ran the interpreter on all questions and manually graded its output. We selected only those questions that were correctly interpreted.

To obtain the training set, we ran the problem solver on the set of correct question interpretations and manually graded their answers and explanations. We then selected those questions that were both correctly answered and explained.

Depending on the number of models required to solve a question we divided the dataset in two:

**single-model questions** - those questions that can be answered by applying a single model. An example of a single-model question is presented in Figure 7.2. There were 49 single-model questions correctly answered by BPS.

[1]See [31, 29, 28, 135, 38] for more details.

**multi-model questions** - the questions that require the use of more than one model in order to be correctly answered. There were only five that were correctly answered by BPS.

### 7.4.2 Background Knowledge

The knowledge base used in this evaluation was developed for Project Halo by a Knowledge-Engineer, other than the author. It contained 13 models of physical phenomena that encoded knowledge from three chapters of a popular college-level Physics textbook, including linear motion with constant velocity, linear motion with constant acceleration, free fall, motion under force, parabolic motion, etc. Answering required only eight of these 13 models.

### 7.4.3 Experimental Setup

In order for the retrieved models to be usable by the problem solver, an adaptation step is required. During this step, each retrieved episode is semantically matched against the given scenario and a set of correspondences is computed. Even though this adaptation step is not part of the memory module, since it is required for compatibility with the Basic Problem Solver, it is part of the overhead incurred when using a memory.

To evaluate the effects of using a memory on problem solving we measured several things:

**problem-solving accuracy** - how many questions were answered correctly; we do not want the memory-enhanced problem solver to sacrifice competence for efficiency.

**problem-solving time** - the time required for a question to be answered. This is the total CPU time required to do problem solving, without question interpretation; this measure includes garbage collection time - as our algorithms were implemented in Lisp we did not have a fine control of the garbage collection mechanism.

While this is the most important measure from a user's stand-point, problem-solving time only tells how fast the overall system is. Problem-solving time includes:

**retrieval time (EM-ret)** - the time required for memory to retrieve appropriate prior episodes; this does not include adaptation time; We are interested in how this grows with the number of stored episodes, as a way to test memory scalability. Memory can be called multiple times while a question is answered. EM-ret is measured as the sum of each of those individual retrieval times.

**adaptation time (EM-adapt)** - the time required for retrieved episodes to be adapted so that they can be used by problem solver; EM-adapt is the sum of the adaptation time for each individual call to memory during the answering of one question.

**search time (PS)** - time spent by the Basic Problem Solver deciding whether one of the models returned by memory provides the answer to the given question; it does not include memory retrieval time, nor adaptation time.

**explanation time (EXPL)** - the time required to generate an explanation once an answer is found.

The following relation holds between these quantities:

**Problem-Solving-Time = EM-ret + EM-adapt + PS + EXPL**

**number of explored nodes in the search** - measures how big the search graph created by BPS was. It is directly influenced by the number of candidate models considered by the problem solver. It has the nice property that, unlike timing data, is machine-independent.

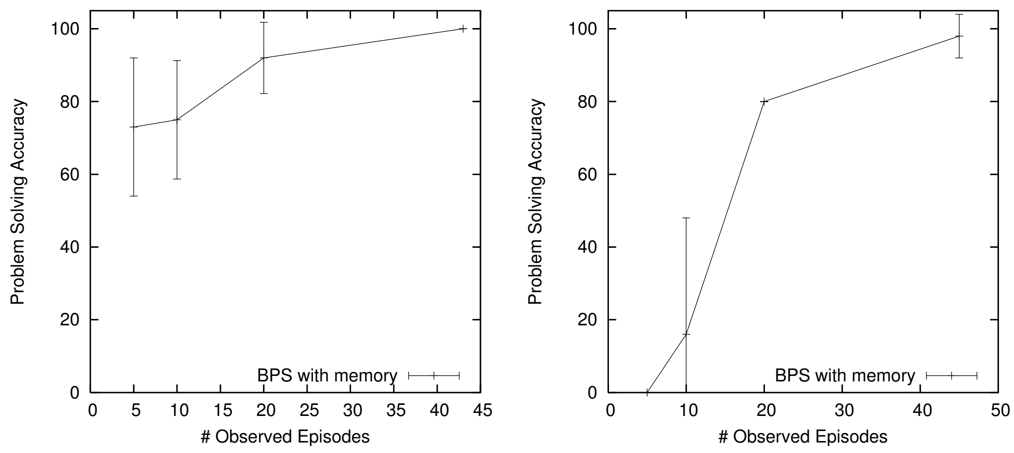
All episodes presented to memory were stored. A maximum of three remindings were used and the best three episodes retrieved by memory were suggested to the problem solver. Similar results were obtained using five remindings and retrieving five episodes. Since a model might match a scenario in multiple ways, memory adaptation might generate more than three (or five) candidates.

For the single-model question dataset, we performed a standard 10-fold cross validation. For the multi-model question set, given its small size, we divided the single-model question set in 10 equal parts and trained on nine of them, testing on the whole multi-model set. We did this 10 times, each time leaving out a different part of the training set.

### 7.4.4 Results

#### Accuracy

Figure 7.3 presents the results for problem solving accuracy for the memory-based problem solver for the single and multi-model question datasets.



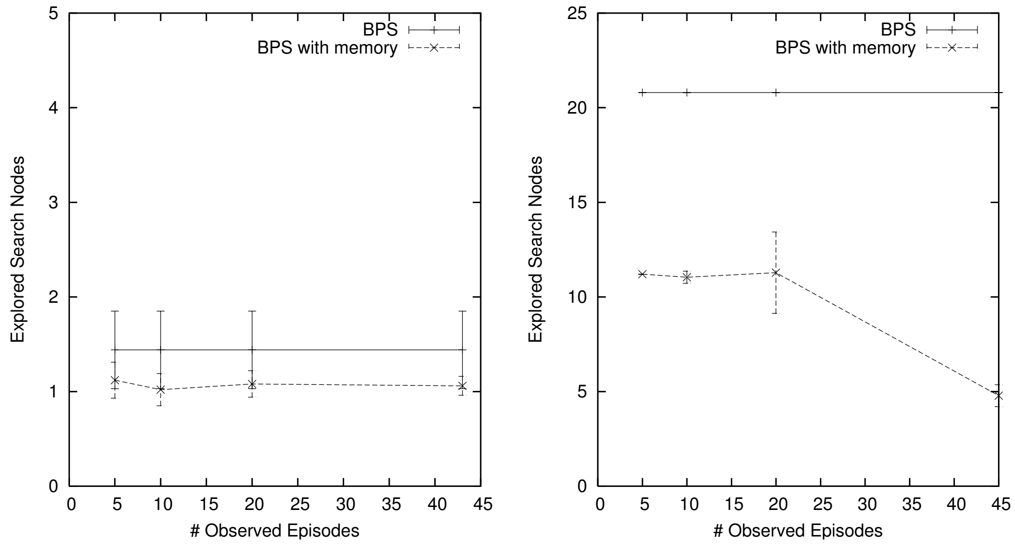
(a) Single model question dataset (b) Multi model question dataset

Figure 7.3: Problem solving accuracy with episodic memory for single and multi-model questions. Error-bars represent the standard deviation.

The memory-enhanced problem solver achieves perfect accuracy on single-model questions and almost perfect accuracy on multi-model questions.

The memory enhanced problem-solver improves faster for single model questions (see Figure 7.3(a)) than for multi-model questions (see Figure 7.3(b)). We attribute this difference to the fact that the multi-model questions used a small subset of the available models, for which there were few training examples. After those examples were stored in memory, performance improved such that there was no significant difference at the end of training. Even though multi-model questions use only a small number of models, the rest of the stored episodes act as a confuser set. This is why in one of the trials, the accuracy after training did not reach exactly 100%.

### Number of explored nodes in the search space



(a) Single model question dataset (b) Multi model question dataset

Figure 7.4: Comparison between number of search space nodes explored by the problem solver with and without episodic memory for single and multiple model questions. Errorbars represent the standard deviation.

Figure 7.4 plots the number of explored nodes in the search space by BPS with and without memory for the single and multi-model question datasets. From the single model question dataset (Figure 7.4(a)) we see that BPS's concept selection heuristic is pretty accurate, suggesting at most one extra model before the correct one (mean = 1.440, sd = 0.408). However, using memory (Figure 7.4(a)) we are able to reduce this even further so that almost always the correct model is ranked first (mean = 1.065, sd = 0.100 after training). The difference between the two is significant at least at the 0.05 level across all test points using a two-tailed t-test.

For single-model questions, this might not be a huge improvement, but for multiple-model questions where the search space grows exponentially, this will have a much greater impact. Figure 7.4(b) shows the number of explored search nodes for the multi-model question dataset. While BPS without memory explored 20.8 nodes, BPS with memory explored less than half of that (mean = 6.540, sd = 0.780 after training). Again the difference is significant at the 0.001 level using a two-tailed t-test.

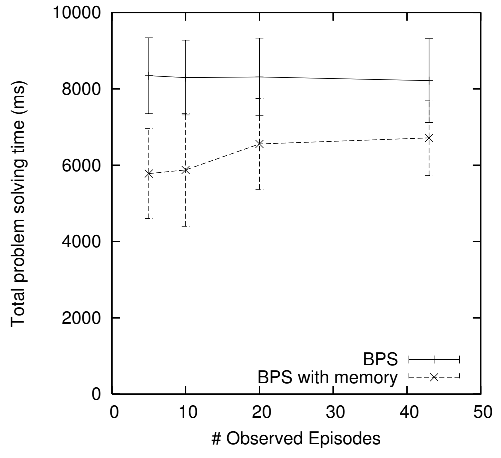
### Total problem-solving time

While memory significantly reduced the number of explored search space nodes for both single and multi-model questions dataset, this measure does not take into account the overhead incurred by using memory. We look now at the total problem-solving time.

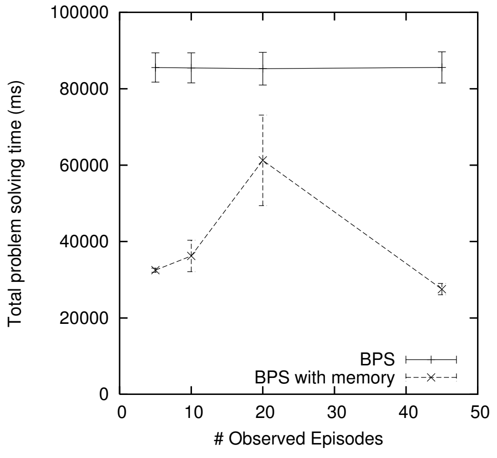
Figure 7.5(a) shows the problem solving time for single model questions for BPS with and without memory. The reduction in number of modes searched, as well as focusing on models that worked in the past, significantly reduces the problem-solving time (difference is significant at least at the 0.01 level across all testing points). This difference is even greater for multi-model questions (Figure 7.5(b)).

Figure 7.6 shows the break-down of the total problem-solving time for single-model questions for BPS with and without memory. It is interesting to note that memory overhead is rather low (Figure 7.6(a)), and decreases as training proceeds. At the end of training memory overhead time is 723.75 ms, sd = 257.58 ms. On average 227.10 ms (sd = 38.87 ms) were spent in memory retrieval alone. This is due to the fact that a mature memory provides more relevant concepts to BPS, which in turn calls memory less often.

For the single model questions, both with and without memory, explanation time and problem solving time are about the same (Figure 7.6(a)-7.7(a)).



(a) Single model question dataset



(b) Multi model question dataset

Figure 7.5: Comparison between problem solving times and without episodic memory for single and multiple model questions. Error-bars represent the standard deviation.

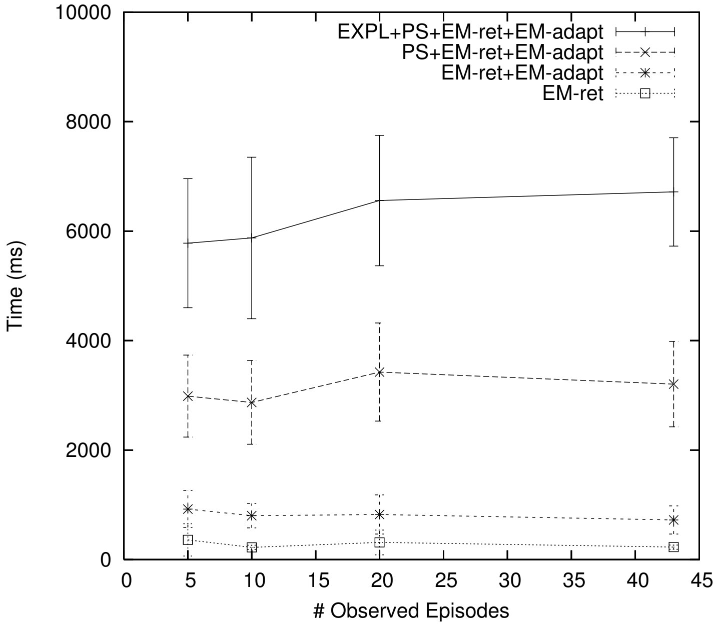
Figure 7.7 shows the break-down of the total problem-solving time for multi-model questions for BPS with and without memory. Memory overhead is again low - 2011.80 ms (sd = 346.84 ms) and represents only a 7.3% of the total problem solving time (mean = 27531.60 ms, sd = 1474.69 ms).

The increase in problem-solving time from the 5th training episode until the 20th can be explained in conjunction with memory performance: the accuracy is not very good, so the problem solver calls memory repeatedly to get the right models.

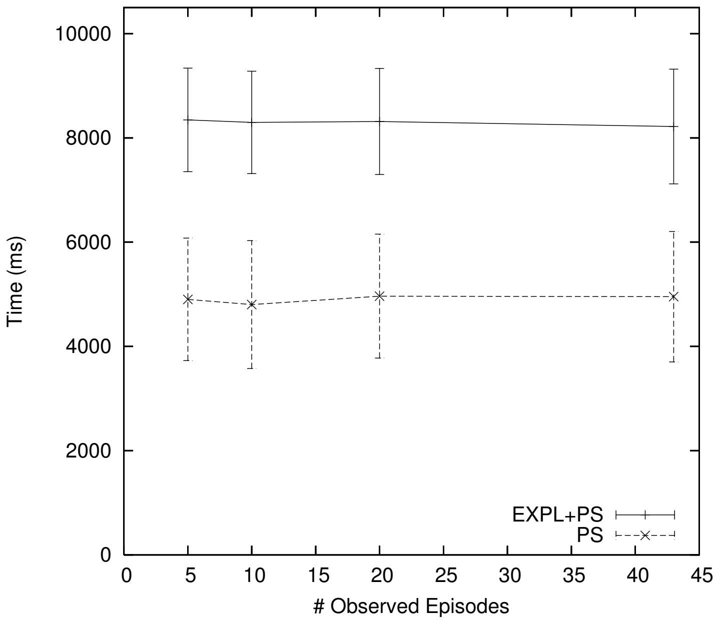
### 7.4.5 Discussion and Future Work

A similar approach is presented in [62]. It uses MAC/FAC [40] as the retrieval model and SME [43] to compute the match between two questions. The authors evaluate the problem solving performance separately according to how different the test questions were compared to the training set (i.e. the type of 'transfer learning' needed). The experiments presented do not address measure retrieval efficiency or the scalability of this approach.

Given the similarity of both the adopted approach and datasets, if would be in-

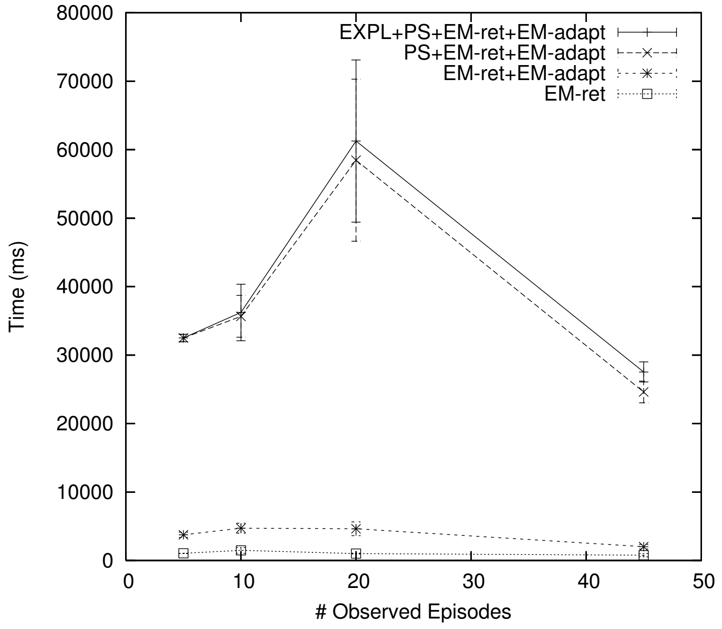


(a) BPS with episodic memory

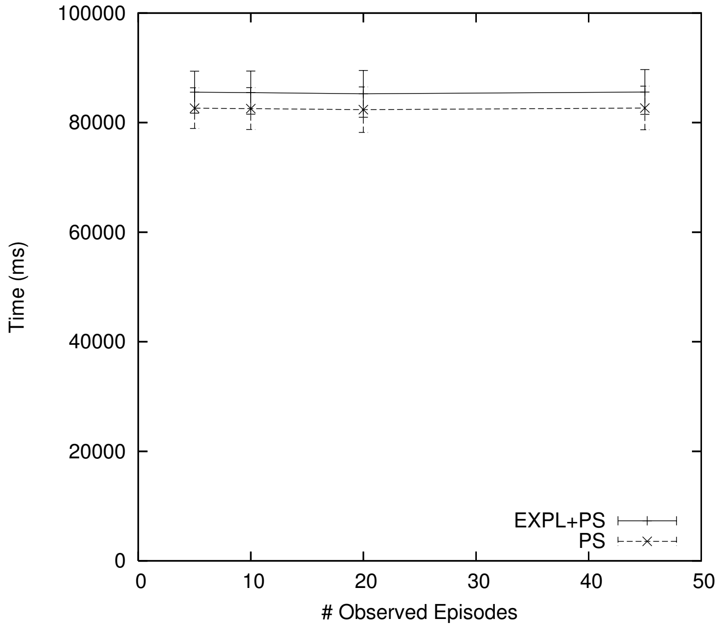


(b) BPS

Figure 7.6: Problem solving time break-down for answering single model questions with and without episodic memory. Error-bars represent the standard deviation.



(a) BPS with episodic memory



(b) BPS

Figure 7.7: Problem solving time break-down for answering multiple model questions with and without episodic memory. Error-bars represent the standard deviation.

teresting to perform a direct comparison of the our episodic approach and this. We are interested in how the episodic-based approach would perform on various ‘transfer learning’ categories, as well as how the MAC/FAC retrieval model would scale.

There are other ways of using memory in problem solving. What we presented here was a rather low-level integration between a problem solver and a memory module. A higher-level one would be to do episodic replay (i.e. analogical replay) by applying the same models as in a previous episode, in the same order. This has the potential of further reducing problem-solving time as the need for search will be drastically reduced.

The other two domains used in Project Halo (biology and chemistry) present another interesting challenge for question answering, as they do not require systems of equations to compute the answers. Rather, the emphasis is on finding the correct analogue in memory that might provide the answer. For example, a typical question found in both chemistry and biology is of the type: ’Two soluble substances react, producing an insoluble substance. What kind of reaction is this?’; the correct answer is `Precipitation-Reaction`.

The episodic memory module is applicable in this case as well, by indexing the concepts in the knowledge base based on their description and suggesting targets for their possible reclassification.

## 7.5 Chapter Summary

In this chapter we presented an application of our episodic memory module to problem solving. We showed that it significantly decreases total processing time for both simple (i.e. single model) and complex (i.e. multi-model) questions. It does so by reducing the number of nodes in the search graph of the problem solver as a result of providing accurate candidates for scenario expansion. This is achieved while incurring a very low overhead.

The use of memory in problem solving is definitely not novel. However, our approach differs from others (e.g. [121]) in that the memory module is *generic* and was not designed to work with this or any other problem solver. This module can be attached to a variety of intelligent systems to enhance them with episodic memory functionality, *in addition* to their own reasoning mechanisms.

# Chapter 8

# Summary and Future Work

## 8.1 Dissertation Summary

### 8.1.1 The Need for Memory

The ability to remember past experiences enables a system to improve its performance as well as its competence. The increasing complexity of such experience and the longer life-expectancy of today's intelligent systems impose additional constraints on their memory subsystems, like the ability to deal in a scalable manner with temporal experience with deep associated semantics.

In order to achieve broad coverage, experience needs to be used in conjunction with other reasoning mechanisms. That is why we need the ability to *add episodic memory* functionality to intelligent systems. This requirement is underlined by the goal of developing such systems in a modular fashion, using generic, reusable components.

To address these requirements, we proposed to *separate the episodic memory functionality* from the system that uses it and to build a *generic, reusable memory module* that can be attached to a variety of applications in order to provide this functionality. The development of such a generic, reusable memory module will allow *easy portability* to different systems and applications. It will enable systems that were not designed to rely on a memory system to benefit from it. Separating memory functionality from the system that uses it should also *reduce the overall complexity* of the system since it will not have to be concerned with this any more. An additional benefit of having reusable memory modules is allowing *research to focus on the generic aspects of memory representation, organization and retrieval* and its interaction with the external application.

### 8.1.2 Memory Requirements

We investigated the set of requirements that such a memory module should follow. In terms of episodic encoding, a memory module should provide *a generic representation of events*, that can be used with different types of events and *a domain independent organization* of these events that supports *flexible retrieval*.

Memory should be able to *store a large number of episodes*, acquired during its functioning, and do so *efficiently*. The storage time should not significantly increase with the number of stored episodes (*scalability*). The decision on what items to store or discard from memory should be based on *preserving competence*.

The retrieval algorithm of a generic memory module for events should be *accurate*, *efficient* and *scalable*. The importance of scalability grows with the life expectancy of the system. Memory items should be *addressable by their content*, which allows external applications to formulate flexible queries. The relevant prior episodes should be retrieved even if they only partially match the current context (*flexible match*).

Given that such a generic memory module is intended to be implemented as a stand-alone application, it needs to provide a clean but *flexible programming interface* (API) that external applications can use. Results of querying memory should also provide an *explanation* of why they were retrieved (e.g. what the similarity between the query and the retrieved memory items was). Such knowledge could be used by the external application in using the retrieval results in its application (e.g. by adapting them).

### 8.1.3 Implementation of the Generic Memory Module

We have implemented such a generic memory module. Episodes are represented along three dimensions: *context* (the general setting in which an episode happened), *contents* (the ordered set of events that make up the episode), and *outcome* (an evaluation of the episode's effect). The underlying knowledge representation is frame-based and relies on the KM language [32]. One of the distinguishing features of our implementation is that it uses semantic knowledge to encode the episodes. This knowledge is represented using the Component Library [14, 48].

Episodes are stored in memory unchanged and are indexed using a multiple-layer indexing scheme: by their feature types (*remindings*) and how they differ structurally (*difference links*). The decision on what constitutes and episode and when to store it was left to the external application using the memory.

We implemented a retrieval algorithm that given a stimulus and a dimension will select the set of most similar prior episodes. It does so by selecting candidate episodes based on their surface similarity to the stimulus, and then searching for the best match using a semantic matcher. An incremental version of this algorithm for retrieving sequences of events was also implemented.

### 8.1.4 Experimental Evaluation

We have evaluated the implementation of our memory module on three different tasks: memory-based planning in the Logistics domain, plan recognition in the Linux and Monroe domains, and memory-based problem solving for question answering for AP-level Physics questions. The memory module was easily applicable to these tasks and domains. We chose to do minimal adaptation of the retrieved episodes, so as to evaluate memory performance separate from adaptation.

Across these tasks and domains, memory proved efficient, accurate and scalable. The proposed indexing mechanism significantly increased performance over systematic search, while preserving competence (memory-based planning, and memory-based problem solving). This increase is considerably larger for search spaces with large branching factors as is the case in memory-based problem solving when multiple models need to be used.

Incremental episodic-based goal schema recognition achieved comparable precision, higher recall and higher convergence when compared to a standard statistical approach. For parameter recognition, the episodic-based approach provided higher recall, but lower precision and convergence than the chosen statistical approach.

Unlike specialized retrieval or recognition algorithms, our memory module builds *multifunctional* structures that can be reused for different tasks than those trained on. The episodic approach acquires a plan library incrementally, and can make use of available domain knowledge for retrieval purposes.

## 8.2 Additional Applications

### 8.2.1 Incremental Recognition

Incremental recognition seems to be a promising way to manage the complexity of matching knowledge structures. Plans are already represented as sequences of actions, which suggests a straight-forward way of dividing such structures into small, coherent pieces. It would be interesting to extend this idea to other kinds of knowledge structures by using some sort of attention focussing mechanism that can serve up small chunks of knowledge to the incremental recognizer.

### 8.2.2 Hierarchical Recognition

Complex plans happen over longer periods of time and consist of events that are presumably at a much lower level of generality than the overall goal of the agent. Recognizing goals at intermediate levels of this taxonomy could reduce the uncertainty introduced by this representation gap [20].

### 8.2.3 Multiple Plan Recognition

In complex domains it is likely that an agent will carry on multiple plans at the same time, interleaving their actions. A recognizer will have to be able to deal with these different plans unfolding at the same time and still perform recognition on such data.

The proposed memory based plan recognition lends itself easily to this task. Episodic-based plan recognition is already able to entertain multiple hypotheses at the same time. We think that the only required change consists in adjusting the similarity measure. The modified similarity measure should reward episodes that match actions not matched by other episodes, actions most likely part of a different ongoing plan.

### 8.2.4 Episodic Memory in Natural Language Question Answering

Interpreting natural language is a memory-intensive task. It involves references to past events, making it well suited for a memory approach ([100, 74]). Answering questions expressed in natural language could suffer from a number of problems such as: missing assumptions from the question formulation, abstraction of details (e.g. linguistic cues need to be expanded in order to recover them), contradiction, alternative representation [27]. Leveraging prior experience could provide assistance to processes such as semantic matching that can help alleviate some of these problems. For example, fast access to relevant knowledge structures could improve the performance of an NL system using a semantic matcher.

# Bibliography

[1] AIPS 2000. The Fifth International Conference on Artificial Intelligence Planning and Scheduling Systems. http://www.cs.toronto.edu/aips2000/, 2000.

[2] Agnar Aamodt and Enric Plaza. Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. *AI Communications*, 7(1), 1994.

[3] David W. Albrecht, Ingrid Zukerman, Ann E. Nicholson, and Ariel Bud. Towards a Bayesian Model for Keyhole Plan Recognition in Large Domains. In Anthony Jameson, Cécile Paris, and Carlo Tasso, editors, *Proceedings of the Sixth International Conference on User Modeling (UM97)*, pages 365–376, 1997.

[4] J Allen and C R Perrault. Analyzing Intention in Utterances. In *Readings in natural language processing*, pages 441–458. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1986.

[5] James F. Allen and George Ferguson. Actions and Events in Interval Temporal Logic. *Journal of Logic Computation*, 4(5):531–579, 1994.

[6] Klaus-Dieter Althoff, Andreas Birk, Christiane Gresse von Wangenheim, and Carsten Tautz. CBR for Experimental Software Engineering. In *Case-Based Reasoning Technology, From Foundations to Applications*, pages 235–254, London, UK, 1998. Springer-Verlag.

[7] Klaus-Dieter Althoff and Rosina Weber. Knowledge Management in Case-Based Reasoning. *The Knowledge Engineering Review*, 20:305–310, 2005.

[8] John R. Anderson. A Spreading Activation Theory of Memory. *Journal of Verbal Learning and Verbal Behavior*, 22:261–295, 1983.

[9] John R. Anderson. *The Architecture of Cognition*. Harvard University Press, Cambridge, MA, USA, 1983.

[10] John R. Anderson. *The Adaptive Character of Thought*. Lawrence Erlbaum Associates, Hillsdale, NJ, USA, 1990.

[11] John R. Anderson. *Cognitive Psychology and Its Implications*. Worth Publishing, New York, Fifth edition, 2000.

[12] Ray Bareiss. *Exemplar-Based Knowledge Acquisition - A Unified Approach to Concept Representation, Classification and Learning*. Academic Press, San Diego, CA, 1989.

[13] Ray Bareiss and J. A. King. Similarity Assessment in Case-Based Reasoning. In *Proceedings of The Second Workshop on Case-Based Reasoning*, pages 67–71, Pensacola Beach, FL, 1989.

[14] Ken Barker, Bruce Porter, and Peter Clark. A Library of Generic Concepts for Composing Knowledge Bases. In *K-CAP 2001: Proceedings of the International Conference on Knowledge Capture*, pages 14–21. ACM Press, 2001.

[15] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T.N. Bhat, H. Weissig, I.N. Shindyalov, and P.E. Bourne. The Protein Data Bank. *Nucleic Acids Res*, 28(1):235–42, 2000.

[16] Stefano Berretti, Alberto Del Bimbo, and Enrico Vicario. Efficient Matching and

Indexing of Graph Models in Content-Based Retrieval. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(10):1089–1105, 2001.

[17] N. Blaylock and J. Allen. Statistical goal parameter recognition. In *The 14th International Conference on Automated Planning and Scheduling (ICAPS'04)*, 2004.

[18] Nate Blaylock and James Allen. Generating Artificial Corpora for Plan Recognition. In Liliana Ardissono, Paul Brna, and Antonija Mitrovic, editors, *Lecture Notes in Artificial Intelligence - User Modeling 2005*, volume 3538. Springer, 2005.

[19] Nate Blaylock and James Allen. Recognizing Instantiated Goals Using Statistical Methods. In Gal Kaminka, editor, *IJCAI Workshop on Modeling Others from Observations (MOO-2005)*, pages 79–86, Edinburgh, 2005.

[20] Nate Blaylock and James Allen. Hierarchical Instantiated Goal Recognition. In *AAAI Workshop on Modeling Others from Observations (MOO-2006)*, 2006. AAAI Technical Report WS-06-13.

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

[22] Jaime G. Carbonell. Derivational Analogy: A Theory of Reconstructive Problem Solving and Expertise Acquisition, 1986.

[23] David Chapman. Planning for Conjunctive Goals. *Artificial Intelligence*, 32(3):333–377, 1987.

[24] Eugene Charniak and Robert P. Goldman. A bayesian model of plan recognition. *Artificial Intelligence*, 64(1):53–79, 1993.

[25] Shaw-Yi. Chaw, Ken Barker, Bruce Porter, and Peter Yeh. Towards an Ontology-

Independent Problem-Solver. Technical report, The University of Texas at Austin, 2007. AI Technical Report TR-07-349.

[26] Shaw-Yi Chaw and Bruce Porter. A Knowledge-Based Approach to Answering Novel Questions. In *Proceedings of the 3rd International Workshop on Knowledge and Reasoning for Answering Questions (KRAQ 2007)*, Hyderabad, India, January 2007.

[27] Shaw-Yi Chaw, Dan Tecuci, Peter Yeh, and James Fan. Capturing a Taxonomy of Failures During Automatic Interpretation of Questions Posed in Natural Language. In *Proceedings to The Fourth International Conference on Knowledge Capture (K-CAP)*, 2007. To Appear.

[28] P. Clark et al. Using a Controlled Language for Posing Questions to a Knowledge-Based System (submitted). 2007.

[29] P. Clark, P. Harrison, T. Jenkins, J. Thompson, and R. Wojcik. Acquiring and Using World Knowledge using a Restricted Subset of English. In *Proceedings to The 18th International FLAIRS Conference (FLAIRS'05)*, 2005.

[30] Peter Clark. PROTOS - A Rational Reconstruction. Technical report, Turing Institute, Glasgow, UK, 1987.

[31] Peter Clark and Phil Harrison. Controlled language processing for Halo question-asking, 2003.

[32] Peter E. Clark and Bruce W. Porter. *KM v2.0 - The Knowledge Machine: User's Manual and Situations Manual*. University of Texas at Austin, 2001.

[33] N.J. Cohen and H. Eichenbaum. *Memory, Amnesia, and the Hippocampal System*. MIT Press, 1995.

[34] M. T. Cox and B. Kerkez. Case-based plan recognition with novel input. *Control and Intelligent Systems*, 34(2):96–104, 2006.

[35] Will Dodd. The Design of Procedural, Semantic, and Episodic Memory Systems for a Cognitive Robot. Master's thesis, Vanderbilt University, 2005.

[36] Herman Ebbinghaus. *Memory: A Contribution to Experimental Psychology*. New York: Teachers College, Columbia University., 1885/1913.

[37] Werner Emde and Dietrich Wettschereck. Relational Instance-Based Learning. In L. Saitta, editor, *Proceedings to the 13th International Conference on Machine Learning*, pages 122–130. Morgan Kaufmann, 1996.

[38] James Fan and Bruce Porter. Interpreting loosely encoded questions. In *Proceedings of Nineteenth National Conference on Artificial Intelligence*. AAAI Press, 2004.

[39] Edward A. Feigenbaum. The Simulation of Verbal Learning Behavior. In *Computers & Thought*, pages 297–309. MIT Press, Cambridge, MA, USA, 1995.

[40] Kenneth Forbus, Dedre Gentner, and Kenneth Law. MAC/FAC: A Model of Similarity-Based Retrieval. *Cognitive Science*, 19(2):141–205, 1995.

[41] N. Friedland, P. Allen, P. Matthews, M. Witbrock, D. Baxter, J. Curtis, B. Shepard, P. Miraglia, J. Angele, S. Staab, E. Moench, H. Oppermann, D. Wenke, D. Israel, V. Chaudhri, B. Porter, K. Barker, J. Fan, S. Chaw, P. Yeh, D. Tecuci, and P. Clark. Project Halo: Towards a Digital Aristotle. *AI Magazine*, 2004.

[42] Peter Friedland and Yumi Iwasaki. The Concept and Implementation of Skeletal Plans. *Journal of Automated Reasoning*, 1(2):161–208, 1985.

[43] Dedre Gentner. Structure-mapping: A Theoretical Framework for Analogy. *Cognitive Science*, 7(2):155–170, April–June 1983.

[44] Dedre Gentner, Keith J. Holyoak, and Boicho N. Kokinov, editors. *The Analogical Mind: Perspectives from Cognitive Science*. MIT Press, Cambridge, MA, 2001.

[45] Ashok K. Goel, Khaled S. Ali, and Eleni Stroulia. Some Experimental Results in Multistrategy Navigation Planning. Technical report, Georgia Institute of Technology, 1996. GIT-CC-95-51.

[46] Bradley A. Goodman and Diane J. Litman. On the Interaction between Plan Recognition and Intelligent Interfaces. *User Modeling and User-Adapted Interaction*, 2(1-2):83–115, 1992.

[47] Barbara J. Grosz and Candace L. Sidner. Attention, intentions, and the structure of discourse. *Computational Linguistics*, 3(12):175–204, 1986.

[48] The MFKB group University of Texas at Austin. The Component Library. http://www.cs.utexas.edu/users/mfkb/tree, 2004.

[49] Rogers P. Hall. Computational Approaches to Analogical Reasoning: a Comparative Analysis. *Artificial Intelligence*, 39(1):39–120, 1989.

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

[51] Kristian J. Hammond. *Case-Based Planning: Viewing Planning as a Memory Task*. Academic Press, 1989.

[52] Kristian J. Hammond. Explaining and repairing plans that fail. *Artificial Intelligence*, 1-2(45):173–228, 1990.

[53] Steve Hanks and Daniel S. Weld. A Dmain-Independent Algorithm for Plan Adaptation. *Journal of Artificial Intelligence Research*, 2:319–360, 1995.

[54] Douglas R. Hofstadter. Epilogue: Analogy as the Core of Cognition. In Dedre Gentner, Keith J. Holyoak, and Boicho N. Kokinov, editors, *The Analogical Mind: Perspectives from Cognitive Science*, chapter 15, pages 499–538. MIT Press, 2001.

[55] Stefan Huwer. Domain independent plan recognition. Master's thesis, Computer Science Department, University of Dundee, Dundee, Scotland, 1993.

[56] Stefan Huwer, Bill Smart, and Alistair Cairns. Domain Independent Plan Recognition, 1995. http://citeseer.ist.psu.edu/huwer94domain.html.

[57] Vulcan Inc. Project HALO. http://www.projecthalo.com.

[58] Anthony G. Francis Jr. *Context-Sensitive Asynchronous Memory*. PhD thesis, Georgia Institute of Technology, August 2000.

[59] Henry A. Kautz. A theory of plan recognition and its implementation. In J. F. Allen, H.A. Kautz, and R.N. Pelavin, editors, *Reasoning about Plans*, chapter 2. Morgan Kaufmann, San Mateo, CA, 1997.

[60] Boris Kerkez and Michael T. Cox. Incremental case-based plan recognition using state indices. In *ICCBR '01: Proceedings of the 4th International Conference on Case-Based Reasoning*, pages 291–305, London, UK, 2001. Springer-Verlag.

[61] Dennis Kibler and David Aha. Learning Representative Exemplars of Concepts: An Initial Case Study. In J.W.Shavlik and T.G.Dietterich, editors, *Readings in Machine Learning*. Morgan Kaufmann, 1990.

[62] Matthew Klenk and Ken Forbus. Measuring the Level of Transfer Learning by an AP Physics Problem-Solver. In *Proceedings of the Twenty-Second Conference on Artificial Intelligence (AAAI-07).*

[63] Boicho Kokinov and Robert M. French. Computational Models of Analogy-making.

In L. Nadel, editor, *Encyclopedia of Cognitive Science*, volume 1, pages 113–118. Nature Publishing Group, London, 2003.

[64] Janet L. Kolodner. Organization and Retrieval in a Conceptual Memory for Events or Con54, Where are You. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence (IJCAI-81)*, pages 227–233, 1981.

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

[66] Richard E. Korf. Planning as Search: a Quantitative Approach. *Artificial Intelligence*, 33(1):65–68, 1987.

[67] John Laird. Episodic Memory vs. Case-Based Reasoning. Invited talk at the Sixth International Conference on Case-Based Reasoning (ICCBR), 2005.

[68] James H. Lawton, Roy M. Turner, and Elise H. Turner. A unified long-term memory system. In K.-D. Althoff, R. Bergmann, and L.K. Branting, editors, *Lecture Notes in Artificial Intelligence - ICCBR-99*, volume 1650, Berlin Heidelberg, 1999. Springer-Verlag.

[69] Michael Lebowitz. Memory-Based Parsing. *Artificial Intelligence*, 21:363–404, 1983.

[70] Douglas B. Lenat and R. V. Guha. *Building Large Knowledge-Based Systems: Representation and Inference in the CYC Project*. Addison-Wesley, Reading, Massachusetts, 1990.

[71] N. Lesh. *Scalable and Adaptive Goal Recognition*. PhD thesis, University of Washington, 1998.

[72] N. Lesh, C. Rich, and C. Sidner. Using Plan Recognition in Human-Computer Col-

laboration. In *Proceedings of the Seventh International Conference on User Modeling*, pages 23–32, 1999.

[73] Diane Litman and James Allen. A plan recognition model for subdialogues in conversation. *Cognitive Science*, 11:163–200, 1987.

[74] Kevin Livingston and Chris Riesbeck. Using Episodic Memory in a Memory Based Parser to Assist Machine Reading. In AAAI Press, editor, *Working notes of the AAAI 2007 Spring Symposium on Machine Reading*, 2007.

[75] Jixin Ma and Brian Knight. A Framework for Historical Case-Based Reasoning. In Kevin D. Ashley and Derek G. Bridge, editors, *ICCBR*, volume 2689 of *Lecture Notes in Computer Science*, pages 246–260. Springer, 2003.

[76] D. Marr. Simple Memory: A Theory for Archicortex. *Royal Society of London Philosophical Transactions Series B*, 262:23–81, July 1971.

[77] James Mayfield. Controlling Inference in Plan Recognition. *User Modeling and User-Adapted Interaction*, 2(1-2):83–115, 1992.

[78] Arthur W. Melton and Edwin Martin, editors. *Coding Processes in Human Memory*. John Wiley and Sons, New York, NY, 1972.

[79] Ray Mooney. Learning for semantic interpretation: Scaling up without dumbing down. In James Cussens, editor, *Proceedings of the 1st Workshop on Learning Language in Logic*, pages 7–15, Bled, Slovenia, 1999.

[80] Erik T. Mueller. *Daydreaming in Humans and Machines*. Ablex Publishing Corp., Norwood, NJ, USA, 1990.

[81] L. Nadel and M. Moscovitch. Memory Consolidation, Retrograde Amnesia and the Hippocampal Complex. *Current Opinion in Neurobiology*, 7(2):217–227, April 1997.

[82] L. Nadel, A. Samsonovitch, L. Ryan, and M. Moscovitch. Multiple Trace Theory of Human Memory: Computational, Neuroimaging and Neuropsychological Results. *Hippocampus*, 10:352–368, 2000.

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

[84] Bernhard Nebel and Jana Koehler. Plan Reuse versus Plan Generation: A Theoretical and Empirical Analysis. *Artificial Intelligence*, 76(1-2):427–454, 1995.

[85] Alan Newell. *Unified Theories of Cognition*. Harvard University Press, Cambridge, Massachusetts, 1990.

[86] Allen Newell and Herbert Simon. *Human Problem Solving*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1972.

[87] Andrew Nuxoll. Enhancing Intelligent Agents with Episodic Memory, 2002.

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

[89] Andrew M. Nuxoll and John E. Laird. Extending Cognitive Architecture with Episodic Memory. In *Proceedings of the Twenty-Second Conference on Artificial Intelligence AAAI 2007*. AAAI Press, 2007.

[90] National Library of Medicine. http://chem.sis.nlm.nih.gov/chemidplus.

[91] Hiroyuki Ogata, Susumu Goto, Kazushige Sato, Wataru Fujibuchi, Hidemasa Bono, and Minoru Kanehisa. KEGG: Kyoto Encyclopedia of Genes and Genomes. *Nucleic Acids Research*, 27(1):29–34, 1999.

[92] C. Raymond Perrault and James F. Allen. A Plan-Based Analysis of Indirect Speech Acts. *American Journal of Computational Linguistics*, 6(3-4):167–182, 1980.

[93] Euripides G. M. Petrakis and Christos Faloutsos. Similarity Searching in Medical Image Databases. *IEEE Transactions on Knowledge and Data Engineering*, 9(3):435–447, 1997.

[94] Martha Pollack. Plans as complex mental attitudes. In Philip R. Cohen, Jerry Morgan, and Martha Pollack, editors, *Intentions in Communication*, pages 77–103. MIT Press, Cambridge, Massachusetts, 1990.

[95] Bruce W. Porter, Ray Bareiss, and Robert C. Holte. Concept Learning and Heuristic Classification in Weak-Theory Domains. *Artificial Intelligence*, 45:229–263, 1990.

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

[97] Carlos Ramirez and Roger Cooley. A theory of the acquisition of episodic memory. In D. Aha and D. Wettschereck, editors, *ECML'97: Case-Based Reasoning Workshop: Beyond Classification of Feature Vectors*, pages 48–56, Prague, March 1997. Springer-Verlag.

[98] John W. Raymond, Eleanor J. Gardiner, and Peter Willet. RASCAL: Calculation of Graph Similarity using Maximum Common Edge Subgraphs. *The Computer Journal*, 45(6):631–644, 2002.

[99] Bradley J. Rhodes. The Wearable Remembrance Agent: A System for Augmented Memory. In *Proceedings of The First International Symposium on Wearable Computers (ISWC '97)*, pages 123–128, Cambridge, Mass., USA, 1997.

[100] Christopher K. Riesbeck. *From conceptual analyzer to Direct Memory Access Parsing: an overview.*, chapter 8. Ellis Horwood Limited, 1986.

[101] Earl D. Sacerdoti. Planning in a Hierarchy of Abstraction Spaces. *Artificial Intelligence*, 5(2):115–135, 1974.

[102] Miquel Sànchez-Marrè, Ulises Cortés, Montserrat Martínez, Joaquim Comas, and Ignasi Rodríguez-Roda. An Approach for Temporal Case-Based Reasoning: Episode-Based Reasoning. In Héctor Muñoz-Avila and Francesco Ricci, editors, *ICCBR*, volume 3620 of *Lecture Notes in Computer Science*, pages 465–476. Springer, 2005.

[103] D. L. Schacter. *Searching for Memory: The Brain, the Mind, and the Past*. Basic Books, 1996.

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

[105] Charles F. Schmidt, N. S. Sridharan, and J. L. Goodson. The plan recognition problem: An intersection of psychology and artificial intelligence. *Artificial Intelligence*, 11:45–83, 1978.

[106] Dennis Shasha, Jason T. L. Wang, and Rosalba Giugno. Algorithmics and Applications of Tree and Graph Searching. In *PODS '02: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 39–52, New York, NY, USA, 2002. ACM Press.

[107] Lokendra Shastri. Episodic memory trace formation in the hippocampal system: a model of cortico-hippocampal interaction. Technical Report TR-01-004, International Computer Science Institute, Berkeley, CA, 2001.

[108] Lokendra Shastri. Episodic memory and cortico-hippocampal interactions. *Trends in Cognitive Sciences*, 6(4):162–168, 2002.

[109] Lokendra Shastri. Calo episodic memory architecture. online presentation, 2004.

[110] Barry Smyth and Mark T. Keane. Remembering to Forget: A Competence-Preserving Case Deletion Policy for Case-Based Reasoning Systems. In *Proceedings to the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 377–383, 1995.

[111] J. F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1984.

[112] Luca Spalazzi. A Survey on Case-Based Planning. *Artificial Intelligence Review*, 16(1):3–36, 2001.

[113] L. R. Squire. Memory and the Hippocampus: A Synthesis from Findings with Rats, Monkeys, and Humans. *Psychological Review*, 99:195–231, 1992.

[114] Srinath Srinivasa and Sujit Kumar. A Platform Based on the Multi-Dimensional Data Model for Analysis of Bio-molecular Structures. In *VLDB*, pages 975–986, 2003.

[115] Dan Tecuci. A Generic Episodic Memory Module. Ph.D. Proposal, August 2005. http://www.cs.utexas.edu/ tecuci/proposal/proposal.pdf.

[116] Dan Tecuci and Bruce Porter. Using an Episodic Memory Module for Pattern Capture and Recognition. In Ken Murray and Ian Harrison, editors, *Capturing and Using Patterns for Evidence Detection: Papers from the 2006 Fall Symposium.*, Menlo Park, CA, 2006. AAAI Press. Technical Report FS-06-02.

[117] Dan Tecuci and Bruce Porter. A Generic Memory Module for Events. Key West, FL, 2007. Proceedings to the 20th Florida Artificial Intelligence Research Society Conference (FLAIRS20).

[118] Endel Tulving. Episodic and semantic memory. In E. Tulving and W. Donaldson, editors, *Organization of Memory*. Academic Press, 1972.

[119] Endel Tulving. *Elements of Episodic Memory*. Clarendon Press, Oxford, 1983.

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

[121] Manuela M. Veloso. *Learning by Analogical Reasoning in General Problem Solving*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, August 1992. Available as technical report CMU-CS-92-174.

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

[123] Steven Vere. Organization of the Basic Agent. *SIGART Bulletin*, 2(4):164–168, 1991.

[124] Steven Vere and Timothy Bickmore. A Basic Agent. *Computational Intelligence*, 4:41–60, 1990.

[125] Rosina Weber, David Aha, and I. Becerra-Fernandez. Intelligent Lessons Learned Systems. *International Journal of Expert Systems — Research & Applications*, 20(1), 2001.

[126] P. Willett, J.M. Barnard, and G.M. Downs. Chemical similarity searching. *Journal of Chemical Information and Modeling*, 38(6):983–996, November 1998.

[127] Terry Winograd. Frame representations and the procedural - declarative controversy. In D.G. Bobrow and A. Collins, editors, *Representation and Understanding*, pages 185–210. Academic Press, 1975.

[128] J. T. Wixted. Analyzing the empirical course of forgetting. *Journal of Experimental Psychology: Learning, Memory & Cognition*, 16:927–935, 1990.

[129] J. T. Wixted and E. B. Ebbesen. On the Form of Forgetting. *Psychological Science*, 2:409–415, 1991.

[130] J. T. Wixted and E. B. Ebbesen. Genuine Power Curves in Forgetting: A Quantitative Analysis of Individual Subject Forgetting Functions. *Memory & Cognition*, 25:731–739, 1997.

[131] Xifeng Yan, Philip S. Yu, and Jiawei Han. Graph Indexing: A Frequent Structure-Based Approach. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 335–346, New York, NY, USA, 2004. ACM Press.

[132] Xifeng Yan, Philip S. Yu, and Jiawei Han. Substructure Similarity Search in Graph Databases. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 766–777, New York, NY, USA, 2005. ACM Press.

[133] Qiang Yang. *Intelligent Planning: A Decomposition and Abstraction Based Approach*. Springer-Verlag, London, UK, 1997.

[134] Peter Z. Yeh. *Flexible Semantic Matching of Rich Knowledge Structures*. PhD thesis, University of Texas at Austin, Aug. 2006.

[135] Peter Z. Yeh, B. Porter, and K. Barker. A unified knowledge based approach for sense disambiguation and semantic role labeling. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI 2006)*, 2006.

[136] Peter Z. Yeh, Bruce Porter, and Ken Barker. Using Transformations to Improve Semantic Matching. In *K-CAP 2003: Proceedings of the Second International Conference on Knowledge Capture*, pages 180–189, 2003.

[137] Peter Z. Yeh, Bruce Porter, and Ken Barker. Matching utterances to rich knowledge

structures to acquire a model of the speaker's goal. In *K-CAP2005: Proceedings of Third International Conference on Knowledge Capture*, pages 129–136, 2005.

# Vita

Dan Gabriel Tecuci was born in Bucharest, Romania, the son of Victoria and Ion Tecuci. He obtained his General Baccalaureate in sciences from the 'Informatics' High-School in Bucharest in 1992 and went on to attend the 'Politehnica' University of Bucharest. He received his B.S. in Computer and Electrical Engineering in 1997, graduating first in his class, and a M.S. in Electrical Engineering in 1998. From 1997 until 1999 he worked as an Assistant Instructor in the Department of Computer Science at the same university. From 1998 until 1999 he also worked as a software developer for CornerSoft Tech in Bucharest, Romania. In the fall of 1999 he enrolled in the Ph.D. program in Computer Sciences at The University of Texas at Austin, where he received his M.S. in December 2001.

Permanent Address: 3463 Lake Austin Blvd. Apt. B
Austin, TX 78703

This dissertation was typeset with LaTeX $2_\varepsilon$[1] by the author.

[1] LaTeX $2_\varepsilon$ is an extension of LaTeX. LaTeX is a collection of macros for TeX. TeX is a trademark of the American Mathematical Society. The macros used in formatting this dissertation were written by Dinesh Das, Department of Computer Sciences, The University of Texas at Austin, and extended by Bert Kay, James A. Bednar, and Ayman El-Khashab.