STARS

Electronic Theses and Dissertations, 2004-2019

2019

Learning Internal State Memory Representations from Observation

Josiah Wong

University of Central Florida

LEARNING INTERNAL STATE MEMORY REPRESENTATIONS FROM OBSERVATION

by

JOSIAH MARK WONG
B.S. University of Central Florida, 2015
M.S. University of Central Florida, 2019

A dissertation submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Department of Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2019

Major Professor: Avelino J. Gonzalez

ABSTRACT

Learning from Observation (LfO) is a machine learning paradigm that mimics how people learn in daily life: learning how to do something simply by watching someone else do it. LfO has been used in various applications, from video game agent creation to driving a car, but it has always been limited by the inability of an observer to know what a performing entity chooses to remember as they act in an environment. Various methods have either ignored the effects of memory or otherwise made simplistic assumptions about its structure. In this dissertation, we propose a new method, Memory Composition Learning, that captures the influence of a performer's memory in an observed behavior through the creation of an auxiliary memory feature set that explicitly models the aspects of the environment with significance for future decisions, and which can be used with a machine learning technique to provide salient information from memory. It advances the state of the art by automatically learning the internal structure of memory instead of ignoring or predefining it. This research is difficult in that memory modeling is an unsupervised learning problem that we elect to solve solely from unobtrusive observation. This research is significant for LfO in that it will allow learning techniques that otherwise could not use information from memory to use a tailored set of learned memory features that capture salient influences from memory and enable decision-making based on these influences for more effective learning performance. To validate our hypothesis, we implemented a prototype for modeling observed memory influences with our approach and applied it to simulated vacuum cleaner and lawn mower domains. Our investigation revealed that MCL was able to automatically learn memory features that describe the influences on an observed actor's internal state, and which improved learning performance of observed behaviors.

To my beloved parents, Kathy and Daniel Wong, to my dear sister, Dee, and to my wonderful

poopers, Theo, Zacchaeus, Zeb, and Reph.

ACKNOWLEDGMENTS

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

Learning from Observation (LfO) of others is a natural form of learning. From a young age, we learn how to perform a variety of tasks, from playing sports to operating complex machinery, by watching others do it. The same kind of learning has been implemented in computer systems for various applications, such as driving a car [18] [66] [70], controlling video game agents [50] [55], flying an airplane [63] [84], learning dance movements in robots [51], learning how to construct parts with machinery [85], and playing simulated soccer [20]. In each of these examples, an actor entity (human or computer-controlled) performs a task or exhibits a behavior (based on potentially numerous and complex external and internal influences) which a learning entity observes. Then, the learning entity learns to perform the actions based on these observations and to subsequently act in the same manner as the actor, even if the actor entity's behavior was not optimal for achieving the objective of the behavior. Thus, the actor does not need to be an "expert" at performing the task.

LfO is an appealing learning method for simulated and robotic agents for several reasons. First, LfO allows non-technical subject matter experts (SMEs) to impart their knowledge to a learning agent without having to specify the constraints and nuances in the knowledge in a machine-readable representation; the formalization of such knowledge is transferred directly from the actor to the learning agent through the learning system [20]. Additionally, it may be difficult for SMEs to articulate certain aspects of their knowledge; such *implicit knowledge* is more readily shared through a demonstration of the task [66]. Furthermore, certain applications benefit from the creation of agents that perform in a human-like manner rather than optimally or "by-the-book", such as military simulations where soldiers can suffer from suboptimal conditions such as fatigue [18]. Finally, LfO can automate significant parts of the knowledge transfer process and lead to a comparable outcome to that of the traditional knowledge engineering approach [22], where a

knowledge engineer interviews a SME and then manually codifies the SME's knowledge into a machine-understandable representation. This was shown by Fernlund et. al. [18] through a direct comparison of LfO agents and hand-crafted agents. Thus, LfO has the potential to more efficiently and more accurately transfer knowledge from an actor to a learning agent than traditional knowledge engineering.

Various challenges in LfO have been investigated in the literature, such as learning high-level behaviors [72], extending LfO to team-based behaviors [35], using LfO to predict actor behavior [17], and using LfO to bootstrap the experiential learning of high-performing and humanlike behaviors [70]. However, all these LfO investigations have been limited to learning *reactive behaviors*; that is, behaviors whose only input is the currently (observable) perception of the environment and not any (unobservable) internal influences on the actor being observed. However, it is possible for an actor's behavior to be influenced by previously observed events or environmental features that are recalled and processed internally by the actor. It is therefore essential for LfO algorithms to account for unobservable internal influences, specifically memory of past events, in order to accurately learn behaviors that rely upon this information.

To address this issue, a few studies have developed techniques to explicitly learn actor behaviors that reason from environmental perceptions as well as from unobservable internal states. An actor's internal state is defined as "everything that the [actor] remembers from previous states" [58] and it depends on "past events or actions that are not in the current perceptual state" [55]. However, the way an actor processes previous states or how the actor represents their internal state varies between studies. The first approach to learning behaviors that reason using an internal state (also called *state-based behaviors*) uses Case-Based Reasoning (CBR) [20] [27]. This approach consults a store of records of the actor's behavior and uses the record that is most similar to a given situation to determine how to behave next. The second approach to learning state-based behavior represents the internal state explicitly in a probabilistic model [58] [76]. It represents conditional probabilities

for each time step and uses them to learn stochastic state-based behaviors.

The LfO studies in the literature that learn behaviors that require memory of events and actions from an arbitrary number of time steps in the past are limited in two important ways. In the CBR approach, reasoning is limited solely to what has been explicitly observed and "does not try to learn what information influences the internal state. If this information was learnt, the agent could attempt to maintain the state rather than infer it on every retrieval" in CBR [20, p. 140]. If the features influencing internal state were learned, then it would no longer be necessary to reconstruct internal state through comparison of arbitrarily long history segments. Also, behaviors with complex internal state structures may require large case libraries to properly learn, which could be computationally expensive to search during CBR's search and retrieval process [20].

In the probabilistic approach, the number of possible values that can be assumed by the internal state of the model must be finite and discrete [58, p. 16, 33]. This is because the actor's internal state "is not observable, and thus, if no assumption is made on its structure, in the general case, there is no direct way to learn its probability distribution" [58, p. 13]. Therefore, the set of possible internal state values is defined a priori in the probabilistic approach. It would be better for a learning agent to automatically infer the best representation for its internal state for learning an observed state-based behavior. Then, the manner by which a learning agent processes events from its history can be tailored to the reasoning process of a specific observed actor.

One example of memory influence in human behavior is that of seeking a supply of gasoline while driving on an interstate highway. Suppose that a woman is driving on an interstate road and notices that the gasoline gauge shows that the fuel tank is nearly empty. She must stop for gas, then continue on her journey. Upon brief introspection, the woman remembers that two minutes ago, she saw a sign that said, "Gas station — 5 miles". She then proceeds to change lanes so that she can make the next exit for the gas station. Even though the woman did not anticipate the need for

gas when she saw the sign, she filed it away in memory so that it could later be retrieved and used in her decision to exit the interstate. To learn this behavior, a memory analysis tool would need to determine that the signs for upcoming gas stations should be part of *what to remember* because they had some utility for future decisions.

Another example of memory influence in human behavior involves submarine combat. Suppose that a naval commander is in charge of a submarine on a mission in enemy waters. When the submarine's sonar readings indicate the presence of another underwater entity, possibly an enemy craft, the commander orders the crew to go on high alert. A while later, the sonar readings no longer detect the entity. However, the commander by no means lessens the alert level of the submarine because the potentially hostile entity could still be there, temporarily undetected. Thus, the memory of the sonar readings still bears significance for the commander's subsequent actions, even when the sonar readings have gone away. To learn this behavior, a memory analysis tool would need to determine *how long* the sonar readings remained important to the naval commander.

To address the needs outlined above, we introduce our approach called *Memory Composition Learning* (MCL), a memory analysis process that allows a learning agent to automatically infer unobservable internal state information that approximates how an observed actor uses memory of past events for reasoning. Given a set learning traces of an observed actor's behavior, MCL learns which features from previous environmental states have the greatest potential for informing the actor's decisions, how they should be remembered (represented), and for how long. MCL's output is a set of memory features that codify relevant memory influences on an observed actor's behavior, specifically what should be remembered and how it should be remembered, and these are used to create a set of memory-enhanced learning traces with information from those memory features at each time step. The enhanced traces and set of memory features can be used with preexisting machine learning algorithms, designed for learning reactive behaviors, because the enhanced trace and memory features remove the need for an agent to retrieve past feature information from its

history. Our approach, therefore, makes learning state-based behaviors conform to the Markov assumption, that all information pertinent to decision-making is consolidated within the current state and not distributed over any past states of the behavior.

This dissertation is structured as follows. In Chapter 2, we discuss the literature for LfO and related research in greater detail. Then, our problem definition is outlined in Chapter 3. Next, we present preliminary discussions concerning representations of behavior in Chapter 4. This is followed by a description of our conceptual approach to our problem in Chapter 5, which sets the foundation for our discussion of our comprehensive conceptual approach in Chapter 6 and the prototype that implements our approach (Chapter 7). In Chapter 8, we evaluate our work to assess our hypothesis through robust human subject testing. Finally, we present a summary, conclusions, and future work in Chapter 9.

CHAPTER 2: LEARNING FROM OBSERVATION AND RELATED RESEARCH

Learning from Observation (LfO) has made many strides toward automating the knowledge acquisition process, traditionally carried out by a knowledge engineer [22]. One accepted limitation of the LfO learning paradigm is that no amount of observation can reveal the precise machinations of an observed entity's internal cognition beyond speculative inference. Therefore, in addition to the current trends in LfO, we discuss several fields of research that have provided insights into how to maximize the inferential power of LfO techniques (e.g. leveraging memory). Such fields of research include temporal-based learning, memory models, and contextual reasoning.

2.1 Learning from Observation

LfO is "a subfield of machine learning that studies how to learn behavior from observation or demonstration" [58, p. 3]. The actions of learning and behaving are done by agents. An *agent* is an entity that *perceives* the environment through sensors, and interacts with the environment through affectors to achieve some goal [20]. Thus, LfO is the process of transferring knowledge from an observed actor (possibly, but not necessarily a human) to a learning agent. LfO is also referred to in the literature as Learning from Demonstration [4], Apprenticeship Learning [1], and Programming by Demonstration [14]. However, in LfO, the actor is not assumed to be a knowing or willing participant in the learning process, thus distinguishing LfO from other research areas that are otherwise very similar. Ontañón, Montaña, and Gonzalez [57] differentiate LfO from other forms of machine learning by stating that "[inputs] to the learning process are temporally based. In supervised learning we have problem/solution pairs, in unsupervised learning, only problems, and in reinforcement learning we have a reward signal. In LfO we have traces that

contain the evolution of the environment over time while the actor is executing actions to achieve a task." [57, p. 2]. In other words, LfO and standard supervised learning are very similar except for how their inputs express cause-effect relationships between a problem and its solution. Standard supervised learning uses training samples (assumed to be independent of each other) that encode explicit cause-effect relationships in the form of problem/solution pairs. In LfO, any cause-effect relationships are hidden implicitly and loosely within a trace of someone's observed performance. In LfO, a learning algorithm is expected to somehow either identify those problem/solution pairs or to learn the procedural knowledge from the trace as a whole. It is in the context of the specific needs of the LfO paradigm that our research is situated.

In order to summarize the state of the art in LfO and emphasize the novelty, difficulty, and significance of our research, we first formalize the learning tasks undertaken by LfO, then we present a brief history of how LfO has evolved to its current standing, and finally we discuss the accomplishments and limitations of the most relevant LfO works with regard to the specific problem addressed by this research.

2.1.1   *Formalization of Learning Tasks and Learning from Observation*

The LfO task is formally defined in [58]. For a given domain, the interaction of an actor C performing task T within an environment E, as shown in Figure 2.1, produces sequences of time-stamped observations called *traces* that are accessible by a learning agent A. Each observation in a trace consists of the set of perception variables X describing the environment, and the set of actions Y executed in response by actor C. Given the traces $[(x_1, y_1) \ldots, (x_n, y_n)]$, learning agent A must learn a behavior B that approximates actor C's policy without having access to task T. Additionally, at time t, the perception X_t may be affected by hidden states in the environment E_t and the action Y_t may be affected by the actor's unobservable internal state, denoted C_t. The authors define

C_t as an abstract representation of the list of past states (internal or environmental) that A chose to retain, and it depends on C_{t-1}, Y_{t-1}, and X_t. In their work, Ontañón, Montana, and Gonzalez [58] use a discrete random variable to represent one of a few possible predefined unobservable internal states. In contrast, our research seeks to automatically learn the most appropriate structure for approximating the internal state, as it is informed by memory of past events.
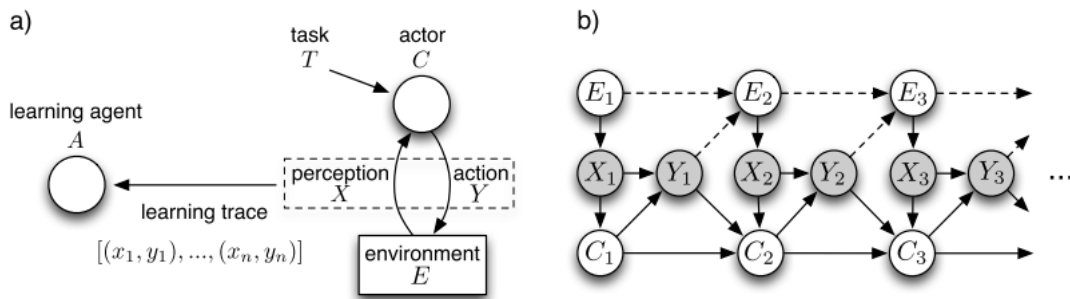
Figure 2.1: The LfO framework, reprinted with permission from [58].

Ontañón, Montaña, and Gonzalez [58] describe three levels of behaviors that can be learned by LfO: strict imitation (Level 1), reactive behavior (Level 2), and memory-influenced behavior (also called state-based behavior) (Level 3). In Level 1 of LfO (strict imitation), an agent learns to imitate a fixed sequence of actions exactly. Once an actor's behavior has been observed, the agent must copy the behavior exactly without any feedback from the environment influencing its control mechanism. An example of this is a robot learning to mimic perceived arm trajectories [34]. In Level 2 of LfO (reactive behavior), an agent's decisions at a given time step are influenced by the current perception of the environment. Most LfO applications fall into this category and traditional supervised learning techniques in isolation can only learn behaviors up to the second level of LfO

[58]. In Level 3 of LfO (memory-influenced behavior), an agent's actions are influenced not only by the current environmental perception but also by an unobservable internal state (C_t at time step t in Figure 2.1). Under the assumption that all task-relevant features are present in the perception of the environment or are predefined domain knowledge (such as explicit goals), the information comprising the unobservable internal state of the observed actor can be retrieved entirely from a subset of previous perceptions and action choices. Therefore, any LfO algorithm that can learn memory-influenced behavior must learn from past perceptions and actions [58]. This third level of LfO requires planning from memory of past states and even consideration for potential future states [57]. Our research investigates how to improve LfO performance for Level 3 behaviors or behaviors composed of a mixture of LfO levels.

A fourth LfO level (Level 4) encapsulates the second and third levels, and learns without prior knowledge of the task to be learned or the domain of deployment [57] [20]. In this level of LfO, learning is independent of rules governing the task or domain and is solely based on the actor's actions in reaction to the environment. However, this level of LfO is beyond the scope of our research because it focuses on minimizing the inputs to the learning task so that it is possible to "learn a variety of tasks from a variety of actors and in a variety of situations... [in a manner that is] general-purpose and task-independent" [20, p. 2-3]. Our research focuses on expanding what is extracted from an observed actor's trace, specifically the actor's internal state structure as it is influenced by memory.

Floyd [20] describes one general framework for LfO, which consists of a cycle of four subtasks: Modeling, Observation, Preprocessing, and Deployment. The first stage, Modeling, determines the representation of inputs and outputs used by a black box — the actor under observation. The inputs and outputs correspond to perception and action generation modules that can be separated from a reasoning module that corresponds to the actor's procedural knowledge. Observation, the second stage, is the process of collecting sequences of interactions between the actor and the environment,

where each interaction is a state-action pair and where the state comprises the observable features of the environment. Without explicit feedback from the actor during the learning that occurs during this stage, the quality of learning is limited by the quality of the observations, which is subject to actor error, the absence of certain situations of interest, and the absence of relevant environmental features that were not included in modeling. The third stage, Preprocessing, handles the extraction of inferential information about the actor's behavior, such as the presence of nondeterminism, stochastic behavior, or inconsistent actor behavior as a result of actor or observation error. In the fourth stage, Deployment, the learning agent executes its learned behavior in the environment. Floyd [20] does not include a separate "learning" or "training" stage in his framework because his framework was designed for CBR, which by its nature, lacks a training stage. For learning algorithms that do have a separate offline training phase, this training would occur during the Preprocessing stage.

The LfO cycle in the general framework for LfO [20] works as follows:

1. Modeling stage: Define the inputs and outputs of the learning task.

2. Repeat the following:

   (a) Observation stage: Collect observations of the actor behaving in the environment.

   (b) Preprocessing stage: Infer effects of nondeterminism, stochasticism, or observation error.

   (c) Deployment: Allow the learning agent to interact with environment.

   (d) If any of the prior stages (Observation, Preprocessing, Deployment) reveal errors in the agent model, return to the Modeling stage. Otherwise, go to the Observation stage if online observation gathering is allowed.

Floyd [20] allowed for interactions with the human observer to make corrections, an aspect of

learning which we elect not to explore in alignment with our more general definition of LfO and its removal from communication with the observed entity. Within this general LfO framework, our research best fits within the Preprocessing stage, the results of which improve performance during the Deployment stage.
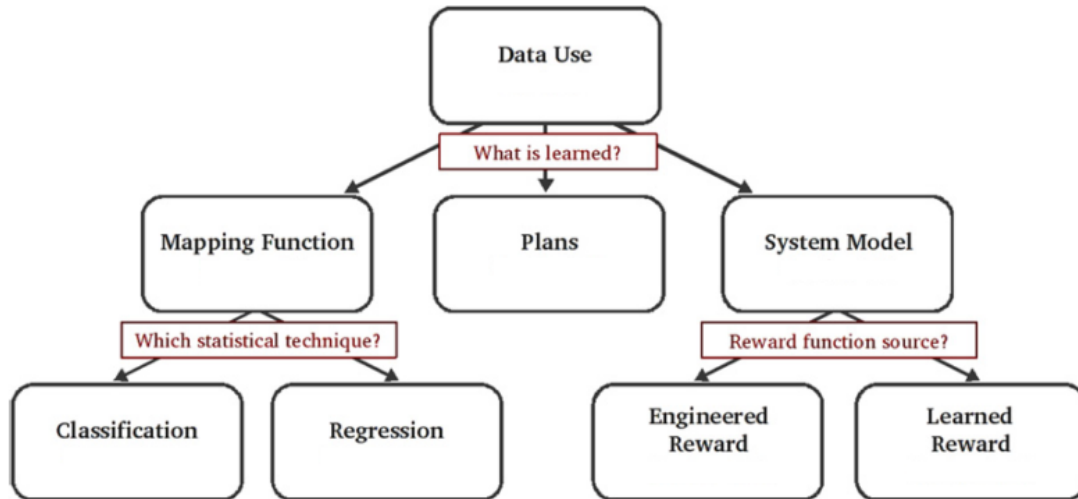
Figure 2.2: Classification of LfD policy derivation methods, adapted with permission from [4].

Argall et. al. [4] classify Learning from Demonstration (LfD) methods in robotics by how the learning agent obtains examples of a behavior to be learned from an actor, and how the learning agent learns this behavior from these demonstrations. Regarding the latter (shown in Figure 2.2), the authors describe three categories of policy derivation: learning a mapping function, using a system model, or developing a plan. The first approach, approximating a mapping function, maps a state to the appropriate action to take, given that state. This approach to learning allows a learning agent to generalize solutions to situations that were not observed. Fernlund et. al. [18]

used this approach by allowing genetic programming to specify agent output for every situation encountered because the mapping function evolved is a series of if-then-else statements in C code. Figure 2.2 shows a further subdivision of this approach into Classification (for a discrete class set) and Regression (for a continuous output spectrum).

The second approach, using a system model, contains a reward function through which action selection within certain states can be evaluated. The fitness function can either be engineered manually or learned automatically. The former choice of creating a fitness function manually was used by Stein and Gonzalez [70] during the experiential learning phase of their technique. After their agents had used observational learning to learn how to mimic the actors' behavior, the agents used experiential learning to further improve their performance at the task to be learned by using a manually-created fitness function to objectively evaluate task performance. This approach was an improvement over starting with a randomly behaving agent and using experiential learning to make it competent.

The latter choice of automatically learning the reward function behind the actor's behavior is used by Inverse Reinforcement Learning (IRL) [52]. IRL determines what the intent of a behavior is, given a description or observations of that behavior. IRL algorithms that use Markov Decision Processes (MDPs) are discussed in [52] in the context of finite and discrete state spaces, infinite and continuous spaces, and partially observable behaviors. The discovered reward function can then be used in Reinforcement Learning (RL) [75]. However, "[one] of the main problems here is that different reward functions may correspond to the observed behavior, and heuristics need to be devised to only consider families of reward functions that are interesting" [58, p. 2].

The systems model approach should not be confused with the work on system identification. System identification [8] observes or interacts with a system of interest in order to model it with an appropriate set of equations. Bongard and Lipson [8] proposed the use of various computational

operations to develop models of synthetic and physical systems. These models consisted of symbolic differential equations. System identification seeks to understand how various features interact with each other within a dynamical system of interest. In contrast, the system model approach to policy derivation [4] seeks to exploit a learned (or manually created) model of how desirable or "rewarding" various states are to an agent behaving within the system/environment to aid agent learning.

The third approach, planning, uses actions to get from an initial state to a goal state, and actions have pre-conditions and post-conditions. This approach sometimes requires guidance from the demonstrator [4]. However, planning approaches requiring human interaction are not applicable to domains where observation must be made non-intrusively, without any direct interaction between the learning agent and the observed actor. In our research, we elect to use unobtrusive observation without any interaction between the observer and the performing actor under observation.

Our research can enhance the performance of all three main branches of policy derivation methods from [4]. This is because our research is *not* aimed at improving a specific learning algorithm but rather, seeks to *model* the influence of *memory* on an observed actor's behavior. Then, these captured memory influences will be made available to an arbitrary machine learning algorithm that will use the model of memory as additional input for better learning the observed behavior.

We have thus framed our research within the formalized LfO task. Our focus is improving learning performance for memory-influenced Level 3 behaviors. Our research serves as a *preprocessing* technique for extracting specific memory influences on an observed behavior and then explicitly modeling them for latter use by a machine learning algorithm. We now describe the evolution of LfO from its roots, and discuss the accomplishments and needs of the field.

2.1.2   A Brief History

LfO is more general than LfD because LfO does not require the actor to be a willing or even a knowing participant in the learning process. In such cases, learning must be done via non-intrusive observation alone. However, much work in LfO has involved willing actors in its relatively short history. One of the earliest works in LfO was by Bauer [5], who created an algorithm that could synthesize computer programs from observation of example program executions. Given a set of sample instructions, Bauer's work could create a procedure or function that generalized these instructions. Later, Michalski and Stepp [48] described conceptual clustering as a more intuitive manner of classifying objects. They proposed using classifications based on conjunctive statements rather than numerical metrics. However, their work defines LfO as "learning without a teacher" [48, p. 2], and thus the authors inaccurately "define it merely as unsupervised learning" [57, p. 1]. Then, Sammut et. al. [63] presented a LfO application that learns to fly an aircraft through a specified flight plan after observing a human actor do it in simulation. The authors manually divided the flight task into different situations and used the C4.5 induction tree algorithm to capture the flight procedural knowledge and later fly the aircraft along the same flight plan. Wang [85] presented OBSERVER, a learning from observation architecture that automatically learned the preconditions and postconditions for operators based on expert traces and from practice problems for operator refinement during subsequent experiential learning. Van Lent and Laird [84] presented Observo-Soar, a LfO architecture that learned operator preconditions and postconditions in a complex flight domain, thus extending the work in [85] and [63] by using the more reactive operator representation by the SOAR architecture as an improvement over the C4.5 induction tree algorithm.

By this time, LfO research heavily emphasized *behavioral cloning*, the modeling of observed behavior [76]. Sidani and Gonzalez [66] described the Situational Awareness Module (SAM),

which used Neural Network Knowledge Units (NNKUs) to learn modular behavioral components in a car driving domain. Disagreements between the NNKUs about which action to perform were resolved by an Action Resolver component that selected which NNKU should determine the next action. SAM learned implicit knowledge that is easier to demonstrate than articulate. However, this approach required that the actor experience the same situation repeatedly to capture variations in the actor's behavior, and it required the developer to manually program the Action Resolver component. Fernlund et. al. [18] presented the Genetic Context Learner (GenCL), which used Genetic Programming (GP) and Context-based Reasoning (CxBR) to facilitate the creation of car driving agents that learned to drive by observing the actor drive a car simulator in specific situations or contexts. GenCL's CxBR component divided control of these agents into different contexts and the agents were trained on each context separately. Then, transition rules were learned in the same manner to recognize when a context should be activated to control the driving agent. The limitation of this work was that Fernlund had to manually identify and partition observational segments in the trace corresponding to each context for training.

Stein and Gonzalez [70] investigated the ability of LfO to bootstrap learning before it was fine-tuned by human-directed haptic feedback and then lastly by experiential learning. Their system was called FALCONET (Force-feedback Approach for Learning from Coaching and Observation Using Natural and Experiential Training). Stein and Gonzalez used neuroevolution to create their agents and found that combining all three learning methods (i.e., observational, human-directed, and experiential) resulted in comparable or superior agent performance in several domains. Luotsinen and Lovlid (2015), like Stein (2009), created a framework that allows LfO to be combined with experiential learning in their Data Driven Behavior Model (DDBM) system, though they only evaluated both learning modules separately. LfO has also been used to create Contextual Game Observation (CONGO) agents for the first-person shooter game, Quake II, by using Neural Networks to learn from a human actor's game play [50]. Ontañón [54] investigated CBR case

15

acquisition strategies in LfO for Real-Time Strategy (RTS) games. Stensrud and Gonzalez [72] created the Fuzzy ARTMAP Template-based Interpretation Learning Engine (FAMTILE), which sought to learn high-level behaviors given predefined low-level behaviors in maze-navigation and poker domains.

LfO has also been applied to domains with multiple agents. Team Nemesis [53] used LfO to learn team formations for RoboCup from the extraction of high-level behaviors from game logfiles. Kamrani, Luotsinen, and Lovlid [36] used LfO to develop military simulated agents that perform a bounding overwatch maneuver, where one agent moves forward while another provides cover. Johnson and Gonzalez [35] combined behavior maps and collaborative CxBR to learn team-based behaviors in bucket brigade and ship pursuit domains. The behavior maps, similar to CBR, memorize perception-to-action pairings for a given context. This method relies heavily on proper context definitions to remain scalable.

Observational learning techniques have also been applied to behavioral analysis of other entities. Fernlund et. al. [17] used GenCL agents to evaluate human choices in controlling simulated tanks for After Action Review (AAR). These agents were modeled after human "experts" controlling these tanks and then used to determine when a human trainee deviated from what the "expert" would have done. Ontañón et. al. [56] used CBR to evaluate teenage car driving behavior to determine the driver's skill level and predict their actions to advance research in crash prediction systems. Similarly, Stanley et. al. [68] used Neuroevolution of Augmenting Topologies (NEAT) to observe crash-prone simulated car drivers and learn models that predicted whether and when a crash would occur.

Other earlier works in the LfO literature include training an air hockey robot [6], training an autonomous land vehicle to follow a road [60], creating rule-based agents in a complex environment [39], and teaching robots to execute dance moves [51].

The literature shows a diversity of investigations and applications in the LfO field that have broadened the impact of this learning paradigm, whether or not the researchers at the time would have classified their work as LfO. The objectives of LfO research have varied over the years, from knowledge transfer to behavior cloning to behavior analysis, and the capabilities of LfO techniques have expanded, from operator planning to behavior modeling in robotic and simulated environments to multi-agent learning and integration with reinforcement learning. These endeavors emphasize *what* LfO can learn (its output) and *how* the resultant output (e.g. behavioral models, procedural knowledge) can be used in various applications. However, the field lacks an emphasis on what LfO learns *from* (its input). Purely observational learning, the focus of our research, is limited by the quality of collected observations and the diversity of situations in which the observed actor behaved. However, LfO need not be limited to learning solely from traces. It is possible to model specific *aspects* of an observed behavior as a primer for learning the behavior as a whole. Trinh and Gonzalez [82] preprocessed traces in order to discover the contexts experienced by an observed entity; these contexts were then used by subsequent LfO (this work is discussed more in depth in Section 2.2.3). Our research seeks to learn *memory influences* that can be used in subsequent LfO. This approach is different from prior approaches to learning memory-aware behaviors, which are discussed next.

2.1.3   Memory-Aware Learning from Observation

In the LfO literature, there have been three approaches to learning memory-influenced behaviors:

1. Ignore the effects of memory or assume that no memory influence exists.

2. Predefine the range of memory influence or its internal structure.

3. Compute memory influence as needed for each decision without explicitly maintaining a

memory structure.

The first approach, ignoring memory, is the approach used in many research endeavors reported in the literature. The advantage to ignoring memory is that learning is much simpler; individual observations can be treated independently of each other and used as separate inputs for standard supervised learning algorithms [58]. The creation of LfO-specific learning algorithms still remains an open problem, but for Level 1 and Level 2 behaviors, such algorithms are unnecessary (though perhaps still helpful). However, supervised learning algorithms are provably insufficient for evaluating or learning Level 3 behaviors [58]. Heninger et. al. [29] also discuss the shortcomings of static evaluation metrics in dynamic environments, but only for reactive behavioral learning. The inability of standard supervised learning to account for temporally-dependent behavior influenced by memory is the main disadvantage of the first approach to learning memory-influenced behaviors and the motivation behind the next two approaches.

The second approach to learning memory-based behaviors makes certain assumptions about the structure of an observed agent's internal state. There are two works that exemplify this approach. The first work, by Tirnauca et. al. [76], used a probabilistic finite automata (PFA) to create three types of behavior models:

1. A learning agent whose only input is the current perception.

2. A learning agent whose inputs are the current and previous perception.

3. A learning agent whose inputs are the current and previous perception and the last action taken.

For a finite and discrete simulated vacuum cleaner domain, the PFA models were tested for proficiency in behavioral recognition and behavior cloning tasks; PFAs were found to outperform other

machine learning techniques for behavior cloning. The third model was advantageous in that it could recognize Level 3 behaviors, but it had some struggles differentiating between deterministic and stochastic variants of the same agent in the behavioral recognition task. Overall, the second and third models in [76] assumed that information from time steps beyond the previous time step was unnecessary for the learning task. The disadvantages of the approach taken in this work are:

1. The assumption that memory influences prior to time step $t - 1$ are negligible may be false. If this is the case, then the PFA models will be insufficient for reacting to these influences and capturing the behavior of entities conforming to these influences.

2. There are *three* different PFA models. The application of the wrong model to learn the wrong behavior can harm recognition and cloning performance, as observed in [76].

The second work to use the second approach to learning memory-based behaviors, by Ontañón, Montaña, and Gonzalez [58], used Dynamic Bayesian Networks (DBNs) to model the conditional probabilities at a given time step t between environmental perception X_t, agent action output Y_t, and agent internal state C_t; specifically, the following probabilities are learned [58, p. 13].

- $Pr(C_1)$

- $Pr(Y_t|C_t, X_t)$

- $Pr(C_t|C_{t-1}, X_t, Y_{t-1})$

Unlike [76], the *internal state* C_t is explicitly specified and it abstractly represents *all* relevant influences from memory of past events. However, in order to use the Expectation Maximization (EM) algorithm to train the DBN to learn memory-based behaviors, the internal state C_t must be a finite, discrete random variable, which had a maximum of four possible values in [58] when

19

learning to replicate the behavior of various simulated vacuum cleaner agents (the rationale for this value was not provided by the authors). This aspect of the research limits the usefulness of this DBN approach in the following ways.

1. The maximum number of possible values N for C_t must be set and/or tuned manually. If N is too low, then the full range of internal state expression cannot be fully realized. If N is too high, then learning in the DBN may be unnecessarily burdened.

2. For each additional C_t value, the computational cost of training increases.

3. The specific human-understandable meaning behind each C_t value is not specified or elucidated by the DBN, which does not allow for human verification or deeper human understanding into the behavior being learned.

The third approach to learning memory-based behaviors uses a lazy machine learning approach, Case Based Reasoning (CBR), to infer the relevant memory influences for every decision during runtime. (It is called a lazy approach because "learning" is done on the fly during run-time; there is no training phase.) Floyd [20] created the Temporal Backtracking (TB) algorithm to execute memory-based behaviors that had been captured in a case base. At every time step, the sequence of interactions that a CBR learning agent has experienced (its *run*) is compared to runs of experts stored in a case base; the action taken by the most relevant run in the case base is applied to the CBR learning agent's current situation. TB compares runs by iteratively going back in time, comparing components of case base runs to the corresponding component in the CBR agent's run, further and further back in time until all similar remaining cases agree on the action to take. In this way, TB only looks as far back in memory as necessary. TB is a significant inspiration for our research and is discussed in more detail in Section 4.4.

The advantages of the TB are that no training is required (CBR is a lazy machine learning algo-

20

rithm) and that no assumption is made as to where memory influences exist in a run in progress. Furthermore, the memory influences can differ from decision to decision and TB can adapt to these differences. The disadvantages of the TB are:

1. The lack of a learned memory structure does not allow for human understanding of recurring memory influence patterns. It also fails to capitalize on computations of memory influences on previous decisions.

2. The greater the size of the case base, the harder it is for TB to deliver real-time decisions. This is the effect of the first limitation.

TB has been evaluated and analyzed by other researchers. Ontañón and Floyd [55] compared various case acquisition strategies for real-time strategy (RTS) games, including TB and Similarity-Based Chunking; the former mimicked player behavior better, but the latter learned to play the RTS game better. Gunaratne, Esfandiari, and Fawaz [27] analyzed TB as a specific instance of a more general case-based memory-aware learning algorithm; TB assumes that more recent memory influences are more relevant, but other learning assumptions were investigated in [27] with varied results for each.

Our research addresses the limitations of each approach to learning memory-influenced behaviors in that it seeks to create a model of an observed entity's use of memory, a model that makes minimal assumptions on the structure of memory and that is human-understandable. Furthermore, this model of memory usage would be compatible with machine learning algorithms limited to learning only up to Level 2 LfO behaviors, enabling them to capitalize on information about memory with minimal (if any) alterations to the algorithm itself. The nature of our research is discussed further in Chapters 5 and 6.

2.1.4 Summary

In summary, LfO is a versatile learning method that has been extensively explored in a short period of time in the areas of agent control, actor emulation, task optimization, and behavior analysis. The levels of behaviors that LfO can learn range from exact memorization of an action sequence to behaviors that are dependent on memory of past events. Only a handful of LfO research endeavors have explicitly investigated how to learn memory-influenced behaviors, but various limitations with human understandability and simplistic assumptions on memory structure still remain. Our research seeks to provide a way to automatically learn memory influences and model them in a transparent manner to improve subsequent LfO. We now continue our literature review of various fields that provide insight into how this research can be conducted.

2.2 Related Research

This section describes three fields of research with significance for our research. Those fields are: temporal-based learning, memory models, and contextual reasoning.

2.2.1 Temporal-Based Learning

LfO is most similar to sequential learning, according to [58]. Sequential learning [13] is the process of assigning a class label to every component of a time series. Given a sequence of inputs $\{x_0, x_1, \ldots, x_{n-1}, x_n\}$, a classifier must generate an output sequence $\{y_0, y_1, \ldots, y_{n-1}, y_n\}$. Dietterich uses the problem of part-of-speech-tagging as an example of sequential learning: given the input sequence "would you like fries with that", a sequential learner should predict the parts of speech output sequence (verb pronoun verb noun prep pronoun) [13].

The sequential learning problem differs from two similar problems:

- **Time-Series Prediction**: Given a subsequence of true output labels, $\{y_0, y_1, \ldots, y_{t-1}\}$, predict the next output label y_t in the sequence. There may also be associated variables $\{x_0, x_1, \ldots, x_{t-1}\}$. One example is the auto-complete feature of a search engine when a user is typing.

- **Sequence Classification**: Given an entire input sequence $\{x_0, x_1, \ldots, x_{n-1}, x_n\}$, predict a single output label y that applies to the entire sequence as a whole. Dietterich uses the problem of determining the author of a sample of written text as an example of sequence classification [13]. (One possible sub-problem is determining when a sequence is complete and ready for classification.)

LfO can be framed as a sequential learning problem in that the sequence of inputs are perceptions of the environment over time and the output sequence is composed of the actions taken at each of those time steps. Thus, a fully trained LfO agent will not have access to what its human trainer actually did in a test scenario the LfO agent did not observe the human perform in when he/she experienced those same perceptions, thus differentiating LfO from time-series prediction. Furthermore, a LfO agent must make a decision at each time step, rather than classifying an entire sequence as in sequence classification.

Dietterich [13] goes on to discuss several issues concerning sequential learning, such as loss functions, feature selection, and computational cost; he also discusses five approaches to the sequential learning problem: sliding windows, recurrent sliding windows, hidden Markov models (HMMs), conditional random fields, and graph transformer networks. The issues for future research in sequential learning that Dietterich identified are:

1. Utilize correlations in sequential data (e.g. knowing that the letter 'u' almost always follows

letter 'q').

2. Use complex loss functions.

3. Identify long-distance interactions in sequential data.

4. Do sequential learning faster.

Our research in learning memory influences in observed behavior addresses the third issue specifically. By learning long-distance interactions (e.g. knowing which events are worth remembering and which remembered events should be recalled), LfO can learn to react to memory influences.

In contrast to Dietterich's definition of sequential learning [13], Graves [26] relaxes the constraint of assigning a label to *every* input element by describing the *sequence labeling* problem: given an n-length input sequence $\{x_0, x_1, \ldots, x_{n-1}, x_n\}$, generate an appropriate m-length output sequence $\{y_0, y_1, \ldots, y_{m-1}, y_m\}$, where n and m are not necessarily the same. The length of m and the alignment of output labels to input labels produce three types of problems:

1. *Sequence Classification*: This is the same problem as that defined by [13]. Given an input sequence, predict a single label Y for the entire sequence. In this case, $m = 1$.

2. *Segment Classification*: In this problem, the alignment of output labels to input labels is known in advance; the subset of inputs to which each output label corresponds is given. Graves [26] uses framewise phoneme classification, the attribution of text to individual acoustic frames, as an example of segment classification.

3. *Temporal Classification*: In this problem, the alignment of output labels to input labels is not known. The only constraint is that $m \leq n$, because we assume that each output label corresponds to at least one input, and each input corresponds to at most one output label.

LfO is an example of segment classification because $m = n$; every perception in the input sequence of the learning problem requires exactly one action (even if that action is to do nothing). However, the problem of learning memory influences is a temporal classification problem. Here's why:

1. The major difference between the case-based approach to learning memory-based behaviors and the predefined memory structure approach (both of which are described in Section 2.1.3) is that the former does not maintain an internal state while the latter does. Our research seeks to represent internal state to overcome the limitations of the former approach. Therefore, the *value* of that internal state may stay constant for several time steps.

2. Given that one internal state value may encompass several time steps, it is necessary for our research to determine over which time steps an internal state value may preside. This alignment of memory influence to input perceptions is not known in advance.

3. Because the output of our research is a model of how a performing entity uses memory over time, the mapping of memory influences to specific input perceptions corresponds to the temporal classification task.
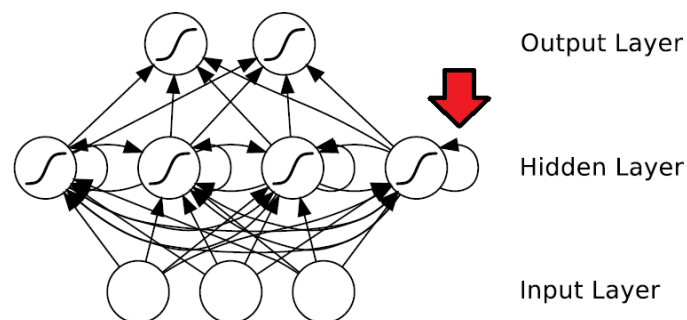
Figure 2.3: A recurrent neural network (RNN), adapted with permission from [26].

Graves continues on by describing *recurrent neural networks* (RNNs) as a type of neural network capable of mapping sequences to sequences. RNNs are basic neural networks that allow recurrent connections or connections from a neuron to itself in a sort of loop; in this way, inputs from prior time steps can exert influence on a neuron in future time steps via the recurrent connection instead of just inputs from the current time step. As Graves [26] puts it, "an RNN can in principle map from the entire *history* of previous inputs to each output." Figure 2.3 shows a simple RNN — all the hidden nodes have recurrent connections (such as the one indicated by a red arrow in the figure), but otherwise the RNN resembles a regular feed-forward neural network.
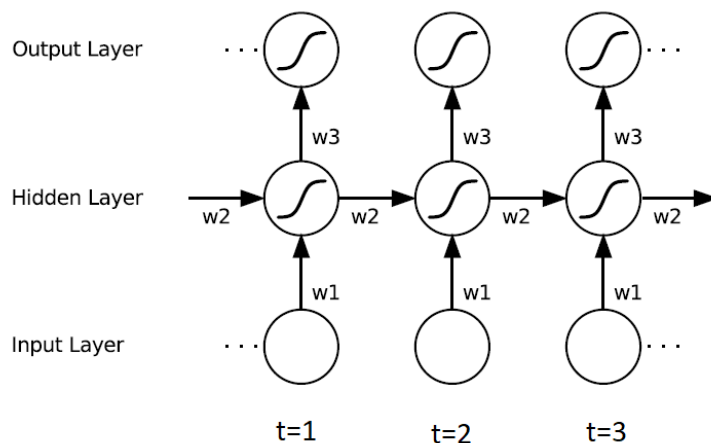
Figure 2.4: An unfolded recurrent neural network, adapted with permission from [26].

Figure 2.4 is further illustrative of the power of RNNs. Here, a RNN is "unfolded" — the information processed by the same set of neurons is seen over time (the time stamps in the figure are ours). The same neural weights are used at each time step, but the input from time step $t = 1$ is used as input to the hidden layer neuron at time step $t = 2$ via the recurrent connection (labeled

"w2") from the hidden layer neuron to itself. Technically, the same input could exert influence on the neuron all the way to time step $t = N$, but in practice, RNNs suffer from the *vanishing gradient problem*, wherein the influence of an input from an early time step diminishes over time such that long-distance relationships between elements of a sequence over long periods of time are harder to capture; this was the third issue for future research in sequential learning that was raised by Dietterich [13]. Figure 2.5 illustrates the vanishing gradient problem in an unfolded RNN. The sensitivity of the neurons to the input from time step $t = 1$ decreases, as shown by the decreasing intensity of black color of the neuron over time as new inputs from latter time steps interfere.
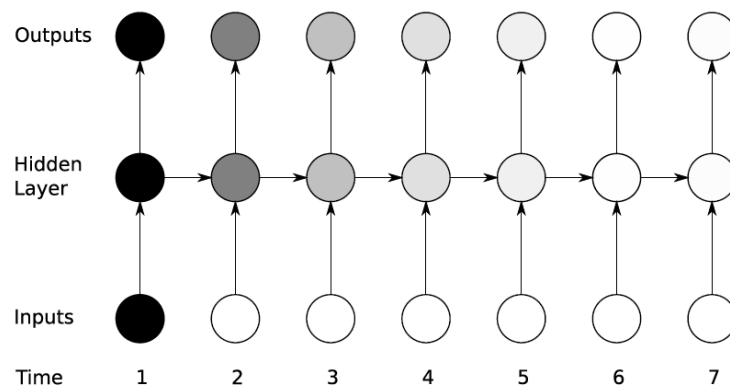
Figure 2.5: The vanishing gradient problem in RNNs, reprinted with permission from [26].

Fortunately, Graves [26] addresses that problem with a discussion of Long Short-Term Memory (LSTM) deep neural networks. LSTMs overcome the vanishing gradient problem via an internal memory cell within the neuron and multiplicative "gates" that can block incoming inputs and preserve the contents of the memory cell for long periods of time.
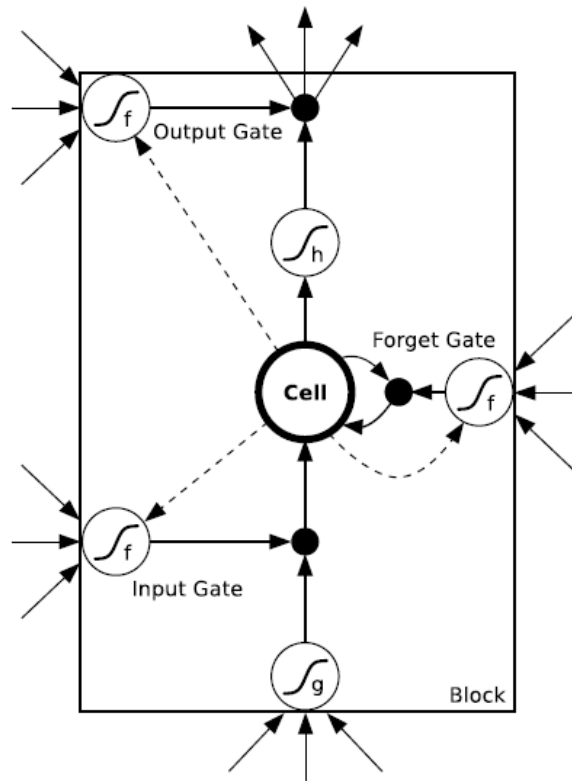
Figure 2.6: LSTM neuron, reprinted with permission from [26].

Reprinted by permission from Springer Nature: Supervised Sequence Labelling with Recurrent Neural Networks by Alex Graves ©2012.

Figure 2.6 shows a single LSTM neuron represented by a rectangular block. Incoming arrows are inputs from neurons in the previous network layer. The *input gate* controls how much these inputs affect the memory cell, represented in the center of the block by a bold circle. The *forget gate* controls how much of the cell's contents are retained for the next time step. The *output gate* controls how much of the memory cell's contents are transmitted to the rest of the network in the current time step. The end result is the preservation of relevant inputs for long periods of time, thus establishing long-distance relationships between temporally distant sequential elements. This

28

is seen in Figure 2.7, where open gates are small circles and closed gates are lines. The input, forget, and output gates are below, left, and above the neuron. The sensitivity of the hidden layer neuron is preserved from time step $t = 1$ to $t = 6$ because the input gate is closed for that duration of time, blocking new inputs that would overwrite the memory cell contents.
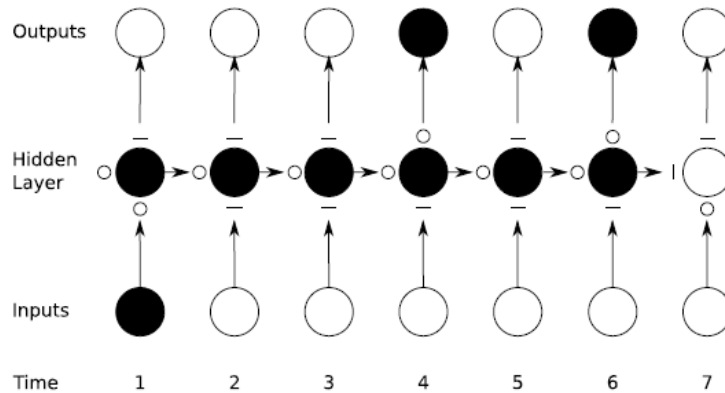
Figure 2.7: LSTM with preserved gradient, reprinted with permission from [26].

The memory cells within LSTMs are an example of "internal memory", but it is possible for deep neural networks to utilize "external memory" (a memory structure external to the learning network). Motivated by the need for "one-shot learning" (the ability to learn on the fly with one attempt at a task), the work in [64] investigated the ability of memory-augmented neural networks (MANNs) to engage in meta-learning. Deep neural networks typically require many data samples to gradually learn how to perform a task and cannot learn quickly from new data without adjusting all network weights (which takes a long time) or experiencing catastrophic interference [38] (the collapse of all learned knowledge). Meta-learning can solve this — rapid task-specific learning is done for new tasks, but gradual inter-task learning is done across tasks. Neural Turing Machines

(NTMs) are MANNs that combine a neural network (e.g. a LSTM) with an external memory model. The neural network facilitates gradual learning across tasks while the external memory model facilitates rapid learning for new tasks. This approach was evaluated in [64] on an image classification task with excellent performance. However, we must note that the memory contents of the external memory module are wiped after each task and any persistent memory that is incorporated into the neural network becomes inaccessible outside of the network. This differs from our work in that we seek to learn a memory model that is persistent across tasks

The reason why sequential learning, sequence labeling, and LSTMs are important for our research is because LfO tasks are temporally based, and the memory influences on behaviors that LfO seeks to learn are also temporally based. We have learned that LfO is a segment classification problem, when framed in the context of sequential learning, but learning memory influences is a temporal classification problem when framed in the context of sequence labeling problems. Connectionist solutions, such as LSTM networks, have the disadvantage of encoding procedural knowledge implicitly in a manner that is difficult for humans to understand, something that we want to avoid in our research. However, the mechanisms of the LSTM neuron are powerful for preserving temporal distant memory influences and can be used in our research.

2.2.2 Memory Models

Memory is essential in everyday living; without it, we would be doomed to unnecessarily repeat the same moments with no ability to learn from the past or plan for future events [43]. We would only be capable of reactive behaviors. Furthermore, without memory, we could only reason with the present moment's perception of the environment. The role of memory in learning from the past during observation is to represent past observations, actions, and decisions in a way that facilitates reasoning from these observations. The role of memory in planning for the future during

deployment is to recall current objectives, gauge progress towards said objectives, and reason about this progress to make the next decision.

Briefly, memory is defined by the Merriam-Webster dictionary [47] as "the power or process of reproducing or recalling what has been learned and retained especially through associative mechanisms" and as "the store of things learned and retained from an organism's activity or experience as evidenced by modification of structure or behavior or by recall and recognition" (Merriam-Webster, 2017). We disambiguate the different aspects of memory as follows:

- Memory contents (or just "memory") – the record of past experiences that have been preserved for future reasoning.

- Memory recall mechanism – the process by which relevant contents from memory are retrieved for reasoning.

- Memory storage mechanism – the process by which memory is updated, either through the incorporation of new contents or the forgetting of old contents.

A cognitive architecture is a framework that mimics the reasoning procedures of the human mind in a computational system [37]. The ability of cognitive architectures to reason about current knowledge and previous experiences to achieve goals necessitates the use of memory structures. Three such cognitive architectures are Soar [40], ACT-R [3], and CLARION [73].

Soar [40] is a cognitive architecture geared to achieving general intelligence. It represents a task in terms of a *problem space* that is searched for an appropriate solution. Problem-solving requires the exploration of several *states* in the problem space. New states are generated from previous ones via *operators*. Whenever progress towards a goal state is halted by an *impasse*, a new *subgoal* is created.

There are two memory systems in Soar: a long-term production memory and a short-term working memory. Working memory stores all information pertinent to the task at hand, such as active goals and current states. Production memory stores information that Soar has learned over time from previous computations. Production memory is able to add to or modify the contents of working memory.

ACT-R (Adaptive Control of Thought - Rational) [3] is a cognitive architecture that is predicated on the notion that human cognition consists of the application of simple procedures on a large and complex body of knowledge. Similar to Soar, it separates procedural knowledge (represented as production rules) from declarative knowledge (represented as data structures called "chunks"). However, while Soar chooses to solve problems by exploring states in a problem space in hopes of achieving a goal state, ACT-R chooses to solve problems by using production rules to derive increasingly complex chunks from preexisting chunks, eventually creating a knowledge chunk that represents the solution to a problem.

In ACT-R, memory is represented as a Bayesian process. The significance of a given knowledge chunk (stored in memory) for a specific situation is affected by the presence of other chunks that are highly associated with it. However, ACT-R explicitly does not factor in the influence of how long ago in the past such associate chunks occurred, having ruled out such influences as trivial. Thus, memory retrieval results in the activation of sufficiently relevant chunks, which can then be modified and combined by production rules.

In the CLARION cognitive architecture [73], there are several types of memory, as shown in Figure 2.8. In Figure 2.8, there are two divisions of memory: procedural versus declarative and explicit versus implicit. Procedural memory stores information on how to perform certain tasks or procedures. Declarative memory stores information not pertaining to tasks/procedures and can be further subdivided into semantic memory and episodic memory. Episodic memory stores personal

experiences while semantic memory stores general information. Sun [73] also makes the distinction between explicit and implicit memory within each of the previously mentioned subsystems. Explicit memory is easily understood, manipulated, and accessible; it can be represented symbolically. However, implicit memory is less accessible and may be represented subsymbolically, such as with a neural network. For clarity, we provide the example of riding a bike; explicit procedural knowledge is knowing to stay upright on the bike while pedaling forward while implicit procedural knowledge consists of the specific muscle contractions and neural transmissions needed to keep one's balance. In Figure 2.8, the flow of information first proceeds through procedural memory, then through declarative memory. Information flow is bidirectional and each memory module facilitates communication between its explicit and implicit components.
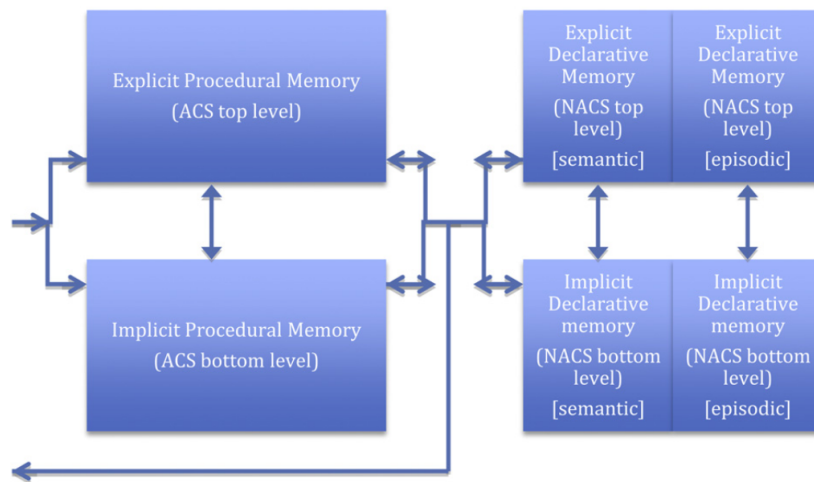
Figure 2.8: CLARION architecture, reprinted with permission from [73].

Complementary Learning Systems (CLS) theory [38] also creates a link between parametric implicit memories and non-parametric instance-based memories, from a neuroscience perspective.

Specifically, CLS theory juxtaposes the roles of the hippocampus and the neocortex in learning. The hippocampus engages in rapid learning of episodic memories, which store details of specific experiences. Then, these memories are slowly integrated into the neocortex, which stores statistical information about the environment, akin to a neural network. This process is called "systems-level consolidation", a process by which the neocortex learns to retrieve memories without help from the hippocampus. Figure 2.9 shows the interactions between these complementary memory systems. The neocortex regions are connected via the green arrows, the hippocampus and surrounding region are shown in gray and facilitate intracommunication paths shown with red arrows. Finally, the neocortex and hippocampus communicate via blue lines.
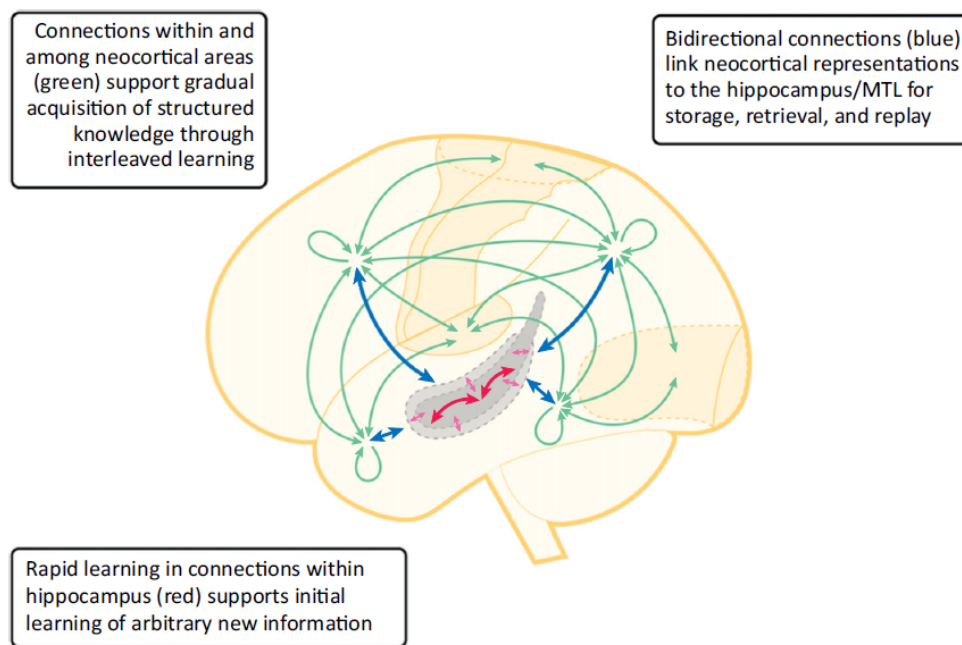
Figure 2.9: Memory systems in the brain, reprinted with permission from [38].

The CLS theory is interesting because it relies both on parametric and non-parametric learning. It is analogous to the combination of CBR (instance-based learning) and LSTM networks (implicit procedural knowledge encoding). It combines the best of both worlds in forming memories. It is also interesting how the hippocampus is useful for remembering particularly traumatic events (e.g. encountering the lion at a watering hole [38]) without having to experience the event repeatedly, as would be necessary for retention in the neocortex.

Given the classifications of memory by [73], "memory" and "learned knowledge" seem to be very similar if not the same. In such a case, memory is just a repository of knowledge. However, we argue that while knowledge is the sum of what one knows (memory contents), the recall and storage mechanisms of memory control how this knowledge is represented, preserved, and summoned during the reasoning process. The question thus becomes: how can the recall and storage mechanisms of memory be trained and how should knowledge be represented in memory to facilitate the construction of internal state? Fortunately, the literature contains several examples of memory in facilitating decision making in social and learning settings.

Conversational agents and Intelligent Social Companions (ISCs) represent a field with extensive use of memory and internal state. Elvir et. al. [15] explore the use of context in a conversational agent. Their system uses what the authors called Memory Interfaces to process requests to memory, a Back-end Database for storing memories, a Contextualization process for managing memories, and Analysis Services for analyzing memories and user input. Within their proposed memory framework, the authors contribute a "gisting" algorithm for extracting main ideas from a conversation. They found memory to be essential for recognizing elements of a question that are unstated but implied. Ho et. al. [31] combine short-term memory (STM) and long-term memory (LTM) for managing active goals and learning from goal outcomes. Their aim was to create a memory system that allowed agent migration between different bodies. Lim et. al. [44] investigate socially-aware memory in ISCs that can learn user preferences and keep information about

certain users secret from other users. Campos and Paiva [12] describe MAY (My Memories Are Yours), which develops shared memory of the user's conversations to bond with the user. It uses Autobiographical Memory (AM), which divides memories into Lifetime Periods, General Events, and Event-Specific Knowledge, where memories in one level may map to multiple memories in higher, more abstract levels. Mohammed Ali [49] developed a conversational system that learned knowledge from casual conversation. This system used a memory module to remember previous interactions with a user, how many times aspects of the knowledge base have been modified, and how confident the system is that aspects of its knowledge base are accurate. It learned to chat normally with a user and then switch to learning mode when the user decides to say something relevant to its knowledge base. Lim [43] presents a survey of memory models in Intelligent Social Companions (ISCs). She discusses human memory and the application of memory in computer systems, specifically ISCs. She identifies various needs for ISC memory research, such as investigating forgetting mechanisms, emotions, and fabricating a life story.

Episodic memory is particularly important in several applications. Brom and Lukavsky [11] initiate a formal description of full episodic memory (FEM) requirements in the context of virtual agents and claim that FEM can be applied to a variety of tasks from remembering past interactions to learning to solve problems from memory of past attempts. Wenwen et. al. (2009) describe electromagnetic adaptive resonance theory (EM-ART) for modeling episodic memory. EM-ART uses three layers of fusion ART: the first layer contains activations for environmental variables; the second layer recognizes single events; the third layer recognizes episodes or sequences of events. Faltersack et. al. [16] present Ziggurat, a general episodic memory system that uses SOAR to control agents that move about grid-like worlds in food-eating and navigation problems. Ziggurat continuously records perception-action pairs that the authors call *episodes*. Sequences of episodes comprise complex episodes and Ziggurat simplifies episodes to achieve the most efficient plan for accomplishing a task. Ho, Dautenhahn, and Nehaniv [32] investigate different memory models

for Artificial Life (AL) agents: purely reactive, short-term, long-term, and short-term/long-term memory. The STM+LTM agents lived the longest, and allowing communication between agents about resources also increased agent longevity.

The goal of our research is to learn internal state representations that are used by an actor to execute behaviors requiring memory. Even though this internal state is unobservable, we have gained insight into the structure of memory, how memory is used in reasoning frameworks in the literature, and how memory is used in conversational agents and ISCs. A common trend in the literature is how memory is compartmentalized by type, purpose, and level of abstraction. We see that it is not necessary to save all of memory, but only the most pertinent components [15]. Episodic memory has been particularly helpful in performing tasks where the current environmental state was not enough for decision making [16] and where knowledge from the past became useful in future computations [32]. However, behaviors requiring memory can be very complex and much uncertainty exists around which aspects of history have the greatest potential utility in future decisions. These needs can be addressed by context, which we discuss next.

2.2.3 Context

Context is the encapsulation of all information relevant to a given situation [24]. More specifically, Trinh [81, p. 157] defines context as "any recognized situation defined by a set of similar practices that can approximate the set of actions, procedures, and expectations employed by an actor within the situation". Reasoning with contexts narrows down the search for solutions in problem solving [81] and it allows situation-specific assumptions to remain unstated, and yet be known [24]. An intuitive example of context involves a pilot who can also cook well; while she's flying an airplane, she does not consult her cooking knowledge and while she's cooking, she does not consult her piloting skills [24]. A more comprehensive overview of context can be found in [9].

Context is the driving force behind the Context-based Reasoning paradigm (CxBR) [24], a framework within which a *tactical* agent (one that acts in real-time) can interact within an environment according to its contextual knowledge. In CxBR, a tactical agent is controlled by one of several *contexts*. Each context encapsulates what the agent can do in a particular situation and it has three parts:

1. Action Knowledge: This part of the context determines what the agent is supposed to do when this context is selected to control the agent, as well as how to do it.

2. Transition Knowledge: This part of the context determines whether this context is appropriate for addressing the agent's current situation.

3. Declarative Knowledge: This part of the context contains information relevant to the agent in this particular context.
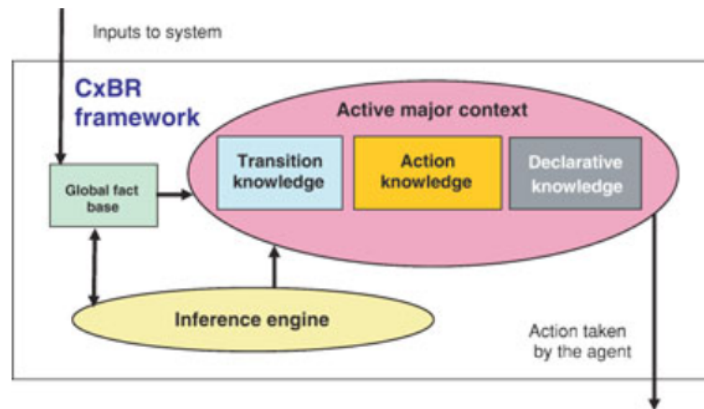
Figure 2.10: An active context in CxBR, reprinted with permission from [24].

Figure 2.10 shows a snapshot of the *active context* within the CxBR framework. There can only be one active context at a time and it is the context that controls the agent. In Figure 2.10, the perception of the environment is used to update a global fact base, which stores all information about the world at a given point in time. This information is made available to the context, which first determines via its transition knowledge whether it is appropriate for the given situation. If it is not, other contexts will be queried for possible activation. Whichever context is selected to be active will use its action knowledge to dictate the agent's next action, which is shown as the output of the CxBR framework in Figure 2.10.
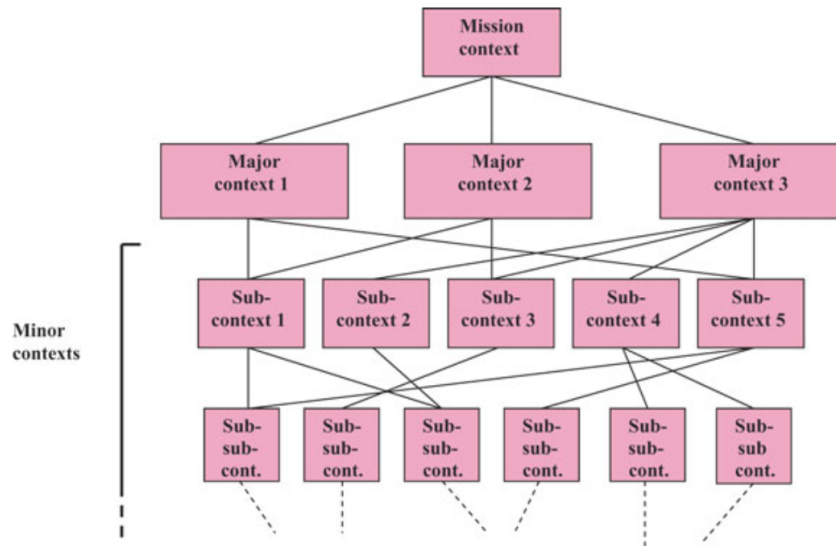
Figure 2.11: Generic context hierarchy, reprinted with permission from [24].

There are different types of contexts, as shown in Figure 2.11. Here, we see that the top context

is the *mission context*, which does not control the agent, but stores the agent's objectives, planned sequence of context activations, and other relevant information pertaining to the mission. The next layer of contexts are *major contexts*, which have been described above. However, major contexts can call sub-contexts to handle a specialized situation, which in turn can call sub-sub-contexts, etc. When one such *minor context* finishes executing, it will return control to the major context that activated it. In this manner, CxBR encourages reuse of sub-contexts among multiple major contexts.

When the active major context's transition knowledge deems the context inappropriate for handling the agent's situation, the active context relinquishes control of the agent and a more suitable context is selected as the next active context. A *context map* is used to determine which context transitions are legal. Figure 2.12 shows an example of a context map, wherein the arrows between major contexts show the horizontal relationships between major contexts. For example, Major context 24 can relinquish control to Major context 31, but not vice versa.
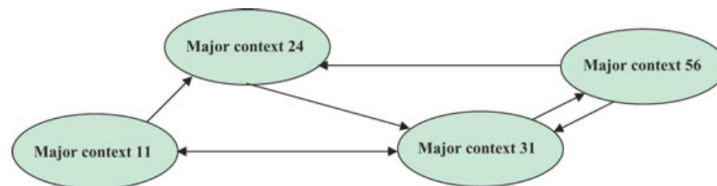
Figure 2.12: Sample CxBR context map, reprinted with permission from [24].

CxBR has been used in several areas of research, including integration with Reinforcement Learning in a driving domain [2], implementation of a project manager agent in a collaborative process

[23], use in modeling hierarchical military agents [46], and in LfO [18] [71] [35] [72]. Fernlund et. al. [18] specifically strove to create CxBR agents and used LfO to learn how to recognize and perform in several manually segmented contexts in a driving domain. Stein and Gonzalez [71] used CxBR to address the poor performance of a context-free LfO technique in a crane-controlling domain. The authors achieved better performance after dividing the crane-controlling task into natural contexts and applying their LfO technique to each context before combining agents created from each context.

Turner [83] presented another context paradigm, Context Mediated Behaviors (CMBs). CMBs represent knowledge as a set of *contextual schemas* (or just c-schemas) that can be "composed" together to describe a particular situation faced by an agent and determine what to do. Turner [83] uses an automous underwater vehicle (AUV) as an example: if the AUV has low power, is under ice, is searching for a sunken ship, and has a malfunctioning sonar reader, then a set of c-schemas describing each of those conditions should be put together to describe the AUV's situation and control its behavior.

The components of a c-schema are:

1. Features that should be present or absent for the c-schema to apply to the situation.

2. Features of the environment that are likely to be present if not yet confirmed.

3. Context-specific definitions of certain terms.

4. Agent goals and their priorities.

5. Procedural knowledge for achieving goals and reacting to unexpected situations.

A *context manager* composes the best c-schemas into a single *context object* that is used to control the agent. New context objects are created as necessary to best fit the agent's situation.

Brézillon [10] presents a third context paradigm, Contextual Graphs (CxGs). CxGs are based on the idea that there are different ways to get from state A to state B, but the manner in which one does is subject to the context. Brézillon [10] defines context as "a collection of relevant conditions and surrounding influences that make a situation unique and comprehensible." In this case, the set of allowable *procedures* are represented by a directed acyclic graph or CxG and the unique paths through the CxG are the specific *practices* or paths that suit the needs of a particular situation, the context. Brézillon [10] uses the example of a company setting allowable procedures for accomplishing certain tasks, but the employees find the best practices or ways of following these procedures to suit the specific needs of their situation.
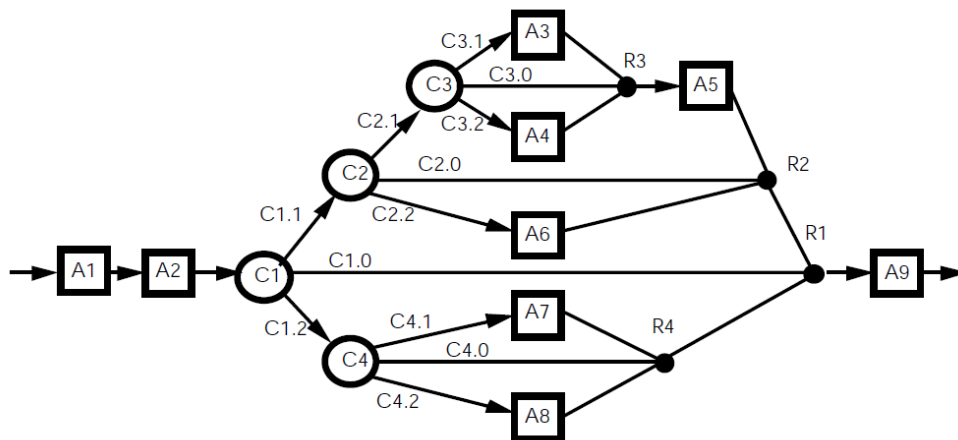
Figure 2.13: Example Contextual Graph, reprinted with permission from [10]

Figure 2.13 shows an example of a CxG. The squares represent *action nodes* wherein a specific action is taken. The contextual nodes are represented by circles with multiple output arrows —

the branch taken depends on the contextual element addressed by that node (e.g. whether a certain condition is true or false) and the responsible contextual element becomes *proceduralized* (i.e. directly relevant to the entity moving through the graph). When the specific goals of the branch have been accomplished, it returns to a *recombination node*, shown in the figure as a bolded dot. Several other node types exist in CxGs, which are not shown in Figure 2.13. Such node types are:

- Sub-graph node: This is a CxG within a CxG, which is feasible, given the acyclic nature of CxGs, and is representative of intermediate goals within the overall mission of traversing the whole CxG [10].

- Parallel Action Grouping node: This node contains several contextual elements which all must be addressed, in any order, before proceeding.

- Activity node: This is a more complex action node that takes into account various contextual elements.

A fourth context method, which combines elements of the prior three methods, is called the Cooperating Context Method (CCM) [33]. CCM, like CMBs, has the capability of combining multiple contexts to solve a set of tasks. However, it also possesses the recursive nature of CxGs' activity nodes and utilizes transitions between contexts like in CxBR. A context in CCM posseses three elements:

1. Descriptive knowledge — the list of tasks that this context can address.

2. Prescriptive knowledge — the know-how behind how to solve tasks, even recursively invoking CCM if necessary.

3. Contextual knowledge — which other contexts are contextually relevant to this context and which ones are irrelevant.

CCM was originally applied to the domain of narrative generation, but its approach to solving tasks is a general-purpose one. Figure 2.14 illustrates the process by which CCM updates the contexts in play to solve tasks in real-time.
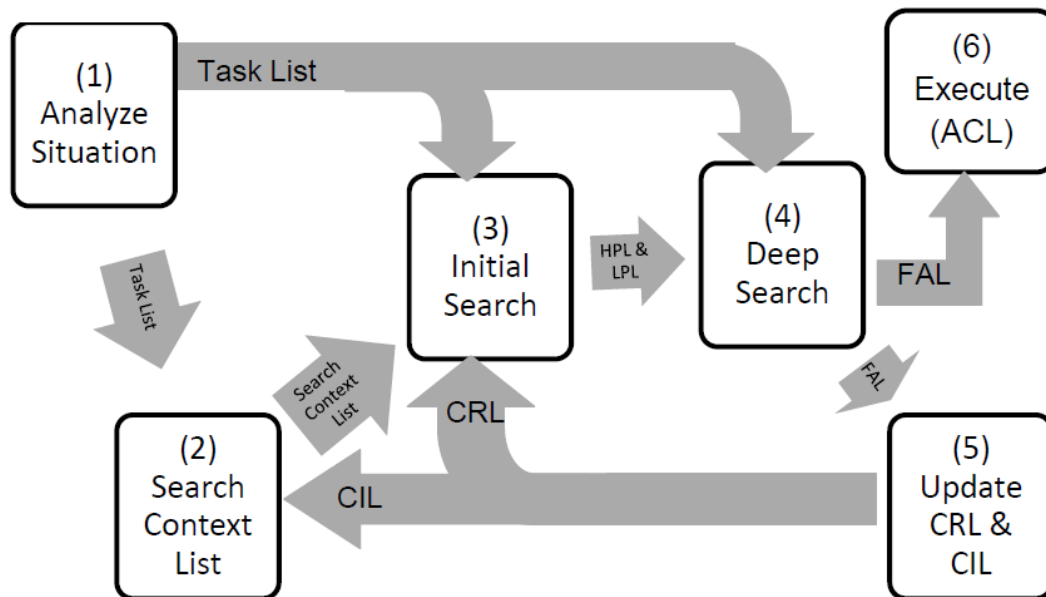
Figure 2.14: The Cooperating Context Method, reprinted with permission from [33]

Reprinted by permission from J. Hollister: A Contextual Approach to Real Time Interactive Narrative Generation by J. Hollister, ©2016.

In a loop, CCM performs the following actions:

1. First, CCM analyzes the situation at hand (e.g. a story in progress) and compiles a list of tasks that need to be completed (e.g. filling in certain story elements).

2. Next, CCM analyzes a predefined list of contexts to determine which contexts are possible solutions for the tasks that must be accomplished. It does this by analyzing the descriptive

knowledge of each context.

3. Given the list of potential contexts, an initial search is performed to separate the contexts into a high priority list (HPL) and low priority list (LPL). The HPL contains the list of contexts that were previously labeled "contextually relevant" and which can solve at least one of the tasks. The LPL contains the list of contexts that were previously labeled "contextually irrelevant" and which can solve at least one of the tasks.

4. Given the HPL and LPL, a deep search is performed to compile the list of contexts that can solve the list of tasks. The HPL is searched first, followed by LPL. The deep search is conducted until all tasks are completed or all contexts have been exhausted. Then, a future active list (FAL) of contexts is compiled. These are the contexts that will be active in the next iteration of CCM.

5. Based on the contexts that are in the FAL, the list of contextually relevant and contextually irrelevant contexts is updated, based on the contextual knowledge of each context.

6. Finally, the prescriptive knowledge of each context in the active context list (ACL) is invoked to solve the compiled list of tasks. In the next iteration of CCM, the contexts in the FAL will comprise the ACL.

We have now discussed four major context paradigms, which have different methods of assessing a given situation and addressing its circumstantial needs. However, all these paradigms seek to exploit the power of context, the inherent divide-and-conquer nature of explicit context definitions [83]. The problem becomes how to best construct the contexts used by any given paradigm.

Trinh and Gonzalez [82] presented the Contextual Partitioning and Clustering (COPAC) method for learning the contexts that were experienced by an observed actor; these contexts can then be used to create CxBR agents through a subsequent LfO algorithm, such as GenCL [18] (which

45

was described in Section 2.1.2). This process of discovering and partitioning contexts in a trace would normally require a developer to manually identify the various contexts in a trace of observed behavior for a desired behavior to be learned.
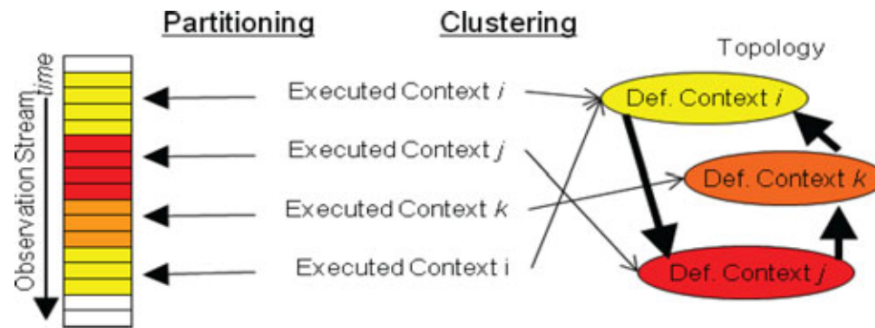
Figure 2.15: COPAC, reprinted with permission from [82].

Reprinted by permission from IEEE: Discovering Contexts from Observed Human Performance by V. Trinh and A. J. Gonzalez ©2013 IEEE.

Figure 2.15 illustrates how COPAC works. Given a sequence of observations (the trace of an actor's behavior), COPAC first *partitions* the trace into segments by finding points in the trace where the actor had a sudden change in behavior in reaction to a salient change in the environment. Then, these segments are clustered by the similarity of the actor's behavior within, per Trinh's definition of context [81]. These clusters comprise the various *defined contexts* and the segments within each cluster are *executed contexts* or *instances* of the defined context. The temporal relationships between executed contexts belonging to different defined contexts inform the context map (the set of legal context transitions).

The work by Trinh and Gonzalez [82] is very relevant to our research and shares several of the same challenges. Instead of applying a predefined context hierarchy to the LfO task for context-

46

based methods, Trinh and Gonzalez [82] sought to discover a context map that was tailored to a specific observed actor. This was a difficult task because the identity and structure of an observed actor's active context is unobservable and often cannot be easily articulated. The same can be said for internal state based on memory. What an observed actor chooses to remember, how such memories are used in decision-making, and how long the memories persist are all unobservable and not always easily articulated. Additionally, contextualization of an observed actor's trace served as a preprocessing step to the LfO task to better inform the learning process. Similarly, inferring the specific internal state structures that influence an observed actor's behavior as a preprocessing step to the learning task can eliminate unnecessary and potentially erroneous assumptions about an actor's internal state that would otherwise have to be predefined prior to learning. Finally, given the potentially large variety and complexity of memory-influenced behaviors, it is advantageous to use context to divide the task of discovering internal state structures that influence an observed actor's behavior into smaller context-specific subtasks.

Thus, we emphasize here that our research does not seek to improve the learning process. Rather, it seeks to model a specific *aspect* of the observed behavior as a preprocessing step that provides additional input to a machine learning algorithm. Similar to how COPAC [82] learns a context map, our research seeks to learn a model of memory usage that accounts for influences of memory on an observed behavior; the structure of this model and the process by which it is obtained is covered in Chapter 6.

2.3   Imitation Learning from Observation

As of the writing of this dissertation, the latest works on LfO have been done under the label of "imitation learning". The general idea behind imitation learning is the same as that for LfO in that a learning agent learns to perform a task by observing another actor doing it. More precisely,

imitation learning is defined as "the process by which one agent tries to learn how to perform a certain task using information generated by another, often more expert agent performing that same task" [79]. Even though this dissertation prefers the use of "learning from observation", we will use the label "imitation learning" in this section to conform to the terminology of the works referenced here and to avoid confusion with "imitation learning from observation", which will be discussed later.

The works in imitation learning are heavily influenced by the constraints surrounding the use of physical robots. In the early days of imitation learning, Schaal [65] discussed the potential of imitation learning to create appropriate control policies for humanoid robots. He proposed imitation learning as an alternative to reinforcement learning, which would learn relatively slowly due to having to explore the large action space available to robots with many degrees of freedom in movement. In contrast, imitation learning would provide a more focused exploration of the action space (e.g. concentrated exploration around reasonably competent behaviors demonstrated by experts) in pursuit of suitably generalizable control policies for robots. Even as early as Schaal's work [65], the need for using proper learning to link perceptions to appropriate actions in the robot was recognized.

The two main approaches to imitation learning according to [79] have been behavior cloning and inverse reinforcement learning. In behavior cloning (BC), a supervised learning algorithm is used to derive a control policy for a learning agent. BC can be achieved relatively quickly because the learning agent can derive a policy from the observation data without having to interact with its environment — the learning agent need only learn how to predict the observed actor's actions given the perceived world state. However, BC can be subject to the "covariate shift" problem, which deserves attention here.

The covariate shift problem [79], in the context of imitation learning, is the reliance on the false

assumption that the distributions for the training data (from which the learning agent learns) and the testing data (on which the learning agent is evaluated) are the same. This often is not the case because if the learning agent deviates from the observed actor's policy even just a little bit, it will encounter a completely new set of situations which the expert never encountered during his run. Ross et. al. [62] developed an imitation learning technique to mitigate the effects of covariate shift by allowing a learning agent to query an expert for the next action during the learning process. This accomplished the dual objectives of 1. teaching the learning agent how to recover from mistakes, and 2. gradually transitioning the learning agent's policy to that of the expert's with minimal deviations (also known as "regret"). However, the authors were only able to bound the effects of covariate shift, not eliminate it completely. Furthermore, the reliance on querying the experts strays from the general spirit of learning from observation alone.

In the second approach to imitation learning, inverse reinforcement learning (IRL) [52], the reward (a.k.a. goal or intent) of the observed actor is inferred so that reinforcement learning techniques can be used to maximize the inferred reward. Thus, IRL techniques are grounded in Markov Decision Processes (MDPs), which are briefly discussed here.

MDPs are a family of problems that are composed of the following components [1]:

- States S - a finite set of states that may be perceived by a learning agent

- Actions A - a finite set of actions that may be executed by the agent to change the world state

- Transitions T - a set of probabilities of transitioning to each state, given the previous state and the agent's choice of action

- Discount factor γ - the amount by which rewards decrease over time

- Initial state distribution D - the possible set of initial states

- Reward R - the function that dictates how much reward an agent receives for being in a certain state. For certain problems, it is assumed that the reward function can be expressed in terms of the features through which the world state is expressed.

In MDPs, the learning agent wishes to learn a policy π that maximizes its reward. Algorithms that learn such policies through trial-and-error are reinforcement learning algorithms [75]. However, it is often difficult to manually specify an appropriate reward function for certain behaviors. In such cases, it is necessary to infer an appropriate reward function from demonstrations of desired behavior. This is accomplished through IRL.

In [1], "apprenticeship learning" is proposed as way to infer a reward function. Under the assumption that the reward function is simply a function of the perception features, apprenticeship learning uses a technique similar to those of support vector machines to find appropriate feature vectors to express the reward function. A learned policy is iteratively modified according to the following algorithm:

1. Initialize a random policy and random reward function.

2. Compute the deviation of the current policy from the observed policy.

3. If the deviation is sufficiently minimal, you're done.

4. Use reinforcement learning to find a policy that best fulfills the current reward function.

5. Update the reward function and go to step 2.

This process finds a reward function that approximates the (unknown) reward function that the observed actor is supposely trying to maximize, but the true reward function remains unknown. It should be noted that this work seeks an agent that is competent at the task at hand, and not an agent

that performs as similarly to the observed actor as possible. However, the agent's performance will be similar to that of the observed actor because the actor's performance is the only notion of correct behavior available in this problem context.

The recent interest in deep learning techniques has sparked new works in imitation learning. For example, within the BC approach, imitation learning was applied to teaching a learning agent to steer a physical car using a convolutional neural network (CNN) [7]. This work was in response to a challenge by the Defense Advanced Research Projects Agency (DARPA) to create autonomous vehicle controllers. The authors wanted to do so without extensive feature engineering and thus elected to use a CNN, which can learn its own features via image processing. Admittedly, the authors had to augment their dataset of human driving images with synthetic images of a car going offroad, so that their CNN could learn how to recover from mistakes. One should note that this work by NVIDIA is a more recent version of Pomerleau's work with ALVINN [60], which used a shallow neural network.

Also along the lines of the BC approach to imitation learning, a quadcopter or Micro Aerial Vehicle (MAV) was trained to navigate forest trails using a deep neural network that was trained on images of forest traversals generated by a human with head-mounted cameras [21]. The authors tackled the problem by converting it into a three-class classification problem where the quadcopter's response to each image was the command to turn left, go straight, or turn right.

However, in conformity to the IRL approach to imitation learning, a few works have used deep adversarial nets, which we briefly cover here.

Generative adversarial networks (GANs) are deep neural networks that seek to learn to generate samples from a distribution of interest, given examples of true samples from said distribution [25]. GANs train two neural networks, a generator and a discriminator, which learn together. The generator learns to mimic the distribution by generating fake samples. The discriminator learns to

distinguish between true samples from the distribution and fake samples from the generator. These networks can be considered "adversaries". As both networks improve at their respective tasks, the generated samples eventually become indistinguishable from the real samples.

The seminal work in [30] introduced a generative adversarial imitation learning (GAIL) technique that learns a policy from demonstrated behavior with a GAN as an alternative to traditional IRL. It learns effectively with only a few demonstrations of desired behavior, but still requires a learning agent to interact with the environment, as do all IRL approaches.

However, the limitation with all the imitation learning works discussed thus far is that all these works assume that an observer has access to the specific actions taken by the observed actor. In reality, an observed action taken by an expert/teacher/demonstrator does not clearly translate to a specific action that should be taken by a learning agent (e.g. a physical robot). For example, if a humanoid robot wants to learn how to make egg-salad sandwiches by watching a Youtube video of someone doing so, then the robot is capable of seeing how the human spreads the egg salad on bread slices and so forth, but the robot does not have access to which muscular contractions the human employed to carry out the task, much less which low-level mechanical operations (limb torque, gripping strength, etc.) it needs to use to copy the human's movements. In other words, the robot learner can see the effects of the human's actions, but does not know the actions themselves. Therefore, the robot must infer which of its actions must be taken to achieve the same effects as those observed in the human's behavior.

In response to this, a specialized form of imitation learning was created: "imitation from observation" (IfO) [45] or "imitation learning from observation" (ILfO). Several works have operated under the constraint that the experts actions are not available to the learning agent, creating a new problem of how to deal with perceptions gathered from observation. Some works have used either teleoperation or sensors on a demonstrator to provide a learning agent (e.g. a robot) with percep-

tions similar to those it would experience during deployment [79] [4]. Otherwise, the learning agent must contend with "viewpoint difference" and "embodiment mismatch" [79]. Viewpoint difference is the problem where an observed actor's perception during a performance differs from what a learning agent perceives within a training environment. Embodiment mismatch is the problem where an observed actor has a different appearance or body than the learning agent and the learning agent must map the observed actor's body movements to appropriate movements in its own body.

To mitigate problems associated with viewpoint difference (ignoring embodiment mismatch), the work in [45] learned a context translation model to convert observations of the actor into suitable inputs for the learning agent. Basically, the context translation model learned how to convert a set of observations from one demonstration (this is called a single "context") into the set of observations present in a second demonstration (called a second "context"), under the assumption that the demonstrations are time-aligned exactly. Then, reinforcement learning is applied by the learning agent, using a handcrafted distance metric reward function. The method was validated in a simulated and physical robotic arm performing various simple tasks.

Another approach for addressing viewpoint difference is perspective-taking. Perspective-taking [80] is a mechanism in human-robot interactions wherein a robot determines how to accomplish a task given by a human by reasoning about information available to the human. Trafton et. al. [80] used the example of a wrench to illustrate this concept. In the example, a robot and a human exist within an environment with two wrenches on a table (one of which is hidden from the human's view). When the human tells the robot, "Give me the wrench," then the robot can either ask the human which wrench should be delivered or the robot can assume that the human must be referring to the wrench she can see. The latter option, perspective-taking, is useful for resolving ambiguity in high-level tasks where a learning robot's perception capabilities differ significantly from those of an observed actor being learned from.

However, if problems with perception are sufficiently addressed, then the next issue ILfO must address is the manner by which a control policy is derived. According to [79], there are two model-based ILfO approaches (inverse and forward) to agent control and two model-free approaches (adversarial and reward-engineered).

In the inverse dynamics model approach, a learning agent must learn which actions in its action set correspond to certain state-transitions. In a forward dynamics model approach, the learning agent must learn which states result from specific actions taken in a given state. The adversarial approach uses GAN structures to learn an appropriate policy without a model of the world, while the reward engineering approach uses a handmade reward function that is approximated based on the observed actor's behavior.

Within this framework, recent work in ILfO has pursued measures to either: 1) reduce the number of times a learning agent must interact with the environment to create an effective control policy or 2) reduce the number of demonstrations required from an observed actor to form an effective policy. Regarding the first reduction, the work in [78] introduced Behavioral Cloning from Observation (BCO), which aims to learn to mimic an observed actor without having to interact with the environment. It works as follows:

1. Learn an inverse dynamics model of the environment. This is akin to how humans already possess prior experience before engaging in an imitation task — they know how the world works.

2. Based on the sequence of states that are perceived during an observed actor's behavior, infer the actions they took to effect those state transitions, using the inverse dynamics model learned previously.

3. Use standard behavior cloning to learn a policy to mimic the state-action sequence for the

observed actor's behavior.

4. If a budget of limited interactions with the environment is allowed, interact with the environ-ment and update the inferred actions of the observed actor and thus the control policy.

The authors showed in various simulated tasks that BCO's performance is generally comparable to other imitation learning techniques (including GAIL [30]) despite being the only method without access to the observed actor's actions.

Regarding the second reduction, pursuant to minimizing the number of required demonstrations from an observed actor for effective learning, two works reported in the literature have worked to increase "sample efficiency". The first work [77] improves upon a previous adversarial imitation learning technique by replacing the model-free reinforcement learning component of the GAN with a model-based reinforcement learning technique (linear quadratic regulators or LQRs). The LQRs assume that the environment operates linearly with a quadratic reward. This technique was found to potentially outperform its predecessor with fewer observations in earlier iterations of learning. (With more iterations, the more complex predecessor technique is better.)

The work in [74] provides a bound on sample efficiency for its proposed Forward Adversarial Imitation Learning (FAIL) technique. FAIL works by learning a separate policy for every time step of a task through a series of min-max games between the learner agent and the observed actor (the expert) in order to minimize regret during learning. The authors make the claim that FAIL is the "first provably efficient algorithm in ILfO setting", meaning that the bound on the number of required samples for FAIL is known.

We now discuss where our research is situated within the imitation learning from observation framework outlined in this section. The biggest distinction between imitation learning from ob-servation and previous LfO research is the constraint on IfO that a learner be able to mimic an

observed actor without having access to the expert's actions. This constraint is particularly important for robotics applications where embodiment mismatch is a significant consideration. However, the objective of our research here is to investigate how LfO may be improved with a learned memory model of an observed actor's performance, and thus we elect to ignore this constraint in our work — we assume that action information by an observed actor is available for learning for the purposes of this dissertation. Schaal proposed "movement primitives" as a way by which robotic movement could be mapped to specific objective-specific actions [65], thus mitigating the problem of embodiment mismatch when processing observations of demonstrated behavior. Inverse dynamics models, such as [78], can be learned in order to infer an observed actor's actions without regard to memory. Moreover, this constraint of not knowing an observed actor's actions would confound our analysis of memory-based behaviors in our work.

Furthermore, the above works in imitation learning do not address the concerns of memory influences on observed behavior. Other than accounting for viewpoint difference [45] or not being able to observe an actor's actions, the works described in this section do not account for unobservable aspects of a task, such as occluded environmental influences or an actor's internal state (e.g. memory influences). Some works use standard supervised learning [7] [21] without regard to the effects of an agent's past decisions. Other works, especially those centered around MDPs, characterize a policy by Markovian (memory-less) states [1], [78]. Even those works that develop control policies iteratively (such as FAIL [74], which conditions a policy for a given time step on the policies derived for prior time steps) do not create a memory model from observed behavior to augment decision making during agent control.

The primary concerns of the ILfO field seem to be centered around capitalizing on the utility of current deep learning techniques and catering to the needs of physical applications, such as robotics. Our research described in this dissertation seeks to investigate the feasibility of learning a memory model of human performance from observation, which can be a separate module that is

added to the lines of research described in this section.

2.4   Summary

In this chapter, we reviewed the history of LfO, discussed the formalization of the LfO task, and analyzed the various works that have addressed the problem of learning memory-influenced behaviors. We then reviewed three related fields, temporal-based learning, memory models, and context, in our search for inspiration and insight into learning the internal state of an observed entity as it is influenced by memory of past events. Finally, we reviwed imitation learning from observation, the latest work in LfO centered around the opportunities and needs surrounding deep learning and robotics. With the establishment of the current state of the art regarding learning memory-influenced behaviors, we now proceed to state the problem addressed by our research, the hypothesis to be investigated, and our contributions to the field.

CHAPTER 3: PROBLEM DEFINITION

3.1 General Problem

Learning from Observation is increasing in importance as the problems tackled by machine learning become increasingly abstract and harder to define via static training samples. Often, a series of observations is more conducive for extracting knowledge from an entity to be modeled than the traditional interviewing process or standard construction of independent training examples. LfO has made great strides toward automating knowledge acquisition and improving the learning process, but its greatest challenge still remains its inability to elucidate the internal state structure used by the observed entity while executing a given behavior. Much research has accomplished the objective of LfO in spite of this challenge of unobservability [19] [70] [7] [21] while other research endeavors have sought to better understand this challenge [58] [20] [27] [76].

3.2 Specific Problem

One aspect of the unobservable nature of an entity's behavior is how memory of past events affects its decision-making process. Specifically, which aspects of the environment does a performer choose to remember? When do these remembered events or objects become relevant to the next decision? For how long are the contents of memory stored and retrieved? The significance of these questions for understanding how to incorporate memory into LfO make it paramount for further research to model how memory is composed, modified, and used in decision-making over time, in order to more fully and accurately extract knowledge from observation.

In our dissertation research, we describe a new technique, *Memory Composition Learning*, that

composes memory structures that approximately model how past interactions with the environment influence an observed entity's behavior. Our aim is to analyze a trace of an entity's behavior and construct an auxiliary set of features that encapsulate the salient aspects of past events that have significant correlation with future decisions. The incorporation of a set of memory features within a trace used as input for LfO would reduce or eliminate perception aliasing [16] in machine learning algorithms limited to learning at most Level 2 LfO behaviors, as discussed in Chapter 2. Our approach more intelligently and explicitly models how an observed entity uses memory for the purposes of analyzing this usage and enabling memoryless learning techniques to capitalize on memory. This technique will alleviate burdens on learning caused by overly simple assumptions on memory influence on an observed behavior; it allows memory to become more explicit and human-readable for behavioral analysis domains; and it will hasten deployment of behaving agents by maintaining a memory-aware internal state instead of recomputing memory influence at every decision.

3.3   Hypothesis

The following statement is the hypothesis of this research:

"The internal state of an observed entity exhibiting a behavior possibly reliant on memory of past events can be modeled through a set of memory features that approximate the influence of memory on the entity's decisions."

3.4   Limitations of the State of the Art Addressed by Dissertation Research

The LfO literature reports many lines of research that generally assume at least one of the following approaches regarding memory-based behaviors:

- The learning technique does not account for memory of past events *at all*. The disadvantage of this approach is that behaviors that rely on memory are common, and failing to account for memory at best reduces the modeled knowledge to a probabilistic emulation of what could otherwise be a deterministic phenomenon.

- The learning technique does not model memory of past events explicitly, but rather computes the influence of memory during the decision-making phase of execution. The disadvantage of this approach is that no understanding is gleaned about how the behavior is influenced by memory. Furthermore, the lack of an internal state model makes decision-making during execution a potentially inefficient process when the influence of memory is complex.

- The learning technique predefines a set of possible states that encapsulate the internal state of an observed entity. The disadvantage of this approach is that assumptions must be made as to the range and structure of the internal state; such assumptions will likely be wrong in many instances. If the assumptions are too conservative, then the internal state representation will be inadequate for fully capturing the influence of memory on the observed behavior. If the assumptions are too liberal, on the other hand, then the learning algorithm will be unnecessarily burdened.

Our proposed approach, Memory Composition Learning (MCL), overcomes the above limitations of the current approaches. MCL accounts for the influence of memory, as dictated by the created memory feature set, which will enable machine learning algorithms to capitalize on pertinent information about memory. Additionally, MCL extracts from the data (trace) the correlative aspects of past events that influence future decisions, and explicitly models them in an auxiliary feature set. Thus, there is no need to go back in time when making a decision at a given time step; the feature set itself reveals which aspects of the past actually matter. Finally, MCL *automatically learns* the structure and range of memory usage, an approximation of an observed entity's internal

state regarding memory that does not rely on unnecessary assumptions.

3.5   Contributions

Our research contributions to the literature for LfO in particular and machine learning in general are as follows:

- A procedure (MCL) that can automatically learn a feature set that encapsulates the salient aspects of past events that influence future decisions. This feature set can be used to improve performance in LfO algorithms. MCL is different from past approaches to learning memory-based behaviors in that it explicitly models memory's influence on a behavior and automatically discovers the structure and range of memory influence. MCL attempts to infer and approximate one aspect of the unobservable aspect of behavior: the retention, modification, and utilization of memory of past events.

- A prototype system implementing the MCL algorithm.

- Data comparing the performance of memory-agnostic learning techniques with that of techniques augmented by MCL preprocessing.

The next chapter discusses behavior representations that are used in our conceptual approach to our defined problem.

CHAPTER 4: BEHAVIORAL REPRESENTATIONS

This chapter discusses how behaviors and memory of behaviors are represented in this research, particularly in the realm of Case Based Reasoning (CBR). Specifically, we provide detailed descriptions of how we choose to represent *behaviors* in terms of how they are generated by agents, stored in a case base, traversed via the Temporal Backtracking algorithm, and condensed into a memory model describing an internal state structure of our choice. These concepts are necessary for understanding the conceptual approach we take to address the problem from Chapter 3.

This chapter is organized as follows:

- Section 4.1 describes three types of agents that can perceive, reason, and act within an environment. It also describes our choice of agent for this research.

- Section 4.2 describes the XYZ domain, a custom-made theoretical environment that we invented and use as a running example in this dissertation.

- Section 4.3 describes CBR conventions that we use in this research for representing recorded behaviors.

- Section 4.4 describes the Temporal Backtracking algorithm, as it was presented in [20]. It is a core mechanic in our conceptual approach.

- Section 4.5 describes Memory Features, novel memory modeling elements of our own invention and the intended output of MCL.

4.1 Behavioral Agents

Agents are the actors within an environment whose actions over time change the state of the environment. Their behaviors and how they are influenced by memory are the object of our investigation in this research. There are different ways to represent how behaviors are generated by agents. We borrow the conventions used by Gunaratne, Esfandiari, and Fawaz [27], which they took from [89]. Those conventions (discussed next) are used throughout the rest of this dissertation.

Let the finite set of possible *states* that the environment may assume be represented as:

$$E = \{e_0, e_1, ..., e_n\}$$

Let the finite set of possible *actions* that an agent can execute in the environment be defined as:

$$A = \{a_0, a_1, ..., a_n\}$$

The agent interacts with the environment by *perceiving* which state the environment is in and then *executing* an action in response. This may change the environmental state that the agent then perceives and responds to with some further action, if necessary. This sequence of perception-action pairs is called a *run* and it is represented as follows:

$$r : e_0 \xrightarrow{a_0} e_1 \xrightarrow{a_1} \ldots \xrightarrow{a_{n-1}} e_n$$

Let R be the set of all finite runs and let $R^E \subseteq R$ be the subset of runs in R that end in a specific environmental state. From a run composed of n perception-action pairs (from environmental state e_0 to state e_{n-1}), we can extract $n - 1$ runs. Those runs are:

- e_0

- $e_0 \xrightarrow{a_0} e_1$

- $e_0 \xrightarrow{a_0} e_1 \xrightarrow{a_1} e_2$

- ...

- $e_0 \xrightarrow{a_0} e_1 \xrightarrow{a_1} e_2 \xrightarrow{a_2} \ldots \xrightarrow{a_{n-2}} e_{n-1}$

Each "sub-run" is a complete run in its own right, but part of a bigger run that continues it. However, if a run is *not* part of some bigger run because the run has reached a terminal state, then we consider this special case a *trace*, a complete, finite, fully expanded record of an observed entity's behavior. (We should note that the term "trace" is not part of the conventions of [27], but we included the concept of a trace in order to eliminate potential confusion between a *run*, which could be continued by further interactions of an observed entity with its environment, and a *trace*, which cannot be continued in this way.)

Given the above, we define three agents (following the conventions by [27]).

- *Standard agent*: chooses the next action Ac based on the run in progress R^E leading up to the most recently perceived environmental state. This agent is informed by memory of previous states and actions.

$$Ag^{Std} : R^E \rightarrow Ac$$

- *Reactive agent*: chooses the next action Ac based solely on the most recently perceived environmental state E.

$$Ag^{Rct} : E \rightarrow Ac$$

- *State-based agent*: chooses the next action Ac based on the most recently perceived environmental state E and an internal state I, which is updated based on E. This agent is also informed by memory that is embedded in the internal state.

$$Ag^{StB} : E \times I \rightarrow Ac$$

$$Ag^{StB} : E \times I \rightarrow I$$

For clarity, we also define I as the finite set of possible internal state values that a state-based agent can assume for its internal state.

$$I = \{i_0, i_1, \ldots, i_n\}$$

For example, a *standard agent* might be operating within the following run in progress:

$$e_0 \xrightarrow{a_0} e_1 \xrightarrow{a_1} e_2$$

The symbols e_0, e_1, e_2 represent the perceived environmental states at time steps $0, 1,$ and $2,$ and the symbols a_0 and a_1 represent the actions taken by the agent at time steps 0 and 1. The specific values represented by these symbols can be anything. Based on the entire run up to the perception of environmental state e_2, the standard agent will then determine that the next action to execute is a_2, and thus continue the run in progress:

$$e_0 \xrightarrow{a_0} e_1 \xrightarrow{a_1} e_2 \xrightarrow{a_2}$$

The environment will change, and in the next time step, the run will be as follows:

$$e_0 \xrightarrow{a_0} e_1 \xrightarrow{a_1} e_2 \xrightarrow{a_2} e_3$$

The new run, ending in the perception of environmental state e_3, will be the input to the standard agent in the next time step.

In contrast, a *reactive agent* has no memory of past events. Suppose it also has the following run in progress:

$$e_0 \xrightarrow{a_0} e_1 \xrightarrow{a_1} e_2$$

Without memory, the reactive agent's only input is the most recent perception of the environment:

$$e_2$$

Solely from this one observation, a reactive agent will determine that the next action to execute is a_2 (which is not necessarily the same action a_2 taken by the standard agent). The environmental state will change, and so its run in progress will be:

$$e_0 \xrightarrow{a_0} e_1 \xrightarrow{a_1} e_2 \xrightarrow{a_2} e_3$$

However, in the next time step, the only information to which the reactive agent has access to is the most recent environmental perception:

$$e_3$$

Therefore, the reactive agent is implicitly relying on the *Markov assumption*, that assumes that all relevant information for making the next decision (the next action to execute) is contained within the most recent perception.

The *state-based agent*, like the reactive agent, uses only the most recent perception in its run and ignores all past interactions. However, like the standard agent, the state-based agent has the capacity for memory because it maintains an internal state I, whose structure is arbitrary. Thus, if

a state-based agent has the following run in progress:

$$e_0 \xrightarrow{a_0} e_1 \xrightarrow{a_1} e_2$$

its only inputs are the most recent perception:

$$e_2$$

and its internal state (with an unspecified structure in our example):

$$i_2$$

The state-based agent, given e_2 and i_2, will determine that the next action is a_2. However, it must also update its internal state from i_2 to i_3 as a result of perceiving e_2. In effect, the state-based agent incorporates e_2 into its "memory" when it updates its internal state from i_2 to i_3 as a result of perceiving environmental state e_2, because the internal state is the encapsulation of all influences from past events. Then, the environmental state will become e_3 as a result of the agent executing action a_2. Then, the inputs to the state-based agent in the next time step will be the perception of environmental state e_3 and the agent's internal state i_3.

In our research, we seek to approximate the internal state I with respect to memory. This internal state approximation can then be used in the creation of state-based agents by a machine learning algorithm. In other words, MCL will learn how to approximate I; then, the machine learning algorithm will learn how to use I in creating state-based agents. The machine learning algorithm that uses I can be any machine learning algorithm, which is not a part of our research.

4.2 The XYZ Domain

The XYZ domain is a simple domain of our own invention, created solely for providing conceptual examples. In the XYZ domain, there are three possible states of the environment: X, Y, and Z. Within this world, an agent can execute one of two actions: a or b. Figure 4.1 illustrates how the different actions (indicated by arrows) change the environment's state. In state X, the environment will transition to state Y if either action a or b is taken by the agent. In state Z, the environment will transition to state X if either action a or b is taken by the agent. In state Y, action a always transitions the environmental state to state Z, but action b has a random effect on the environment: 50% of the time, action b will transition the environmental state to state Z, and the rest of the time, the environmental state will not change (it will remain in state Y).
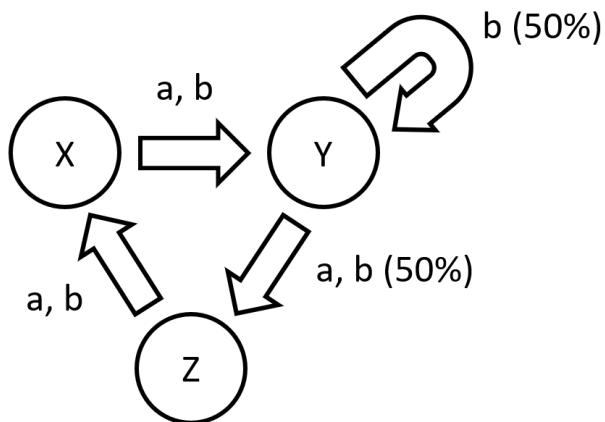
Figure 4.1: XYZ Domain Finite State Machine

We define one *agent* for this XYZ domain; we simply call it the XYZ agent. The XYZ agent's behavior is defined by the following rules:

1. If the agent perceives environmental state X, choose the action that was taken one time step

68

ago. Choose action a by default if it's the first time step.

2. If the agent perceived environmental state Y, choose action b.

3. If the agent perceives environmental state Z, choose the opposite of the action taken when the environment was previously in state Z. So, if the agent chose action a when it previously perceived environmental state Z, then it should choose action b; and if the agent chose action b when it previously perceived environmental state Z, then it should choose action a. Choose action a by default if this is the first time that environmental state Z has been perceived by the agent.

This agent's behavior is trivial in that different actions will in some cases affect the environment in the same way (e.g. when transitioning from state X or Z, any action gets the agent to the same next state), but it is interesting because the actions chosen depend on memory of past events. Again, the XYZ domain and the behavior of the XYZ agent exist only for the purpose of providing examples in this dissertation.

We now provide an example *run* of the XYZ agent's behavior in the XYZ world. Suppose that the XYZ agent perceives environmental state X.

$$run : X$$

According to rule 1, the agent should choose the action taken one time step ago, but this is the first time step, so the agent chooses action a by default.

$$run : X \xrightarrow{a}$$

According to the finite state machine in Figure 4.1, the environment will transition from state X to

69

state Y.

$$run : X \xrightarrow{a} Y$$

A single *time step* encompasses a single *perception* followed by a single *action*. For clarity, we provide subscripts indicating which time step a given perception or action belongs to. We rewrite the current run of the XYZ agent with this convention and use it in the rest of this dissertation. Thus, our run so far consists of time steps 0 and 1.

$$run : X_0 \xrightarrow{a_0} Y_1$$

Upon perceiving environmental state Y, the agent will execute action b according to rule 2.

$$run : X_0 \xrightarrow{a_0} Y_1 \xrightarrow{b_1}$$

According to the finite state machine in Figure 4.1, the environmental state will either remain the same or transition to state Z. The probability of both outcomes is 50%, so for this example, let's assume that the next environmental state becomes Z.

$$run : X_0 \xrightarrow{a_0} Y_1 \xrightarrow{b_1} Z_2$$

Upon perceiving environmental state Z, the agent will execute the action taken when the environmental state was previously in state Z (according to rule 3). However, this is the first time the agent has perceived environmental state Z, so it chooses action a by default.

$$run : X_0 \xrightarrow{a_0} Y_1 \xrightarrow{b_1} Z_2 \xrightarrow{a_2}$$

According to the finite state machine in Figure 4.1, the environmental state will change from state

Z to state X.

$$run : X_0 \xrightarrow{a_0} Y_1 \xrightarrow{b_1} Z_2 \xrightarrow{a_2} X_3$$

Upon perceiving environmental state X at time step 3, the agent will execute the action from the previous time step (time step 2), according to rule 1. This action is action a.

$$run : X_0 \xrightarrow{a_0} Y_1 \xrightarrow{b_1} Z_2 \xrightarrow{a_2} X_3 \xrightarrow{a_3}$$

According to the finite state machine in Figure 4.1, the environmental state will change from X to Y.

$$run : X_0 \xrightarrow{a_0} Y_1 \xrightarrow{b_1} Z_2 \xrightarrow{a_2} X_3 \xrightarrow{a_3} Y_4$$

Upon perceiving environmental state Y, the agent will execute action b, according to rule 2.

$$run : X_0 \xrightarrow{a_0} Y_1 \xrightarrow{b_1} Z_2 \xrightarrow{a_2} X_3 \xrightarrow{a_3} Y_4 \xrightarrow{b_4}$$

According to the finite state machine in Figure 4.1, the environmental state will either remain the same (Y) or become Z. Let's say that it remains the same in this example.

$$run : X_0 \xrightarrow{a_0} Y_1 \xrightarrow{b_1} Z_2 \xrightarrow{a_2} X_3 \xrightarrow{a_3} Y_4 \xrightarrow{b_4} Y_5$$

Upon perceiving environmental state Y in time step 5, rule 2 dictates that the agent will perform action b.

$$run : X_0 \xrightarrow{a_0} Y_1 \xrightarrow{b_1} Z_2 \xrightarrow{a_2} X_3 \xrightarrow{a_3} Y_4 \xrightarrow{b_4} Y_5 \xrightarrow{b_5}$$

According to Figure 4.1, the environmental state will either remain the same (Y) or become Z.

Let's say that the environment transitions to state Z in this example.

$$run : X_0 \xrightarrow{a_0} Y_1 \xrightarrow{b_1} Z_2 \xrightarrow{a_2} X_3 \xrightarrow{a_3} Y_4 \xrightarrow{b_4} Y_5 \xrightarrow{b_5} Z_6$$

Upon perceiving environmental state Z, the agent will execute the action taken when the environmental state was previously in state Z (according to rule 3). The last time the XYZ agent had perceived environmental state Z was in time step 2. In that time step, it performed action a, so according to rule 3, the agent should choose action b in the current time step (time step 6).

$$run : X_0 \xrightarrow{a_0} Y_1 \xrightarrow{b_1} Z_2 \xrightarrow{a_2} X_3 \xrightarrow{a_3} Y_4 \xrightarrow{b_4} Y_5 \xrightarrow{b_5} Z_6 \xrightarrow{b_6}$$

According to the finite state machine in Figure 4.1, the environmental state will transition from Z to X.

$$run : X_0 \xrightarrow{a_0} Y_1 \xrightarrow{b_1} Z_2 \xrightarrow{a_2} X_3 \xrightarrow{a_3} Y_4 \xrightarrow{b_4} Y_5 \xrightarrow{b_5} Z_6 \xrightarrow{b_6} X_7$$

Upon perceiving environmental state X in time step 7, the agent will select the action taken in the previous time step (time step 6), which is action b.

$$run : X_0 \xrightarrow{a_0} Y_1 \xrightarrow{b_1} Z_2 \xrightarrow{a_2} X_3 \xrightarrow{a_3} Y_4 \xrightarrow{b_4} Y_5 \xrightarrow{b_5} Z_6 \xrightarrow{b_6} X_7 \xrightarrow{b_7}$$

There is no terminal condition specified in the behavior of the XYZ agent, so it will proceed until the end of time.

$$run : X_0 \xrightarrow{a_0} Y_1 \xrightarrow{b_1} Z_2 \xrightarrow{a_2} X_3 \xrightarrow{a_3} Y_4 \xrightarrow{b_4} Y_5 \xrightarrow{b_5} Z_6 \xrightarrow{b_6} X_7 \xrightarrow{b_7} \dots$$

However, let's assume that we cease to care what the agent does after time step 100, because our

research requires agent runs to be finite in duration.

$$run : X_0 \xrightarrow{a_0} Y_1 \xrightarrow{b_1} Z_2 \xrightarrow{a_2} X_3 \xrightarrow{a_3} Y_4 \xrightarrow{b_4} Y_5 \xrightarrow{b_5} Z_6 \xrightarrow{b_6} X_7 \xrightarrow{b_7} \ldots \xrightarrow{Ac_{99}} perception_{100}$$

The above constitutes an example run from the XYZ domain. Next, we describe CBR conventions and demonstrate how they can be used to represent this example run in a case base.

4.3   Case Based Reasoning Conventions

This section describes the conventions we borrow from CBR for this research. In our research, we use CBR to store traces by an observed human. Problem-solving occurs in CBR as follows [20]:

1. A *test problem* is presented to the CBR framework. The structure of that problem is domain-dependent.

2. The CBR framework examines a prebuilt *case base* for a possible solution. This case base is filled with many *cases*. Each case is a problem-solution pair.

3. During the *retrieval* stage, each case's problem component is compared to the test problem to be solved. All cases with problem components that have a sufficient similarity to the test problem are retrieved. These are *candidate cases*.

4. During the *reuse* stage, the candidate cases are evaluated for their ability to solve the test problem and the best one's solution component is used.

5. Other stages of the CBR cycle include *revision* of a used case if it is proven to be inadequate for solving the test problem and its *retention* in the case base, but our research does not involve these stages.

In our research, each case's *problem* component is a run (at some point in time) ending in an environmental state (R^E), and its *solution* component is an action (Ac) which the observed entity took at that point in the run. In our research, a *case base* is composed of cases. Each case C is represented as follows:

$$C : R^E \times Ac$$

Therefore, each sub-run in a trace is represented as a separate case in the case base.

Table 4.1: Case Base for XYZ Agent

Case Number	Problem	Solution
0	X_0	a_0
1	$X_0 \xrightarrow{a_0} Y_1$	b_1
2	$X_0 \xrightarrow{a_0} Y_1 \xrightarrow{b_1} Z_2$	a_2
3	$X_0 \xrightarrow{a_0} Y_1 \xrightarrow{b_1} Z_2 \xrightarrow{a_2} X_3$	a_3
4	$X_0 \xrightarrow{a_0} Y_1 \xrightarrow{b_1} Z_2 \xrightarrow{a_2} X_3 \xrightarrow{a_3} Y_4$	b_4
5	$X_0 \xrightarrow{a_0} Y_1 \xrightarrow{b_1} Z_2 \xrightarrow{a_2} X_3 \xrightarrow{a_3} Y_4 \xrightarrow{b_4} Y_5$	b_5
6	$X_0 \xrightarrow{a_0} Y_1 \xrightarrow{b_1} Z_2 \xrightarrow{a_2} X_3 \xrightarrow{a_3} Y_4 \xrightarrow{b_4} Y_5 \xrightarrow{b_5} Z_6$	b_6
7	$X_0 \xrightarrow{a_0} Y_1 \xrightarrow{b_1} Z_2 \xrightarrow{a_2} X_3 \xrightarrow{a_3} Y_4 \xrightarrow{b_4} Y_5 \xrightarrow{b_5} Z_6 \xrightarrow{b_6} X_7$	b_7
$\vdots$	$\vdots$	$\vdots$
100	$X_0 \xrightarrow{a_0} Y_1 \xrightarrow{b_1} Z_2 \xrightarrow{a_2} X_3 \xrightarrow{a_3} Y_4 \xrightarrow{b_4} Y_5 \xrightarrow{b_5} Z_6 \xrightarrow{b_6} X_7 \xrightarrow{b_7}$ $\ldots \xrightarrow{Ac_{99}} perception_{100}$	Ac_{100}

We can build a case base from the *sub-runs* comprising the example run by the XYZ agent from Section 4.2, using the same procedure from [20]. Each sub-run is the subset of the example run up to and including the environmental state perceived at a given time step. Thus, the cases generated by the example run are shown in Table 4.1.

Case 0 corresponds to the sub-run up to time step 0. The run up to and including the environmental state at time step 0 is just X_0; this comprises the *problem* component of the case. The action taken in that time step (time step 0) is a_0; this comprises the *solution* component of the case. Case 1 corresponds to the sub-run up to time step 1. The run up to and including the environmental state at time step 1 is $X_0 \xrightarrow{a_0} Y_1$; this is the problem component of case 1. The action taken in time step 1 is b_1; this is the solution component of case 1. The rest of the cases in the case base created from the example run of the XYZ agent are similarly defined.

However, it is possible for a case base to contain cases generated from *multiple* traces. For example, suppose that our XYZ agent generates two separate traces, with different seeds for the random number generator that decides what action to take when at state Y. Suppose those two runs, up to time step 4, are as follows:

1. $X_0 \xrightarrow{a_0} Y_1 \xrightarrow{b_1} Z_2 \xrightarrow{a_2} X_3 \xrightarrow{a_3} Y_4$

2. $X_0 \xrightarrow{a_0} Y_1 \xrightarrow{b_1} Y_2 \xrightarrow{b_2} Z_3 \xrightarrow{a_3} X_4$

To differentiate between the traces from which each run element comes from (either trace 1 or trace 2), we ascribe a superscript for each run element, denoting the trace of origin. This convention is our own invention and it is applied as follows:

1. $X_0^1 \xrightarrow{a_0^1} Y_1^1 \xrightarrow{b_1^1} Z_2^1 \xrightarrow{a_2^1} X_3^1 \xrightarrow{a_3^1} Y_4^1$

2. $X_0^2 \xrightarrow{a_0^2} Y_1^2 \xrightarrow{b_1^2} Y_2^2 \xrightarrow{b_2^2} Z_3^2 \xrightarrow{a_3^2} X_4^2$

If we create a case base from these two runs, it would that shown in Table 4.2. In this case base, cases 0-3 correspond to trace 1 and cases 4-7 correspond to trace 2. Cases must have unique identifiers.

Table 4.2: XYZ Agent Case Base from Two Complete Runs

Case Number	Problem	Solution
0	X_0^1	a_0^1
1	$X_0^1 \xrightarrow{a_0^1} Y_1^1$	b_1^1
2	$X_0^1 \xrightarrow{a_0^1} Y_1^1 \xrightarrow{b_1^1} Z_2^1$	a_2^1
3	$X_0^1 \xrightarrow{a_0^1} Y_1^1 \xrightarrow{b_1^1} Z_2^1 \xrightarrow{a_2^1} X_3^1$	a_3^1
4	X_0^2	a_0^2
5	$X_0^2 \xrightarrow{a_0^2} Y_1^2$	b_1^2
6	$X_0^2 \xrightarrow{a_0^2} Y_1^2 \xrightarrow{b_1^2} Y_2^2$	b_2^2
7	$X_0^2 \xrightarrow{a_0^2} Y_1^2 \xrightarrow{b_1^2} Y_2^2 \xrightarrow{b_2^2} Z_3^2$	a_3^2

With the above terminology pertaining to agents, runs, and CBR in place, we are now ready to describe Temporal Backtracking, as designed by Floyd [20].

4.4 Temporal Backtracking

Temporal Backtracking (TB) is a CBR approach for learning memory-influenced behaviors [20]. A modified version of TB is used within our MCL algorithm, so a detailed discussion of TB as it was originally set forth in [20] is warranted. The modifications to TB in our research are *not* described in this section. This section only describes TB as it was *originally designed*; our modifications to TB for use in our research are described in Chapter 5.

TB relies on the CBR framework described in Section 4.3. TB agents use the CBR approach of case retrieval and case reuse to solve the problem of choosing the next action to take within a *test run* in progress for a standard agent (described in Section 4.1). This is shown in Algorithm 1.

Algorithm 1 Temporal Backtracking

1: **procedure** GET-NEXT-ACTION(test-run, case-base, time-step)
2:     next-action ← *state-retrieve*(test-run, case-base, time-step, 0)
3:     **return** next-action
4: **procedure** STATE-RETRIEVE(test-run, case-pool, time-step, time-back)
5:     candidate-actions ← (empty set)
6:     candidate-cases ← (empty set)
7:     test-percep ← test-run.*percep-at*(time-step − time-back)
8:     **for** each *case* in case-pool **do**
9:         case-percep ← case.*percep-at*(time-step − time-back)
10:         similiarity ← *percep-sim*(test-percep, case-percep)
11:         **if** similarity > THRESHOLD **then**
12:             candidate-cases.*add*(case)
13:             **if** *not* candidate-actions.*contains*(case.action) **then**
14:                 candidate-actions.*add*(case.action)
15:     **if** *count*(candidate-actions) *is* 0 **then return** random-action
16:     **if** *count*(candidate-actions) *is* 1 **then return** candidate-actions[0]
17:     **return** `action-retrieve`(test-run, candidate-cases, time-step, time-back − 1)
18: **procedure** ACTION-RETRIEVE(test-run, case-pool, time-step, time-back)
19:     candidate-actions ← (empty set)
20:     candidate-cases ← (empty set)
21:     test-action ← test-run.*action-at*(time-step − time-back)
22:     **for** each *case* in case-pool **do**
23:         case-action ← case.*action-at*(time-step − time-back)
24:         **if** case-action == test-action **then**
25:             candidate-cases.*add*(case)
26:             **if** *not* candidate-actions.*contains*(case.action) **then**
27:                 candidate-actions.*add*(case.action)
28:     **if** *count*(candidate-actions) *is* 0 **then return** random-action
29:     **if** *count*(candidate-actions) *is* 1 **then return** candidate-actions[0]
30:     **return** `state-retrieve`(test-run, candidate-cases, time-step, time-back)
31: **procedure** PERCEP-SIM(percep1, percep2)
32:     sim ← 0
33:     **for** feature in percep1 **do**
34:         p1 *gets* percep1.*value-of*(feature)
35:         p2 *gets* percep2.*value-of*(feature)
36:         sim ← sim + $(1 - abs(p1 - p2)/(p1 + p2))$
37:     **return** sim

Basically, TB works as follows:

1. Invoke procedure `get-next-action` and pass it the test run in progress, the case base, and the current time step.

2. Set the pool of *candidate cases* or "case-pool" to all cases in the case base. Set the value of "time-back" to zero (this tracks how many time steps back in the test run TB is looking at during recursion).

3. Do the following until all cases in the case-pool agree on the next action (e.g. their solution components are all the same). Each repetition of this step is one *iteration* of the TB algorithm.

   (a) Invoke procedure `state-retrieve`. Do the following:

      i. Let τ be the last time step in the test run. Set "test-percep" to the *state* at time $\tau -$ time-back in the test run.

      ii. Compare all cases in the case pool to the test run. For each case, do the following:

         A. Let γ be the last time step in the case's problem component run. Set "case-percep" to the *state* for this run at time step $\gamma -$ time-back.

         B. Compare "case-percep" and "test-percep" with procedure `percep-sim`. If their similarity exceeds a predefined threshold, retain the case in the "case-pool", otherwise discard it.

      iii. If the solution component of all remaining cases in the case pool is the same, return the action for this solution component and go to Step 4.

   (b) Increment "time-back" and invoke procedure `action-retrieve`. Do the following:

      i. Let τ be the last time step in the test run. Set "test-action" to the *action* at time $\tau -$ time-back in the test run.

ii. Compare all cases in the case pool to the test run. For each case, do the following:

    A. Let γ be the last time step in the case's problem component run. Set "case-action" to the *action* for this run at time step $\gamma -$ time-back.

    B. Compare "case-action" and "test-action". If they are the same, retain the case in the "case-pool", otherwise discard it.

iii. If the solution component of all remaining cases in the case pool is the same, return the action for this solution component and go to Step 4.

4. The procedure `get-next-action` will return the next action for the agent to take.

We now proceed with an example from the XYZ domain (described in Section 4.2) to show how TB works. Suppose we want to use TB to control an agent that is supposed to behave like the XYZ agent. Suppose that our TB agent has the following *test run T* in progress, where "???" means that the next action to take is unknown.

$$\texttt{test run T:} \ldots \xrightarrow{b^T_{t-5}} Z^T_{t-4} \xrightarrow{b^T_{t-4}} X^T_{t-3} \xrightarrow{b^T_{t-3}} Y^T_{t-2} \xrightarrow{b^T_{t-2}} Y^T_{t-1} \xrightarrow{b^T_{t-1}} Z^T_t \xrightarrow{???}$$

As described in Section 4.2, each element of the run has a subscript denoting the time step it belongs to and a superscript denoting its trace of origin. The run is currently at time step t; the correct action to take at time step t in the above run is unknown. The trace of origin is T, which is currently being implemented by the test run in progress.

TB will require a case base constructed from observations of the XYZ agent's interactions with the environment. Suppose we have the case base shown in Table 4.3. For simplicity, let us assume that all cases in the case base come from the same trace of origin, trace C (C for "case base"). Trace C has at most T time steps and each case contains the sub-run of the trace up to some time t. In other words, for every time step, from $t = 0$ to $t = T$, there is a case in the case base containing

the sub-run of the trace up to that value t. Therefore, the value of t is different for each case. For example, t could be 30 for case 0, 15 for case 1, 17 for case 2, etc. The important thing is that t is the last time step for the sub-run contained within any given case.

Table 4.3: Temporal Backtracking XYZ Case Base

Case Number	Problem	Solution
0	$\ldots \xrightarrow{a_{t-1}^C} X_t^C$	a_t^C
1	$\ldots \xrightarrow{b_{t-1}^C} X_t^C$	b_t^C
2	$\ldots \xrightarrow{a_{t-1}^C} Y_t^C$	b_t^C
3	$\ldots \xrightarrow{b_{t-1}^C} Y_t^C$	b_t^C
4	$\ldots \xrightarrow{b_{t-5}^C} Z_{t-4}^C \xrightarrow{b_{t-4}^C} X_{t-3}^C \xrightarrow{b_{t-3}^C} Y_{t-2}^C \xrightarrow{b_{t-2}^C} Y_{t-1}^C \xrightarrow{b_{t-1}^C} Z_t^C$	a_t^C
5	$\ldots \xrightarrow{b_{t-5}^C} Z_{t-4}^C \xrightarrow{a_{t-4}^C} X_{t-3}^C \xrightarrow{a_{t-3}^C} Y_{t-2}^C \xrightarrow{b_{t-2}^C} Y_{t-1}^C \xrightarrow{b_{t-1}^C} Z_t^C$	b_t^C
6	$\ldots \xrightarrow{b_{t-5}^C} X_{t-4}^C \xrightarrow{b_{t-4}^C} Y_{t-3}^C \xrightarrow{b_{t-3}^C} Y_{t-2}^C \xrightarrow{b_{t-2}^C} Y_{t-1}^C \xrightarrow{b_{t-1}^C} Z_t^C$	a_t^C
7	$\ldots \xrightarrow{a_{t-5}^C} X_{t-4}^C \xrightarrow{a_{t-4}^C} Y_{t-3}^C \xrightarrow{b_{t-3}^C} Y_{t-2}^C \xrightarrow{b_{t-2}^C} Y_{t-1}^C \xrightarrow{b_{t-1}^C} Z_t^C$	b_t^C
8	$\ldots \xrightarrow{a_{t-5}^C} Y_{t-4}^C \xrightarrow{b_{t-4}^C} Z_{t-3}^C \xrightarrow{b_{t-3}^C} X_{t-2}^C \xrightarrow{b_{t-2}^C} Y_{t-1}^C \xrightarrow{b_{t-1}^C} Z_t^C$	a_t^C
9	$\ldots \xrightarrow{b_{t-5}^C} Y_{t-4}^C \xrightarrow{b_{t-4}^C} Z_{t-3}^C \xrightarrow{a_{t-3}^C} X_{t-2}^C \xrightarrow{a_{t-2}^C} Y_{t-1}^C \xrightarrow{b_{t-1}^C} Z_t^C$	b_t^C
$\vdots$	$\vdots$	$\vdots$

In Table 4.3, we only show ten cases. There are potentially many other cases that would resemble the ten shown in the final elements of their problem components. They are treated like the cases they resemble during the TB algorithm, but we ignore them for simplicity in this example.

The rules governing the behavior of the XYZ agent are *not* known to the TB agent (because this is the behavior it must emulate solely from observations structured within a case base). However, for the benefit of the reader, the rules described in Section 4.2 for dictating the behavior of the XYZ agent are repeated below.

1. If the agent perceives environmental state X, choose the action that was taken one time step ago. Choose action a by default if it's the first time step.

2. If the agent perceived environmental state Y, choose action b.

3. If the agent perceives environmental state Z, choose the opposite of the action taken when the environment was previously in state Z. So, if the agent chose action a when it previously perceived environmental state Z, then it should choose action b; and if the agent chose action b when it previously perceived environmental state Z, then it should choose action a. Choose action a by default if this is the first time that environmental state Z has been perceived by the agent.

Therefore, we know from rule 3 that the next action that should be executed in the test run is action a, if the TB agent is to truly behave like the XYZ agent.

$$\texttt{test run T:} \ldots \xrightarrow{b^T_{t-5}} Z^T_{t-4} \xrightarrow{b^T_{t-4}} X^T_{t-3} \xrightarrow{b^T_{t-3}} Y^T_{t-2} \xrightarrow{b^T_{t-2}} Y^T_{t-1} \xrightarrow{b^T_{t-1}} Z^T_t \xrightarrow{\mathbf{a^T_t}}$$

We know this because the last time step when the TB agent perceived environmental state Z in the test run was at time step $t-4$ and the action taken in that time step was action b; thus, the TB agent must take the opposite action (action a) when it perceives environmental state Z again. However, the TB agent does *not* know the behavior rules governing the XYZ agent, so the TB agent must deduce the next action to take using its case base; we now show how TB does this.

In the *first* iteration of TB, all the cases' problem components are compared to the test run. Specifically, the perception at time step t for each case is compared to the perception in the test run at time step t. (Recall that the value of t is different for the test run and for each case.) In the test run, the perception at time step t is environmental state Z. TB will select those cases with the same

perception at time step t.

$$\texttt{test run T:}\ldots \xrightarrow{b^T_{t-5}} Z^T_{t-4} \xrightarrow{b^T_{t-4}} X^T_{t-3} \xrightarrow{b^T_{t-3}} Y^T_{t-2} \xrightarrow{b^T_{t-2}} Y^T_{t-1} \xrightarrow{b^T_{t-1}} \mathbf{Z^T_t} \xrightarrow{???}$$

Table 4.4: Temporal Backtracking XYZ Example - Iteration 1

Case Number	Problem	Solution
0	$[\ldots \xrightarrow{a^C_{t-1}} \mathbf{X^C_t}]$	a^C_t
1	$[\ldots \xrightarrow{b^C_{t-1}} \mathbf{X^C_t}]$	b^C_t
2	$[\ldots \xrightarrow{a^C_{t-1}} \mathbf{Y^C_t}]$	b^C_t
3	$[\ldots \xrightarrow{b^C_{t-1}} \mathbf{Y^C_t}]$	b^C_t
4	$\ldots \xrightarrow{b^C_{t-5}} Z^C_{t-4} \xrightarrow{b^C_{t-4}} X^C_{t-3} \xrightarrow{b^C_{t-3}} Y^C_{t-2} \xrightarrow{b^C_{t-2}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} \mathbf{Z^C_t}$	a^C_t
5	$\ldots \xrightarrow{b^C_{t-5}} Z^C_{t-4} \xrightarrow{a^C_{t-4}} X^C_{t-3} \xrightarrow{a^C_{t-3}} Y^C_{t-2} \xrightarrow{b^C_{t-2}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} \mathbf{Z^C_t}$	b^C_t
6	$\ldots \xrightarrow{b^C_{t-5}} X^C_{t-4} \xrightarrow{b^C_{t-4}} Y^C_{t-3} \xrightarrow{b^C_{t-3}} Y^C_{t-2} \xrightarrow{b^C_{t-2}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} \mathbf{Z^C_t}$	a^C_t
7	$\ldots \xrightarrow{a^C_{t-5}} X^C_{t-4} \xrightarrow{a^C_{t-4}} Y^C_{t-3} \xrightarrow{b^C_{t-3}} Y^C_{t-2} \xrightarrow{b^C_{t-2}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} \mathbf{Z^C_t}$	b^C_t
8	$\ldots \xrightarrow{a^C_{t-5}} Y^C_{t-4} \xrightarrow{b^C_{t-4}} Z^C_{t-3} \xrightarrow{b^C_{t-3}} X^C_{t-2} \xrightarrow{b^C_{t-2}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} \mathbf{Z^C_t}$	a^C_t
9	$\ldots \xrightarrow{b^C_{t-5}} Y^C_{t-4} \xrightarrow{b^C_{t-4}} Z^C_{t-3} \xrightarrow{a^C_{t-3}} X^C_{t-2} \xrightarrow{a^C_{t-2}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} \mathbf{Z^C_t}$	b^C_t

In our example, the run elements at time t for each case are bolded in Table 4.4. All the cases from the case base whose problem components are sub-runs ending with perceptions of environmental state Z are selected and the rest are discarded. Therefore, cases 0 and 1 are discarded because their sub-runs end in environmental state X. Cases 2 and 3 are discarded because their sub-runs end in environmental state Y. The discarded cases are surrounded by square brackets in Table 4.4; the cases that were not discarded constitute the pool of remaining candidate cases.

TB then checks to determine whether all the remaining candidate cases' solution components

agree. If they do agree, then that agreed-upon solution is returned. In our example, they do not agree. Cases 4, 6, and 8 have action a as their solution, but cases 5, 7, and 9 have action b as their solution. Therefore, TB must go back in time by one time step and compare the *actions* of each case at time $t - 1$ to that of the test run.

Table 4.5: Temporal Backtracking XYZ Example - Iteration 2

Case Number	Problem	Solution
4	$\ldots \xrightarrow{b_{t-5}^C} Z_{t-4}^C \xrightarrow{b_{t-4}^C} X_{t-3}^C \xrightarrow{b_{t-3}^C} Y_{t-2}^C \xrightarrow{b_{t-2}^C} Y_{t-1}^C \xrightarrow{\mathbf{b_{t-1}^C}} Z_t^C$	a_t^C
5	$\ldots \xrightarrow{b_{t-5}^C} Z_{t-4}^C \xrightarrow{a_{t-4}^C} X_{t-3}^C \xrightarrow{a_{t-3}^C} Y_{t-2}^C \xrightarrow{b_{t-2}^C} Y_{t-1}^C \xrightarrow{\mathbf{b_{t-1}^C}} Z_t^C$	b_t^C
6	$\ldots \xrightarrow{b_{t-5}^C} X_{t-4}^C \xrightarrow{b_{t-4}^C} Y_{t-3}^C \xrightarrow{b_{t-3}^C} Y_{t-2}^C \xrightarrow{b_{t-2}^C} Y_{t-1}^C \xrightarrow{\mathbf{b_{t-1}^C}} Z_t^C$	a_t^C
7	$\ldots \xrightarrow{a_{t-5}^C} X_{t-4}^C \xrightarrow{a_{t-4}^C} Y_{t-3}^C \xrightarrow{b_{t-3}^C} Y_{t-2}^C \xrightarrow{b_{t-2}^C} Y_{t-1}^C \xrightarrow{\mathbf{b_{t-1}^C}} Z_t^C$	b_t^C
8	$\ldots \xrightarrow{a_{t-5}^C} Y_{t-4}^C \xrightarrow{b_{t-4}^C} Z_{t-3}^C \xrightarrow{b_{t-3}^C} X_{t-2}^C \xrightarrow{b_{t-2}^C} Y_{t-1}^C \xrightarrow{\mathbf{b_{t-1}^C}} Z_t^C$	a_t^C
9	$\ldots \xrightarrow{b_{t-5}^C} Y_{t-4}^C \xrightarrow{b_{t-4}^C} Z_{t-3}^C \xrightarrow{a_{t-3}^C} X_{t-2}^C \xrightarrow{a_{t-2}^C} Y_{t-1}^C \xrightarrow{\mathbf{b_{t-1}^C}} Z_t^C$	b_t^C

In the *second* iteration of TB, the *action* at time step $t - 1$ of each case in the pool of surviving cases is compared to the action of the test run at time step $t - 1$. These actions are bolded in Table 4.5. The cases that have the same action as the test run at time step $t - 1$ will be selected. The action at time step $t - 1$ in the test run is action b.

$$\texttt{test run T:} \ldots \xrightarrow{b_{t-5}^T} Z_{t-4}^T \xrightarrow{b_{t-4}^T} X_{t-3}^T \xrightarrow{b_{t-3}^T} Y_{t-2}^T \xrightarrow{b_{t-2}^T} Y_{t-1}^T \xrightarrow{\mathbf{b_{t-1}^T}} Z_t^T \xrightarrow{???}$$

Those cases with action b at time step $t - 1$ are retained within the group of candidate cases and the rest are discarded. At this iteration, no cases are rejected. However, the solution components of the candidate cases are still not all the same; cases 4, 6, and 8 have action a as their solution and cases 5, 7, and 9 have action b as their solution. Therefore, TB will next examine the perception at

time step $t-1$ for each case as it compares to the perception at time step $t-1$ in the test run.

Table 4.6: Temporal Backtracking XYZ Example - Iteration 3

Case Number	Problem	Solution
4	$\ldots \xrightarrow{b^C_{t-5}} Z^C_{t-4} \xrightarrow{b^C_{t-4}} X^C_{t-3} \xrightarrow{b^C_{t-3}} Y^C_{t-2} \xrightarrow{b^C_{t-2}} \mathbf{Y^C_{t-1}} \xrightarrow{b^C_{t-1}} Z^C_t$	a^C_t
5	$\ldots \xrightarrow{b^C_{t-5}} Z^C_{t-4} \xrightarrow{a^C_{t-4}} X^C_{t-3} \xrightarrow{a^C_{t-3}} Y^C_{t-2} \xrightarrow{b^C_{t-2}} \mathbf{Y^C_{t-1}} \xrightarrow{b^C_{t-1}} Z^C_t$	b^C_t
6	$\ldots \xrightarrow{b^C_{t-5}} X^C_{t-4} \xrightarrow{b^C_{t-4}} Y^C_{t-3} \xrightarrow{b^C_{t-3}} Y^C_{t-2} \xrightarrow{b^C_{t-2}} \mathbf{Y^C_{t-1}} \xrightarrow{b^C_{t-1}} Z^C_t$	a^C_t
7	$\ldots \xrightarrow{a^C_{t-5}} X^C_{t-4} \xrightarrow{a^C_{t-4}} Y^C_{t-3} \xrightarrow{b^C_{t-3}} Y^C_{t-2} \xrightarrow{b^C_{t-2}} \mathbf{Y^C_{t-1}} \xrightarrow{b^C_{t-1}} Z^C_t$	b^C_t
8	$\ldots \xrightarrow{a^C_{t-5}} Y^C_{t-4} \xrightarrow{b^C_{t-4}} Z^C_{t-3} \xrightarrow{b^C_{t-3}} X^C_{t-2} \xrightarrow{b^C_{t-2}} \mathbf{Y^C_{t-1}} \xrightarrow{b^C_{t-1}} Z^C_t$	a^C_t
9	$\ldots \xrightarrow{b^C_{t-5}} Y^C_{t-4} \xrightarrow{b^C_{t-4}} Z^C_{t-3} \xrightarrow{a^C_{t-3}} X^C_{t-2} \xrightarrow{a^C_{t-2}} \mathbf{Y^C_{t-1}} \xrightarrow{b^C_{t-1}} Z^C_t$	b^C_t

Table 4.7: Temporal Backtracking XYZ Example - Iteration 4

Case Number	Problem	Solution
4	$\ldots \xrightarrow{b^C_{t-5}} Z^C_{t-4} \xrightarrow{b^C_{t-4}} X^C_{t-3} \xrightarrow{b^C_{t-3}} Y^C_{t-2} \xrightarrow{\mathbf{b^C_{t-2}}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} Z^C_t$	a^C_t
5	$\ldots \xrightarrow{b^C_{t-5}} Z^C_{t-4} \xrightarrow{a^C_{t-4}} X^C_{t-3} \xrightarrow{a^C_{t-3}} Y^C_{t-2} \xrightarrow{\mathbf{b^C_{t-2}}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} Z^C_t$	b^C_t
6	$\ldots \xrightarrow{b^C_{t-5}} X^C_{t-4} \xrightarrow{b^C_{t-4}} Y^C_{t-3} \xrightarrow{b^C_{t-3}} Y^C_{t-2} \xrightarrow{\mathbf{b^C_{t-2}}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} Z^C_t$	a^C_t
7	$\ldots \xrightarrow{a^C_{t-5}} X^C_{t-4} \xrightarrow{a^C_{t-4}} Y^C_{t-3} \xrightarrow{b^C_{t-3}} Y^C_{t-2} \xrightarrow{\mathbf{b^C_{t-2}}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} Z^C_t$	b^C_t
8	$\ldots \xrightarrow{a^C_{t-5}} Y^C_{t-4} \xrightarrow{b^C_{t-4}} Z^C_{t-3} \xrightarrow{b^C_{t-3}} X^C_{t-2} \xrightarrow{\mathbf{b^C_{t-2}}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} Z^C_t$	a^C_t
9	$[\ldots \xrightarrow{b^C_{t-5}} Y^C_{t-4} \xrightarrow{b^C_{t-4}} Z^C_{t-3} \xrightarrow{a^C_{t-3}} X^C_{t-2} \xrightarrow{\mathbf{a^C_{t-2}}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} Z^C_t]$	b^C_t

In the *third* iteration of TB, the perception at time step $t-1$ in the test run is environmental state Y.

$$\texttt{test run T:} \ldots \xrightarrow{b^T_{t-5}} Z^T_{t-4} \xrightarrow{b^T_{t-4}} X^T_{t-3} \xrightarrow{b^T_{t-3}} Y^T_{t-2} \xrightarrow{b^T_{t-2}} \mathbf{Y^T_{t-1}} \xrightarrow{b^T_{t-1}} Z^T_t \xrightarrow{???}$$

The candidate cases in Table 4.6 with perceptions of environmental state Y at time step $t-1$ are

retrieved. In our example, all candidate cases are retained; their solutions still do not agree, so TB will examine the cases' *actions* at time step $t - 2$ in its next iteration.

In the *fourth* iteration of TB, the action at time step $t - 2$ in the test run is action b.

$$\texttt{test run T:} \ldots \xrightarrow{b^T_{t-5}} Z^T_{t-4} \xrightarrow{b^T_{t-4}} X^T_{t-3} \xrightarrow{b^T_{t-3}} Y^T_{t-2} \xrightarrow{\mathbf{b^T_{t-2}}} Y^T_{t-1} \xrightarrow{b^T_{t-1}} Z^T_t \xrightarrow{???}$$

Those cases in Table 4.7 with action b at time step $t - 2$ are selected as the new set of candidate cases and the rest are discarded. Case 9 is discarded (and surrounded by square brackets in Table 4.7) because the action at time step $t - 2$ of its sub-run is action a, not action b. The remaining cases (cases 4-8) are shown in Table 4.7 without square brackets; their solutions still do not match, so TB will next compare the *perception* of each case at time step $t - 2$ to that of the test run.

Table 4.8: Temporal Backtracking XYZ Example - Iteration 5

Case Number	Problem	Solution
4	$\ldots \xrightarrow{b^C_{t-5}} Z^C_{t-4} \xrightarrow{b^C_{t-4}} X^C_{t-3} \xrightarrow{b^C_{t-3}} \mathbf{Y^C_{t-2}} \xrightarrow{b^C_{t-2}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} Z^C_t$	a^C_t
5	$\ldots \xrightarrow{b^C_{t-5}} Z^C_{t-4} \xrightarrow{a^C_{t-4}} X^C_{t-3} \xrightarrow{a^C_{t-3}} \mathbf{Y^C_{t-2}} \xrightarrow{b^C_{t-2}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} Z^C_t$	b^C_t
6	$\ldots \xrightarrow{b^C_{t-5}} X^C_{t-4} \xrightarrow{b^C_{t-4}} Y^C_{t-3} \xrightarrow{b^C_{t-3}} \mathbf{Y^C_{t-2}} \xrightarrow{b^C_{t-2}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} Z^C_t$	a^C_t
7	$\ldots \xrightarrow{a^C_{t-5}} X^C_{t-4} \xrightarrow{a^C_{t-4}} Y^C_{t-3} \xrightarrow{b^C_{t-3}} \mathbf{Y^C_{t-2}} \xrightarrow{b^C_{t-2}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} Z^C_t$	b^C_t
8	$[\ldots \xrightarrow{a^C_{t-5}} Y^C_{t-4} \xrightarrow{b^C_{t-4}} Z^C_{t-3} \xrightarrow{b^C_{t-3}} \mathbf{X^C_{t-2}} \xrightarrow{b^C_{t-2}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} Z^C_t]$	a^C_t

In the *fifth* iteration of TB, the perception at time step $t - 2$ in the test run is environmental state Y.

$$\texttt{test run T:} \ldots \xrightarrow{b^T_{t-5}} Z^T_{t-4} \xrightarrow{b^T_{t-4}} X^T_{t-3} \xrightarrow{b^T_{t-3}} \mathbf{Y^T_{t-2}} \xrightarrow{b^T_{t-2}} Y^T_{t-1} \xrightarrow{b^T_{t-1}} Z^T_t \xrightarrow{???}$$

The cases in Table 4.8 with perceptions of environmental state Y at time step $t - 2$ are kept and the rest are discarded. Case 8 is discarded because the environmental state perceived in its sub-run

85

at time step $t - 2$ is X. The solutions of the remaining cases (cases 4-7) still disagree, so TB will next compare the *action* of each case at time step $t - 3$ to that of the test run at time step $t - 3$.

Table 4.9: Temporal Backtracking XYZ Example - Iteration 6

Case Number	Problem	Solution
4	$\ldots \xrightarrow{b^C_{t-5}} Z^C_{t-4} \xrightarrow{b^C_{t-4}} X^C_{t-3} \xrightarrow{\mathbf{b^C_{t-3}}} Y^C_{t-2} \xrightarrow{b^C_{t-2}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} Z^C_t$	a^C_t
5	$[\ldots \xrightarrow{b^C_{t-5}} Z^C_{t-4} \xrightarrow{a^C_{t-4}} X^C_{t-3} \xrightarrow{\mathbf{a^C_{t-3}}} Y^C_{t-2} \xrightarrow{b^C_{t-2}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} Z^C_t]$	b^C_t
6	$\ldots \xrightarrow{b^C_{t-5}} X^C_{t-4} \xrightarrow{b^C_{t-4}} Y^C_{t-3} \xrightarrow{\mathbf{b^C_{t-3}}} Y^C_{t-2} \xrightarrow{b^C_{t-2}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} Z^C_t$	a^C_t
7	$\ldots \xrightarrow{a^C_{t-5}} X^C_{t-4} \xrightarrow{a^C_{t-4}} Y^C_{t-3} \xrightarrow{\mathbf{b^C_{t-3}}} Y^C_{t-2} \xrightarrow{b^C_{t-2}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} Z^C_t$	b^C_t

In the *sixth* iteration of TB, the action at time step $t - 3$ of the test run is action b.

$$\texttt{test run T:} \ldots \xrightarrow{b^T_{t-5}} Z^T_{t-4} \xrightarrow{b^T_{t-4}} X^T_{t-3} \xrightarrow{\mathbf{b^T_{t-3}}} Y^T_{t-2} \xrightarrow{b^T_{t-2}} Y^T_{t-1} \xrightarrow{b^T_{t-1}} Z^T_t \xrightarrow{???}$$

Those cases in Table 4.9 with action b at time step $t - 3$ in their sub-runs are kept and the rest are discarded. In this iteration of our example, case 5 is discarded because the action at time step $t - 3$ of its sub-run is action a instead of action b like in the test run. The remaining cases are cases 4, 6, and 7. Their solutions still do not agree; cases 4 and 6 have action a for their solution, but case 7 has action b as its solution. Therefore, TB will next compare the *perception* of each case at time step $t - 3$ to that of the test run at time step $t - 3$.

In the *seventh* iteration of TB, the perception of the test run at time step $t - 3$ is environmental state X.

$$\texttt{test run T:} \ldots \xrightarrow{b^T_{t-5}} Z^T_{t-4} \xrightarrow{b^T_{t-4}} \mathbf{X^T_{t-3}} \xrightarrow{b^T_{t-3}} Y^T_{t-2} \xrightarrow{b^T_{t-2}} Y^T_{t-1} \xrightarrow{b^T_{t-1}} Z^T_t \xrightarrow{???}$$

The cases in Table 4.10 with environmental state X as the perception at time step $t - 3$ of their sub-runs are kept and the rest are rejected. In this case, cases 6 and 7 are rejected because the

86

environmental state perceived at time step $t - 3$ of their sub-runs is Y, instead of X as in the test run. Therefore, the only remaining case is case 4. Therefore, its solution, action a, is returned by TB and the recursion stops.

Table 4.10: Temporal Backtracking XYZ Example - Iteration 7

Case Number	Problem	Solution
4	$\dots \xrightarrow{b^C_{t-5}} Z^C_{t-4} \xrightarrow{b^C_{t-4}} \mathbf{X^C_{t-3}} \xrightarrow{b^C_{t-3}} Y^C_{t-2} \xrightarrow{b^C_{t-2}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} Z^C_t$	a^C_t
6	$[\dots \xrightarrow{b^C_{t-5}} X^C_{t-4} \xrightarrow{b^C_{t-4}} \mathbf{Y^C_{t-3}} \xrightarrow{b^C_{t-3}} Y^C_{t-2} \xrightarrow{b^C_{t-2}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} Z^C_t]$	a^C_t
7	$[\dots \xrightarrow{a^C_{t-5}} X^C_{t-4} \xrightarrow{a^C_{t-4}} \mathbf{Y^C_{t-3}} \xrightarrow{b^C_{t-3}} Y^C_{t-2} \xrightarrow{b^C_{t-2}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} Z^C_t]$	b^C_t

Thus, TB has (correctly) determined that the next action to take in the test run is action a.

$$\texttt{test run T:} \dots \xrightarrow{b^T_{t-5}} Z^T_{t-4} \xrightarrow{b^T_{t-4}} X^T_{t-3} \xrightarrow{b^T_{t-3}} Y^T_{t-2} \xrightarrow{b^T_{t-2}} Y^T_{t-1} \xrightarrow{b^T_{t-1}} Z^T_t \xrightarrow{\mathbf{a^T_t}}$$

Originally designed by Floyd [20] and described in this section, TB can replicate memory-influenced behaviors. Its limitation is that it operates like a standard agent (defined in Section 4.1). Thus, it does not try to explicitly model an observed entity's internal state as in a state-based agent (defined in Section 4.1). As a result, it must travel back in time recursively for potentially every decision. As the case base grows in size and the complexity of memory influence increases (potentially increasing the distance in the past a memory influence exists), agent deployment becomes increasingly costly.

This is an important justification for MCL in our research here. A model of the internal state of an observed entity that is *learned*, and not *predefined*, is more accessible than raw memory influences that must be recursively retrieved every time step by the agent, and it can be used by learning algorithms that create state-based agents. Therefore, TB in isolation cannot accomplish

the objective of this research as stated by the dissertation hypothesis.

In the next subsection, we describe Memory Features. Memory Features are one of the innovations introduced in our research and are the intended output of the MCL algorithm.

4.5   Memory Features

Memory Features (MFs) (also memory "attributes" or "variables") capture some information about the past. MFs are the output of the MCL algorithm and a core component of our research. MFs are the formalized codification of memory influence that we have invented within the context of LfO. The success of MCL is completely dependent on the proper definition and discovery of MFs from observable behavior, and on their integration into the original traces to complement the features already there describing the perception of the environment. First, we describe a simple letter sequence example showing why MFs are essential for capturing influences of memory in our research. Then, we describe our MF formalization.

4.5.1   Memory Features - Conceptual Description

Table 4.11: Simple Letter Sequence

Time	0	1	2	3	4	5	6	7	8	9	10	11	12	...
Letter	A	B	B	A	B	B	A	B	B	A	B	B	A	...

Suppose that we have a repeating sequence of letters, A and B, as seen in Table 4.11. Here, we can see that the subsequence ABB repeats itself. We can also represent this sequence as a run, per

the conventions from Section 4.1.

$$A_0 \rightarrow B_1 \rightarrow B_2 \rightarrow A_3 \rightarrow B_4 \rightarrow B_5 \rightarrow A_6 \rightarrow B_7 \rightarrow B_8 \rightarrow A_9 \rightarrow B_{10} \rightarrow B_{11} \rightarrow A_{12} \rightarrow \ldots$$

Here, we assume that the action taken, that would normally be written over each arrow in the sequence, is either "Go to A" or "Go to B". The finite state machine shown in Figure 4.2 dictates the effects of these actions. The environmental state is sufficient for knowing which action by the agent was taken to bring about that state, so we omit the action label from our runs because it is implicit.
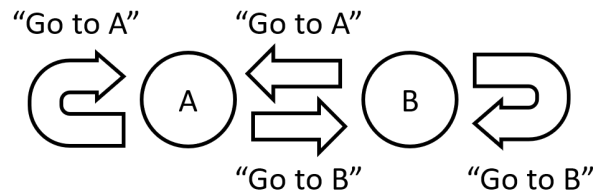
Figure 4.2: Finite state machine for ABB example

Now, if we were told that the current letter is $\underline{B}$ and if we were asked to determine which letter comes after it, we could not do so with certainty.

$$\cdots \rightarrow \underline{B} \rightarrow ??$$

This is because there are instances when B comes after $\underline{B}$ and other instances when A comes after $\underline{B}$. However, if we were told what the previous letter was, then we could answer the question with absolute confidence.

$$\cdots \rightarrow B \rightarrow \underline{B} \rightarrow ??$$

89

$$\cdots \rightarrow A \rightarrow \underline{B} \rightarrow ??$$

If the letter before $\underline{B}$ is B, then the next letter is A. If the letter before $\underline{B}$ is A, then the next letter is B.

$$\cdots \rightarrow B \rightarrow \underline{B} \rightarrow \mathbf{A}$$

$$\ldots, A \rightarrow \underline{B} \rightarrow \mathbf{B}$$

The conventional description of an observed performer's *run*, provided by [27], represents an observed behavior as a sequence of *interactions* of an actor with its environment. Each interaction consists of a *perception* of the environment and an *action* executed in response by the actor. However, this representation of an actor's run does not include the internal state of the actor. The actor's internal state cannot be observed by an external observer, but it plays a vital role in the actor's decisions. The hypothesis of our research is that the internal state of an observed actor, specifically the component of internal state that is influenced by memory of past observable interactions of the observed actor with its environment, can be approximated and modeled by a preprocessing technique and then used by an agent that learns to emulate the observed actor's behavior. Therefore, we present a modified representation of a run that explicitly represents the internal state of an *agent* as it changes over time. We call a run that accounts for internal state an *internal run* and we call a run that does *not* account for internal state an *observed run*. We note here that our definition of an internal run only accounts for the actor's memory of past events. We do not account for other components of an actor's behavior that may affect his or her internal state, such as motivation, agenda, goals, or emotional state. Such components of internal state and their influences on behavior are beyond the scope of our research and are explicitly left for future research.

In addition to the finite set $E = \{e_0, e_1, \ldots, e_n\}$ of possible *states* that the environment may assume and the finite set $A = \{a_0, a_1, \ldots, a_n\}$ of possible *actions* that an agent may execute in response,

our notation includes the finite set I of possible *internal state instances* that an agent's singular internal state may assume:

$$I = \{i_0, i_1, \ldots, i_n\}$$

It is worth noting that for the purposes of our research, an agent has only *one* internal state, but that internal state can assume one of several *values* or *instances* at any given time. Also, I is finite because every run has a beginning and therefore every run up to a given time step t is finite. Thus, the number of configurations for an internal state based on memory of a finite run is itself finite.

An *internal run* is represented as a sequence of *interactions* of the agent with its environment, each of which consists of an internal state instance, a perception of the environmental state, and an action taken in response. In a state-based agent, the juxtaposition of the internal state instance and the perception of the environment within a given interaction is called the agent's *awareness*. From that awareness, the agent chooses an *action* to do in response (even if that action is to do nothing). Again, this convention is our own invention. An internal run may be represented as:

$$(i_0, e_0) \xrightarrow{a_0} (i_1, e_1) \xrightarrow{a_1} (i_2, e_2) \xrightarrow{a_2} \ldots \xrightarrow{a_{n-1}} (i_n, e_n)$$

An alternative (more compact) notation is:

$$(i, e)_0 \xrightarrow{a_0} (i, e)_1 \xrightarrow{a_1} (i, e)_2 \xrightarrow{a_2} \ldots \xrightarrow{a_{n-1}} (i, e)_n$$

The *internal run* representation is more detailed than the representational convention of a run presented in [27] and described in Section 4.1. This is because an internal run accounts for the change in an agent's internal state over time. However, an internal run is only accessible by the agent itself because only an agent has access to its own internal state. All external observers of a performing entity's behavior have access to the *observed* run, but the performing entity is the only

one with access to its *internal* run. Our research tries to *approximate* the internal state (based on memory) within an internal run of an observed entity, but the *actual* internal state is inaccessible to us, the observers.

For one last clarification, we return to the car driving example from Chapter 1. In the example where a car driver on an interstate highway passes a sign indicating a gas station at the next exit and then takes the exit, the *observed run* (known by anyone watching) could be represented by the following sequence:

- Car on interstate passes sign indicating a gas station at the next exit.

- Car continues driving on the interstate.

- Car changes to the right-most lane.

- Car takes the exit.

In contrast, the *internal run* (known only to the driver) could be represented by the following sequence:

- *Observe*: Car passes sign for gas station. *Internally*: *[Okay, gas station coming up.]*

- *Observe*: Car drives on interstate. *Internally*: *[Oh, hold on, I need gas! Where can I get gas? Oh, the sign before... gas at next exit.]*

- *Observe*: Car changes to right lane. *Internally*: *[Need to get off interstate to get gas.]*

- *Observe*: Car exits interstate. *Internally*: *[Now, off to the gas station.]*

If the observer happened to be inside of the car with the driver and could also observe the level of the fuel gauge (which would be at a low level throughout this example), then the observer could

92

also associate a low fuel gauge reading with memory of the gas station sign when learning to proceed to an exit when gas is needed.

We now proceed to convert the following *observed* run from our "ABB" example

$$A_0 \rightarrow B_1 \rightarrow B_2 \rightarrow A_3 \rightarrow B_4 \rightarrow B_5 \rightarrow A_6 \rightarrow B_7 \rightarrow B_8 \rightarrow A_9 \rightarrow B_{10} \rightarrow B_{11} \rightarrow A_{12} \rightarrow \ldots$$

into the format of an internal run. We have established that it is possible to determine the next letter in the observed run if we know the current letter and the letter that came before it. Therefore, the current letter comprises the *perception* and the letter before it comprises the *internal state instance*, both of which make up the *awareness* within an internal run. The first awareness in our internal run is thus composed of the first letter in the observed run, A, as the perception, and $null$ as the internal state instance (because there are no previous letters). The internal run is now:

$$(null, \mathbf{A})_0$$

The next letter (perception) in the observed run is B. The action taken in the first time step to get there is "Go to B". (Again, we omit action labels from the run because the action is intuitive and implicit.) Within the internal run, the internal state instance is the letter of the previous time step, which is A. The run is now:

$$(null, \mathbf{A})_0 \rightarrow (A, \mathbf{B})_1$$

The letter (perception) in the third time step of the observed run is again B. In the internal run, in addition to the perception, the awareness is composed of the letter in the previous time step, which is B. The run is now:

$$(null, \mathbf{A})_0 \rightarrow (A, \mathbf{B})_1 \rightarrow (B, \mathbf{B})_2$$

In the fourth time step of the observed run, the perception is A. In the corresponding time step

of the internal run, the perception is also A and the internal state instance is the letter from the previous time step, which is B. The run is now:

$$(null, \mathbf{A})_0 \rightarrow (A, \mathbf{B})_1 \rightarrow (B, \mathbf{B})_2 \rightarrow (B, \mathbf{A})_3$$

The rest of the internal run corresponding to our observed run is instantiated in a similar fashion, using the same procedure we have illustrated:

$$(null, \mathbf{A})_0 \rightarrow (A, \mathbf{B})_1 \rightarrow (B, \mathbf{B})_2 \rightarrow (B, \mathbf{A})_3 \rightarrow (A, \mathbf{B})_4 \rightarrow (B, \mathbf{B})_5 \rightarrow (B, \mathbf{A})_6 \ldots$$

Now, given an awareness in the internal run in our "ABB" example, we can deduce the next perception (i.e. letter) in the observed run. For example, if the latest awareness in the internal run is (B, B), then the next awareness in that internal run will be (B, A). This means that the next perception in the corresponding position in the observed run is simply A (the perception component of the awareness (B, A)). We can make such a deduction because there is a now a *one-to-one* mapping between each awareness in the internal run and the appropriate next perception (letter) in the observed run, as seen in Table 4.12. For example, from the second entry (row) in Table 4.12, if the current awareness in our internal run is (A, B), then we know with certainty that the next awareness in the internal run should be (B, B). This means that the next perception in the observed run will be the perception component of the next awareness (B, B), which is B.

The internal run thus *Markovian* [86] - at a given time step in the internal run, the information needed to determine the *next* awareness in the internal run is entirely contained within the awareness of the *current* time step and absolutely no information is needed from any prior awarenesses in the internal run.

Table 4.12: One-to-One Mapping From Internal Run Awareness to Next Observed Run Perception

Entry	Current Awareness (Internal Run)	Next Awareness (Internal Run)	Next Perception (Observed Run)
1	$(null, \text{A})$	$\rightarrow (A, \text{B})$	$\rightarrow \text{B}$
2	(A, B)	$\rightarrow (B, \text{B})$	$\rightarrow \text{B}$
3	(B, B)	$\rightarrow (B, \text{A})$	$\rightarrow \text{A}$
4	(B, A)	$\rightarrow (A, \text{B})$	$\rightarrow \text{B}$

In our "ABB" example, each awareness in the internal run contained both a perception and an internal state instance that consisted of a single *memory feature* (MF), where the MF stored the value of the previous time step's letter (perception). The purpose of MFs are therefore to make internal runs Markovian. For the "ABB" example, the observed run is not Markovian because information from previous time steps is required to determine the next action ("Go to A" or "Go to B") needed to institute the next desired letter, but the internal run is Markovian because the current awareness has all the information necessary to produce the next time step's awareness contents. The set of MFs necessary to make an internal run Markovian thus comprises the *internal state I* that is used by state-based agents, described in Section 4.1. It is the job of the MCL algorithm to learn which MFs can help produce the internal state instances within each awareness of an internal run that would make the internal run Markovian. We now describe these MFs formally.

4.5.2 Memory Features - Formal Description

There are two kinds of MFs in our research: *value-back* MFs and *time-back* MFs. Value-back MFs store the *value μ* of a *feature f* at time $t - K$, where K is some positive constant less than t. Time-back MFs store the *number of time steps σ* prior to a given time step t when a specific *value*

V of a given *feature f* was last observed.

Up until this point, within a run we have only considered each *perception* as a whole, but not the individual components or *features* that comprise it. All of our examples until now have only used perceptions that had just one feature, so it is important for us to note here that a perception can have several features that have varying ramifications for memory-influenced behaviors. For example, "speed" and "steer angle" may be two possible features in a car-driving domain. In our "ABB" example, our only feature was "Letter". Temporal Backtracking [20] [27] considers each feature as equally important in determining a case's relevance to a test run in progress (as seen in the `Percep-Sim` function in Algorithm 1 in Section 4.4). However, our research decomposes these perceptions to investigate the significance to memory of each individual *feature*. MFs can also track the *action* taken at some prior time step.

For a given behavior of an agent, let the finite set of all features F that are contained in an agent's perception and action sets be:

$$F = \{f_0, f_1, \ldots, f_n\}$$

Let the finite set of all values V that may be assumed by a feature in the agent's perception be:

$$V = \{v_0, v_1, \ldots, v_n\}$$

Let the set T be the set of all numbers in the range [1, N], where N is the maximum number of time steps that an agent's run may have.

$$T = \{1, 2, \ldots, N\}$$

For a given time step value $t \in T$, let K be the set of positive integers less than or equal to t.

$$K = \{1, 2, \ldots, t - 1\}, t \in T$$

Therefore, a MF, like an agent, is a conversion from some cross product of its input parameters to the space of its output parameter. The value-back MF, MF^{VB}, is parameterized by a feature f, a time step t, and a constant k, and it stores the value v of f at time step $t - k$. The value-back MF can be represented as:

$$MF^{VB} : F \times K \times T \rightarrow V$$

The time-back MF, MF^{TB}, is parameterized by a feature f, a time step t, and a value v of f, and it stores the number of time steps k *before* time step t when value v was last observed for feature f. The time-back MF can be represented as:

$$MF^{TB} : F \times V \times T \rightarrow K$$

It is important to note that if value v occurs for feature f at time step t, a time-back MF tracking v for f will *not* return the value $k = 0$, because all values from time step t comprise the *perception* at time step t and not *memory* of *past* events — such a value cannot affect the internal state instance at time step t. Instead, the time-back MF will return the number of time steps *before* time step t when v last occurred.

An example of a value-back MF would be the MF we used in the "ABB" example, the *value* of the feature *letter* at time $t - k$, where $k = 1$. We represent this MF as:

```
vLetter(1)
```

In the general case, a value-back MF is represented as:

$$vFeature(k)$$

We exclude the time step t from the naming convention of a value-back MF because its value is expected to change over time, similar to how a perception feature (e.g. "Throttle" in a car driving domain) will change over time.

Table 4.13 shows the value of the value-back MF `vLetter(1)` for each time step of the "ABB" run from our example. This corresponds to the same letter sequence shown in Table 4.11. At time step 0, the value of the feature *Letter* is A. The value-back MF `vLetter(1)` stores the value of feature *Letter* at time $t - 1$. At time step 0, $t - 1 = -1$, which is less than 0, and so the MF stores "null" (which also bears the same meaning as "N/A", "nothing", or "-"). At time step 1, the value of feature *Letter* is B. The value-back MF `vLetter(1)` stores the value of *Letter* at time step $t - 1 = 0$; thus, the MF stores "A" because that was the value of *Letter* at the previous time step, time step 0.

Table 4.13: Value-Back MF for ABB Example

Time	0	1	2	3	4	5	6	...
`vLetter(1)`	null	A	B	B	A	B	B	...
Letter	A	B	B	A	B	B	A	...

The contents of Table 4.13 can be represented in the format of an internal run (as described in Section 4.5.1), where each awareness within an interaction is of the form (*vLetter(1)*, Letter). This internal representation is:

$$(null, \text{A})_0 \rightarrow (A, \text{B})_1 \rightarrow (B, \text{B})_2 \rightarrow (B, \text{A})_3 \rightarrow (A, \text{B})_4 \rightarrow (B, \text{B})_5 \rightarrow (B, \text{A})_6 \rightarrow \ldots$$

98

An example of a time-back MF would be an MF that stores the number of time steps k *before* time step t when the value A for feature *Letter* was last observed in our example "ABB" letter sequence. We represent this MF as:

$$\texttt{tLetter(A)}$$

In the general case, a time-back MF is represented as:

$$\texttt{tFeature(value)}$$

For the same reason we excluded the time step t from the naming convention for the value-back MF, we also do so for the time-back MF. The `t` prefix stands for *time-back*.

Table 4.14: Time-Back MF for ABB Example

Time	0	1	2	3	4	5	6	...
tLetter(A)	null	1	2	3	1	2	3	...
Letter	A	B	B	A	B	B	A	...

Table 4.14 shows the value of the time-back MF `tLetter(A)` for each time step of the "ABB" run from our example. This corresponds to the same letter sequence shown in Table 4.11. At time step 0, the value of the feature *Letter* is A. The time-back MF `tLetter(A)` stores the number of time steps before time step t when the value of feature *Letter* was A. There are no time steps before time step 0, and so the MF stores "null".

At time step 1, the value of feature *Letter* is B. The time-back MF `tLetter(A)` stores the value $k = 1$, because the last observation of value A for feature *Letter* was at time step 0, which comes $k = 1$ time steps before the current time step, time step 1.

At time step 2, the value of feature *Letter* is B. The last observation of value A for feature *Letter* is still time step 0, which comes two time steps before the current time step, time step 2. Therefore, `tLetter(A)` stores the value $k = 2$.

At time step 3, the value of feature *Letter* is A. The last observation of value A for feature *Letter* before the current time step $t = 3$ is time step 0. So, `tLetter(A)` stores the value $k = 3$.

The contents of Table 4.14 can be represented in the format of an internal run, where each awareness within an interaction is of the form (*tLetter(A)*, Letter). This internal run representation is:

$$(null, A)_0 \rightarrow (1, B)_1 \rightarrow (2, B)_2 \rightarrow (3, A)_3 \rightarrow (1, B)_4 \rightarrow (2, B)_5 \rightarrow (3, A)_6 \rightarrow \ldots$$

The power of MFs lies in their ability to capture knowledge from the past and augment an internal run to make it Markovian. If the appropriate MFs can be learned by MCL via analysis of a trace of an observed behavior, then an internal state approximation for the observed behavior composed of these MFs can be constructed for this purpose.

4.6   Behavioral Representations Summary

In this chapter, we described terminology and topics that are important for understanding the implementation of the MCL algorithm. We discussed the various agents that can be created through learning algorithms. We then described the XYZ domain and a memory-influenced behavior by one agent within this domain, which will be addressed in Chapter 5. This was followed by a discussion of the terminology behind a Case-Based Reasoning (CBR) framework that facilitates learning memory-influenced behaviors. Next, we described the Temporal Backtracking (TB) algorithm, as originally designed by Floyd [20], and provided an example of it operating within the XYZ do-

main. Finally, we presented our novel Memory Features (MFs), the intended output of MCL. With these concepts in place, we now present a minimal conceptual description of the MCL algorithm.

CHAPTER 5: BASELINE CONCEPTUAL APPROACH

In this chapter, we introduce our initial or "baseline" conceptual approach to the problem described in Chapter 3. Here, we describe the Memory Composition Learning (MCL) algorithm; this chapter covers a preliminary version of MCL that is unencumbered by improvements made in the advanced version described in Chapter 6. We call this baseline approach "MCLv0". MCLv0 was used to verify the basic concepts of MCL in a simple vacuum cleaner domain before enhancing the approach through other useful concepts.

This chapter is organized as follows:

- Section 5.1 presents the high-level view of the overall MCL algorithm and which aspects of the algorithm constitute the MCLv0 approach.

- Section 5.2 describes the first phase of MCLv0, Memory Feature Extraction.

- Section 5.3 describes the second phase of MCLv0, Memory Feature Refinement.

- Section 5.4 describes the third phase of MCLv0, Trace Enhancement.

- Section 5.5 describes our assessment of MCLv0 in a simulated vacuum cleaner domain.

5.1 High Level Conceptual Approach

In this section, we describe the Memory Composition Learning (MCL) algorithm, our conceptual approach to the problem defined in Chapter 3. We furthermore show which aspects of this algorithm constitute our baseline conceptual approach, MCLv0.

The purpose of MCL is to learn a set of features that encapsulate the pertinent information from memory of past events that have significance for decision-making in an observed behavior. We accomplish this in three phases, as seen in Figure 5.1.
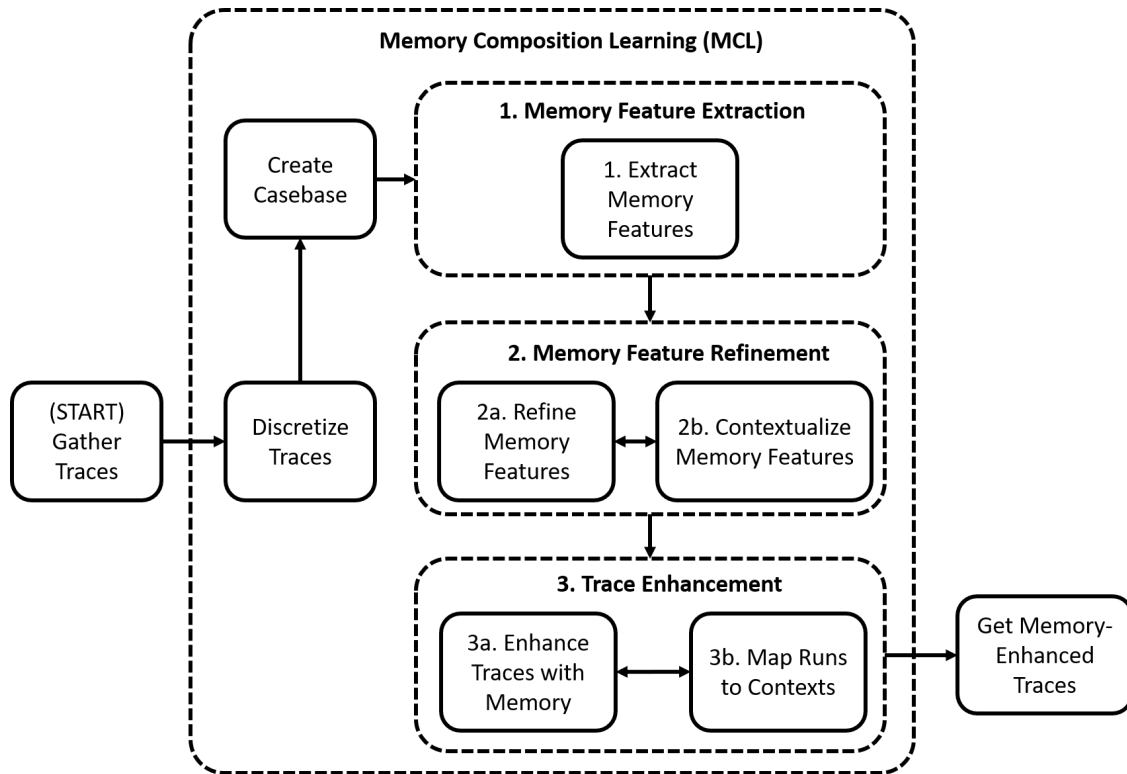
Figure 5.1: MCL Full Approach

MCL can be viewed as a pipeline that takes as input several traces (described in Section 4.1) that have been converted into a CBR case base (see Section 4.3). The three main steps in the MCL approach are:

1. Extract atomic MFs for each case in the case base.

2. Refine the extracted MFs into a minimal subset that applies to all cases.

3. Enhance the traces with memory by incorporating the refined MFs into the traces themselves.

In Figure 5.1, the first step is to gather raw traces that record the observed behavior from which we want to learn. If the traces encode continuous-valued data, they would need to first be converted into discrete-valued traces. These traces are then converted into a case base that is structured according to the conventions outlined in Section 4.3. Then, the operations of our MCL approach begin.

In the first phase, *Memory Feature Extraction*, the objective is to learn the appropriate MFs that make the problem component sub-run for each case conform to the Markov assumption. This phase operates on the micro-level; each case is addressed individually. In Step 1 "Extract Memory Features", the memory influences at work for a given case are expressed as MFs (see Section 4.5). The end result of this phase is that each case will have its own set of extracted MFs.

In the second phase, *Memory Feature Refinement*, the objective is to reduce or *refine* the many sets of MFs extracted for each case into just *one* refined set of MFs that applies to *all cases*. This phase operates on the macro-level; all cases must be taken into account simultaneously. This occurs in Step 2a "Refine Memory Features". This also provides the opportunity to see which groups of MFs frequently occurred together; from this information, various memory-based *contexts* may be generated in Step 2b "Contextualize Memory Features".

In the third phase, *Trace Enhancement*, the objective is to compute the contents of memory (the values of the MFs) for each time step in the original traces. This creates new *memory-enhanced traces*, which are the ultimate output of MCL. This occurs in Step 3a "Enhance Traces with Memory". If contexts were generated in the Memory Feature Refinement phase, then portions of each trace that map to the generated contexts will be written to file in Step 3b "Map Runs to Contexts".

Steps 1, 2a, and 3a are absolutely necessary for MCL to work. These constitute the *minimal MCL approach*, MCLv0, which is described in this chapter. The addition of steps 1b, 2b, and 3b along with the appropriate modification of steps 1, 2a, and 3a constitutes a *comprehensive approach*,

which is described Chapter 6.

Figure 5.2 depicts the transformation of memoryless traces into memory-enhanced traces as they are processed through the MCLv0 pipeline. At the start of the pipeline for MCLv0, *Gather Raw Traces*, we obtain one or more traces that recorded an observed entity's behavior. As described in Section 4.1, a trace is a record of an observed run that cannot be expanded or continued because that run has reached a terminating state. In other words, a trace is a record of what an observed entity did in one scenario, from start to finish. Each trace is represented graphically within Figure 5.2 as a sequence of alternating perception-action pairs, where a *perception* is a group of three squares, and an *action* is a circle. Each square in the perception is a single *feature* that captures one aspect of the environment at a given time step. After the observed entity perceives the environment, it executes an action in response, which then changes the environment. Upon perceiving the updated environment, the entity will choose an action to take in response, and the cycle continues. (Note: There is no particular reason why there are three squares — three is just an arbitrary number because each perception tracks three features in this example.)

In the second module of the pipeline in Figure 5.2, *Create Case Base*, a case base is formed from all the memoryless traces. Each rectangle in the stack is a single case that contains a problem component (the run from one trace up to a given time step's perception of the environment) and a solution component, the action which the observed entity took at that point of the observed run. This formulation of the case base, as was described in Section 4.3, is necessary to execute a modified version of the Temporal Backtracking algorithm [20], which is executed in the third module of the MCLv0 pipeline.
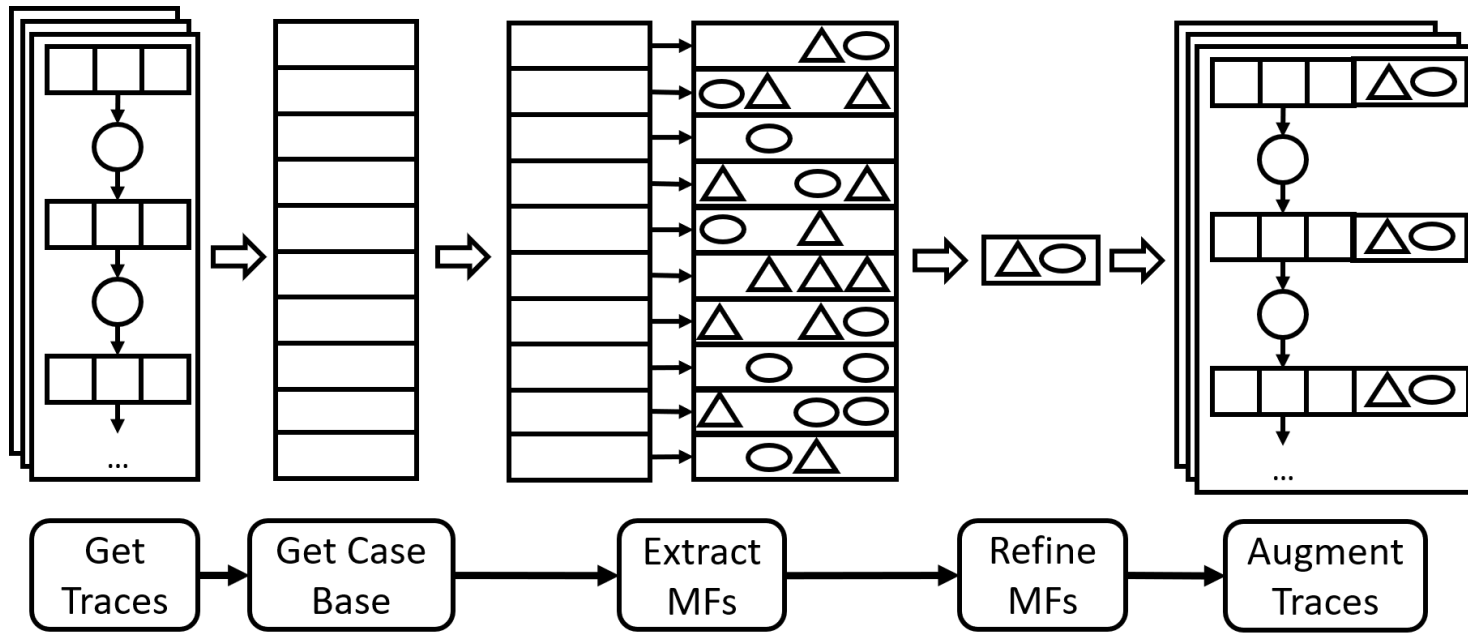
Figure 5.2: MCLv0 Pipeline

In this third module of the pipeline, *Extract Memory Features*, a list of value-back and time-back MFs (represented as ovals and triangles, respectively, in Figure 5.2) is extracted for each case via our version of the Temporal Backtracking algorithm called *Memory Temporal Backtracking* (MemTB). MemTB is applied to each case in the case base to get a list of MFs that converts the observed sub-run within the case into a Markovian internal sub-run. However, because each case is handled independently of the other cases, many extraneous MFs may be extracted during this process.

The fourth module of the MCLv0 pipeline, *Refine Memory Features*, refines the list of MFs extracted from each case in the case base and determines which MFs are common across all the cases. In some cases, certain MFs may be redundant and can be discarded. In other cases, the extracted MFs are the result of artifacts in the traces instead of being indicative of memory influence on the behavior as a whole. Thus, the set of extracted MFs is reduced to a minimal *refined* set of MFs that is applicable to the majority of cases in the case base.

In the fifth module of the MCLv0 pipeline, *Enhance Traces with Memory*, the refined MFs are incorporated into the memoryless traces to produce a new set of memory-enhanced traces. The value of each MF is computed for every time step of each trace. The MFs can then be treated like the originally included trace features by a machine learning algorithm.

The problem that MCL seeks to address is formally defined as follows:

- *Input*: A set of traces that record the observed behavior of an actor.

- *Output*: A set of *memory-enhanced* traces that record values for MFs that summarize relevant memory influences at each time step.

In other words, if an observed behavior requires information from memory in order to explain the

actions taken by the observed actor, then MCL learns which MFs express those memory influences and then computes their values for each time step of the observed behavior in order to create memory-enhanced traces.

Algorithm 2 Memory Composition Learning

---

    procedure MEM-COMP-LEARN(traces)
        casebase ← `create-casebase`(traces)
        mem-feats ← (empty set)
        for each *case* in casebase **do**
            tmp-mem-feats ← `extract-mem-feats`(*case*, casebase, *case*.last-time-step)
            mem-feats.`add`(tmp-mem-feats)
        min-mem-feat-set ← `refine-mem-feats`(traces, casebase, mem-feats)
        for each *trace* in traces **do**
            `enhance-trace`(*trace*, min-mem-feat-set)

The MCL algorithm (see Algorithm 2), as it has been implemented in the MCLv0 approach, works as follows:

- Given a set of traces, structure the information within as a case base, as described in Section 4.3.

- Once a set of traces have been converted into a case base (see Section 4.3), a set of MFs is *extracted* for each individual case via the `extract-mem-feats` procedure.

- All of these MFs are appended to a large cumulative list of MFs, which is then *refined* in the `refine-mem-feats` procedure to reduce the MF list to just a minimal subset that has the most utility across the entire case base.

- Finally, the final MF set is used to enhance each of the original traces in the `enhance-trace` procedure.

Each of the major steps will be discussed at length in the following sections and clarified with our running example from the XYZ domain.

5.2   Memory Feature Extraction

The Memory Feature Extraction step of the MCL algorithm analyzes each case in the case base and extracts a list of MFs for each case that would make the case's problem component sub-run Markovian. First, we provide an algorithmic description of the Memory Feature Extraction step, then we clarify it with our ongoing example in the XYZ domain.

5.2.1   *Memory Feature Extraction Algorithm*

For each case in the case base, a call is made to the procedure `extract-mem-feats` in Algorithm 2. The Memory Feature Extraction algorithm invoked by this procedure is shown in Algorithm 3. This procedure uses a modified version of the Temporal Backtracking (TB) algorithm (the original version was described in Section 4.4). We call our modified TB version *Memory Temporal Backtracking* or just "MemTB". Unlike the original TB, which gets the next *action* to take at a given time step in a test run in progress, MemTB will get a MF that codifies the most recent memory influence on a case's problem sub-run that resolves the proper action to take next in the sub-run. By invoking MemTB multiple times and modifying the case's sub-run each time to account for the extracted memory influences, a list of MFs that maximally reduces a case's dependence on memory can be otained. The objective is to find the MFs that will convert the case's problem component test-run into an internal run that is Markovian.

Algorithm 3 Memory Feature Extraction

1: **procedure** EXTRACT-MEM-FEATS(test-run, case-base, time-step)
2:     mem-feats ← (empty set)
3:     **while** True **do**
4:         last-mf, call-depth ← `mem-tb`(test-run, case-base, time-step)
5:         **if** call-depth $== 0$ **then**
6:             **break**
7:         vb-test-run, vb-case-base ← `add-val-back-mf`(last-mf, test-run, case-base)
8:         tb-test-run, tb-case-base ← `add-time-back-mf`(last-mf, test-run, case-base)
9:         temp-mf, vb-call-depth ← `mem-tb`(vb-test-run, vb-case-base, time-step)
10:        temp-mf, tb-call-depth ← `mem-tb`(tb-test-run, tb-case-base, time-step)
11:        **if** vb-call-depth $\leq$ tb-call-depth **then**
12:            test-run, case-base ← `add-val-back-mf`(last-mf, test-run, case-base)
13:            mem-feats.*add*(val-back-mf)
14:        **if** tb-call-depth $\leq$ vb-call-depth **then**
15:            test-run, case-base ← `add-time-back-mf`(last-mf, test-run, case-base)
16:            mem-feats.*add*(time-back-mf)
17:     **return** mem-feats

The procedure `extract-mem-feats` is invoked for a given *test case* for which MFs are extracted. This procedure is called once for every case in the case base; every case in the case base gets a chance to be the *test case*. The procedure `extract-mem-feats` has four major steps that it iterates through until all memory influences for the current test case have been obtained in the form of MFs:

1. Use MemTB to extract one *raw* MF for the test run. The "test run" is the problem component sub-run in the *test case* for which MCL is extracting MFs. A *raw* MF is a MF that has not been designated as either value-back or time-back. Record how many MemTB iterations were required. Note, MemTB is a recursive function.

2. Convert the raw MF into a value-back MF. Temporarily add this MF to the internal test run. Run MemTB and record how many MemTB iterations are required.

3. Convert the raw MF into a time-back MF. Temporarily add this MF to the internal test run.

Run MemTB and record how many MemTB iterations are required.

4. Convert the raw MF into the MF type that has the greatest reduction in MemTB iterations. Permanently add it to the internal test run. If the number of TB iterations was greater than one, return to Step 1. Otherwise, terminate.

To clarify, the procedure `extract-mem-feats` will iterate through the above steps until the memory influences for a given test case have been captured in the form of MFs. However, within a given iteration, three calls to the recursive function MemTB are made, the result of each is the extraction of a single raw MF, which may or may not be converted into a specialized (value-back or time-back) MF.

The procedure `mem-tb` is shown in Algorithm 4. It consists of an initial call to the recursive procedure `Mem-State-Retrieve`, shown in Algorithm 5. Similar to how the original TB algorithm alternates between recursive functions `State-Retrieve` and `Action-Retrieve` until all cases agree on an action to take, MemTB will recursively alternate between `Mem-State-Retrieve` (Algorithm 5) and `Mem-Action-Retrieve` (Algorithm 6) until all retained cases agree upon the next action to take. The algorithmic design of MF Extraction in MCLv0 is a simple and intuitive one that is meant to validate the basic mechanism of extracting MFs. However, the reader may notice that the algorithmic design for MF Extraction in MCLv0 is computationally inefficient. This limitation is addressed in Chapter 6.

Algorithm 4 Memory Temporal Backtracking

 1: **procedure** MEM-TB(test-run, case-base, time-step)
 2:     **return** `mem-state-retrieve`(test-run, case-base, time-step, 0, 1)

Algorithm 5 Memory State Retrieval

1: **procedure** MEM-STATE-RETRIEVE(test-run, case-pool, time-step, time-back, call-depth)
2:    candidate-actions ← (empty set)
3:    candidate-cases ← (empty set)
4:    test-percep ← test-run.*percep-at*(time-step − time-back)
5:    **for** each *case* in case-pool **do**
6:        case-percep ← case.*percep-at*(time-step − time-back)
7:        similiarity ← *percep-sim*(test-percep, case-percep)
8:        **if** similarity >= THRESHOLD **then**
9:            candidate-cases.*add*(case)
10:           **if** *not* candidate-actions.*contains*(case.action) **then**
11:               candidate-actions.*add*(case.action)
12:    **if** *count*(candidate-actions) *is* 0 **then**
13:        **return** *null*, call-depth   ▷ Memory feature can't be determined
14:    **if** *count*(candidate-actions) *is* 1 **then**
15:        raw-mem-feat ← `get-raw-mem-feat`(test-run, candidate-cases, case-pool, time-step, time-back)
16:        **return** raw-mem-feat, call-depth   ▷ Return raw memory feature
17:    **return** `mem-action-retrieve`(test-run, candidate-cases, time-step, time-back − 1, call-depth + 1)   ▷ Recursion

---

Algorithm 6 Memory Action Retrieval

1: **procedure** MEM-ACTION-RETRIEVE(test-run, case-pool, time-step, time-back, call-depth)
2:    candidate-actions ← (empty set)
3:    candidate-cases ← (empty set)
4:    test-action ← test-run.*action-at*(time-step − time-back)
5:    **for** each *case* in case-pool **do**
6:        case-action ← case.*action-at*(time-step − time-back)
7:        **if** case-action == test-action **then**
8:            candidate-cases.*add*(case)
9:            **if** *not* candidate-actions.*contains*(case.action) **then**
10:               candidate-actions.*add*(case.action)
11:    **if** *count*(candidate-actions) *is* 0 **then return** *null*, call-depth   ▷ Memory feature can't be determined
12:    **if** *count*(candidate-actions) *is* 1 **then return** *new* **MemFeat**("Action", test-action, time-back), call-depth   ▷ Return raw memory feature
13:    **return** `mem-state-retrieve`(test-run, candidate-cases, time-step, time-back, call-depth + 1)   ▷ Recursion

---

If a raw MF is created in the procedure `Mem-State-Retrieve`, the raw MF can only track one of the perception features in the perception. The original TB algorithm treats all features in the perception equally and only cares about the similarity of a case's problem component sub-run's perception *as a whole*, but MemTB can only create a raw MF from one of the features in the perception. Therefore, a heuristic for determining the most *influential* perception feature is used in the procedure `Get-Raw-Mem-Feat` (Algorithm 7). The heuristic is composed of the following steps.

1. Create two lists, *candidate-cases* and *reject-cases*. The former contains cases with a solution component matching that of the *test-run* (the problem component sub-run of the test case). The latter contains cases with different solution components.

2. For each feature in the perception of the *test-run*, compute the similarity of the corresponding feature of each case in the *candidate-cases* list. Do the same for each case in the *reject-cases* list. A case's similarity score for a feature is in the range [0.0, 1.0], where a higher score indicates higher similarity.

3. Given the feature similarities for candidate cases and rejected cases, a feature's *influence* score is equal to the average candidate case feature similarity plus one minus the average reject case feature similarity. The assumption behind this computation is that an influential feature should yield high similarity scores for candidate cases and low similarity scores for rejected cases. Future work could investigate more nuanced metrics for finding the appropriate feature for the raw MF

4. Create a raw MF that tracks the feature with the highest influence score and stores the value of that feature and the number of time steps back in time when that value was taken.

Algorithm 7 Raw Memory Feature Computation

1: **procedure** GET-RAW-MEM-FEAT(test-run, candidate-cases, case-pool, time-step, time-back)
2:     reject-cases ← (empty-set)
3:     candidate-feat-scores ← (empty-set)
4:     reject-feat-scores ← (empty-set)
5:     all-features ← case-pool[0].*features*
6:     test-percep ← test-run.*percep-at*(time-step − time-back)
7:     **for** *case* in case-pool **do**
8:         **if** *not* case-pool.contains(*case*) **then**
9:             reject-cases.add(*case*)
10:     **for** *feat* in all-features **do**
11:         candidate-feat-scores.add(0.0)
12:         reject-feat-scores.add(0.0)
13:     **for** *feat* in all-features **do**
14:         **for** *cand-case* in candidate-cases **do**
15:             case-percep ← cand-case.percep-at(time-step - time-back)
16:             temp-val ← case-percep.value−of(*feat*)
17:             old-val ← candidate-feat-scores.value−of(*feat*)
18:             candidate-feat-scores.value−of(*feat*) ← old-val + temp-val
19:         **for** *rej-case* in reject-cases **do**
20:             case-percep ← rej-case.percep-at(time-step - time-back)
21:             temp-val ← case-percep.value−of(*feat*)
22:             old-vale ← reject-feat-scores.value−of(*feat*)
23:             reject-feat-scores.value−of(*feat*) ← old-val + temp-val
24:         old-val ← candidate-feat-scores.value−of(*feat*)
25:         candidate-feat-scores.value−of(*feat*) ← old-val / candidate-feat-scores.size()
26:         old-val ← reject-feat-scores.value−of(*feat*)
27:         reject-feat-scores.value−of(*feat*) ← old-val / reject-feat-scores.size()
28:     max-score ← 0.0
29:     best-feat ← null
30:     **for** *feat* in all-features **do**
31:         good-score ← candidate-feat-scores.value−of(*feat*)
32:         bad-score ← reject-feat-scores.value−of(*feat*)
33:         tmp-score ← good-score + (1 - bad-score)
34:         **if** tmp-score > max-score **then**
35:             max-score ← tmp-score
36:             best-feat ← *feat*
37:     feat-val ← test-percep.value−of(best-feat)
38:     **return MemFeat**(best-feat, feat-val, time-back)

It is important to note that MCL does *not* learn how to emulate the behavior of an observed performer. That is done by a machine learning algorithm later. TB, as originally designed, was meant to learn memory-influenced behavior; however, as we discussed in Section 4.4, TB is insufficient by itself to capture the influence of memory explicitly. This limitation inspired our research. In contrast to TB, which retrieves the next *action* to take in a sub-run in progress, MemTB retrieves the most discriminative memory influence used to separate cases in a case base that are sufficiently similar to a given sub-run in progress from those cases that are not sufficiently similar. This distinction and its consequences are further discussed next.

5.2.2 *Memory Temporal Backtracking Analysis*

This section briefly outlines the commonalities and differences between TB [20] and MemTB. They are the same in their implementation of case comparison and backtracking elements. They differ in purpose, in input parameters, and return output. We discuss the elements that TB and MemTB share in common, then we discuss how they are different.

First, we define what a *test run* is. In the context of TB, a test run is the performance of a TB-controlled agent in progress, for which TB seeks to determine the next action that should be taken by the agent. In the context of MemTB (within MCLv0), the test run is the problem sub-run component of the specific case in the case base for which MCL is extracting MFs. In TB, the test run has never been seen before (e.g. it is not necessarily reflected exactly in the case base). In MemTB, the test run comes from one of the cases in the case base because MemTB is used to find memory influences for each case in the case base.

Regarding case comparisons, TB and MemTB both rely on two assumptions:

1. A test-run and a case's sub-run should be compared temporal element by temporal element.

115

Given a time step t^T for the test run and t^C for the case's sub-run, an element from k time steps prior to t^T in the test run can only be compared to the corresponding element k time steps prior to t^C in the case's sub-run.

2. The procedure for comparing perceptions or actions from two different runs is the same for both TB and MemTB. For perceptions with multiple features, each feature contributes equally to the similarity score of two perceptions.

This means that when comparing two runs, both TB and MemTB will compare the same elements and compare them in the same manner.

Regarding backtracking, TB and MemTB both rely on the assumption, as stated in [27], that the most recent events of a run in progress are the most relevant to making the next decision. This is evidenced in the manner by which both TB and MemTB backtrack, which is defined by these steps:

1. Set $k = 0$ and include all cases in the case base within the initial candidate cases pool.

2. Repeat the following until $t - k = 0$ for all candidate cases' sub-runs and the test run.

   (a) Compare the candidate cases' *perceptions* to the corresponding perception in the test run at time $t - k$. Remove cases with similarity scores below a certain predefined threshold.

   (b) This is the end of one iteration. If all remaining candidate cases agree on the same action, terminate.

   (c) Increment k by one.

   (d) Compare the candidate cases' *actions* to the corresponding action in the test run at time $t - k$. Remove cases with similarity scores below a certain predefined threshold.

116

(e) This is the end of a second iteration. If all remaining candidate cases agree on the same action, terminate.

By first examining the most recent perceptions and actions, both TB and MemTB are more likely to inform their results with influences from recent time steps than from those in older time steps, thus biasing their computations toward influences from recent events. Additionally, TB and MemTB, given the same test run and initial pool of candidate cases, will terminate after the same number of iterations.

However, TB and MemTB differ in terms of what happens when they do *terminate*. This is because they differ in *purpose*, which informs their *input parameters* and *return output*. The purpose of TB is to determine the next *action* to take in a test run in progress by using the same action as that of the most similar case sub-runs in the case base. The purpose of MemTB is to determine the most recent memory influence (encoded by a raw MF) that in and of itself distinguishes all agreeing and sufficiently similar candidate cases from all disagreeing and sufficiently dissimilar reject cases; in other words, MemTB determines which memory influence was responsible for resolving case disagreements over the right action to take in the test run during MemTB's final iteration.

The difference in purpose between TB and MemTB affects the final output. For TB, the final output is simply the action (the solution component) of all remaining candidate cases in the final TB iteration (when all candidate cases have the same solution component). For MemTB, the output is a raw MF that tracks a feature f (even if it is "Action"), the number of time steps k to backtrack, and value v of f. If there are multiple features in the perception and the test run perception was an element of comparison in MemTB's final iteration, then a heuristic must be applied to determine which feature of the perception was the most influential in discriminating between kept cases and reject cases when MemTB terminates recursion (see Algorithm 7 from Section 5.2.1).

117

Finally, the input parameters of TB and MemTB differ, in accordance to these two algorithms' difference in purpose. Specifically, MemTB has two changes in input parameters:

1. MemTB has an additional input parameter, `call-depth`, which is incremented every iteration. In other words, `call-depth` will increment approximately twice as quickly as the backtracking parameter k does. This input parameter is tracked because its value in the final MemTB iteration is returned in addition to the discovered raw MF.

2. The *test run* parameter of TB does not necessarily appear as a sub-run in any of the cases in the case base (if it does, it's only by coincidence). However, the test run parameter of MemTB will *always* be equivalent to a sub-run of one of the cases in the case base; this is because MemTB is used to find the memory influences for each case in the case base and, therefore, each case in the case base is used as the test case exactly once so that MFs can be extracted for it.

Regarding the difference between TB and MemTB for the *test run* parameter, further explanation is required. In Algorithm 2 (from Section 5.2.1), the argument given to the procedure `extract-mem-feats` as the test run parameter is the sub-run from a *case* from the case base. In fact, the procedure `extract-mem-feats` is called for *every* case in the case base. This is important because this guarantees that for deterministic behaviors, the correct memory influence (subject to the recency bias mentioned earlier that TB and MemTB share) will be found, even if the case from which the test run was derived is the only remaining candidate case in the last MemTB iteration.

In summary, MemTB uses the same case comparison and backtracking mechanisms as TB. However, MemTB's purpose differs in that it seeks to determine the most recent memory influence that resolves case retrieval. In order to do this, MemTB has additional mechanisms to *create* a raw MF in its final iteration. In addition, MemTB tracks its recursion depth and uses sub-runs from

its own case base as test runs because it is learning the memory influences from its own case base instead of resolving test runs outside of its case base. The differences between the original TB and MemTB are as follows:

1. `Mem-State-Retrieve` and `Mem-Action-Retrieve` track an additional parameter, *call-depth*, which is incremented at each level of recursion. This is different from the parameter *time-back*, which tracks how many time steps prior to the parameter *time-step* recursion has traversed (two function calls, one to `Mem-Action-Retrieve` and then one to `Mem-State-Retrieve`, constitutes one time step). The final value of this parameter is returned in the return statements in lines 13/16 and 11/12 of `Mem-State-Retrieve` (Algorithm 5) and `Mem-Action-Retrieve` (Algorithm 6), respectively.

2. When all retained cases in the *case-pool* agree on the next action to take in MemTB, instead of returning that action, a raw MF is returned instead. In line 12 of `Mem-Action-Retrieve` (Algorithm 6), this raw MF consists of the tracked feature ("Action" in this case), the *value* of the tracked feature (*test-action*, the observed action taken in this case), and the number of time steps back when all retained cases agreed (which is stored in the *time-back* parameter). In `Mem-State-Retrieve` (Algorithm 5), a raw MF is also returned, albeit with a call to `get-raw-mem-feat` (Algorithm 7).

5.2.3 Memory Feature Extraction Example

We now show MCL in action during the Memory Feature Extraction step, using the same example from the XYZ domain that was used in Section 4.4. For the benefit of the reader, the rules described in Section 4.2 for governing the behavior of the XYZ agent are repeated below.

1. If the agent perceives environmental state X, choose the action that was taken one time step

ago. Choose action a by default if it's the first time step.

2. If the agent perceived environmental state Y, choose action b.

3. If the agent perceives environmental state Z, choose the opposite taken when the environment was previously in state Z. So, if the agent chose action a when it previously perceived environmental state Z, then it should choose action b; and if the agent chose action b when it previously perceived environmental state Z, then it should choose action a. Choose action a by default if this is the first time that environmental state Z has been perceived by the agent.

By the end of the Memory Feature Extraction step, MCL will have discovered a list of MFs for each case in the case base. Suppose we have the case base shown in Table 5.1. We note that there are *many* cases that resemble the ten shown in Table 5.1, but these ten are representative of the possible *unique* cases that can be generated by observing the XYZ agent.

Table 5.1: XYZ Case Base: Unique Cases

Case Number	Problem	Solution
0	$\ldots \xrightarrow{a_{t-1}^C} X_t^C$	a_t^C
1	$\ldots \xrightarrow{b_{t-1}^C} X_t^C$	b_t^C
2	$\ldots \xrightarrow{a_{t-1}^C} Y_t^C$	b_t^C
3	$\ldots \xrightarrow{b_{t-1}^C} Y_t^C$	b_t^C
4	$\ldots \xrightarrow{b_{t-5}^C} Z_{t-4}^C \xrightarrow{b_{t-4}^C} X_{t-3}^C \xrightarrow{b_{t-3}^C} Y_{t-2}^C \xrightarrow{b_{t-2}^C} Y_{t-1}^C \xrightarrow{b_{t-1}^C} Z_t^C$	a_t^C
5	$\ldots \xrightarrow{b_{t-5}^C} Z_{t-4}^C \xrightarrow{a_{t-4}^C} X_{t-3}^C \xrightarrow{a_{t-3}^C} Y_{t-2}^C \xrightarrow{b_{t-2}^C} Y_{t-1}^C \xrightarrow{b_{t-1}^C} Z_t^C$	b_t^C
6	$\ldots \xrightarrow{b_{t-5}^C} X_{t-4}^C \xrightarrow{b_{t-4}^C} Y_{t-3}^C \xrightarrow{b_{t-3}^C} Y_{t-2}^C \xrightarrow{b_{t-2}^C} Y_{t-1}^C \xrightarrow{b_{t-1}^C} Z_t^C$	a_t^C
7	$\ldots \xrightarrow{a_{t-5}^C} X_{t-4}^C \xrightarrow{a_{t-4}^C} Y_{t-3}^C \xrightarrow{b_{t-3}^C} Y_{t-2}^C \xrightarrow{b_{t-2}^C} Y_{t-1}^C \xrightarrow{b_{t-1}^C} Z_t^C$	b_t^C
8	$\ldots \xrightarrow{a_{t-5}^C} Y_{t-4}^C \xrightarrow{b_{t-4}^C} Z_{t-3}^C \xrightarrow{b_{t-3}^C} X_{t-2}^C \xrightarrow{b_{t-2}^C} Y_{t-1}^C \xrightarrow{b_{t-1}^C} Z_t^C$	a_t^C
9	$\ldots \xrightarrow{b_{t-5}^C} Y_{t-4}^C \xrightarrow{b_{t-4}^C} Z_{t-3}^C \xrightarrow{a_{t-3}^C} X_{t-2}^C \xrightarrow{a_{t-2}^C} Y_{t-1}^C \xrightarrow{b_{t-1}^C} Z_t^C$	b_t^C

For cases where the perception at time step t is environmental state X, we know that the action chosen is always the same as that taken at time step $t-1$. Therefore, the first call to procedure `mem-tb` (Algorithm 4) in procedure `extract-mem-feats` (Algorithm 3) will result in the MemTB iterations shown in Table 5.2 if the test run is $\ldots \xrightarrow{a} X$ (action a leading to state X) or those shown in Table 5.3 if the test run is $\ldots \xrightarrow{b} X$ (action b leading to state X). (By *test run*, we are referring to the problem sub-run component of the specific case for which we are extracting MFs. Even though this case is part of the case base, it is treated as a test case.)

Table 5.2: Memory Temporal Backtracking Iterations - Test Run: $\ldots \xrightarrow{a} X$

Iteration	Test Run	Candidate Cases	Candidate Actions
0	$\ldots \xrightarrow{a^T_{t-1}} \mathbf{X^T_t}(\xrightarrow{???})$	$\ldots \xrightarrow{a^C_{t-1}} \mathbf{X^C_t}(\xrightarrow{a^C_{t-1}})$ $\ldots \xrightarrow{b^C_{t-1}} \mathbf{X^C_t}(\xrightarrow{b^C_{t-1}})$	a, b
1	$\ldots \xrightarrow{\mathbf{a^T_{t-1}}} X^T_t(\xrightarrow{???})$	$\ldots \xrightarrow{\mathbf{a^C_{t-1}}} X^C_t(\xrightarrow{a^C_{t-1}})$ $[\ldots \xrightarrow{\mathbf{b^C_{t-1}}} X^C_t(\xrightarrow{b^C_{t-1}})]$	a

Table 5.3: Memory Temporal Backtracking Iterations - Test Run: $\ldots \xrightarrow{b} X$

Iteration	Test Run	Candidate Cases	Candidate Actions
0	$\ldots \xrightarrow{b^T_{t-1}} \mathbf{X^T_t}(\xrightarrow{???})$	$\ldots \xrightarrow{a^C_{t-1}} \mathbf{X^C_t}(\xrightarrow{a^C_{t-1}})$ $\ldots \xrightarrow{b^C_{t-1}} \mathbf{X^C_t}(\xrightarrow{b^C_{t-1}})$	a, b
1	$\ldots \xrightarrow{\mathbf{b^T_{t-1}}} X^T_t(\xrightarrow{???})$	$[\ldots \xrightarrow{\mathbf{a^C_{t-1}}} X^C_t(\xrightarrow{a^C_{t-1}})]$ $\ldots \xrightarrow{\mathbf{b^C_{t-1}}} X^C_t(\xrightarrow{b^C_{t-1}})$	b

Briefly, in Iteration 0, all cases with sub-runs ending with perceptions of environmental state X are retained and all others are discarded. (For the purposes of this example, let us call the one

121

perception feature within each perception "State".) In this iteration, the solutions from these cases are not the same (possible solutions are actions a and b), so MemTB iterates one more time to analyze the *actions* at time step $t - 1$. In Iteration 1, those cases with the same action as the test run at time step $t - 1$ are retained and the others are rejected (shown by square brackets in Tables 5.2 and Table 5.3). After Iteration 1, all retained cases have the same solution component.

Therefore, MemTB will create a *raw MF* (a MF that is not yet designated as value-back or time-back). This raw MF stores:

1. The name of the last feature used to resolve conflicts in MemTB ("Action" in this example).

2. The number of time steps back when all retained cases' solution components agreed (*one time step back* in this example).

3. The value of the feature used to resolve conflicts in MemTB (for test run $\ldots \xrightarrow{a} X$ from Table 5.2, that value is action a, and for test run $\ldots \xrightarrow{b} X$ from Table 5.3, that value is action b).

Once MemTB returns this raw MF, the next step in the procedure `extract-mem-feats` (Algorithm 3) is to determine if the MF should be a value-back MF or a time-back MF. The designation as value-back or time-back should maximize the MF's usefulness. The usefulness of the MF is determined by how much it reduces the number of iterations MemTB would have to make if the MF was included as part of the perception. Therefore, MemTB tries both MF types and sees which one is more "useful".

In our XYZ example, for the test run $\cdots \rightarrow X \xrightarrow{a}$, the retrieved raw MF becomes `vAction(1)` if it becomes a value-back MF or `tAction(a)` if it becomes a time-back MF. The former would track the Action taken in the previous time step $(t - 1)$. The latter would track the number of

time steps before time step t when the Action taken was a. Tables 5.4 and 5.5 show the iterations of MemTB when the raw MF (stores these components: feature $f =$ "Action" , number of time steps back $k = 1$, value $v = a$) is added as a value-back MF `vAction(1)` and as a time-back MF `tAction(a)`, respectively, when extracting MFs for time step t in a case whose sub-run is $\ldots \xrightarrow{a} X$.

Table 5.4: Memory Temporal Backtracking Iterations - Value-Back MF Test Run: $\ldots \xrightarrow{a} X$

Iter	Test Run	Candidate Cases	Candidate Actions
n	$\ldots \xrightarrow{Ac_{t-1}^T} (State, \texttt{vAction(a)})_t^T(\xrightarrow{???})$	Retained cases $[\,\text{Discarded cases}\,]$	$\{Ac_t\}$
0	$\ldots \xrightarrow{a_{t-1}^T} (\mathbf{X_t^T}, \mathbf{a})(\xrightarrow{???})$	$\ldots \xrightarrow{a_{t-1}^C} (\mathbf{X}, \mathbf{a})_{\mathbf{t}}^{\mathbf{C}}(\xrightarrow{a_{t-1}^C})$ $[\ldots \xrightarrow{b_{t-1}^C} (\mathbf{X}, \mathbf{b})_{\mathbf{t}}^{\mathbf{C}}(\xrightarrow{b_{t-1}^C})]$	a

Table 5.5: Memory Temporal Backtracking Iterations - Time-Back MF Test Run: $\ldots \xrightarrow{a} X$

Iter	Test Run	Candidate Cases	Candidate Actions
n	$\ldots \xrightarrow{Ac_{t-1}^T} (State, \texttt{tAction(a)})_t^T(\xrightarrow{???})$	Retained cases $[\,\text{Discarded cases}\,]$	$\{Ac_t\}$
0	$\ldots \xrightarrow{a_{t-1}^T} (\mathbf{X}, \mathbf{1})_{\mathbf{t}}^{\mathbf{C}}(\xrightarrow{???})$	$\ldots \xrightarrow{a_{t-1}^C} (\mathbf{X}, \mathbf{1})_{\mathbf{t}}^{\mathbf{C}}(\xrightarrow{a_{t-1}^C})$ $[\ldots \xrightarrow{b_{t-1}^C} (\mathbf{X}, > \mathbf{1})_{\mathbf{t}}^{\mathbf{C}}(\xrightarrow{b_{t-1}^C})]$	a

As we can see, within the procedure `extract-mem-feats` (Algorithm 3), when either type of MF is added to the awareness of the internal test run for this particular case, only one iteration of MemTB is required. Therefore, MCL will add both types of MFs to the list MFs extracted for this test case. Additionally, MemTB goes back exactly zero time steps in the past with either

MF, because the awareness of the current time step t contains all information (environmental state perception plus internal state) necessary to arrive at the correct solution. Therefore, the logic in the procedure `extract-mem-feats` will terminate.

Table 5.6: Memory Temporal Backtracking Iterations - Value-Back MF Test Run: $\dots \xrightarrow{b} X$

Iter	Test Run	Candidate Cases	Candidate Actions
n	$\dots \xrightarrow{Ac_{t-1}^T} (State, \texttt{vAction(1)})_t^T (\xrightarrow{???})$	Retained cases [Discarded cases]	$\{Ac_t\}$
0	$\dots \xrightarrow{b_{t-1}^T} (\mathbf{X_t^T}, \mathbf{b})(\xrightarrow{???})$	$[\dots \xrightarrow{a_{t-1}^C} (\mathbf{X}, \mathbf{a})_\mathbf{t}^\mathbf{C} (\xrightarrow{a_{t-1}^C})]$ $\dots \xrightarrow{b_{t-1}^C} (\mathbf{X}, \mathbf{b})_\mathbf{t}^\mathbf{C} (\xrightarrow{b_{t-1}^C})$	b

Table 5.7: Memory Temporal Backtracking Iterations - Time-Back MF Test Run: $\dots \xrightarrow{b} X$

Iter	Test Run	Candidate Cases	Candidate Actions
n	$\dots \xrightarrow{Ac_{t-1}^T} (State, \texttt{tAction(b)})_t^T (\xrightarrow{???})$	Retained cases [Discarded cases]	$\{Ac_t\}$
0	$\dots \xrightarrow{b_{t-1}^T} (\mathbf{X}, \mathbf{1})_\mathbf{t}^\mathbf{C} (\xrightarrow{???})$	$[\dots \xrightarrow{a_{t-1}^C} (\mathbf{X}, \mathbf{1})_\mathbf{t}^\mathbf{C} (\xrightarrow{a_{t-1}^C})]$ $\dots \xrightarrow{b_{t-1}^C} (\mathbf{X}, > \mathbf{1})_\mathbf{t}^\mathbf{C} (\xrightarrow{b_{t-1}^C})$	a

A similar trend is observed when MemTB (via the procedure `extract-mem-feats` (Algorithm 3)) is extracting MFs for a case whose sub-run is $\dots \xrightarrow{b} X$. Tables 5.6 and 5.7 show the iterations of MemTB when the raw MF (stores: feature $f = $ *Action*, number of time steps back $k = 1$, value $v = b$) is added to the perception as a value-back MF (`vAction(1)`) or as a time-back MF (`tAction(b)`), respectively. With either type of MF (value-back or time-back), MemTB ends

after one iteration, so both types of MFs are extracted for this case. Additionally, MemTB goes back in time exactly zero time steps, so it terminates.

If a case contains a sub-run ending with a perception of environmental state Y, then the iterations of MemTB are shown in Table 5.8. The action b always follows a perception of environmental state Y, so all retrieved cases with a perception of environmental state Y at time step t have the same solution component. Therefore, MemTB has gone back in time exactly *zero* time steps and will terminate without extracting any MFs, because they are not needed for such cases with sub-runs ending with environmental state Y.

Table 5.8: Memory Temporal Backtracking Iterations - Test Run: $\ldots Y$

Iteration	Test Run	Candidate Cases	Candidate Actions
0	$\ldots Y_t^T \xrightarrow{???}$	$\ldots Y_t^T \xrightarrow{b_t^T}$	b

For cases in the case base that have sub-runs ending with perceptions of environmental state Z, the influence of memory becomes more complex and the true power of MCL becomes evident. Suppose we are extracting MFs for a case in the case base from the XYZ domain whose sub-run is as follows (same as the example run from Section 4.4):

$$\texttt{test run T:} \ldots \xrightarrow{b_{t-5}^T} Z_{t-4}^T \xrightarrow{b_{t-4}^T} X_{t-3}^T \xrightarrow{b_{t-3}^T} Y_{t-2}^T \xrightarrow{b_{t-2}^T} Y_{t-1}^T \xrightarrow{b_{t-1}^T} Z_t^T \xrightarrow{???}$$

Suppose that the unique cases in our case base are those in Table 5.9. In the procedure `extract-mem-feats`, the initial call to the procedure `mem-tb` (Algorithm 4) will proceed through the same iterations as those described in Section 4.4. Those iterations are shown in Tables 5.10 and 5.11. (The enormity of the contents requires them to be split among two tables.)

125

Table 5.9: Temporal Backtracking XYZ Unique Cases

Case Number	Problem	Solution
0	$\ldots \xrightarrow{a^C_{t-1}} X^C_t$	a^C_t
1	$\ldots \xrightarrow{b^C_{t-1}} X^C_t$	b^C_t
2	$\ldots \xrightarrow{a^C_{t-1}} Y^C_t$	b^C_t
3	$\ldots \xrightarrow{b^C_{t-1}} Y^C_t$	b^C_t
4	$\ldots \xrightarrow{b^C_{t-5}} Z^C_{t-4} \xrightarrow{b^C_{t-4}} X^C_{t-3} \xrightarrow{b^C_{t-3}} Y^C_{t-2} \xrightarrow{b^C_{t-2}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} Z^C_t$	a^C_t
5	$\ldots \xrightarrow{b^C_{t-5}} Z^C_{t-4} \xrightarrow{a^C_{t-4}} X^C_{t-3} \xrightarrow{a^C_{t-3}} Y^C_{t-2} \xrightarrow{b^C_{t-2}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} Z^C_t$	b^C_t
6	$\ldots \xrightarrow{b^C_{t-5}} X^C_{t-4} \xrightarrow{b^C_{t-4}} Y^C_{t-3} \xrightarrow{b^C_{t-3}} Y^C_{t-2} \xrightarrow{b^C_{t-2}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} Z^C_t$	a^C_t
7	$\ldots \xrightarrow{a^C_{t-5}} X^C_{t-4} \xrightarrow{a^C_{t-4}} Y^C_{t-3} \xrightarrow{b^C_{t-3}} Y^C_{t-2} \xrightarrow{b^C_{t-2}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} Z^C_t$	b^C_t
8	$\ldots \xrightarrow{a^C_{t-5}} Y^C_{t-4} \xrightarrow{b^C_{t-4}} Z^C_{t-3} \xrightarrow{b^C_{t-3}} X^C_{t-2} \xrightarrow{b^C_{t-2}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} Z^C_t$	a^C_t
9	$\ldots \xrightarrow{b^C_{t-5}} Y^C_{t-4} \xrightarrow{b^C_{t-4}} Z^C_{t-3} \xrightarrow{a^C_{t-3}} X^C_{t-2} \xrightarrow{a^C_{t-2}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} Z^C_t$	b^C_t

As demonstrated in Section 4.4 and shown in Tables 5.10 and 5.11, it takes seven iterations for the given case to be resolved by MemTB. In the final iteration of MemTB, a raw MF is created, which stores the feature analyzed ("State", the only feature), the value of that feature ("X" in this case), and the number of time steps back (3 time steps in this case).

It is important to note here that MCL only returns a raw MF from the *last* iteration of MemTB instead of one raw MF from *every* iteration. The rationale for this decision is that the raw MF from the last MemTB iteration is *essential* for separating the last few dissenting cases from the group of cases that are sufficiently similar to the test run; no other information from memory that was analyzed in the prior MemTB iterations was sufficient evidence for discarding those last few dissenting cases.

Table 5.10: Memory TB Iterations for Test Run Ending with State Z

Iter.	Test Elem	Candidate Cases	Solns.
1	Z_t^T	$[\ldots \xrightarrow{a_{t-1}^C} \mathbf{X_t^C}(\xrightarrow{a_t^C})]$ $[\ldots \xrightarrow{b_{t-1}^C} \mathbf{X_t^C}(\xrightarrow{b_t^C})]$ $[\ldots \xrightarrow{a_{t-1}^C} \mathbf{Y_t^C}(\xrightarrow{b_t^C})]$ $[\ldots \xrightarrow{b_{t-1}^C} \mathbf{Y_t^C}(\xrightarrow{b_t^C})]$ $\ldots \xrightarrow{b_{t-5}^C} Z_{t-4}^C \xrightarrow{b_{t-4}^C} X_{t-3}^C \xrightarrow{b_{t-3}^C} Y_{t-2}^C \xrightarrow{b_{t-2}^C} Y_{t-1}^C \xrightarrow{b_{t-1}^C} \mathbf{Z_t^C}(\xrightarrow{a_t^C})$ $\ldots \xrightarrow{b_{t-5}^C} Z_{t-4}^C \xrightarrow{a_{t-4}^C} X_{t-3}^C \xrightarrow{a_{t-3}^C} Y_{t-2}^C \xrightarrow{b_{t-2}^C} Y_{t-1}^C \xrightarrow{b_{t-1}^C} \mathbf{Z_t^C}(\xrightarrow{b_t^C})$ $\ldots \xrightarrow{b_{t-5}^C} X_{t-4}^C \xrightarrow{b_{t-4}^C} Y_{t-3}^C \xrightarrow{b_{t-3}^C} Y_{t-2}^C \xrightarrow{b_{t-2}^C} Y_{t-1}^C \xrightarrow{b_{t-1}^C} \mathbf{Z_t^C}(\xrightarrow{a_t^C})$ $\ldots \xrightarrow{a_{t-5}^C} X_{t-4}^C \xrightarrow{a_{t-4}^C} Y_{t-3}^C \xrightarrow{b_{t-3}^C} Y_{t-2}^C \xrightarrow{b_{t-2}^C} Y_{t-1}^C \xrightarrow{b_{t-1}^C} \mathbf{Z_t^C}(\xrightarrow{b_t^C})$ $\ldots \xrightarrow{a_{t-5}^C} Y_{t-4}^C \xrightarrow{b_{t-4}^C} Z_{t-3}^C \xrightarrow{b_{t-3}^C} X_{t-2}^C \xrightarrow{b_{t-2}^C} Y_{t-1}^C \xrightarrow{b_{t-1}^C} \mathbf{Z_t^C}(\xrightarrow{a_t^C})$ $\ldots \xrightarrow{b_{t-5}^C} Y_{t-4}^C \xrightarrow{b_{t-4}^C} Z_{t-3}^C \xrightarrow{a_{t-3}^C} X_{t-2}^C \xrightarrow{a_{t-2}^C} Y_{t-1}^C \xrightarrow{b_{t-1}^C} \mathbf{Z_t^C}(\xrightarrow{b_t^C})$	a, b
2	b_{t-1}^T	$\ldots \xrightarrow{b_{t-5}^C} Z_{t-4}^C \xrightarrow{b_{t-4}^C} X_{t-3}^C \xrightarrow{b_{t-3}^C} Y_{t-2}^C \xrightarrow{b_{t-2}^C} Y_{t-1}^C \xrightarrow{\mathbf{b_{t-1}^C}} Z_t^C(\xrightarrow{a_t^C})$ $\ldots \xrightarrow{b_{t-5}^C} Z_{t-4}^C \xrightarrow{a_{t-4}^C} X_{t-3}^C \xrightarrow{a_{t-3}^C} Y_{t-2}^C \xrightarrow{b_{t-2}^C} Y_{t-1}^C \xrightarrow{\mathbf{b_{t-1}^C}} Z_t^C(\xrightarrow{b_t^C})$ $\ldots \xrightarrow{b_{t-5}^C} X_{t-4}^C \xrightarrow{b_{t-4}^C} Y_{t-3}^C \xrightarrow{b_{t-3}^C} Y_{t-2}^C \xrightarrow{b_{t-2}^C} Y_{t-1}^C \xrightarrow{\mathbf{b_{t-1}^C}} Z_t^C(\xrightarrow{a_t^C})$ $\ldots \xrightarrow{a_{t-5}^C} X_{t-4}^C \xrightarrow{a_{t-4}^C} Y_{t-3}^C \xrightarrow{b_{t-3}^C} Y_{t-2}^C \xrightarrow{b_{t-2}^C} Y_{t-1}^C \xrightarrow{\mathbf{b_{t-1}^C}} Z_t^C(\xrightarrow{b_t^C})$ $\ldots \xrightarrow{a_{t-5}^C} Y_{t-4}^C \xrightarrow{b_{t-4}^C} Z_{t-3}^C \xrightarrow{b_{t-3}^C} X_{t-2}^C \xrightarrow{b_{t-2}^C} Y_{t-1}^C \xrightarrow{\mathbf{b_{t-1}^C}} Z_t^C(\xrightarrow{a_t^C})$ $\ldots \xrightarrow{b_{t-5}^C} Y_{t-4}^C \xrightarrow{b_{t-4}^C} Z_{t-3}^C \xrightarrow{a_{t-3}^C} X_{t-2}^C \xrightarrow{a_{t-2}^C} Y_{t-1}^C \xrightarrow{\mathbf{b_{t-1}^C}} Z_t^C(\xrightarrow{b_t^C})$	a, b
3	Y_{t-1}^T	$\ldots \xrightarrow{b_{t-5}^C} Z_{t-4}^C \xrightarrow{b_{t-4}^C} X_{t-3}^C \xrightarrow{b_{t-3}^C} Y_{t-2}^C \xrightarrow{b_{t-2}^C} \mathbf{Y_{t-1}^C} \xrightarrow{b_{t-1}^C} Z_t^C(\xrightarrow{a_t^C})$ $\ldots \xrightarrow{b_{t-5}^C} Z_{t-4}^C \xrightarrow{a_{t-4}^C} X_{t-3}^C \xrightarrow{a_{t-3}^C} Y_{t-2}^C \xrightarrow{b_{t-2}^C} \mathbf{Y_{t-1}^C} \xrightarrow{b_{t-1}^C} Z_t^C(\xrightarrow{b_t^C})$ $\ldots \xrightarrow{b_{t-5}^C} X_{t-4}^C \xrightarrow{b_{t-4}^C} Y_{t-3}^C \xrightarrow{b_{t-3}^C} Y_{t-2}^C \xrightarrow{b_{t-2}^C} \mathbf{Y_{t-1}^C} \xrightarrow{b_{t-1}^C} Z_t^C(\xrightarrow{a_t^C})$ $\ldots \xrightarrow{a_{t-5}^C} X_{t-4}^C \xrightarrow{a_{t-4}^C} Y_{t-3}^C \xrightarrow{b_{t-3}^C} Y_{t-2}^C \xrightarrow{b_{t-2}^C} \mathbf{Y_{t-1}^C} \xrightarrow{b_{t-1}^C} Z_t^C(\xrightarrow{b_t^C})$ $\ldots \xrightarrow{a_{t-5}^C} Y_{t-4}^C \xrightarrow{b_{t-4}^C} Z_{t-3}^C \xrightarrow{b_{t-3}^C} X_{t-2}^C \xrightarrow{b_{t-2}^C} \mathbf{Y_{t-1}^C} \xrightarrow{b_{t-1}^C} Z_t^C(\xrightarrow{a_t^C})$ $\ldots \xrightarrow{b_{t-5}^C} Y_{t-4}^C \xrightarrow{b_{t-4}^C} Z_{t-3}^C \xrightarrow{a_{t-3}^C} X_{t-2}^C \xrightarrow{a_{t-2}^C} \mathbf{Y_{t-1}^C} \xrightarrow{b_{t-1}^C} Z_t^C(\xrightarrow{b_t^C})$	a, b
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Table 5.11: Memory TB Iterations for Test Run Ending with State Z

Iter.	Test Elem	Candidate Cases	Solns.
$\vdots$	$\vdots$	$\vdots$	$\vdots$
4	b_{t-2}^T	$\ldots \xrightarrow{b_{t-5}^C} Z_{t-4}^C \xrightarrow{b_{t-4}^C} X_{t-3}^C \xrightarrow{b_{t-3}^C} Y_{t-2}^C \xrightarrow{\mathbf{b_{t-2}^C}} Y_{t-1}^C \xrightarrow{b_{t-1}^C} Z_t^C(\xrightarrow{a_t^C})$ $\ldots \xrightarrow{b_{t-5}^C} Z_{t-4}^C \xrightarrow{a_{t-4}^C} X_{t-3}^C \xrightarrow{a_{t-3}^C} Y_{t-2}^C \xrightarrow{\mathbf{b_{t-2}^C}} Y_{t-1}^C \xrightarrow{b_{t-1}^C} Z_t^C(\xrightarrow{b_t^C})$ $\ldots \xrightarrow{b_{t-5}^C} X_{t-4}^C \xrightarrow{b_{t-4}^C} Y_{t-3}^C \xrightarrow{b_{t-3}^C} Y_{t-2}^C \xrightarrow{\mathbf{b_{t-2}^C}} Y_{t-1}^C \xrightarrow{b_{t-1}^C} Z_t^C(\xrightarrow{a_t^C})$ $\ldots \xrightarrow{a_{t-5}^C} X_{t-4}^C \xrightarrow{a_{t-4}^C} Y_{t-3}^C \xrightarrow{b_{t-3}^C} Y_{t-2}^C \xrightarrow{\mathbf{b_{t-2}^C}} Y_{t-1}^C \xrightarrow{b_{t-1}^C} Z_t^C(\xrightarrow{b_t^C})$ $\ldots \xrightarrow{a_{t-5}^C} Y_{t-4}^C \xrightarrow{b_{t-4}^C} Z_{t-3}^C \xrightarrow{b_{t-3}^C} X_{t-2}^C \xrightarrow{\mathbf{b_{t-2}^C}} Y_{t-1}^C \xrightarrow{b_{t-1}^C} Z_t^C(\xrightarrow{a_t^C})$ $[\ldots \xrightarrow{b_{t-5}^C} Y_{t-4}^C \xrightarrow{b_{t-4}^C} Z_{t-3}^C \xrightarrow{a_{t-3}^C} X_{t-2}^C \xrightarrow{\mathbf{a_{t-2}^C}} Y_{t-1}^C \xrightarrow{b_{t-1}^C} Z_t^C(\xrightarrow{b_t^C})]$	a, b
5	Y_{t-2}^T	$\ldots \xrightarrow{b_{t-5}^C} Z_{t-4}^C \xrightarrow{b_{t-4}^C} X_{t-3}^C \xrightarrow{b_{t-3}^C} \mathbf{Y_{t-2}^C} \xrightarrow{b_{t-2}^C} Y_{t-1}^C \xrightarrow{b_{t-1}^C} Z_t^C(\xrightarrow{a_t^C})$ $\ldots \xrightarrow{b_{t-5}^C} Z_{t-4}^C \xrightarrow{a_{t-4}^C} X_{t-3}^C \xrightarrow{a_{t-3}^C} \mathbf{Y_{t-2}^C} \xrightarrow{b_{t-2}^C} Y_{t-1}^C \xrightarrow{b_{t-1}^C} Z_t^C(\xrightarrow{b_t^C})$ $\ldots \xrightarrow{b_{t-5}^C} X_{t-4}^C \xrightarrow{b_{t-4}^C} Y_{t-3}^C \xrightarrow{b_{t-3}^C} \mathbf{Y_{t-2}^C} \xrightarrow{b_{t-2}^C} Y_{t-1}^C \xrightarrow{b_{t-1}^C} Z_t^C(\xrightarrow{a_t^C})$ $\ldots \xrightarrow{a_{t-5}^C} X_{t-4}^C \xrightarrow{a_{t-4}^C} Y_{t-3}^C \xrightarrow{b_{t-3}^C} \mathbf{Y_{t-2}^C} \xrightarrow{b_{t-2}^C} Y_{t-1}^C \xrightarrow{b_{t-1}^C} Z_t^C(\xrightarrow{b_t^C})$ $[\ldots \xrightarrow{a_{t-5}^C} Y_{t-4}^C \xrightarrow{b_{t-4}^C} Z_{t-3}^C \xrightarrow{b_{t-3}^C} \mathbf{X_{t-2}^C} \xrightarrow{b_{t-2}^C} Y_{t-1}^C \xrightarrow{b_{t-1}^C} Z_t^C(\xrightarrow{a_t^C})]$	a, b
6	b_{t-3}^T	$\ldots \xrightarrow{b_{t-5}^C} Z_{t-4}^C \xrightarrow{b_{t-4}^C} X_{t-3}^C \xrightarrow{\mathbf{b_{t-3}^C}} Y_{t-2}^C \xrightarrow{b_{t-2}^C} Y_{t-1}^C \xrightarrow{b_{t-1}^C} Z_t^C(\xrightarrow{a_t^C})$ $[\ldots \xrightarrow{b_{t-5}^C} Z_{t-4}^C \xrightarrow{a_{t-4}^C} X_{t-3}^C \xrightarrow{\mathbf{a_{t-3}^C}} Y_{t-2}^C \xrightarrow{b_{t-2}^C} Y_{t-1}^C \xrightarrow{b_{t-1}^C} Z_t^C(\xrightarrow{b_t^C})]$ $\ldots \xrightarrow{b_{t-5}^C} X_{t-4}^C \xrightarrow{b_{t-4}^C} Y_{t-3}^C \xrightarrow{\mathbf{b_{t-3}^C}} Y_{t-2}^C \xrightarrow{b_{t-2}^C} Y_{t-1}^C \xrightarrow{b_{t-1}^C} Z_t^C(\xrightarrow{a_t^C})$ $\ldots \xrightarrow{a_{t-5}^C} X_{t-4}^C \xrightarrow{a_{t-4}^C} Y_{t-3}^C \xrightarrow{\mathbf{b_{t-3}^C}} Y_{t-2}^C \xrightarrow{b_{t-2}^C} Y_{t-1}^C \xrightarrow{b_{t-1}^C} Z_t^C(\xrightarrow{b_t^C})$	a, b
7	X_{t-3}^T	$\ldots \xrightarrow{b_{t-5}^C} Z_{t-4}^C \xrightarrow{b_{t-4}^C} \mathbf{X_{t-3}^C} \xrightarrow{b_{t-3}^C} Y_{t-2}^C \xrightarrow{b_{t-2}^C} Y_{t-1}^C \xrightarrow{b_{t-1}^C} Z_t^C(\xrightarrow{a_t^C})$ $[\ldots \xrightarrow{b_{t-5}^C} X_{t-4}^C \xrightarrow{b_{t-4}^C} \mathbf{Y_{t-3}^C} \xrightarrow{b_{t-3}^C} Y_{t-2}^C \xrightarrow{b_{t-2}^C} Y_{t-1}^C \xrightarrow{b_{t-1}^C} Z_t^C(\xrightarrow{a_t^C})]$ $[\ldots \xrightarrow{a_{t-5}^C} X_{t-4}^C \xrightarrow{a_{t-4}^C} \mathbf{Y_{t-3}^C} \xrightarrow{b_{t-3}^C} Y_{t-2}^C \xrightarrow{b_{t-2}^C} Y_{t-1}^C \xrightarrow{b_{t-1}^C} Z_t^C(\xrightarrow{b_t^C})]$	a

Therefore, the raw MF that is created in the last MemTB iteration is indicative of a *memory influence* that has significance for the observed behavior (the behavior of the XYZ agent in this case). Raw MFs that could be extracted from earlier MemTB iterations would not be sufficient for filtering out the mismatching cases that made it to the final MemTB iteration. However, the raw MF from the final MemTB iteration may be sufficient to filter out cases in earlier iterations, as we shall see in our continuing example.

After MemTB extracts the raw MF from its last iteration, MCL must decide whether the raw MF is better as a value-back MF or a time-back MF. So, MCL makes it a value-back MF `vState(3)` and temporarily adds it to the perceptions in the test run and in all cases in the case base. In the procedure `extract-mem-feats` (Algorithm 3), MCL runs MemTB again (with a call to the procedure `mem-tb`, Algorithm 4) and counts the number of iterations it takes to make all surviving candidate cases agree. Such iterations for our example with the case with the sub-run

$$\texttt{test run T:} \ldots \xrightarrow{b^T_{t-5}} Z^T_{t-4} \xrightarrow{b^T_{t-4}} X^T_{t-3} \xrightarrow{b^T_{t-3}} Y^T_{t-2} \xrightarrow{b^T_{t-2}} Y^T_{t-1} \xrightarrow{b^T_{t-1}} Z^T_t \xrightarrow{???}$$

are shown in Tables 5.12 and 5.13 (the contents needed to be split into two tables). Each awareness in Tables 5.12 and 5.13 has the form $(State, \texttt{vAction(3)})_t$. In Iteration 1, we do not show cases not ending with perceptions of environmental state Z, to save space (such cases would be discarded within that iteration).

When we add a value-back MF `vState(3)` to the perception, the number of iterations required by MemTB drops from seven to six. Next, MCL will remove the value-back MF `vState(3)` from the test case's sub-run and the cases in the case base, then it will convert the raw MF it retrieved earlier into the time-back MF `tState(X)`. MCL will incorporate `tState(X)` into the perceptions of the test case's sub-run and those of the cases in the case base. MCL's logic within the procedure `extract-mem-feats` (Algorithm 3) will then call MemTB again (procedure

mem-tb, Algorithm 4) to see if the time-back MF can reduce the number of iterations even further than the value-back MF did. These iterations can be seen in Table 5.14. Each awareness in the sub-runs is of the form $(State, \text{tState}(X))_t$.

Table 5.12: Memory TB Iterations for Test Run Ending with State Z with Value-Back MF

Iter.	Test Elem	Candidate Cases	Solns.
1	$(Z,Y)_t^T$	$\ldots \xrightarrow{b_{t-3}^C} (Y,X)_{t-2}^C \xrightarrow{b_{t-2}^C} (Y,Y)_{t-1}^C \xrightarrow{b_{t-1}^C} (\mathbf{Z},\mathbf{Y})_\mathbf{t}^\mathbf{C}(\xrightarrow{a_t^C})$ $\ldots \xrightarrow{a_{t-3}^C} (Y,X)_{t-2}^C \xrightarrow{b_{t-2}^C} (Y,Y)_{t-1}^C \xrightarrow{b_{t-1}^C} (\mathbf{Z},\mathbf{Y})_\mathbf{t}^\mathbf{C}(\xrightarrow{b_t^C})$ $\ldots \xrightarrow{b_{t-3}^C} (Y,Y)_{t-2}^C \xrightarrow{b_{t-2}^C} (Y,Y)_{t-1}^C \xrightarrow{b_{t-1}^C} (\mathbf{Z},\mathbf{Y})_\mathbf{t}^\mathbf{C}(\xrightarrow{a_t^C})$ $\ldots \xrightarrow{b_{t-3}^C} (Y,Y)_{t-2}^C \xrightarrow{b_{t-2}^C} (Y,Y)_{t-1}^C \xrightarrow{b_{t-1}^C} (\mathbf{Z},\mathbf{Y})_\mathbf{t}^\mathbf{C}(\xrightarrow{b_t^C})$ $\ldots \xrightarrow{b_{t-3}^C} (X,Z)_{t-2}^C \xrightarrow{b_{t-2}^C} (Y,X)_{t-1}^C \xrightarrow{b_{t-1}^C} (\mathbf{Z},\mathbf{Y})_\mathbf{t}^\mathbf{C}(\xrightarrow{a_t^C})$ $\ldots \xrightarrow{a_{t-3}^C} (X,Z)_{t-2}^C \xrightarrow{a_{t-2}^C} (Y,X)_{t-1}^C \xrightarrow{b_{t-1}^C} (\mathbf{Z},\mathbf{Y})_\mathbf{t}^\mathbf{C}(\xrightarrow{b_t^C})$	a,b
2	b_{t-1}^T	$\ldots \xrightarrow{b_{t-3}^C} (Y,X)_{t-2}^C \xrightarrow{b_{t-2}^C} (Y,Y)_{t-1}^C \xrightarrow{\mathbf{b_{t-1}^C}} (Z,Y)_t^C(\xrightarrow{a_t^C})$ $\ldots \xrightarrow{a_{t-3}^C} (Y,X)_{t-2}^C \xrightarrow{b_{t-2}^C} (Y,Y)_{t-1}^C \xrightarrow{\mathbf{b_{t-1}^C}} (Z,Y)_t^C(\xrightarrow{b_t^C})$ $\ldots \xrightarrow{b_{t-3}^C} (Y,Y)_{t-2}^C \xrightarrow{b_{t-2}^C} (Y,Y)_{t-1}^C \xrightarrow{\mathbf{b_{t-1}^C}} (Z,Y)_t^C(\xrightarrow{a_t^C})$ $\ldots \xrightarrow{b_{t-3}^C} (Y,Y)_{t-2}^C \xrightarrow{b_{t-2}^C} (Y,Y)_{t-1}^C \xrightarrow{\mathbf{b_{t-1}^C}} (Z,Y)_t^C(\xrightarrow{b_t^C})$ $\ldots \xrightarrow{b_{t-3}^C} (X,Z)_{t-2}^C \xrightarrow{b_{t-2}^C} (Y,X)_{t-1}^C \xrightarrow{\mathbf{b_{t-1}^C}} (Z,Y)_t^C(\xrightarrow{a_t^C})$ $\ldots \xrightarrow{a_{t-3}^C} (X,Z)_{t-2}^C \xrightarrow{a_{t-2}^C} (Y,X)_{t-1}^C \xrightarrow{\mathbf{b_{t-1}^C}} (Z,Y)_t^C(\xrightarrow{b_t^C})$	a,b
3	$(Y,Y)_{t-1}^T$	$\ldots \xrightarrow{b_{t-3}^C} (Y,X)_{t-2}^C \xrightarrow{b_{t-2}^C} (\mathbf{Y},\mathbf{Y})_\mathbf{t-1}^\mathbf{C} \xrightarrow{b_{t-1}^C} (Z,Y)_t^C(\xrightarrow{a_t^C})$ $\ldots \xrightarrow{a_{t-3}^C} (Y,X)_{t-2}^C \xrightarrow{b_{t-2}^C} (\mathbf{Y},\mathbf{Y})_\mathbf{t-1}^\mathbf{C} \xrightarrow{b_{t-1}^C} (Z,Y)_t^C(\xrightarrow{b_t^C})$ $\ldots \xrightarrow{b_{t-3}^C} (Y,Y)_{t-2}^C \xrightarrow{b_{t-2}^C} (\mathbf{Y},\mathbf{Y})_\mathbf{t-1}^\mathbf{C} \xrightarrow{b_{t-1}^C} (Z,Y)_t^C(\xrightarrow{a_t^C})$ $\ldots \xrightarrow{b_{t-3}^C} (Y,Y)_{t-2}^C \xrightarrow{b_{t-2}^C} (\mathbf{Y},\mathbf{Y})_\mathbf{t-1}^\mathbf{C} \xrightarrow{b_{t-1}^C} (Z,Y)_t^C(\xrightarrow{b_t^C})$ $[\ldots \xrightarrow{b_{t-3}^C} (X,Z)_{t-2}^C \xrightarrow{b_{t-2}^C} (\mathbf{Y},\mathbf{X})_\mathbf{t-1}^\mathbf{C} \xrightarrow{b_{t-1}^C} (Z,Y)_t^C(\xrightarrow{a_t^C})]$ $[\ldots \xrightarrow{a_{t-3}^C} (X,Z)_{t-2}^C \xrightarrow{a_{t-2}^C} (\mathbf{Y},\mathbf{X})_\mathbf{t-1}^\mathbf{C} \xrightarrow{b_{t-1}^C} (Z,Y)_t^C(\xrightarrow{b_t^C})]$	a,b
⋮	⋮	⋮	⋮

Table 5.13: Memory TB Iterations for Test Run Ending with State Z with Value-Back MF

Iter.	Test Elem	Candidate Cases	Solns.
$\vdots$	$\vdots$	$\vdots$	$\vdots$
4	b_{t-2}^T	$\ldots \xrightarrow{b_{t-3}^C} (Y,X)_{t-2}^C \xrightarrow{\mathbf{b_{t-2}^C}} (Y,Y)_{t-1}^C \xrightarrow{b_{t-1}^C} (Z,Y)_t^C (\xrightarrow{a_t^C})$ $\ldots \xrightarrow{a_{t-3}^C} (Y,X)_{t-2}^C \xrightarrow{\mathbf{b_{t-2}^C}} (Y,Y)_{t-1}^C \xrightarrow{b_{t-1}^C} (Z,Y)_t^C (\xrightarrow{b_t^C})$ $\ldots \xrightarrow{b_{t-3}^C} (Y,Y)_{t-2}^C \xrightarrow{\mathbf{b_{t-2}^C}} (Y,Y)_{t-1}^C \xrightarrow{b_{t-1}^C} (Z,Y)_t^C (\xrightarrow{a_t^C})$ $\ldots \xrightarrow{b_{t-3}^C} (Y,Y)_{t-2}^C \xrightarrow{\mathbf{b_{t-2}^C}} (Y,Y)_{t-1}^C \xrightarrow{b_{t-1}^C} (Z,Y)_t^C (\xrightarrow{b_t^C})$	a, b
5	$(Y,X)_{t-2}^T$	$\ldots \xrightarrow{b_{t-3}^C} \mathbf{(Y,X)_{t-2}^C} \xrightarrow{b_{t-2}^C} (Y,Y)_{t-1}^C \xrightarrow{b_{t-1}^C} (Z,Y)_t^C (\xrightarrow{a_t^C})$ $\ldots \xrightarrow{a_{t-3}^C} \mathbf{(Y,X)_{t-2}^C} \xrightarrow{b_{t-2}^C} (Y,Y)_{t-1}^C \xrightarrow{b_{t-1}^C} (Z,Y)_t^C (\xrightarrow{b_t^C})$ $[\ldots \xrightarrow{b_{t-3}^C} \mathbf{(Y,Y)_{t-2}^C} \xrightarrow{b_{t-2}^C} (Y,Y)_{t-1}^C \xrightarrow{b_{t-1}^C} (Z,Y)_t^C (\xrightarrow{a_t^C})]$ $[\ldots \xrightarrow{b_{t-3}^C} \mathbf{(Y,Y)_{t-2}^C} \xrightarrow{b_{t-2}^C} (Y,Y)_{t-1}^C \xrightarrow{b_{t-1}^C} (Z,Y)_t^C (\xrightarrow{b_t^C})]$	a, b
6	b_{t-3}^T	$\ldots \xrightarrow{\mathbf{b_{t-3}^C}} (Y,X)_{t-2}^C \xrightarrow{b_{t-2}^C} (Y,Y)_{t-1}^C \xrightarrow{b_{t-1}^C} (Z,Y)_t^C (\xrightarrow{a_t^C})$ $[\ldots \xrightarrow{\mathbf{a_{t-3}^C}} (Y,X)_{t-2}^C \xrightarrow{b_{t-2}^C} (Y,Y)_{t-1}^C \xrightarrow{b_{t-1}^C} (Z,Y)_t^C (\xrightarrow{b_t^C})]$	a

Now, the test run takes the form:

$$\ldots \xrightarrow{b_{t-3}^T} (Y,1)_{t-2}^T \xrightarrow{b_{t-2}^T} (Y,2)_{t-1}^T \xrightarrow{b_{t-1}^T} (Z,3)_t^T$$

Note that the first part of the test run was truncated, because MemTB will terminate before processing those elements of the test run in this example.

Table 5.14: Memory TB Iterations for Test Run Ending with State Z with Time-Back MF

Iter.	Test Elem	Candidate Cases	Solns.
1	$(Z,3)_t^T$	$\ldots \xrightarrow{b_{t-3}^C} (Y,1)_{t-2}^C \xrightarrow{b_{t-2}^C} (Y,2)_{t-1}^C \xrightarrow{b_{t-1}^C} (\mathbf{Z},\mathbf{3})_\mathbf{t}^\mathbf{C}(\xrightarrow{a_t^C})$ $\ldots \xrightarrow{a_{t-3}^C} (Y,1)_{t-2}^C \xrightarrow{b_{t-2}^C} (Y,2)_{t-1}^C \xrightarrow{b_{t-1}^C} (\mathbf{Z},\mathbf{3})_\mathbf{t}^\mathbf{C}(\xrightarrow{b_t^C})$ $[\ldots \xrightarrow{b_{t-3}^C} (Y,2)_{t-2}^C \xrightarrow{b_{t-2}^C} (Y,3)_{t-1}^C \xrightarrow{b_{t-1}^C} (\mathbf{Z},\mathbf{4})_\mathbf{t}^\mathbf{C}(\xrightarrow{a_t^C})]$ $[\ldots \xrightarrow{b_{t-3}^C} (Y,2)_{t-2}^C \xrightarrow{b_{t-2}^C} (Y,3)_{t-1}^C \xrightarrow{b_{t-1}^C} (\mathbf{Z},\mathbf{4})_\mathbf{t}^\mathbf{C}(\xrightarrow{b_t^C})]$ $[\ldots \xrightarrow{b_{t-3}^C} (X,3)_{t-2}^C \xrightarrow{b_{t-2}^C} (Y,1)_{t-1}^C \xrightarrow{b_{t-1}^C} (\mathbf{Z},\mathbf{2})_\mathbf{t}^\mathbf{C}(\xrightarrow{a_t^C})]$ $[\ldots \xrightarrow{a_{t-3}^C} (X,3)_{t-2}^C \xrightarrow{a_{t-2}^C} (Y,1)_{t-1}^C \xrightarrow{b_{t-1}^C} (\mathbf{Z},\mathbf{2})_\mathbf{t}^\mathbf{C}(\xrightarrow{b_t^C})]$	a, b
2	b_{t-1}^T	$\ldots \xrightarrow{b_{t-3}^C} (Y,1)_{t-2}^C \xrightarrow{b_{t-2}^C} (Y,2)_{t-1}^C \xrightarrow{\mathbf{b_{t-1}^C}} (Z,3)_t^C(\xrightarrow{a_t^C})$ $\ldots \xrightarrow{a_{t-3}^C} (Y,1)_{t-2}^C \xrightarrow{b_{t-2}^C} (Y,2)_{t-1}^C \xrightarrow{\mathbf{b_{t-1}^C}} (Z,3)_t^C(\xrightarrow{b_t^C})$	a, b
3	$(Y,2)_{t-1}^T$	$\ldots \xrightarrow{b_{t-3}^C} (Y,1)_{t-2}^C \xrightarrow{b_{t-2}^C} (\mathbf{Y},\mathbf{2})_\mathbf{t-1}^\mathbf{C} \xrightarrow{b_{t-1}^C} (Z,3)_t^C(\xrightarrow{a_t^C})$ $\ldots \xrightarrow{a_{t-3}^C} (Y,1)_{t-2}^C \xrightarrow{b_{t-2}^C} (\mathbf{Y},\mathbf{2})_\mathbf{t-1}^\mathbf{C} \xrightarrow{b_{t-1}^C} (Z,3)_t^C(\xrightarrow{b_t^C})$	a, b
4	b_{t-2}^T	$\ldots \xrightarrow{b_{t-3}^C} (Y,1)_{t-2}^C \xrightarrow{\mathbf{b_{t-2}^C}} (Y,2)_{t-1}^C \xrightarrow{b_{t-1}^C} (Z,3)_t^C(\xrightarrow{a_t^C})$ $\ldots \xrightarrow{a_{t-3}^C} (Y,1)_{t-2}^C \xrightarrow{\mathbf{b_{t-2}^C}} (Y,2)_{t-1}^C \xrightarrow{b_{t-1}^C} (Z,3)_t^C(\xrightarrow{b_t^C})$	a, b
5	$(Y,1)_{t-2}^T$	$\ldots \xrightarrow{b_{t-3}^C} (\mathbf{Y},\mathbf{1})_\mathbf{t-2}^\mathbf{C} \xrightarrow{b_{t-2}^C} (Y,2)_{t-1}^C \xrightarrow{b_{t-1}^C} (Z,3)_t^C(\xrightarrow{a_t^C})$ $\ldots \xrightarrow{a_{t-3}^C} (\mathbf{Y},\mathbf{1})_\mathbf{t-2}^\mathbf{C} \xrightarrow{b_{t-2}^C} (Y,2)_{t-1}^C \xrightarrow{b_{t-1}^C} (Z,3)_t^C(\xrightarrow{b_t^C})$	a, b
6	b_{t-3}^T	$\ldots \xrightarrow{\mathbf{b_{t-3}^C}} (Y,1)_{t-2}^C \xrightarrow{b_{t-2}^C} (Y,2)_{t-1}^C \xrightarrow{b_{t-1}^C} (Z,3)_t^C(\xrightarrow{a_t^C})$ $[\ldots \xrightarrow{\mathbf{a_{t-3}^C}} (Y,1)_{t-2}^C \xrightarrow{b_{t-2}^C} (Y,2)_{t-1}^C \xrightarrow{b_{t-1}^C} (Z,3)_t^C(\xrightarrow{b_t^C})]$	a

The time-back MF tState(X) also reduces the number of iterations for MemTB from seven to six, so it is not more advantageous than the value-back MF type in that regard. However, it is interesting to note that in the first iteration of MemTB, the value-back MF reduced the number of unique candidates cases to four while the time-back MF reduced the number of unique candidate cases to two. The question of whether or not this indicates an advantage to using the time-back MF instead of the value-back MF for this particular test case is left for future research.

Table 5.15: MemTB, Recursion Level 2, Test Run Ending with State Z

Iter.	Test Elem	Candidate Cases	Solns.
1	$(Z,Y,3)_t^T$	$\ldots \xrightarrow{b_{t-3}^C} (Y,X,1)_{t-2}^C \xrightarrow{b_{t-2}^C} (Y,Y,2)_{t-1}^C \xrightarrow{b_{t-1}^C}$ $(\mathbf{Z},\mathbf{Y},\mathbf{3})_{\mathbf{t}}^{\mathbf{C}}(\xrightarrow{a_t^C})$ $\ldots \xrightarrow{a_{t-3}^C} (Y,X,1)_{t-2}^C \xrightarrow{b_{t-2}^C} (Y,Y,2)_{t-1}^C \xrightarrow{b_{t-1}^C}$ $(\mathbf{Z},\mathbf{Y},\mathbf{3})_{\mathbf{t}}^{\mathbf{C}}(\xrightarrow{b_t^C})$ $[\ldots \xrightarrow{b_{t-3}^C} (Y,Y,2)_{t-2}^C \xrightarrow{b_{t-2}^C} (Y,Y,3)_{t-1}^C \xrightarrow{b_{t-1}^C}$ $(\mathbf{Z},\mathbf{Y},\mathbf{4})_{\mathbf{t}}^{\mathbf{C}}(\xrightarrow{a_t^C})]$ $[\ldots \xrightarrow{b_{t-3}^C} (Y,Y,2)_{t-2}^C \xrightarrow{b_{t-2}^C} (Y,Y,3)_{t-1}^C \xrightarrow{b_{t-1}^C}$ $(\mathbf{Z},\mathbf{Y},\mathbf{4})_{\mathbf{t}}^{\mathbf{C}}(\xrightarrow{b_t^C})]$ $[\ldots \xrightarrow{b_{t-3}^C} (X,Z,3)_{t-2}^C \xrightarrow{b_{t-2}^C} (Y,X,1)_{t-1}^C \xrightarrow{b_{t-1}^C}$ $(\mathbf{Z},\mathbf{Y},\mathbf{2})_{\mathbf{t}}^{\mathbf{C}}(\xrightarrow{a_t^C})]$ $[\ldots \xrightarrow{a_{t-3}^C} (X,Z,3)_{t-2}^C \xrightarrow{a_{t-2}^C} (Y,X,1)_{t-1}^C \xrightarrow{b_{t-1}^C}$ $(\mathbf{Z},\mathbf{Y},\mathbf{2})_{\mathbf{t}}^{\mathbf{C}}(\xrightarrow{b_t^C})]$	a,b
2	b_{t-1}^T	$\ldots \xrightarrow{b_{t-3}^C} (Y,Y,1)_{t-2}^C \xrightarrow{b_{t-2}^C} (Y,Y,2)_{t-1}^C \xrightarrow{\mathbf{b_{t-1}^C}}$ $(Z,Y,3)_t^C(\xrightarrow{a_t^C})$ $\ldots \xrightarrow{a_{t-3}^C} (Y,Y,1)_{t-2}^C \xrightarrow{b_{t-2}^C} (Y,Y,2)_{t-1}^C \xrightarrow{\mathbf{b_{t-1}^C}}$ $(Z,Y,3)_t^C(\xrightarrow{b_t^C})$	a,b
3	$(Y,Y,2)_{t-1}^T$	$\ldots \xrightarrow{b_{t-3}^C} (Y,Y,1)_{t-2}^C \xrightarrow{b_{t-2}^C} (\mathbf{Y},\mathbf{Y},\mathbf{2})_{\mathbf{t-1}}^{\mathbf{C}} \xrightarrow{b_{t-1}^C}$ $(Z,Y,3)_t^C(\xrightarrow{a_t^C})$ $\ldots \xrightarrow{a_{t-3}^C} (Y,Y,1)_{t-2}^C \xrightarrow{b_{t-2}^C} (\mathbf{Y},\mathbf{Y},\mathbf{2})_{\mathbf{t-1}}^{\mathbf{C}} \xrightarrow{b_{t-1}^C}$ $(Z,Y,3)_t^C(\xrightarrow{b_t^C})$	a,b
$\vdots$	$\vdots$	$\vdots$	$\vdots$

MCLv0 will add both `vState(3)` and `tState(`X`)` as MFs to the test case's sub-run and those of the rest of the cases in the case base. However, the minimum number of iterations achieved was six, so MCLv0 will continue to search for additional MFs that will reduce this iteration count to one when combined with the current set of MFs, which currently consists of two MFs.

Thus, MCLv0 will extract a raw MF from its last iteration when including the two MFs that were just added to the awareness. Those iterations appear in Tables 5.15 and 5.16. Awarenesses in these sub-runs now take the form (State, `vState(3)`, `tState(`X`)`).

Table 5.16: MemTB, Recursion Level 2, Test Run Ending with State Z

Iter.	Test Elem	Candidate Cases	Solns.
$\vdots$	$\vdots$	$\vdots$	$\vdots$
4	b_{t-2}^{T}	$\ldots \xrightarrow{b_{t-3}^{C}} (Y,Y,1)_{t-2}^{C} \xrightarrow{\mathbf{b_{t-2}^{C}}} (Y,Y,2)_{t-1}^{C} \xrightarrow{b_{t-1}^{C}}$ $(Z,Y,3)_{t}^{C}(\xrightarrow{a_{t}^{C}})$ $\ldots \xrightarrow{a_{t-3}^{C}} (Y,Y,1)_{t-2}^{C} \xrightarrow{\mathbf{b_{t-2}^{C}}} (Y,Y,2)_{t-1}^{C} \xrightarrow{b_{t-1}^{C}}$ $(Z,Y,3)_{t}^{C}(\xrightarrow{b_{t}^{C}})$	a,b
5	$(Y,Y,1)_{t-2}^{T}$	$\ldots \xrightarrow{b_{t-3}^{C}} (\mathbf{Y},\mathbf{Y},\mathbf{1})_{\mathbf{t-2}}^{\mathbf{C}} \xrightarrow{b_{t-2}^{C}} (Y,Y,2)_{t-1}^{C} \xrightarrow{b_{t-1}^{C}}$ $(Z,Y,3)_{t}^{C}(\xrightarrow{a_{t}^{C}})$ $\ldots \xrightarrow{a_{t-3}^{C}} (\mathbf{Y},\mathbf{Y},\mathbf{1})_{\mathbf{t-2}}^{\mathbf{C}} \xrightarrow{b_{t-2}^{C}} (Y,Y,2)_{t-1}^{C} \xrightarrow{b_{t-1}^{C}}$ $(Z,Y,3)_{t}^{C}(\xrightarrow{b_{t}^{C}})$	a,b
6	b_{t-3}^{T}	$\ldots \xrightarrow{\mathbf{b_{t-3}^{C}}} (Y,Y,1)_{t-2}^{C} \xrightarrow{b_{t-2}^{C}} (Y,Y,2)_{t-1}^{C} \xrightarrow{b_{t-1}^{C}} (Z,3)_{t}^{C}(\xrightarrow{a_{t}^{C}})$ $[\ldots \xrightarrow{\mathbf{a_{t-3}^{C}}} (Y,Y,1)_{t-2}^{C} \xrightarrow{b_{t-2}^{C}} (Y,Y,2)_{t-1}^{C} \xrightarrow{b_{t-1}^{C}}$ $(Z,Y,3)_{t}^{C}(\xrightarrow{b_{t}^{C}})]$	a

After six iterations, MemTB creates a raw MF storing the feature f = Action, value $v = b$,

number of time steps back $k = 3$. Then, MemTB converts the raw MF into a value-back MF `vAction(3)`. Adding this MF reduces the number of MemTB iterations to one, as seen in Table 5.17. The perceptions take the form (State, `vState(3)`, `tState(`X`)`, `vAction(3)`).

Table 5.17: MemTB, Recursion Level 2, Test Run Ending with State Z, Value-Back MF

Iter.	Test Elem	Candidate Cases	Solns.
1	$(Z,Y,3,b)_t^T$	$\dots \xrightarrow{b_{t-3}^C} (Y,X,1,b)_{t-2}^C \xrightarrow{b_{t-2}^C} (Y,Y,2,b)_{t-1}^C \xrightarrow{b_{t-1}^C}$ $(\mathbf{Z},\mathbf{Y},\mathbf{3},\mathbf{b})_\mathbf{t}^\mathbf{C}(\xrightarrow{a_t^C})$ $[\dots \xrightarrow{a_{t-3}^C} (Y,X,1,b)_{t-2}^C \xrightarrow{b_{t-2}^C} (Y,Y,2,a)_{t-1}^C \xrightarrow{b_{t-1}^C}$ $(\mathbf{Z},\mathbf{Y},\mathbf{3},\mathbf{a})_\mathbf{t}^\mathbf{C}(\xrightarrow{b_t^C})]$ $[\dots \xrightarrow{b_{t-3}^C} (Y,Y,2,b)_{t-2}^C \xrightarrow{b_{t-2}^C} (Y,Y,3,b)_{t-1}^C \xrightarrow{b_{t-1}^C}$ $(\mathbf{Z},\mathbf{Y},\mathbf{4},\mathbf{b})_\mathbf{t}^\mathbf{C}(\xrightarrow{a_t^C})]$ $[\dots \xrightarrow{b_{t-3}^C} (Y,Y,2,b)_{t-2}^C \xrightarrow{b_{t-2}^C} (Y,Y,3,b)_{t-1}^C \xrightarrow{b_{t-1}^C}$ $(\mathbf{Z},\mathbf{Y},\mathbf{4},\mathbf{b})_\mathbf{t}^\mathbf{C}(\xrightarrow{b_t^C})]$ $[\dots \xrightarrow{b_{t-3}^C} (X,Z,3,a)_{t-2}^C \xrightarrow{b_{t-2}^C} (Y,X,1,b)_{t-1}^C \xrightarrow{b_{t-1}^C}$ $(\mathbf{Z},\mathbf{Y},\mathbf{2},\mathbf{b})_\mathbf{t}^\mathbf{C}(\xrightarrow{a_t^C})]$ $[\dots \xrightarrow{a_{t-3}^C} (X,Z,3,b)_{t-2}^C \xrightarrow{a_{t-2}^C} (Y,X,1,b)_{t-1}^C \xrightarrow{b_{t-1}^C}$ $(\mathbf{Z},\mathbf{Y},\mathbf{2},\mathbf{a})_\mathbf{t}^\mathbf{C}(\xrightarrow{b_t^C})]$	a

However, a time-back MF may be equally advantageous. The value-back MF is then removed from the test case sub-run and those of the other cases in the case base. Then, MCL will create a time-back MF `tAction(`b`)`, add it to the test case sub-run and to all other cases in the case base, and run MemTB. The first iteration is shown in in Table 5.18. Here, the surviving candidate cases in this first iteration do not have matching solution components. TempTB will require more than one iteration to resolve the temporal conflicts with this MF set. It is evident that the value-back

MF is more useful than the time-back MF in reducing the number of MemTB iterations, so the rest of the iterations involving the time-back MF are not shown.

Table 5.18: Memory TB Iterations for Test Run Ending with State Z, Time-Back MF

Iter.	Test Elem	Candidate Cases	Solns.
1	$(Z, Y, 3, 1)_t^T$	$\ldots \xrightarrow{b_{t-3}^C} (Y, X, 1, 1)_{t-2}^C \xrightarrow{b_{t-2}^C} (Y, Y, 2, 1)_{t-1}^C \xrightarrow{b_{t-1}^C}$ $(\mathbf{Z}, \mathbf{Y}, \mathbf{3}, \mathbf{1})_{\mathbf{t}}^{\mathbf{C}}(\xrightarrow{a_t^C})$ $\ldots \xrightarrow{a_{t-3}^C} (Y, X, 1, 3)_{t-2}^C \xrightarrow{b_{t-2}^C} (Y, Y, 2, 1)_{t-1}^C \xrightarrow{b_{t-1}^C}$ $(\mathbf{Z}, \mathbf{Y}, \mathbf{3}, \mathbf{1})_{\mathbf{t}}^{\mathbf{C}}(\xrightarrow{b_t^C})$ $[\ldots \xrightarrow{b_{t-3}^C} (Y, Y, 2, 1)_{t-2}^C \xrightarrow{b_{t-2}^C} (Y, Y, 3, 1)_{t-1}^C \xrightarrow{b_{t-1}^C}$ $(\mathbf{Z}, \mathbf{Y}, \mathbf{4}, \mathbf{1})_{\mathbf{t}}^{\mathbf{C}}(\xrightarrow{a_t^C})]$ $[\ldots \xrightarrow{b_{t-3}^C} (Y, Y, 2, 1)_{t-2}^C \xrightarrow{b_{t-2}^C} (Y, Y, 3, 1)_{t-1}^C \xrightarrow{b_{t-1}^C}$ $(\mathbf{Z}, \mathbf{Y}, \mathbf{4}, \mathbf{1})_{\mathbf{t}}^{\mathbf{C}}(\xrightarrow{b_t^C})]$ $[\ldots \xrightarrow{b_{t-3}^C} (X, Z, 3, 1)_{t-2}^C \xrightarrow{b_{t-2}^C} (Y, X, 1, 1)_{t-1}^C \xrightarrow{b_{t-1}^C}$ $(\mathbf{Z}, \mathbf{Y}, \mathbf{2}, \mathbf{1})_{\mathbf{t}}^{\mathbf{C}}(\xrightarrow{a_t^C})]$ $[\ldots \xrightarrow{a_{t-3}^C} (X, Z, 3, 2)_{t-2}^C \xrightarrow{a_{t-2}^C} (Y, X, 1, 1)_{t-1}^C \xrightarrow{b_{t-1}^C}$ $(\mathbf{Z}, \mathbf{Y}, \mathbf{2}, \mathbf{1})_{\mathbf{t}}^{\mathbf{C}}(\xrightarrow{b_t^C})]$	a
⋮	⋮	⋮	⋮

MCL will thus terminate its logic in the procedure `extract-mem-feats` (Algorithm 3) for the test case with following sub-run

$$\text{test run T}: \ldots \xrightarrow{b_{t-5}^T} Z_{t-4}^T \xrightarrow{b_{t-4}^T} X_{t-3}^T \xrightarrow{b_{t-3}^T} Y_{t-2}^T \xrightarrow{b_{t-2}^T} Y_{t-1}^T \xrightarrow{b_{t-1}^T} Z_t^T \xrightarrow{???}$$

and it will extract following MFs

- `vState(3)`: The value of *State* at time step $t-3$.

- `tState(X)`: The number of time steps before time step t when *State* stored the value X.

- `vAction(3)`: The value of *Action* at time step $t-3$.

Table 5.19: Extracted Memory Features, XYZ Example

Case	Problem	Soln	Extracted MFs
0	$\ldots \xrightarrow{a^C_{t-1}} X^C_t$	a^C_t	`vAction(1)` `tAction(a)`
1	$\ldots \xrightarrow{b^C_{t-1}} X^C_t$	b^C_t	`vAction(1)` `tAction(b)`
2	$\ldots \xrightarrow{a^C_{t-1}} Y^C_t$	b^C_t	None
3	$\ldots \xrightarrow{b^C_{t-1}} Y^C_t$	b^C_t	None

4	$\ldots \xrightarrow{b^C_{t-5}} Z^C_{t-4} \xrightarrow{b^C_{t-4}} X^C_{t-3} \xrightarrow{b^C_{t-3}} Y^C_{t-2} \xrightarrow{b^C_{t-2}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} Z^C_t$	a^C_t	`vState(3)`
			`tState(X)`
			`vAction(3)`

5	$\ldots \xrightarrow{b^C_{t-5}} Z^C_{t-4} \xrightarrow{a^C_{t-4}} X^C_{t-3} \xrightarrow{a^C_{t-3}} Y^C_{t-2} \xrightarrow{b^C_{t-2}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} Z^C_t$	b^C_t	`tAction(a)`

6	$\ldots \xrightarrow{b^C_{t-5}} X^C_{t-4} \xrightarrow{b^C_{t-4}} Y^C_{t-3} \xrightarrow{b^C_{t-3}} Y^C_{t-2} \xrightarrow{b^C_{t-2}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} Z^C_t$	a^C_t	`vAction(4)`
			`tAction(b)`
			`vState(3)`

7	$\ldots \xrightarrow{a^C_{t-5}} X^C_{t-4} \xrightarrow{a^C_{t-4}} Y^C_{t-3} \xrightarrow{b^C_{t-3}} Y^C_{t-2} \xrightarrow{b^C_{t-2}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} Z^C_t$	b^C_t	`tAction(a)`

8	$\ldots \xrightarrow{a^C_{t-5}} Y^C_{t-4} \xrightarrow{b^C_{t-4}} Z^C_{t-3} \xrightarrow{b^C_{t-3}} X^C_{t-2} \xrightarrow{b^C_{t-2}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} Z^C_t$	a^C_t	`vState(2)`
			`tState(X)`
			`vAction(2)`

9	$\ldots \xrightarrow{b^C_{t-5}} Y^C_{t-4} \xrightarrow{b^C_{t-4}} Z^C_{t-3} \xrightarrow{a^C_{t-3}} X^C_{t-2} \xrightarrow{a^C_{t-2}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} Z^C_t$	b^C_t	`vAction(2)` `tAction(a)`

Up to this point, we have demonstrated how MCL extracts MFs for cases with the following test

runs, which were selected to demonstrate MCLv0's capabilities in a variety of runs:

- $\ldots \xrightarrow{a_{t-1}} X_t \xrightarrow{a_t}$

- $\ldots \xrightarrow{b_{t-1}} X_t \xrightarrow{b_t}$

- $\ldots Y_t \xrightarrow{b_t}$

- $\ldots \xrightarrow{b_{t-5}} Z_{t-4} \xrightarrow{b_{t-4}} X_{t-3} \xrightarrow{b_{t-3}} Y_{t-2} \xrightarrow{b_{t-2}} Y_{t-1} \xrightarrow{b_{t-1}} Z_t \xrightarrow{a_t}$

A similar procedure can be used to extract the MFs for the other cases in our example. (Walking through this procedure for the other cases would be repetitive.) The MFs for all 10 example cases are shown in Table 5.19. These MFs will be refined in our ongoing example in Section 5.3.

5.3   Memory Feature Refinement

This section describes how MCL, given a list of MFs for every case in a case base as generated by the Memory Feature Extraction step, will refine the MFs into a minimal subset that can be applied to all cases in the case base. We first describe the algorithm behind the Memory Feature Refinement step. Then, we demonstrate this step in our ongoing example from the XYZ domain.

5.3.1   Memory Feature Refinement Algorithm

The implementation of the Memory Feature Refinement step in MCL is described in this section. The objective of the Memory Feature Refinement step is to isolate those MFs that appear consistently across multiple cases from multiple traces of origin in the case base. Such MFs are likely indicative of a *memory influence* that informs the observed behavior encoded by the case base.

During the Memory Feature Extraction step described in Section 5.2, various MFs may be extracted that are artifacts of a particular trace. Here, *artifacts* are aspects of the environment that coincide with a specific situation, but are irrelevant to the behavior at that situation. As an example from the car driving domain, suppose that there just happens to be a tree branch that fell on top of the stop sign when an observed human stops at the stop sign before crossing the intersection, just like the human is observed doing at any other stop sign. If MCL extracts a MF encoding the presence of the tree branch at this one particular intersection, then the "tree branch MF" would be an artifact because the tree branch has no bearing on the behavior of the human driver and it is present at none of the other observations of driving at stop signs. A MF encoding the presence of a *stop sign* would be more relevant to the car driving behavior at this situation. Therefore, the objective of the Memory Feature *Refinement* step is to filter out the MF artifacts from the MF set.

Algorithm 8 Memory Feature Refinement

 1: **procedure** MEM-FEAT-REFINE(trace-list, case-base, mem-feat-lists)
 2:     refined-mem-feats ← (empty list)
 3:     case-threshold ← 0.05   ▷ Min percent of cases a MF must appear in
 4:     trace-threshold ← 0.80   ▷ Min percent of traces a MF must appear in
 5:     unique-mem-feats ← `get-unique-mem-feats`(mem-feat-lists)
 6:     **for** each *mem-feat* in unique-mem-feats **do**
 7:         num-cases-appear ← 0
 8:         traces-appear ← (empty list)
 9:         **for** each *case* in case-base **do**
10:             tmp-mem-feat ← mem-feat-lists.`at`(*case*)
11:             **if** same-mem-feats(tmp-mem-feat, *mem-feat*) **then**
12:                 num-cases-appear ← num-cases-appear $+1$
13:                 **for** each *trace* in trace-list **do**
14:                     **if** *case*.trace-name equals *trace* **then**
15:                         **if** trace-appear.`contains`(*trace*) is `False` **then**
16:                             trace-appear.`add`(*trace*)
17:                             **break**
18:         case-percent ← num-cases-appear / case-base.`size`()
19:         trace-percent ← traces-appear.`size`() / trace-list.`size`()
20:         **if** case-percent $\geq$ case-threshold and trace-percent $\geq$ trace-threshold **then**
21:             refined-mem-feats.`add`(*mem-feat*)
22:     **return** refined-mem-feats

In MCL, the Memory Feature Refinement step is shown in Algorithm 8. It is called in the procedure `memory-composition-learning`, which was shown in Algorithm 2 in Section 5.1. Its inputs are the list of traces containing observations of the behavior, the case base of the observations created from those traces, and the list of case-specific MF sets from the prior Memory Feature Extraction step. As described in Section 5.2, a list of value-back and/or time-back MFs is extracted for each case, *independent* of other cases. Therefore, MCL must take this list of MF sets and refine it into a set of MFs that is applicable to *all* cases (i.e. the behavior as a whole). The Memory Feature Refinement step can be summarized by the following substeps.

1. Get a list of MFs from every case-specific MF set. Make sure there are no duplicate MFs.

2. For each MF:

    (a) Compute the percentage of *cases* in which the MF appears.

    (b) Compute the percentage of *traces* in which the MF appears.

    (c) If the case-percentage and trace-percentage coverages exceed the coverage thresholds for the case base and trace set, respectively, add the MF to the set of refined MFs.

3. Return the set of refined MFs.

From the procedure `mem-feat-refine` in Algorithm 8, we can see that substep 1 is accomplished with a call to the procedure `get-unique-mem-feats`, shown in Algorithm 9. In order to compare MFs (to ensure that one is not a copy of the other), the procedure `get-unique-mem-feats` calls the procedure `same-mem-feats` (shown in Algorithm 10) and passes two MFs to compare. As described in Section 4.5.2, MFs are either of type "Time-Back" or 'Value-Back". Just like a perception feature, a MF will store a potentially different return value for each time step t in a trace. However, independent of t, we define MF *equivalence* as follows:

140

- Two time-back MFs are equivalent if they track the same feature f and the same value v of f.

- Two value-back MFs are equivalent if they track the same feature f at time step $t - k$ for the same constant k.

Algorithm 9 Get Unique Memory Features

 1: **procedure** GET-UNIQUE-MEM-FEATS(mem-feat-lists)
 2:     unique-mem-feats ← (empty set)
 3:     **for** each *mf-list* in mem-feat-lists **do**
 4:         **for** each *mem-feat* in *mf-list* **do**
 5:             found ← False
 6:             **for** each *uniq-mf* in unique-mem-feats **do**
 7:                 **if** `same-mem-feats`(*uniq-mf*, *mem-feat*) is `False` **then**
 8:                     found ← `True`
 9:                     **break**
10:             **if** found is `False` **then**
11:                 unique-mem-feats.*add*(*mem-feat*)
12:     **return** unique-mem-feats

Algorithm 10 Memory Feature Comparison

 1: **procedure** SAME-MEM-FEATS(mem-feat1, mem-feat2)
 2:     **if** mem-feat1.*mem-feat-type*.`equals`(mem-feat2.*mem-feat-type*) is `False` **then**
 3:         **return** `False`
 4:     **if** mem-feat1.*feat-name*.`equals`(mem-feat2.*feat-name*) is `False` **then**
 5:         **return** `False`
 6:     **if** mem-feat1.*mem-feat-type*.`equals`("`Time-Back`") **then**
 7:         **if** mem-feat1.*value*.`equals`(mem-feat2.*value*) is `False` **then**
 8:             **return** `False`
 9:     **if** mem-feat1.*mem-feat-type*.`equals`("`Value-Back`") **then**
10:         **if** mem-feat1.*time-look-back*.`equals`(mem-feat2.*time-look-back*) is `False` **then**
11:             **return** `False`
12:     **return** `True`

For substep 2a, the for-loop in line 9 of the `mem-feat-refine` procedure (Algorithm 8) iterates through each case to check its list of MFs for an equivalent copy of a specific unique MF (specified

in the for-loop on line 6); if so, the variable `num-cases-appear` is incremented by one. For substep 2b, the for-loop in line 13 of the procedure `mem-feat-refine` adds the trace of origin of the case to a list of traces that the unique MF appears in. For substep 2c, the case coverage percentage and the trace coverage percentage are computed in lines 18-19 of procedure `mem-feat-refine` and are compared to the thresholds set on lines 3-4 in the same procedure. For MCL, these thresholds are hard-coded (we discuss the selection of these threshold values later). Finally, the unique MFs with sufficient coverage in the set of traces and cases are compiled and returned in line 22 of the procedure `mem-feat-refine`, according to substep 3.

The Memory Feature Refinement step uses two thresholds: a case coverage threshold and a trace coverage threshold. These are hyperparameters that are tuned for each domain and set by hand. The case coverage threshold is the minimum percentage of cases in the case base in whose extracted MF lists (generated by the Memory Feature Extraction stage) a MF must appear in order to be considered applicable to a *significant* proportion of the observed behavior encoded by the case base. The trace coverage threshold is the minimum percentage of traces in the trace set for which a MF must appear at least once in order to be considered applicable to a *significant* proportion of unique scenarios.

The use of two thresholds (a case coverage threshold and a trace coverage threshold) will yield four categories of MFs:

- *Blip Artifact*: This MF passes neither threshold. Its existence indicates either an aberration in the prior MF Extraction step, described in Section 5.2, or an artifact that occurs once or twice in a very small fraction of the traces and that likely does not inform the observed behavior as a whole because of its low frequency.

- *Rare Memory Influence*: This MF passes the trace coverage threshold, but not the case coverage threshold. This indicates that the encoded memory influence is present in the majority

142

of traces, but occurs so infrequently in a trace that including the memory influence as a member of the refined MF set would be spurious and unnecessarily comprehensive; such memory influences are still potentially important, but were excluded in the experiments for MCLv0 and will not be investigated in our research.

- *Trace Artifact*: This MF passes the case coverage threshold, but not the trace coverage threshold. This means that a particular memory influence occurred frequently in a minority of the traces and is likely a result of the intricacies of a particular trace scenario and not the behavior itself. This is based on the assumption that a memory influence will exert itself on a behavior in a wide variety of circumstances because it is inherent to the behavior, not a particular situation.

- *True Memory Influence*: This MF passes both the case coverage threshold and the trace coverage threshold. This means that the MF encodes a memory influence that occurs in the majority of trace scenarios and it occurs frequently enough in those scenarios to be helpful in a significant percentage of the cases. This MF will be included in the set of refined MFs.

In MCLv0, the case and trace coverage thresholds are set to 0.05 and 0.80, respectively. These particular thresholds gave good results in the experiments described in Section 5.5. However, we note here that the case coverage threshold is relatively low while the trace coverage threshold is relatively high. The case coverage threshold is low because certain memory influences may indeed only affect behavior at a single time step to signify a single decision (e.g. a Rhumba vacuum selecting a new direction when bumping into a wall based on memory of its last known location). However, such memory influences should occur enough times to comprise at least a minimal portion of a behavior's decisions, which we set at 5%. The trace coverage threshold, in contrast, is much higher because we assume that the salient aspects of a behavior, and the essence of its reliance on memory, will remain constant no matter what situational circumstances a behaving

entity finds itself in. Thus, MCLv0 assumes that a memory influence will manifest itself in most trace scenarios, at least 80% according to our threshold. We will challenge this assumption in our upcoming work, to be discussed in Chapter 6.

We now proceed to illustrate the Memory Feature Refinement step using our ongoing example from the XYZ domain.

5.3.2   Memory Feature Refinement Example

The Memory Feature Extraction step resulted in a list of MFs for each case that made the problem component sub-run of the case Markovian. Table 5.20 shows the extracted MFs for each case in our example; it has the same contents as Table 5.19 from the end of Section 5.2.

Table 5.20 shows the extracted MFs for all *unique* cases, but the Memory Feature Refinement process uses *all* cases, even redundant ones, to identify the True Memory Influence MFs that were described in Section 5.3.1. Therefore, we need to specify how many times each of the unique cases occurs in our case base. Let us assume that the state Y repeats itself at most three times consecutively and that the case base is created from ten traces, each 100 time steps in duration for a total of 1000 cases. For our example, the unique cases shown in Table 5.21 comprise all possible cases. Table 5.21 also shows the frequency of each unique case in the case base and the trace set. (Note: For mathematical simplicity in this example, we round frequencies to the nearest percentage, resulting in only 970 total cases in this example.)

As described in Section 5.3.1, the first step is to get a list of unique MFs from the list of case-specific extracted MF sets. The parameters of the MFs are encoded in feature name f (which is either "State" or "Action" in our example) and the parameter in parentheses: a time-look-back constant k for value-back MFs (which are prefixed by a v) or a value v of feature f for time-back

144

MFs (whose names are prefixed by a `t`). The set of unique MFs will not be equivalent to each other, as described in Section 5.3.1.

Table 5.20: Extracted Memory Features, XYZ Example

Case	Problem	Soln	Extracted MFs
0	$\ldots \xrightarrow{a^C_{t-1}} X^C_t$	a^C_t	`vAction(1)` `tAction(`a`)`
1	$\ldots \xrightarrow{b^C_{t-1}} X^C_t$	b^C_t	`vAction(1)` `tAction(`b`)`
2	$\ldots \xrightarrow{a^C_{t-1}} Y^C_t$	b^C_t	None
3	$\ldots \xrightarrow{b^C_{t-1}} Y^C_t$	b^C_t	None

4	$\ldots \xrightarrow{b^C_{t-5}} Z^C_{t-4} \xrightarrow{b^C_{t-4}} X^C_{t-3} \xrightarrow{b^C_{t-3}} Y^C_{t-2} \xrightarrow{b^C_{t-2}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} Z^C_t$	a^C_t	`vState(3)`
			 `tState(`X`)`
			 `vAction(3)`

5	$\ldots \xrightarrow{b^C_{t-5}} Z^C_{t-4} \xrightarrow{a^C_{t-4}} X^C_{t-3} \xrightarrow{a^C_{t-3}} Y^C_{t-2} \xrightarrow{b^C_{t-2}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} Z^C_t$	b^C_t	`tAction(`a`)`

6	$\ldots \xrightarrow{b^C_{t-5}} X^C_{t-4} \xrightarrow{b^C_{t-4}} Y^C_{t-3} \xrightarrow{b^C_{t-3}} Y^C_{t-2} \xrightarrow{b^C_{t-2}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} Z^C_t$	a^C_t	`vAction(4)`
			 `tAction(`b`)`
			 `vState(3)`

7	$\ldots \xrightarrow{a^C_{t-5}} X^C_{t-4} \xrightarrow{a^C_{t-4}} Y^C_{t-3} \xrightarrow{b^C_{t-3}} Y^C_{t-2} \xrightarrow{b^C_{t-2}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} Z^C_t$	b^C_t	`tAction(`a`)`

8	$\ldots \xrightarrow{a^C_{t-5}} Y^C_{t-4} \xrightarrow{b^C_{t-4}} Z^C_{t-3} \xrightarrow{b^C_{t-3}} X^C_{t-2} \xrightarrow{b^C_{t-2}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} Z^C_t$	a^C_t	`vState(2)`
			 `tState(`X`)`
			 `vAction(2)`

9	$\ldots \xrightarrow{b^C_{t-5}} Y^C_{t-4} \xrightarrow{b^C_{t-4}} Z^C_{t-3} \xrightarrow{a^C_{t-3}} X^C_{t-2} \xrightarrow{a^C_{t-2}} Y^C_{t-1} \xrightarrow{b^C_{t-1}} Z^C_t$	b^C_t	`vAction(2)` `tAction(`a`)`

Table 5.21: XYZ Example, Proportions in Case Base

Case	Problem	Soln	Extracted MFs	Case %	Num Cases	Num Traces
0	$\dots \xrightarrow{a} X$	a	`vAction(1)` `tAction(a)`	12.0%	120	10
1	$\dots \xrightarrow{b} X$	b	`vAction(1)` `tAction(b)`	12.0%	120	10
2	$\dots \xrightarrow{a} Y$	b	None	12.0%	120	10
3	$\dots \xrightarrow{b} Y$	b	None	37.0%	370	10
4	$\dots \xrightarrow{b} Z \xrightarrow{b} X \xrightarrow{b} Y \xrightarrow{b} Y \xrightarrow{b} Z$	a	`vState(3)` `tState(X)` `vAction(3)`	4.0%	40	10
5	$\dots \xrightarrow{b} Z \xrightarrow{a} X \xrightarrow{a} Y \xrightarrow{b} Y \xrightarrow{b} Z$	b	`tAction(a)`	4.0%	40	10
6	$\dots \xrightarrow{b} X \xrightarrow{b} Y \xrightarrow{b} Y \xrightarrow{b} Y \xrightarrow{b} Z$	a	`vAction(4)` `tAction(b)` `vState(3)`	4.0%	40	10
7	$\dots \xrightarrow{a} X \xrightarrow{a} Y \xrightarrow{b} Y \xrightarrow{b} Y \xrightarrow{b} Z$	b	`tAction(a)`	4.0%	40	10
8	$\dots \xrightarrow{a} Y \xrightarrow{b} Z \xrightarrow{b} X \xrightarrow{b} Y \xrightarrow{b} Z$	a	`vState(2)` `tState(X)` `vAction(2)`	4.0%	40	10
9	$\dots \xrightarrow{b} Y \xrightarrow{b} Z \xrightarrow{a} X \xrightarrow{a} Y \xrightarrow{b} Z$	b	`vAction(2)` `tAction(a)`	4.0%	40	10

Table 5.22: XYZ Example, Memory Feature Analysis

ID	Mem Feat	Cases	Traces	Case %	Trace %	Include
0	vAction(1)	0, 1	All	24%	100%	YES
1	vAction(2)	8, 9	All	8%	100%	YES
2	vAction(3)	4	All	4%	100%	NO
3	vAction(4)	6	All	4%	100%	NO
4	vState(2)	8	All	4%	100%	NO
5	vState(3)	4, 6	All	8%	100%	YES
6	tAction(a)	0, 5, 7, 9	All	24%	100%	YES
7	tAction(b)	1, 6	All	16%	100%	YES
8	tState(X)	4, 8	All	8%	100%	YES

Table 5.22 shows the list of unique MFs extracted. For each unique MF, the table also shows the cases and traces the MF appears in, its case and trace coverage percentages, and whether it passes the coverage thresholds for inclusion in the refined MF set. The list of unique MFs are as follows:

- MFs 0-3: value-back MFs tracking the Action taken at time steps $t-1$, $t-2$, $t-3$, and $t-4$, respectively.

- MFs 4-5: value-back MFs tracking the value of the feature State (the perception) at time steps $t-2$ and $t-3$, respectively.

- MFs 6-7: time-back MFs tracking the number of time steps before time step t when actions a and b, respectively, were last taken.

- MF 8: time-back MF tracking the number of time steps before time step t when State last stored value X.

All MFs pass the trace coverage threshold of 80%, but MFs 2-4 do not pass the case coverage threshold of 5%. Therefore, the memory influences encoded by MFs 2-4 did not affect a significant portion of the behavior as a whole and will not be included in the final refined MF set.

One of the advantages of using MCL is that the memory influences are explicitly enumerated. We can tell much from which MFs were retained in the final refined MF set, which MFs were filtered out during Memory Feature Refinement, and which MFs were never extracted in the previous Memory Feature Extraction stage. We analyze all of these now:

- MFs that were extracted, then retained in the refined MF set:

  1. Time-back MFs for both actions a and b: This makes sense, because the proper action to take often depended on *when* that action (or its opposite) occurred previously.

  2. Value-back MFs for Action at $t - 1$ and $t - 2$: Here, we see the importance of looking back a fixed number of time steps back to see *which action* occurred, in order to inform the next action to take at time step t.

  3. Time-back MF for state X: The perception of X, which immediately prompts a repeat of the action from the prior time step, was found by MCLv0 as a *shortcut* for determining the action taken when the perception Z last occurred (because X perceptions immediately follow Z perceptions). This helped in a couple Z cases, which depend on this information.

- MFs that were extracted, but *not* retained for the final refined MF set:

  1. Value-back MFs for Action at time steps $t - 3$ and $t - 4$: These MFs are only useful for Z cases that involve longer Y chains (sequences of consecutive perceptions of Y), which represent a smaller proportion of the case base. In contrast, value-back MFs for

148

Action at time steps $t - 1$ and $t - 2$ (see item 2 above) are useful for both Z cases with short Y chains and X cases in addition to Z cases with longer Y chains.

2. Value-back MF for State perception at time step $t - 2$. This MF is only useful for Z cases with a single Y perception preceding Z where action a is not observed after X (case 8). Furthermore, this MF does not help the ubiquitous X cases, which only have to look one time step back into the past.

- MFs that were never extracted during Memory Feature Extraction:

1. Value-back MF for State at time step $t - 1$: It is notable that this MF was never extracted because its absence indicates that all value-back memory influences based on perception exist at least two time steps back.

2. Time-back MFs for perceptions Y and Z. The former is never informative because a perception of Y is followed by an arbitrary number of Y perceptions and always evokes action b. The latter is also unnecessary because a perception of X always follows a Z perception and utilizes the same action taken when Z is perceived; that is why `tState(X)` was added as part of the refined MF set.

3. Value-back MFs tracking either State or Action for time steps before $t - 4$: state Y can never repeat itself more than three times consecutively for this example. The absence of these MFs indicates that memory influences before the most recent Z perception do not exist.

The above insights about the behavior of the XYZ agent are fully realized because the memory influences were explicitly encoded in the MFs that MCL extracted, then refined. The insight about the timing of the previous X perception is particularly surprising, given that the human-made rules governing the XYZ agent (which MCL does not have access to) codify action selection when Z is perceived according to the action taken at the previous Z perception; MCL used the timing of

149

the previous X perception as a more efficient shortcut for codifying the same memory influence, because the X perception exists closer to the present than the prior Z perception.

At the end of the Memory Feature Refinement step, the following MFs are selected as the codification of memory influence on the behavior of the XYZ agent as a whole.

- `vAction(1)`

- `vAction(2)`

- `vState(3)`

- `tAction(`a`)`

- `tAction(`b`)`

- `tState(`X`)`

The exponentially large space of possible memory influences has been reduced to just six MFs, which will be incorporated into each of the 10 traces to augment them with memory in the next section.

5.4 Trace Enhancement

This section describes the final step of the MCLv0 approach. Given the final set of MFs that codify all significant memory influences on an observed behavior, the Trace Enhancement step computes the return values of each MF for every time step of every trace. The final memory-enhanced traces will then contain all pertinent information from memory at each time step so that the sun-run contained therein is as close to Markovian as possible. The MFs will be indistinguishable from

normal perception features to any application (e.g. a machine learning algorithm) that uses the memory-enhanced traces.

Algorithm 11 Trace Enhancement

1: **procedure** ENHANCE-TRACE(trace, refined-mem-feats)
2:     **for** $t = 0$ to trace.`size()` **do**
3:        **for** each *mem-feat* in refined-mem-feats **do**
4:           mem-feat-name $\leftarrow$ mem-feat.*mem-feat-name*
5:           feat-name $\leftarrow$ mem-feat.*feat-name*
6:           **if** mem-feat.*type* is "Value-Back" **then**
7:               time-back $\leftarrow$ mem-feat.*k-steps-back*
8:               feat-val $\leftarrow$ `null`
9:               **if** $t \geq$ time-back **then**
10:                  feat-val $\leftarrow$ trace.`at`(time-back).`value-of`(feat-name)
11:               trace.`at`(t).`value-of`(mem-feat-name) $\leftarrow$ feat-val
12:           **if** mem-feat.*type* is "Time-Back" **then**
13:               time-back $\leftarrow$ `null`
14:               feat-val $\leftarrow$ mem-feat.*feat-val*
15:               tmp-time-back $\leftarrow$ 1
16:               **while** $t -$ tmp-time-back ≥ 0 **do**
17:                  tmp-feat-val $\leftarrow$ trace.`at`(tmp-time-back).`value-of`(feat-name)
18:                  **if** tmp-feat-val $=$ feat-val **then**
19:                     **break**
20:                  tmp-time-back $\leftarrow$ tmp-time-back $- 1$
21:               **if** tmp-time-back > -1 **then**
22:                  time-back $\leftarrow$ tmp-time-back
23:               trace.`at`(t).`value-of`(mem-feat-name) $\leftarrow$ time-back
24:     trace.`write-to-file`()
25:     **return**

Algorithm 11 contains the trace enhancement process. The procedure `enhance-trace` is called for each trace in the procedure `mem-comp-learn` in Algorithm 2 in Section 5.1. The basic steps of this process are as follows:

1. For each time step in the trace, do the following:

   - For a value-back MF, the feature name f and number of time steps k to look before

time step t are fixed. Store the value v of f at time step $t - k$ as the value stored for the value-back MF at time step t.

- For a time-back MF, the feature name f and the value v of f for which to search before time step t are fixed. Iterate from time step $t - 1$ to 0 for a time step $t - k$ when the value of feature f is v. Store k as the value of the time-back MF at time step t.

2. After the values of every MF for every time step in the trace have been computed, write the memory-enhanced trace to file.

Table 5.23: XYZ Example, Trace Sample

Time	State	Action
0	X	a
1	Y	b
2	Z	a
3	X	a
4	Y	b
5	Y	b
6	Z	b
7	X	b
8	Y	b
9	Y	b
10	Y	b
11	Z	a
$\vdots$	$\vdots$	$\vdots$

We now proceed with illustrating the Trace Enhancement process for our ongoing example in the XYZ domain. Suppose we have the trace shown in Table 5.23. The Trace Enhancement process will be applied using the following MFs that were extracted in Section 5.2.3, then refined in Section 5.3.2.

- vAction(1): value of Action at time step $t - 1$

152

- vAction(2): value of Action at time step $t - 2$

- vState(3): value of State perception at time step $t - 3$

- tAction(a): number of time steps before t when Action was a

- tAction(b): number of time steps before t when Action was b

- tState(X): number of time steps before t when State was X

Section 4.5.2 illustrated by detailed example the process of adding time-back and value-back MFs to a trace. The same process is applied here for each MF. The end results are shown in Table 5.24. (We substitute Ac for Action and St for State to save space.)

Table 5.24: XYZ Example, Memory-Enhanced Trace Sample

Time	State	Action	vAc(1)	vAc(2)	vSt(3)	tAc(a)	tAc(b)	tSt(X)
0	X	a	-	-	-	-	-	-
1	Y	b	a	-	-	1	-	1
2	Z	a	b	a	-	2	1	2
3	X	a	a	b	X	1	2	3
4	Y	b	a	a	Y	1	3	1
5	Y	b	b	a	Z	2	1	2
6	Z	b	b	b	X	3	1	3
7	X	b	b	b	Y	4	1	4
8	Y	b	b	b	Y	5	1	1
9	Y	b	b	b	Z	6	1	2
10	Y	b	b	b	X	7	1	3
11	Z	a	b	b	Y	8	1	4
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

With the creation of memory-enhanced traces, we have provided information at each time step in these traces that encodes the influences of memory. These traces can then be used by one of several applications, such as behavioral analysis or agent control applications.

153

5.5 Assessment of MCLv0 Approach

This section describes how we evaluated whether MCLv0, our initial implementation of the MCL algorithm, functioned as intended and we present test results showing how it accomplishes its main purpose of capturing and modeling the influence of memory within a set of memory features that are automatically learned. This evaluation was also reported in our previous publication [87].

5.5.1 *The Vacuum Cleaner Domain*

MCLv0 was assessed using the vacuum cleaner domain (see Figure 5.3) from [58]. In this domain, a vacuum cleaner agent (green square) moves about a 2D grid world cleaning dirt patches (red squares) and bumping into various obstacles (gray rectangles), both of which are perceived by eight binary features, two per side indicating whether the closest object in that direction is dirt or an obstacle and whether that object is near (adjacent to the agent) or far. An agent can move up, down, left, right, or stand still.

The authors in [58] defined eight agents, each exhibiting a behavior that is Level 1, 2, or 3, and that is either deterministic, stochastic, or random. These agents are as follows:

- Fixed Sequence agent (SEQ, Level 1): repeatedly executes a fixed sequence of actions.

- Random agent (RD, Level 2): performs a random action at each time step.

- Straight Line agent (SL, Level 3): moves in a straight line until encountering an obstacle, then picks a new random direction and continues.

- Wall Follower agent (WF, Level 2): If there is a wall on the left, it follows that wall, otherwise it moves right.

154

- Zig Zag agent (ZZ, Level 3): Moves left until encountering an obstacle, then moves down one spot and reverses its horizontal direction. When it bumps into an obstacle when moving down, it continues the pattern moving up.

- Smart agents: The RD, SL, and WF agents have "smart" variations (SRD, SSL, and SWF, respectively) which perform similarly to their non-smart counterparts, but will instead move towards dirt if it is sighted in any given direction. The smart agents are the same LfO level as their non-smart counterparts.

$$
\begin{array}{ll}
x^1 = wall & \text{(element up)} \\
x^2 = far & \text{(distance)} \\
x^3 = dirt & \text{(element right)} \\
x^4 = far & \text{(distance)} \\
x^5 = obstacle & \text{(element down)} \\
X^6 = close & \text{(distance)} \\
x^7 = wall & \text{(element left)} \\
x^8 = close & \text{(distance)}
\end{array}
$$
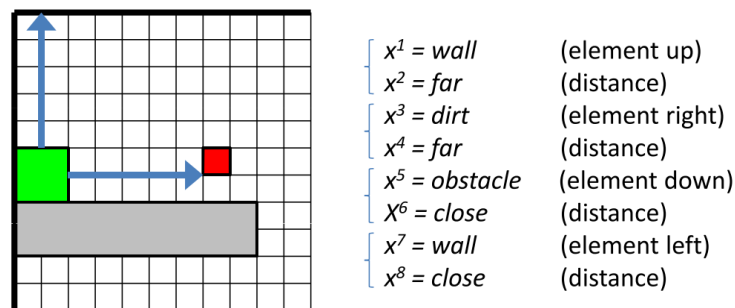
Figure 5.3: Vacuum cleaner world, reprinted with permission from [58].

The SEQ agent exhibits deterministic Level 1 behavior, which is purely a function of time. The WF and SWF agents exhibit deterministic Level 2 behaviors, which do not require memory. The ZZ agent exhibits deterministic Level 3 behavior, because the immediate short-term direction and long-term vertical direction must be remembered. The SL and SSL agents also exhibit Level 3 behavior, but it is non-deterministic. The RD and SRD agents exhibit Level 2 non-deterministic behavior.

5.5.2 Experiments and Results

This section describes the results of experiments in the vacuum cleaner domain that were used to evaluate our MCLv0 approach. These results were also presented in our previous publication [87].

Each of the vacuum cleaner agents had *raw* traces, each corresponding to one of seven maps, and each trace was 1000 time steps long. For each agent, we ran MCLv0 on the seven traces containing observations of that agent's behavior. Then, MCLv0 extracted MFs appropriate for summarizing information from memory for that agent's behavior and generated *memory-enhanced* traces containing the values of these MFs.

We fed each trace into the Weka machine learning framework [28] and used three classifiers: J.48 tree, Multilayer Perceptron, and Bayes Net. These three classifiers are standard supervised learning algorithms, which we use in this evaluation to demonstrate the effects of using the original memoryless traces and the memory-enhanced traces generated by MCL. (In contrast, the work in [58] used Dynamic Bayesian Networks, which consider internal state.) We collected results when Weka was fed the raw traces and when Weka was fed the memory-enhanced traces. Here, we present the F1-score for each agent's trace (raw or MCL-enhanced) for each classifier when 10-fold cross validation is used. The F1-score ensures that agents were not improperly considered to be high-performing when they only learned to predict the majority class (or action) to the detriment of minority classes [27]. The F1-score is computed for some class X as follows:

- TP: True Positive rate, the number of class X instances that were correctly predicted

- FP: False Positive rate, the number of class X instances that were not correctly predicted

- FN: False Negative rate, the number of instances not of class X that were incorrectly classified as class X

- P: Precision rate, the fraction of class X classifications that were correct: $TP/(TP + FP)$

- R: Recall rate, the fraction of class X instances that were correctly classified: $TP/(TP + FN)$

- $F1$: F1-score, the combination of precision and recall rates: $2PR/(P + R)$

The chosen classifiers are normally limited to learning Level 2 behaviors, but MCL provides these learning algorithms with information from memory via the learned memory features. The impact of MCL on learning performance for the Bayes Net, J.48 Decision Tree, and Multilayer Perceptron learners is shown in Figures 5.4, 5.5, and 5.6, respectively. The reported F1 score for a given agent is the average F1 score over all seven maps the agent behaved in. Also, results for the stochastic Bayes Net and Multilayer Perceptron learners were generated from a single trial rather than being averaged over multiple trials. Thus, all results are analyzed via absolute comparisons rather than through statistical analysis.
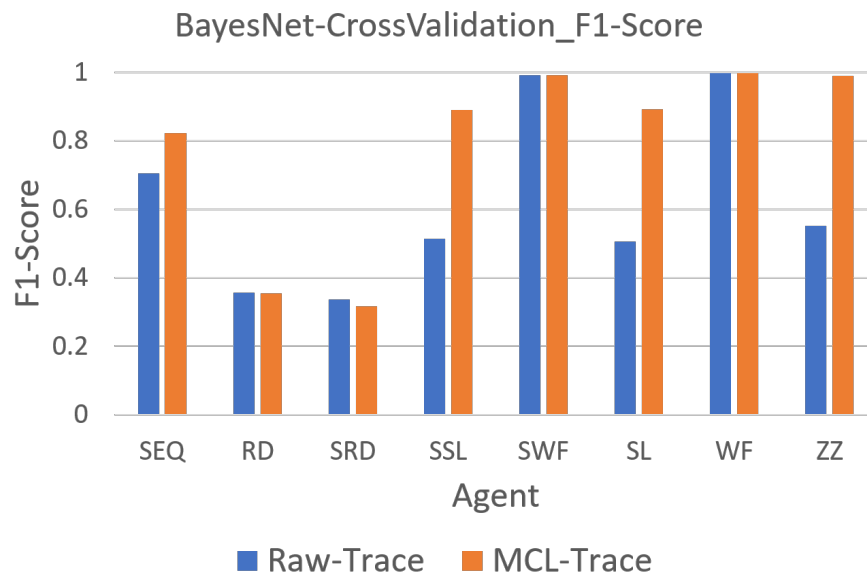
Figure 5.4: Bayes Net Learning Performance

157
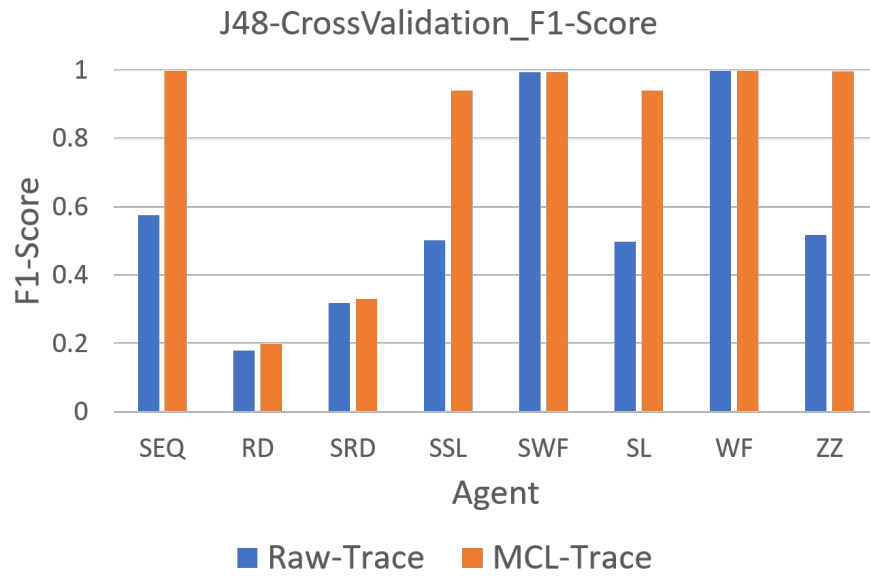
Figure 5.5: J.48 Learning Performance
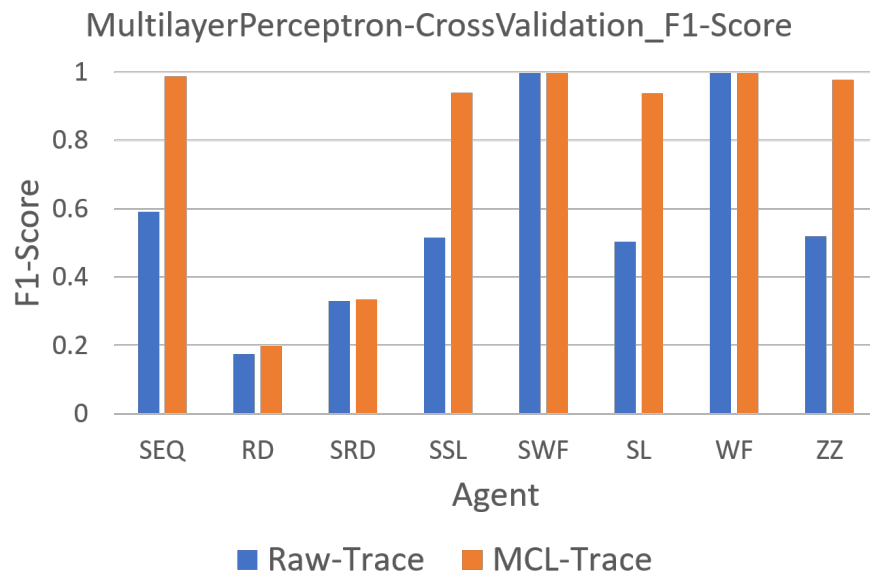
Figure 5.6: Multilayer Perceptron Learning Performance

Table 5.25: F1 Scores for MCLv0 Evaluation

Agent	Bayes Net		J48 Tree		MLP	
	Orig	MCL	Orig	MCL	Orig	MCL
SEQ	0.71	**0.82**	0.58	**1.00**	0.59	**0.99**
RD	0.35	0.35	0.17	0.20	0.18	0.20
SRD	0.35	0.32	0.30	0.33	0.33	0.33
SSL	0.53	**0.89**	0.52	**0.94**	0.52	**0.94**
SWF	0.99	0.99	0.99	0.99	1.00	1.00
SL	0.50	**0.89**	0.49	**0.94**	0.50	**0.94**
WF	1.00	1.00	1.00	1.00	1.00	1.00
ZZ	0.55	**0.99**	0.52	**1.00**	0.52	**0.98**

The specific F1 scores are shown in Table 5.25, where F1 scores for learning with MCL-generated traces are bolded if they exceed the corresponding F1 score for learning the original (memory-less) traces. For all three classifiers, the MCL-augmented traces for agents SEQ, SL, SSL, and ZZ resulted in a higher learning performance than their memory-less trace counterparts (with F1 scores that improved by at least 10%). There was no improvement for RD, SRD, WF or SWF because these behaviors do not depend on memory at all; there was also no performance degradation for these agents, with the exception of the SRD agent for the Bayes Net clasifier (degradation of only 3%). In other words, the memory-enhanced traces generated by MCL resulted in higher performance for classifying the behavior of agents with memory-dependent Level 3 behavior, both of stochastic and deterministic variants. It was also able to improve classification for the time-dependent Level 1 SEQ agent by learning MFs that encoded where the agent was in its 20-move action sequence. MCL traces did not result in better behavior for Level 2 agents (which did not rely on memory at all), but it did not harm their performance either. These trends are observed for all three machine learning algorithms. All these results are desirable and suggest that MCL has

worked as intended for improving machine learning performance of algorithms that were previously limited to learning up to Level 2 behaviors.

We now demonstrate how the learned MFs achieve the second advantage of MCL: explicitly enumerating the memory influences on agent behavior for human understanding. The perception features for the vacuum cleaner domain track the following for each cardinal direction: the sighted *element* (dirt or obstacle) and its *distance away* (far or close). The perception features are thus named: `ElemUp`, `DistUp`, `ElemRight`, `DistRight`, `ElemDown`, `DistDown`, `ElemLeft`, `DistLeft`. The MFs use the same naming convention used in Section 4.5.2. The MFs that were extracted and then refined for each agent are as follows:

- **SEQ**: Sequential Agent, Level 1 Deterministic Behavior

  - Time-Back MFs: `tDistUp(`*Far*`)`, `tAction(`*Down*`)`

  - Value-Back MFs: `vDistUp(1)`, `vDistUp(2)`, `vDistUp(4)`, `vAction(3)`, `vAction(5)`

  - Analysis: The only perception features that exert a memory influence are `ElemUp` and `Action`. The fixed action sequence is up, up, right, right, down, down, left, left, up, up, up, right, right, right, down, down, down, left, left, left, repeat.

- **RD**: Random Agent, Level 2 Non-Deterministic Behavior

  - Time-Back MFs: `tElemUp(`*obstacle*`)`, `tDistUp(`*Close*`)`, `tDistUp(`*Far*`)`, `tDistRight(`*Close*`)`, `tDistRight(`*Far*`)`, `tDistDown(`*Close*`)`, `tDistDown(`*Far*`)`, `tDistLeft(`*Close*`)`, `tDistLeft(`*Far*`)`, `tAction(`*Still*`)`, `tAction(`*Up*`)`, `tAction(`*Right*`)`, `tAction(`*Left*`)`, `tAction(`*Down*`)`

  - Value-Back MFs: `vDistUp(1)`, `vDistUp(2)`, `vDistUp(4)`, `vDistRight(2)`,

160

`vDistRight(2), vDistDown(2), vDistLeft(1), vDistLeft(2), vAc-`
`tion(1),vAction(2),vAction(3),vAction(4),vACtion(5),vAction(6)`

- Analysis: MCLv0 has no way of knowing that memory is useless for random behavior, so it learned 14 time-back MFs and 14 value-back MFs. All possible time-back MFs for `DistUp, DistRight, DistDown, DistLeft,` and `Action` were learned. Furthermore, various value-back MFs were learned, including those for `Action` up to six time steps back in the past. This might indicate a danger that MCLv0 will attempt to overfit memory influence.

• **SRD**: Smart Random Agent, Level 2 Non-Deterministic Behavior

- Time-Back MFs: `tDistUp(`*Close*`), tDistUp(`*Far*`), tDistRight(`*Close*`),`
`tDistRight(`*Far*`),tDistDown(`*Close*`),tDistDown(`*Far*`),tDistLeft(`*Close*`),`
`tDistLeft(`*Far*`),tAction(`*Up*`),tAction(`*Right*`),tAction(`*Left*`),tAc-`
`tion(`*Down*`)`

- Value-Back MFs: `vDistUp(1),vDistUp(2),vDistRight(2),vDistDown(1),`
`vDistDown(2),vDistLeft(1),vDistLeft(2),vAction(1),vAction(2),`
`vAction(3),vAction(4),vAction(5),vAction(6),vAction(7),vAc-`
`tion(8)`

- Analysis: MCLv0 similarly learned many MFs for SRD, which ultimately did not improve learning performance. Most of the time-back MFs and value-back MFs that were learned by MCLv0 for the SRD agent are the same as those learned for the RD agent.

• **SSL**: Smart Straight Line Agent, Level 3 Non-Deterministic Behavior

- Time-Back MFs: `tAction(`*Up*`), tAction(`*Right*`), tAction(`*Left*`), tAc-`
`tion(`*Down*`),DistUp(`*Far*`)`

- Value-Back MFs: `vDistDown(2),vAction(1)`

161

– Analysis: MCLv0 learned a small MF set for SSL. The MFs tracking the `Action` feature recorded which direction the cleaner agent was going in previous time steps, and thus were useful for improving the performance of machine learning when using the memory-enhanced traces. The MFs `vDistDown(2)` and `tDistUp(`*`Far`*`)` may be attempts by MCL to capture memory influence for the "smart" behavior of this agent, but we know that smart behavior is not influenced by memory, so it was an error on MCLv0's part.

- **SWF**: Smart Wall Follower Agent, Level 2 Deterministic Behavior

  – Time-Back MFs: None

  – Value-Back MFs: None

  – Analysis: No MFs were learned for SWF. This is good because SWF is not influenced by memory and any additional features that otherwise could have been added by MCL would have only confused learning.

- **SL**: Straight Line Agent, Level 3 Non-Deterministic Behavior

  – Time-Back MFs: `tAction(`*`Up`*`)`, `tAction(`*`Right`*`)`, `tAction(`*`Left`*`)`, `tAction(`*`Down`*`)`

  – Value-Back MFs: `vAction(1)`

  – Analysis: MCL learned MFs pertaining to just "Action". In the absence of "smart" behavior, SL is only influenced by the previous action taken. Intuitively, only Val-Action-1 is necessary, but MCL also added the time-back MFs because each individual MF extraction provided no evidence that one type was better than the other, so both types were added. Therefore, a more advanced version of MCL should further refine the MF set.

- **WF**: Wall Follower Agent, Level 2 Deterministic Behavior

  - Time-Back MFs: None

  - Value-Back MFs: None

  - Analysis: No MFs were learned for WF, which is the ideal situation because WF is not influenced by memory.

- **ZZ**: Zig Zag Agent, Level 3 Deterministic Behavior

  - Time-Back MFs: `tAction(`*Up*`)`, `tAction(`*Right*`)`, `tAction(`*Left*`)`, `tAction(`*Down*`)`

  - Value-Back MFs: `vAction(1)`, `vAction(8)`

  - Analysis: MCL learned MFs for "Action", including an interesting `vAction(8)` MF (storing the action taken eight time steps in the past), which is likely an artifact of the maps experienced. Otherwise, the learned MFs correspond to the intuitive notion that the Zig Zag agent must memorize its short-term horizontal direction (so that it moves in a straight line) and its long-term vertical direction (so that it knows whether to move up or down after bumping into the left or right wall).

The MCLv0 approach has generally succeeded in learning MFs that codify memory influences on each of the vacuum cleaner agents in a sensible fashion. Except for the random agents, the MF sets that were learned were small and may be further reduced as the algorithm is further refined in our research.

5.6 Minimal Conceptual Approach Summary

In this section, we described the MCLv0 approach to the MCL algorithm. The MCLv0 approach implements the essential components of the MCL algorithm which form its core mechanism. It is geared toward learning memory influences in discrete-valued behavioral traces and encoding them with appropriate memory features that are used to create memory-enhanced traces. We described each step in the MCLv0 approach in detail: Memory Feature Extraction, Memory Feature Refinment, and Trace Enhancement. Now that we have presented the MCL algorithm, as it is implemented in the MCLv0 approach and has been explained and illustrated by example up to this point, we are now ready to justify and describe improvements to the algorithm, which constitute our comprehensive conceptual approach.

CHAPTER 6: COMPREHENSIVE CONCEPTUAL APPROACH

In this chapter, we describe our full or "comprehensive" conceptual approach to the problem described in Chapter 3. We call this comprehensive approach "MCLvN". Here, we describe how MCLvN addresses the limitations in MCLv0, which we described in Chapter 5, and how MCLvN has been designed to handle the general class of memory-influenced behaviors.

This chapter is organized as follows:

- Section 6.1 summarizes the MCLv0 approach.

- Section 6.2 outlines the limitations of the MCLv0 approach.

- Section 6.3 provides a high-level description of the modifications present in MCLvN and how each of them addresses a specific limitation of MCLv0.

- Section 6.4 discusses preprocessing steps that were added to MCLvN to handle continuously-valued behavioral traces.

- Section 6.5 discusses improvements made to the Memory Feature Extraction phase in MCLvN.

- Section 6.6 discusses improvements made to the Memory Feature Refinement phase in MCLvN.

- Section 6.7 discusses improvements made to the Trace Enhancement phase in MCLvN.

6.1 Baseline Approach Synopsis

This section summarizes the MCLv0 baseline approach upon which MCLvN improves. It is organized as follows:

- Section 6.1.1 describes the output of MCLv0, Memory Features (MFs).

- Section 6.1.2 describes the overall MCL algorithm and its inputs and outputs.

- Section 6.1.3 describes the MF Extraction step of MCLv0.

- Section 6.1.4 describes the MF Refinement step of MCLv0.

- Section 6.1.5 describes the Trace Enhancement step of MCLv0.

This section will serve as a brief summary of Chapter 5 and may be skipped by those who have a sufficient understanding of that chapter's contents.

6.1.1   Memory Features

Memory Features (MFs) are features that encode aspects of an agent's past performance. There are raw MFs and specialized MFs. Raw MFs record an aspect of recorded performance at a single point in time and they have three components:

1. A feature f, which is one of the variables recording the perception of the environment.

2. A time step t in the range $[0, T]$, for some maximum number of time steps T.

3. A value v that is recorded for f at time t.

For example, suppose we have the trace seen Table 6.1, which records the values of two features, F and G, over five time steps such that time step $t \in \{0, 1, 2, 3, 4\}$.

One possible raw MF is one which records that feature F stored value - at time $t = 2$. We denote this as MF($F = $ -, $t = 2$). Another possible MF is one recording that feature G stored value x at

time $t = 3$, denoted MF($G = $ x, $t = 3$). One should note that these raw MFs are only applicable to one particular time step t.

Table 6.1: Example Trace with Two Features

t	0	1	2	3	4
F	+	-	-	+	+
G	a	b	z	x	y

Specialized MFs are raw MFs that are parameterized and formatted to store information about memory relative to any time step. These are MFs for which an appropriate value can be computed at every time step. There are two types: value-back and time-back.

Value-back MFs (VMFs) are MFs that store the *value* of some feature at a fixed time prior to the present. VMFs have two parameters, which are:

1. A feature f, which is one of the variables that records the perception of the environment.

2. A time-back constant tb, which is the number of time steps prior to some time step t at which value v was recorded for feature f for this MF.

Thus, for a given time step t, a VMF stores the value v of feature f at time $t - tb$. We call it a "value-back" MF because it stores a value from memory or a *value back in time*.

Time-back MFs (TMFs) are MFs that store the amount of time since the last occurrence of some value of a feature. TMFs have two parameters, which are:

1. A feature f, which is one of the variables recording the perception of the environment.

2. A fixed value v of feature f.

Thus, for a given time step t, a TMF stores the number of time steps tb before time t when value v for feature f was last observed. It is a called a "time-back" MF because it stores the amount of *time back* to the most recent observation of a specific feature value.

For example, suppose we have a sequence of values for a single feature F over five time steps (where time t takes on values 0-4), as seen in Table 6.2:

Table 6.2: Example Trace with One Feature

t	0	1	2	3	4
F	+	-	-	+	+

If we add a VMF that tracks the value of feature F at time $t - 1$, the values of this VMF, denoted $vF(1)$, are as follows (see Table 6.3):

Table 6.3: Example Trace with Value-Back Memory Feature

t	0	1	2	3	4
F	+	-	-	+	+
$vF(1)$		+	-	-	+

VMF $vF(1)$ stores the symbol from the immediately prior time step. At time $t = 0$, there is no such symbol, so $vF(1)$ stores nothing.

168

Table 6.4: Example Trace with Time-Back Memory Feature

t	0	1	2	3	4
F	+	-	-	+	+
$tF(\text{-})$			1	1	2

If we add a TMF tracking the number of time-back tb steps *before* time t since the last occurrence of the value '-' of feature F, the values of this TMF, denoted $tF(\text{-})$, are as follows (see Table 6.4):

The first occurrence of the value '-' for feature F is at time step $t = 1$. Therefore, TMF $tF(\text{-})$ stores nothing for times $t = 0$ and $t = 1$ because at those times, there is no prior occurrence of '-' before t.

6.1.2 *Memory Composition Learning, Version 0*

Memory Composition Learning (MCL) is an algorithm for learning a memory model that explains the memory influences for an observed behavior. A preliminary version of this algorithm, which we call MCLv0, was implemented using three major steps: 1) MF Extraction, 2) MF Refinement, 3) Trace Enhancement.

The input to MCL is a series of *traces* that record a single start-to-end *run* of the observed behavior. A *run* is a set of variables (including agent actions) and their values up to a certain point in time. (The actual definition of a run is more nuanced and is described in Chapter 5, but this definition suffices for this section.) A perception is a set of features or variables that encode the state of the environment at a single point in time. Actions are variables that encode what an acting agent did, possibly in reaction to the perceived state of the environment, at a point in time. Thus, for a trace with N time steps, there are N runs (the run up to time 0, the run up to time 1, etc.).

After one or more traces recording an actor's observed behavior are obtained, the traces are converted into a *case base*. The environmental variables are then copied over to the cases themselves, because the cases replace the traces for the purposes of learning a memory model of the observed behavior. Each case in the case base has two components:

- *Problem Component*: This component stores a run of the observed behavior up to and including the values of the perception features at a given time step t.

- *Solution Component*: This component stores the action(s) taken by the observed actor at time t.

Next, MCL automatically learns a *memory model* of the observed behavior to explain the memory influences at work. This memory model, which is a set of appropriate MFs, is incorporated into the original traces to create memory-enhanced traces that store all pertinent information from memory at each time step. The objective is to restore the Markov assumption — that all information needed to predict the action taken by the observed actor at a given time step is present in the values of the variables recorded for that time step. This would allow standard supervised learning algorithms to learn Level 3 memory-based behaviors because the influences of the actor's memory are reflected as part of the present state.

MCL does this through three steps:

1. Memory Feature Extraction: For each case in the case base, a set of MFs explaining the memory influences for that case is computed.

2. Memory Feature Refinement: Given the sets of MFs extracted for each case, a single *refined* set of MFs explaining the memory influences on the behavior as a whole is computed.

170

3. Trace Enhancement: The value of all MFs in the refined set is computed for each time step in each trace. These are incorporated into each corresponding trace to create memory-enhanced traces.

The following sections describe each of these steps in greater detail.

6.1.3 Baseline Memory Feature Extraction

MF Extraction is the process in MCLv0 by which a set of MFs is computed for each case in a case base. For each case, the set of extracted MFs for a given case encodes the memory influences at work for the problem component run in the case.

In MCLv0, MF Extraction is done *iteratively*. The objective of this step is to extract the memory influences present for each individual case. Therefore, each case gets its turn to be the *test case*, for which MCLv0 uses the following procedure to extract memory influences in the form of specialized MFs. MCLv0 repeats the following steps for each test case.

1. Instantiate the set of extracted MFs, which is initially an empty set. Copy every case in the case base (including the test case) to a pool of candidate cases.

2. Do the following until an exit condition is encountered:

   (a) Use Memory Temporal Backtracking (MemTB) to retrieve a raw MF from the earliest time step where the test case differed from other cases in the case base that prescribed a different action for time t.

   (b) If no raw MF was retrieved with MemTB because the perception features at time t were sufficient to predict the test case's solution component (the action at time t), then exit the loop and go to Step 3.

(c) Convert the raw MF into a VMF. Add it to copies of the test case and cases in the case base.

(d) Use MemTB with the copies of the test case and the case base with the VMF to retrieve a second raw MF. Note how many time steps later this second raw MF occurs than the first raw MF.

(e) Convert the first raw MF into a TMF. Add it to copies of the test case and cases in the case base.

(f) Use MemTB with the copies of the test case and the case base with the TMF to retrieve a third raw MF. Note how many time steps later this third raw MF occurs than the first raw MF.

(g) If the second raw MF (the one retrieved using the VMF) occurs later than the third raw MF (the one retrieved using the TMF), add the VMF to the set of extracted MFs and permanently add the VMF to the test case and case base.

(h) If the third raw MF (the one retrieved using the TMF) occurs later than the second raw MF (the one retrieved using the VMF), add the TMF to the set of extracted MFs and permanently add the TMF to the test case and case base.

(i) If both the second and third raw MFs occur at the same time step, include both the VMF and TMF in the set of extracted MFs and permanently add both to the test case and case base.

3. Return the set of extracted MFs.

In each iteration of step 2, two MFs (one value-back and one time-back) are created and evaluated — one or both will then be added to the set of MFs extracted for this case. If the problem component sub-run has T time steps, then at most $2T$ MFs can be extracted for this case, two per time step. If there are C cases in the case base, then the above procedure will be performed C times,

once per case. This is because each case in the case base will act as the test case one time, so that MCLv0 can extract specialized MFs for it. Thus, the end result will be C sets of MFs that are used as input in the following MF Refinement step, which is discussed next.

6.1.4 Baseline Memory Feature Refinement

MF Refinement (MFR) is the process in MCLv0 by which the sets of MFs extracted for each case are condensed into a single set of refined MFs that describe the memory influences on the observed behavior as a whole.

In MCLv0, MF Refinement assesses each MF *individually*. The MF Extraction process was done for each case independently to find the memory influences that help predict the solution component of each case. The MF Refinement process, in contrast, assesses the memory influences that are prevalent in an observed behavior as a whole (e.g. over all cases in the case base) in order to determine which MFs are most significant. It does so through these steps:

1. Instantiate the set of refined MFs, which is initially an empty set.

2. Get the set of *unique* MFs that were extracted for any case. (The same specialized MF may have been extracted for multiple cases, but such an MF is only processed once in the refinement stage.) For each of these MFs, do the following:

   (a) Compute the proportion α of cases for which this MF was extracted.

   (b) Compute the proportion β of traces for which this MF was extracted for at least one case derived from the trace.

   (c) Compare α and β to predefined thresholds τ and ρ, respectively. If $\alpha > \tau$ and $\beta > \rho$, add this MF to the set of refined MFs.

173

The purpose of the thresholds τ and ρ (which are tuned by hand for each domain) is to filter out *artifacts* or memory influences that are based on aspects of the environment, rather than aspects of the behavior. There are three artifacts:

1. *Blip Artifact*: This is a MF that appears in few cases ($\alpha < \tau$) and few traces ($\beta < \rho$) and is therefore meaningless; an aberration of the extraction process may have resulted from noise in the behavioral data.

2. *Trace Artifact*: This is a MF that appears in many cases ($\alpha > \tau$), but very few traces ($\beta < \rho$). It is memory of an aspect of the environment of certain traces rather than memory that affects the behavior in general in all traces.

3. *Rare Memory Influence*: This is a MF that appears in many traces ($\beta > \rho$), but in very few cases ($\alpha < \tau$). If is a memory influence that affects the behavior in general, but it occurs so infrequently that we do not retain it in our refined MF set.

Once all artifacts are filtered out and our refined MF set is obtained, we have a set of MFs that explain how memory influences the behavior as a whole. This is the input to the next step of MCLv0, Trace Enhancement.

6.1.5 Baseline Trace Enhancement

Trace Enhancement (TE) is the process in MCLv0 by which a set of memory-enhanced traces is created by incorporating the learned memory model of the observed behavior into the original traces.

The learned memory model is a set of MFs that explains the memory influences on the observed behavior as a whole. For each time step in each of the traces, the return value of each MF is

computed and incorporated into the original trace, thereby converting it into a memory-enhanced trace. The MFs in the enhanced traces can be treated the same as the originally-included perception features and action features by a standard supervised learning algorithm without modifying the learning algorithm. The hope is that all pertinent information from memory will be present at each time step, thereby asserting the Markov assumption that the information at the current time step is sufficient for predicting the action(s) taken at that time step.

6.2   Limitations of MCLv0 Approach

Chapter 5 presented MCLv0, a preliminary version of the Memory Composition Learning (MCL) algorithm. However, MCLv0 suffers from the following limitations:

1. *Context Agnosticism*: MCLv0 assumes that the influence of memory is consistent throughout the execution of the behavior; in reality, however, the usefulness of MFs and their relevance may rise and fall throughout the behavior's execution depending on the situation (context). MFs should thus be context-sensitive.

2. *Extraction Inefficiency*: MCLv0 is computationally expensive in the MF extraction phase and is unlikely to scale up well to domains with more complex perceptions.

3. *Inability to Handle Continuous Data*: MCLv0 only operates in finite, discrete domains. Future versions should be adapted to continuous domains and use more complex similarity metrics when comparing case runs.

4. *Extraneous Memory Representations*: MCLv0 currently only learns two MFs (time-back and value-back), which are often generated unnecessarily when a more concise MF set could be used to express certain memory influences.

5. *Stochastic Considerations*: MCLv0 has no mechanism for addressing random or stochastic behavior when generating its learned MF set. Learning from completely random agents, for example, can wreak havoc on the MCLv0 process, as was shown in Section 5.5.

MCLv0's purpose was to show that the idea behind the hypothesis of our research was both credible and feasible. However, MCLv0's shortcomings do not allow it to properly validate the hypothesis of our research (stated in Section 3.3). Either MCLv0 makes simplistic assumptions or does not account for certain aspects of behavior that would allow it to be generally applied to real-world human behaviors.

In order to prove that the concept of the MCL algorithm is applicable to domains and learning tasks in the general case, and that it is better than other techniques that learn memory-influenced behaviors, the MCLv0 approach must be enhanced to overcome all the limitations mentioned. These improvements are discussed next.

6.3   MCLvN Approach Overview

This section provides a high-level overview of the extensions to the MCLv0 approach that enable MCL to learn memory models in behaviors with continuous-valued data, complex memories, and contextual considerations. We call this modified approach "MCLvN", the ultimate comprehensive conceptual approach to the MCL algorithm presented in this dissertation. Figure 6.1 shows the final MCLvN approach. (It has similar contents to those shown in Figure 5.1 from Section 5.1 except with shaded boxes showing where new modules/functionality are to be located for MCLvN.) In Section 5.1, we discussed these modules (and improvements to preexisting modules from MCLv0).

We discuss here how these improvements of MCLvN are meant to address the limitations of MCLv0 here; other sections discuss the specific mechanics of the MCLvN components in greater

176

detail. Put another way, we justify the modifications/additions in this section and *why* we included them, but we describe in later sections *how* these updated modules work. Also, we do not claim that the design choices in this approach are optimal, but only that their synergistic combination adequately works to address the overall problem of inferring a model of memory influences at work in observed behavior as described in Chapter 3.
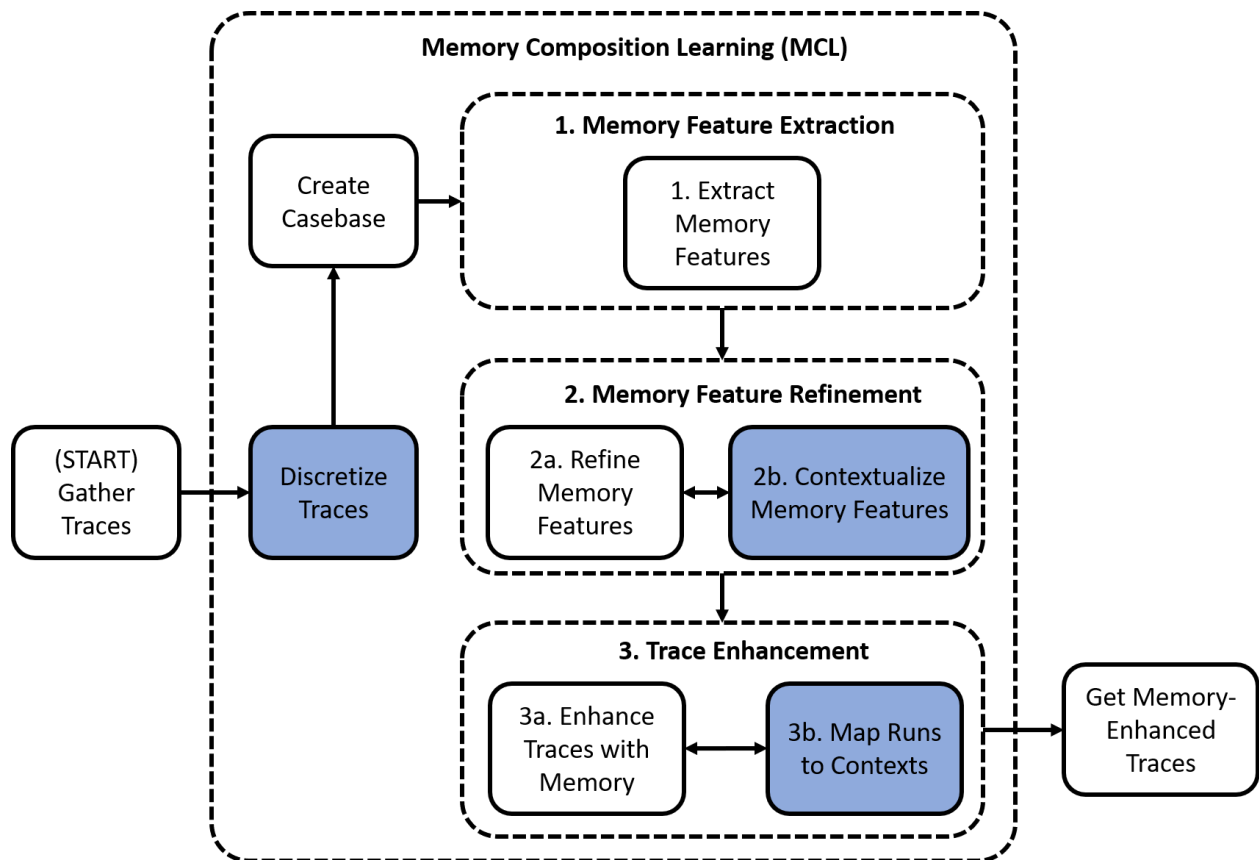
Figure 6.1: MCL Comprehensive Approach

6.3.1   MCLvN Preprocessing Overview

Before the major phases of MCL are invoked, we added a "discretizer" module to preprocess continuous-valued features in behavioral traces. This module will convert all continuous-valued

features into discrete values. Then, all feature values can be processed by MCL without regard to the range and distributions in the data. This would address the "Inability to Handle Continuous Data" limitation discussed in Section 6.2.

The alternative was to rewrite all other modules to process continuous-valued data directly, which would essentially create a second MCL algorithm specifically geared toward continuous-valued data. However, there are three issues with this approach:

1. Having two versions of MCL makes it difficult to directly test the efficacy of various modules of MCL. There will always be two sets of tests and any differences in memory modeling performance could be attributed to the type of data processed rather than to the MCL modules themselves. It would also make it difficult to manage applications when there are two versions of the resulting tool.

2. The Memory Temporal Backtracking algorithm treats time steps as discrete units. Unless there is a feature tracking time itself (e.g. the number of seconds that have passed since the beginning of the simulation), there is no way to create time-back MFs that record a continuous amount of time (instead of a discrete number of time steps as in MCLv0).

3. MCL requires memory influences to repeatedly appear in order for such influences to be learned and modeled. High-precision memory values for a continuous-valued feature are less likely to be repeated (unless similar values are clustered) and would make learning value-back MFs difficult.

Discretization was seen as a better alternative to handling continuous-valued traces and expanding the MCLv0 approach.

6.3.2 MCLvN Memory Feature Extraction Overview

The updates that were intended for the Memory Feature Extraction phase in MCLvN are twofold:

1. Update Step 1 "Extract Memory Features" from Figure 6.1 to use a new principle, called the Memory Singularity Principle (MSP), to reduce redundant computations and make this phase more efficient. This will address the "Extraction Inefficiency" limitation. We discuss the MSP in greater detail later.

2. A consequence of using MSP is that the specialization of MFs (converting raw MFs into value-back or time-back MFs) is no longer possible in the MF Extraction stage and therefore must be transferred to the subsequent MF Refinement stage.

The algorithmic complexity of the MF Extraction phase in MCLv0 was determined to be O(N^4) in Appendix A, where N is the number of time steps in the observed behavior. This algorithmic complexity is derived in Appendix A. Nevertheless, O(N^4) is much too computationally intensive to be useful in real-world applications.

6.3.3 MCLvN Memory Feature Refinement Overview

The updates to the Memory Feature Refinement phase in MCLvN will be twofold:

1. Update Step 2a to generate specialized MFs that express the memory influences codified by the raw MFs that were extracted for each case in the case base. By waiting until the MF Refinement stage to generate specialized MFs, a more concise set of specialized MFs can be generated that more effectively expresses the memory influences on observed behavior. This addresses the "Extraneous Memory Representations" limitation.

179

2. Add Step 2b "Contextualize Memory Features" from Figure 6.1 to create memory-based contexts. Such contexts will encode sets of MFs that repeatedly occur together throughout a trace. This will address the "Context Agnosticism" limitation. The creation of these contexts will also address the "Stochastic Considerations" limitation because MFs that do not map to a specific context can be said to be a product of non-determinism (such as human inconsistency in behavior or noise) or some aspect of the environment that was not modeled in the trace and thus cannot be modeled by memory.

6.3.4 MCLvN Trace Enhancement Overview

The updates to the Trace Enhancement phase in MCLvN will be twofold:

1. Update Step 3a "Enhance Traces with Memory" to be able to add MFs to continuous traces. The MFs are generated from discretized traces and their values for each time step are computed with those same discretized traces. However, the MFs' values are now incorporated into the original continuously-valued traces.

2. Add Step 3b "Map Runs to Contexts" to write to file which segments of a trace map to which contexts. This will allow context-based LfO algorithms to know which segments of the traces to use for training knowledge for the generated contexts. This addresses the "Context Agnosticism" limitation.

These updates accommodate the outputs generated by the MF Refinement stage that are beyond the simple value-back and time-back MFs that were generated in MCLv0.

6.3.5   MCLvN Holistic Overview

We have provided a high-level description of the modifications/additions that have been made as part of MCLvN and how they address specific limitations in the MCLv0 approach. We list below the limitations of MCLv0 and which modules address them in MCLvN.

Table 6.5: How MCLvN address limitations in MCLv0

Limitation	Description	MCLvN modules
Context Agnosticism	Group memory influences by context.	Steps 2b, 3b
Extraction Inefficiency	Leverage repeated computations.	Step 1
Need for Discretization	Process continuous-valued data.	Discretizer module
Extra. Mem. Represent.	Make MFs more concise.	Step 2a
Stochastic Considerations	Handle non-determinism.	Step 2b

The following sections will now provide more detailed explanations for the mechanisms behind these MCLvN novelties.

6.4   Trace Preprocessing

This section addresses how MCLvN addresses continuously-valued data. In this step, MCLvN converts every continuous value of a trace into a discretized value. In order to do this, the following information needs to be specified for each feature:

- A base value B — the value to which other values are compared. For example, for a feature measuring a vehicle's speed, we would set $B = 0$ because we want to know how many units of speed above 0 a vehicle is going.

- A range size R — the maximum difference that two feature values can have and still correspond to the same discrete value (e.g. if $R = 0.2$, then values 5.2 and 5.1 will be the same value when discretized).

Then, for a given feature F with base value B and range size R, we can convert a continuous value V observed for F into a discretized value $D = \frac{V-B}{R}$. It should be noted that this formula uses integer division, because D must assume an integer value. Thus, D represents the number of units of size R away from B that V is. This formulation uses a uniform distribution, but future work may use a different distribution (e.g. Gaussian) for discretization. Now, we discuss the mechanism behind MF Extraction in MCLvN.

6.5   Improved Memory Feature Extraction

This section describes the improved MF Extraction step of MCLvN. The improvements in this step address two limitations in the MCLv0 version:

1. The iterative MF extraction algorithm made three calls to MemTB in order to procure a single MF. This is grossly inefficient.

2. The iterative MF extraction algorithm made the simplistic assumption that the best specialized MF (value-back or time-back) was the one that resulted in the retrieval of raw MFs that are more recent. This resulted in extracted MF sets that were at times not concise.

Given a *test-run,* a *pool* of all cases in the casebase, and *k* to 0. Call MemTB+.

Initialize set of *extracted-MFs* to an empty set. Initialize *rejected-cases* set to empty set.

Create a *new-raw-MF* storing the most discriminative feature, its value, and *k*.

Yes ← Have all cases been compared to the test-*run*? → *No* Load next case from *pool*.

Do all cases in *pool* agree on action for time *t*?

Yes → Add *new-raw-MF* to *extracted-MFs* set. Return *extracted-MFs.*

Yes ← Is this case similar to the *test-run* at time *t-k*.

No

Remove this case from the *pool*. Add case to *rejected-cases* set.

No → Recursively call MemTB+ with (*k*+1). Obtain a list of *earlier-MFs* from time steps before (*t-k*-1).

Is the *rejected-cases set* empty?

No → Do the MFs in *earlier-MFs* resolve all cases in *pool* that have a different solution component than the test-*run?* → *No* Add *earlier-MFs* and *new-raw-MF* to *extracted-MFs* set. Return *extracted-MFs* set.

Yes

Return set of *earlier-MFs*. ← *Yes*

Figure 6.2: Memory Temporal Backtracking Plus — Flowchart

The improved MF Extraction step of MCLvN, rather than *iteratively* retrieving a *single* MF after every three calls to MemTB), it does so *recursively* by retrieving *all* MFs from just a single call to MemTB for a given case. In other words, in MCLv0, if M MFs are to be retrieved for a given case, MemTB would be called $3M$ times. However, in MCLvN, the extraction process can retrieve M MFs with just one MemTB invocation, that reduces the run-time complexity of the MF Extraction step significantly.

The MF Extraction step relies on what we call the *Memory Singulary Principle* (MSP), which states that under the assumption of perfect memory recall, if two elements in a run are the same, then the memories of those elements will also be the same. The corollary is that if two elements in a run are not the same, then the memories of those elements will also not be the same. In other words, there is a one-to-one mapping between a memory and the element it is "pointing" to, to use a programming analogy.

In order to leverage the MSP, the algorithm for MemTB is modified — we call it MemTB+. In this way, future improvements to MemTB+ can simply add additional + signs (e.g. MemTB++). Figure 6.2 shows the steps for MemTB+. Terminating events in the flowchart have a black outline.

We note here that a consequence of leveraging MSP is that the output of MF Extraction in MCLvN is *not* a set of specialized MFs for each case in the case base. Instead, MF Extraction outputs a set of *raw* MFs for each case in the case base. The specialization of these raw MFs into value-back or time-back MFs is thus transitioned to the MF Refinement stage in MCLvN.

We describe the steps in Figure 6.2 in detail below. Note that the MF Extraction+ procedure *iteratively* processes each case in the case base, but it processes an individual case *recursively*.

1. The inputs to MemTB+ are a *test-run*, for which MFs are being extracted, a *pool* of cases that are similar to *test-run*, and a time-back parameter k. The first call to MemTB+ will pass

all cases in the case base as the *pool* of similar cases and a time-back parameter $k = 0$. The value of k will be incremented as the recursion depth increases. The current time t of the *test-case* is known.

2. MemTB+ will create two empty sets:

   (a) *extracted-MFs*, which stores the extracted MFs that comprise the final output of MemTB+.

   (b) *rejected-cases*, which stores those cases that are removed from *pool* in a given MemTB+ call.

3. The following is done for each case in *pool*.

   (a) The current case is compared to the *test-case* at time $t - k$.

   (b) If the case is dissimilar to the *test-case* at time $t - k$, it is added to the *rejected-cases* set and removed from *pool*.

4. A *new-raw-MF* is created and it stores three components:

   (a) The *feature* for which the most cases remaining in *pool* are similar to *test-case* and for which the most cases in *rejected-cases* are dissimilar to *test-case* is selected as the feature recorded by the *new-raw-MF*.

   (b) The *value* of the discriminative feature is recorded.

   (c) The time $t - k$ is recorded.

5. If all *pool* cases share the same solution component, then a MF set with just the *new-raw-MF* will be returned. This MF was responsible for resolving all memory-influences for this MemTB+ iteration. This is the terminating case (a.k.a. termination of recursion).

6. Otherwise, a recursive call to MemTB+ will be made to obtain the list of *earlier-MFs* that resolve memory influences for time steps before time $t - k$. The parameters to this MemTB+

call will be the same *test-case*, the remaining cases in *pool*, and an incremented $k + 1$ parameter. In Figure 6.2, the flow would return to "Initialize set of *extracted-MFs* to an empty set. Initialize *rejected-cases* set to empty set."

7. In the original MemTB (no plus sign), the output of this recursive call to MemTB (a single raw MF) would have been returned. However, MemTB+ adds a few crucial post-processing steps, which are as follows:

    (a) If the *rejected-cases* set is empty, this means that no cases were removed from *pool*. This means that no memory influence was present at time $t - k$. Therefore, the *earlier-MFs* set is returned.

    (b) Otherwise, there is a memory influence that caused certain cases to be removed from *pool* at time $t-k$, namely the *new-raw-MF* that was created. However, it is possible that memory influences from *earlier-MFs* are sufficient for explaining memory influence up to time $t-k$, thus making *new-raw-MF* unnecessary. This is because of the MSP, which we explain later. For now, if *earlier-MFs* are sufficient for resolving (i.e. explaining) memory influences at time $t - k$, then just return *earlier-MFs*. Otherwise, add *earlier-MFs* and *new-raw-MF* to the *extracted-MFs* set and return *extracted-MFs*.

Following from the MSP, we can make the following assertions:

1. At time $t-k$, all cases in *pool* have similar elements to the *test-case*. Therefore, all *memories* between times $t - k$ and t are also similar to those of the *test-case* and cannot be used to differentiate between the *test-case* and other cases with different solution-components in that time range.

2. Therefore, in the absence of memory influences prior to $t-k$, any memory influences present at time $t - k$ are *absolutely necessary* for explaining why the action taken in the *test-case*

186

differs from that of other cases in *pool* at time $t - k$.

3. However, because of MemTB+'s recency bias (the assumption that more recent memories are more influential than earlier memories, hence the reason why TB and thus MemTB+ inspect cases from present to past), memory influences prior to $t - p$, for some $p < k$, are not considered when removing cases from *pool* at time $t - p$.

4. Therefore, it is possible that memory influences from times $t - k$ to $t - p - 1$ (all time steps prior to time $t - p$, but after and including $t - k$) may be sufficient to remove cases from *pool* with different solution components than that of the *test-case*. If this is the case, then any memory influences from time $t - p$ are unnecessary.

Furthermore, we should note that the at the end of the improved MF Extraction phase for MCLvN, the output is a set of *raw* MFs, not specialized MFs; MFs are not designed as value-back or time-back in the MF Extraction step for MCLvN. This is because the MSP only applies to raw MFs, not specialized MFs. Therefore, using the MSP has three important ramifications:

1. We can now guarantee that all extracted MFs are necessary, provided that they were correctly formed in the first place. This is because we no longer rely on the potentially faulty recency bias, allowing earlier MFs to potentially phase out or suppress memory influences that come later. This is in contrast to MCLv0, which relied on the possibly faulty assumption that selecting MFs that cause subsequent MFs to be more recent was the optimal procedure. There is currently no definitive evidence that this recency bias is appropriate for all domains.

2. MCLvN, by delaying MF specialization to a later stage, avoids the overhead of computing MF values for all cases in the case base. The MSP allows us to compare elements directly instead of first computing MF values and comparing those. This is because raw MFs have a one-to-one relationship with the elements being remembered/tracked.

3. Raw MFs cannot be included in memory-enhanced traces because they only record memory information for a single point in time. They cannot be adapted to other time steps. Thus, it is up to a subsequent stage, such as the MF Refinement+ step, to specialize these raw MFs. (In contrast, MCLv0 specialized its raw MFs during the MF Extraction stage.)

However, the limitation of the current MF Extraction algorithm we have described thus for MCLvN is that the extraction process is invoked for every case in the case base. With a few modifications to the algorithm, we can run the extraction process once and get the extracted raw MF lists for every case in the case base. Those modifications are as follows:

1. Instead of having MemTB+ return the set of extracted MFs to be assigned to a specific test case, just have each case in the case base update its own extracted MF set on the fly as the MF Extraction operation is being carried out. Every time a raw MF is created during recursion, the appropriate cases in the case base incorporate it into their raw MF sets immediately instead of waiting for the entire set of raw MFS to be computed. This eliminates the need for return statements during MF Extraction.

2. Instead of discarding cases that are dissimilar to the test case, extract the raw MFs those cases as well. Do this by arbitrarily picking one of those cases (it doesn't matter which one) as the new test case and running another MemTB+ operation from that point in the recursive call stack.

If one views each recursive call as a node in a tree, then MemTB+ is like traversing a tree. Each leaf node represents the end of recursion. Previously, the path to each leaf node began all the way from the root node (the first recursive call to MemTB+). Now, the path to each leaf node need only begin from the lowest node in the tree (the closest ancestor) that has more than one child node.

Given a *test-run,* a *pool* of all cases in the casebase, and *k* to 0. Call MemTB+.

Initialize set of *extracted-MFs* to an empty set. Initialize *rejected-cases* set to empty set.

If k is 0, assign each case an initially empty set of extracted MFs.

Create a *new-raw-MF* storing the most discriminative feature, its value, and *k*.

Yes ← Have all cases been compared to the *test-run*?

No → Load next case from *pool*.

Is this case similar to the *test-run* at time *t-k*.

Yes

No

Do all cases in *pool* agree on action for time *t*?

Yes → Add *new-raw-MF* to the extracted MFs set of each case in *pool.*

Remove this case from the *pool*. Add case to *rejected-cases* set.

Yes

No

Recursively call MemTB+ with (*k*+1). Obtain a list of *earlier-MFs* from time steps before (*t-k*-1).

[Extract for rejected cases]

Is the *rejected-cases set* empty?

No → Do the MFs in *earlier-MFs* resolve all cases in *pool* that have a different solution component than the *test-run?*

No → Add *new-raw-MF* to the extracted MFs set of each case in *pool.*

Yes

Done

Yes

Figure 6.3: Tree Memory Temporal Backtracking Plus — Flowchart

Figure 6.3 shows the modified extraction algorithm. The two major changes from Figure 6.2 are:

- The return statements are modified such that any new MF generated in that call is immediately assigned to all cases in the pool.

- The last task that is done in MemTB+ is a resolution of the rejected cases that were not similar the test case.

[Extract for rejected cases]
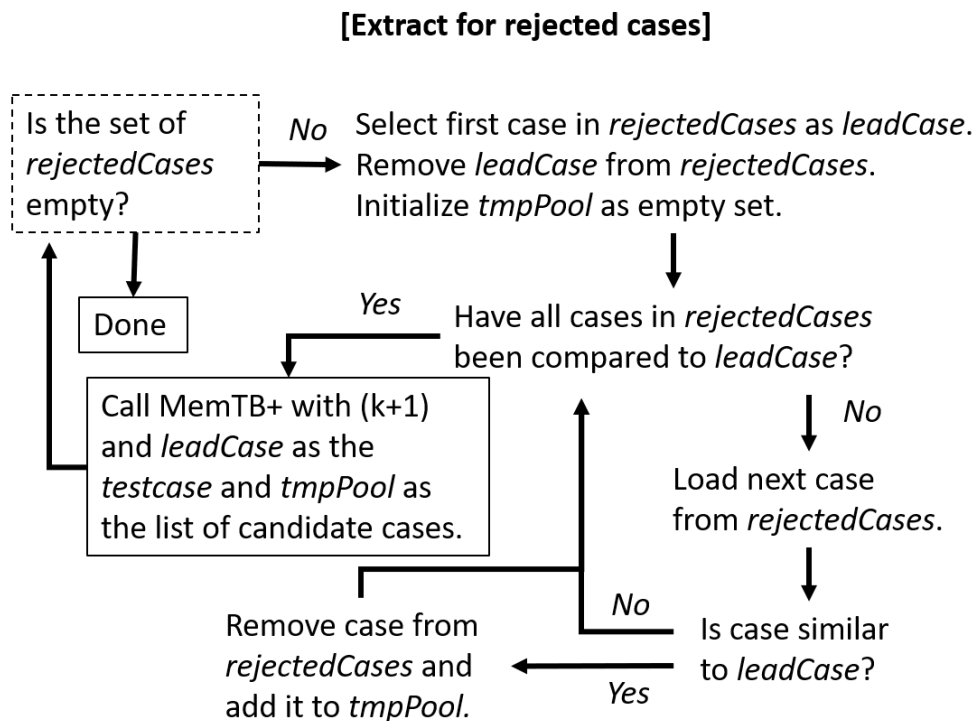
Figure 6.4: Tree Memory Temporal Backtracking Plus — Addressing Rejected Cases

The MF extraction process for the rejected cases is shown in Figure 6.4. Basically, while the set of rejected cases is not empty, the first case is chosen to be a lead case (a new test case) to which all rejected cases are compared. All cases that are similar to the lead case are put in a new set and removed from the rejected cases set. Then, MemTB+ is invoked on this new group of similar cases and the loop continues.

The algorithmic complexity of this new MemTB+ algorithm is O(N^2). Its derivation can be found in Appendix A. This is much more scalable to practical applications than its former O(N^4) complexity.

6.5.1 Comparison of MF Extraction Methods

The following section contains a graphical representation of how the different MF Extraction methods discussed in this dissertation compare to each other. The objective is to demonstrate how the MF Extraction procedure in MCLvN has been improved over the version in MCLv0.

6.5.1.1 MCLv0 Memory Feature Extraction

In MCLv0, the MF Extraction process is performed repeatedly, once for each case in the case base. Figure 6.5 shows the first iteration of MF Extraction for a single case.

Figure 6.5: MCLv0 MF Extraction — Iteration 1, Recursion Level 1

In this figure, cases are represented by miniature shapes. The test case (a solid circle in this example) is surrounded by square brackets. All cases that are possibly similar to the test case are encapsulated in a group with a solid outline — this constitutes the candidate cases pool. A single raw MF will be extracted by a call to MemTB. In its first level of recursion, MemTB will compare all cases to the test case at time t (where t for each case is the latest time step in that case's problem

component sub-run). Each case will be either be retained in the candidate cases pool or be rejected.
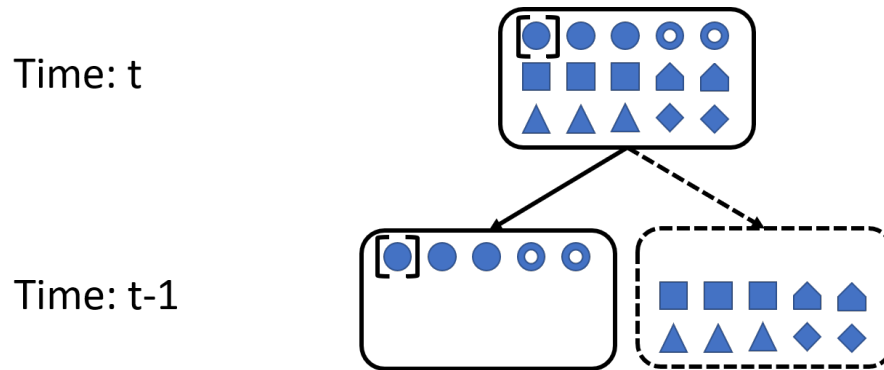
Figure 6.6: MCLv0 MF Extraction — Iteration 1, Recursion Level 2

Figure 6.6 shows the result of the case comparisons from the first MemTB recursion level. All dissimilar cases (non-circular shapes) have been rejected and set aside in a group with a dashed outline — they will no longer be considered by MemTB. All other cases that are still similar to the test case are retained in the candidate case pool. However, the cases in the candidate case pool do not agree on the next action to be taken at time step t (since not each case is the exact same shape), so MemTB will delve into a second level of recursion, comparing the cases in the candidate case pool with the test case at time $t - 1$. (Note: This process is more nuanced, as described in Chapter 5, but the simplification of the process serves the purposes of this example.)

Figure 6.7 shows the result of the case comparisons from the second MemTB recursion levels. The cases that were dissimilar from the test case were rejected and set aside in a group with a dashed outline while the other cases were retained in the candidate case pool. All cases in the candidate case pool now prescribe the same action at time t (since they are all the exact same shape, a solid circle), so MemTB terminates and a single raw MF is extracted. The raw MF is represented by a fuzzy cloud shape. This ends the first iteration of MF Extraction for the given test case. (Again, this is a simplification of the process described in Chapter 5.)

192
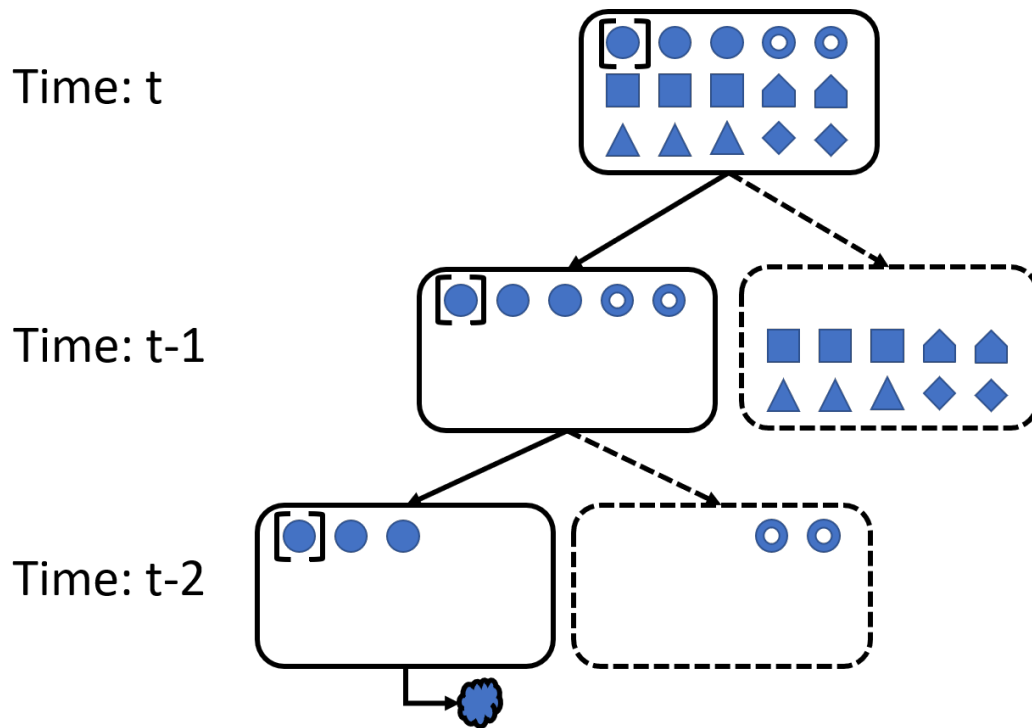
Figure 6.7: MCLv0 MF Extraction — Iteration 1, Recursion Level 3

In the second iteration of MF Extraction for the same test case, the raw MF that was extracted in the previous iteration is then converted into a value-back MF (VMF) and temporarily incorporated into the test case and all cases in the case base. The VMF is represented as a yellow cube. This is seen in Figure 6.8.

In Figure 6.8, all the cases are changed from a blue color to a yellow color to represent the incorporation of the VMF. (Unfortunately, this color change will not be reflected in black and white copies of this dissertation.) As a result of the VMF, we can see that MemTB only reaches a recursion level of 2 instead of 3.
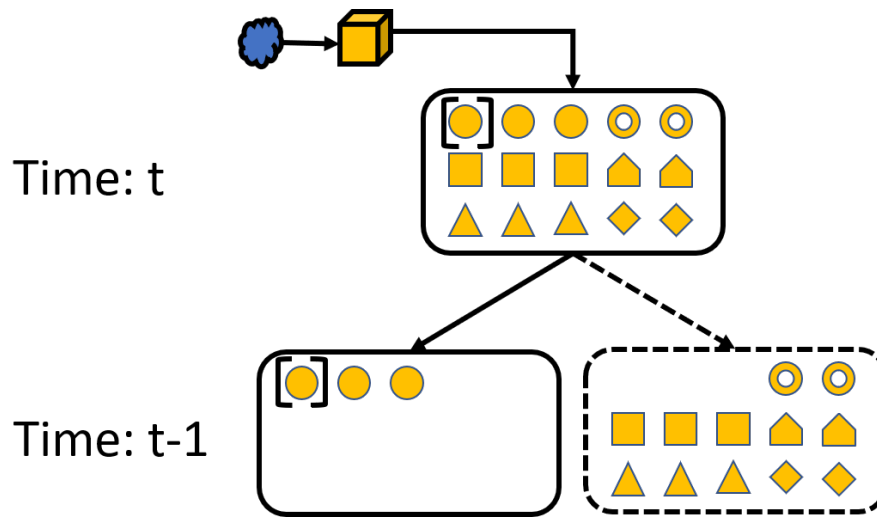
Figure 6.8: MCLv0 MF Extraction — Iteration 2, Testing VMF
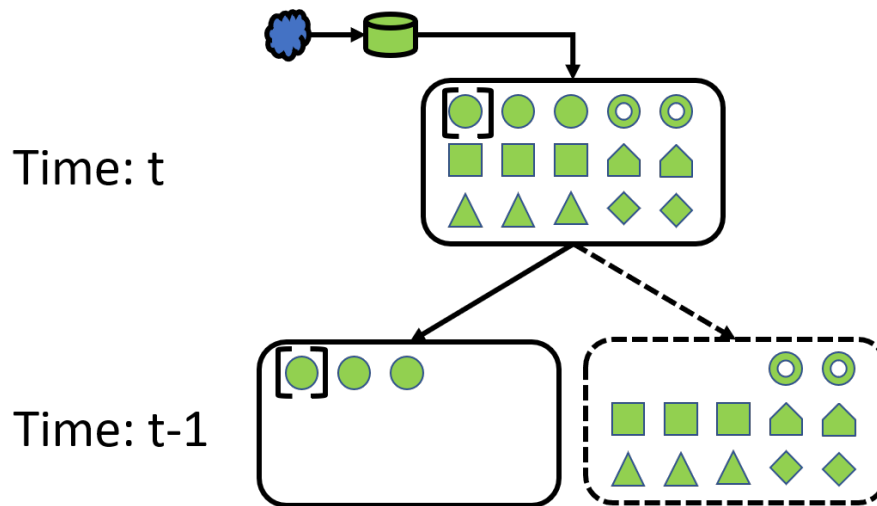
Figure 6.9: MCLv0 MF Extraction — Iteration 2, Testing TMF

The next step is to remove the VMF from the cases in the case base. The raw MF from iteration 1 is then converted into a time-back MF (TMF) and temporarily incorporated into the cases in the case base. The TMF is represented as a green cylinder shape. MemTB is run again, as seen in

194

Figure 6.9. In this figure, all the cases are green to signify the incorporation of the TMF.

As seen in Figure 6.9, when the TMF is used, MemTB also results in a recursion level of 2. There-fore, according to the recency bias that MCLv0 uses, both the VMF and TMF versions of the raw MF that was extracted in Iteration 1 are equally advantageous and thus they are both permanently incorporated into the cases of the case base (at least for the duration of the MF Extraction pro-cedure for this test case). Even though it would seem that having both the VMF and the TMF is redundant, they are both incorporated in order to avoid potentially filtering out an MF that is actually necessary.
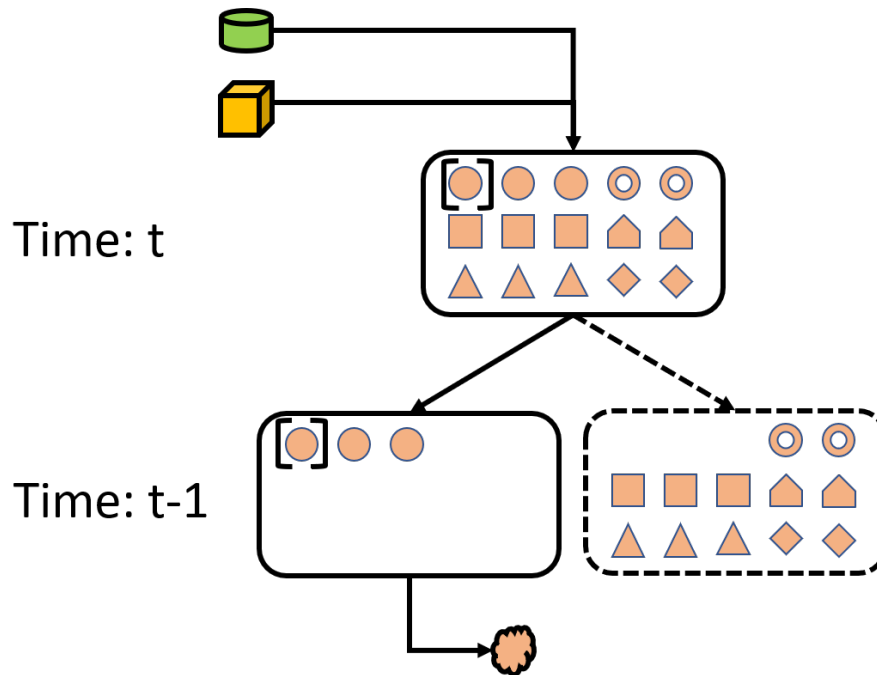
Figure 6.10: MCLv0 MF Extraction — Iteration 2, Extracting Raw MF

MemTB is then run a third time with this case base (which now contains two MFs in each case) in order to extract a second raw MF. This is shown in Figure 6.10. The VMF and TMF (the cube and cylinder) are permanently incorporated into the cases in the case base, which are now beige

195

color to signify the incorporation of both MFs. The result of this call to MemTB, a raw MF, is represented as a cloud shape in the figure. This concludes the second iteration of MF Extraction for the current test case.

The third iteration of MF Extraction for the same test case begins with the conversion of the raw MF from the second iteration into a VMF and its temporary incorporation into the cases in the case base. This is shown in Figure 6.11, where the VMF is a purple cube and all cases are shown as red to signify the incorporation of the new VMF. (Note: The cases already possess the TMF and VMF that were incorporated in the previous iteration).
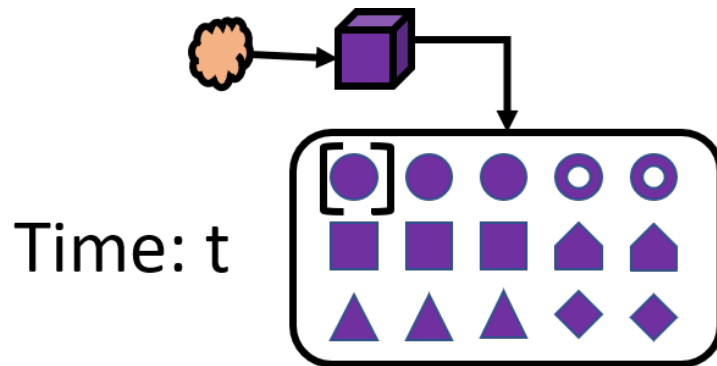
Figure 6.11: MCLv0 MF Extraction — Iteration 3, Testing VMF

In Figure 6.11, we can see MemTB terminates in its first level of recursion, which indicates that the MFs have captured all memory influences relevant to this case. In order to see if a TMF version of the second raw MF is useful, the raw MF that was extracted in the second iteration is converted into a TMF and temporarily added to the cases in the case base. This is shown by Figure 6.12, where the TMF is a red cube and all cases in the case base are red to signify the incorporation of the TMF.
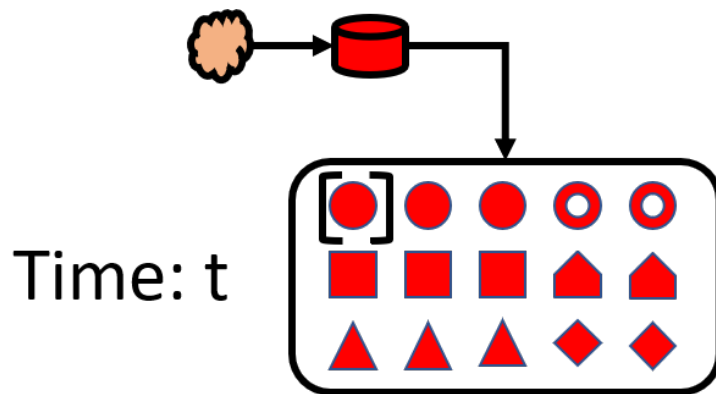
Figure 6.12: MCLv0 MF Extraction — Iteration 3, Testing TMF

As seen in Figure 6.12, MemTB terminates in its first level of recursion, indicating that the TMF and VMF versions of the raw MF that was extracted in the second iteration of MF Extraction are equally advantageous, so both are permanently added to the cases in the case base. Thus, four MFs (the VMF and TMF versions of the first extracted raw MF, and the VMF and TMF versions of the second extracted raw MF) have been extracted for the test case surrounded by square brackets. The end result of MF Extraction for a single test case is shown in Figure 6.13.
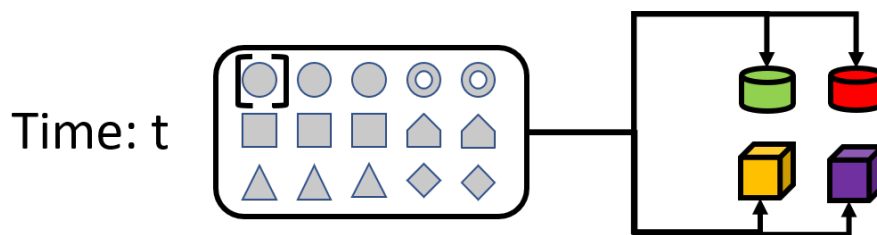
Figure 6.13: MCLv0 MF Extraction — End Result

In MCLv0. the process we just described is run for every single case in the case base (a total of 15 times in our example). A small optimization can be added to this process if we assume that the MFs that are extracted for one test case are the same MFs that would be extracted for the other

cases that remain in the candidate case pool when MemTB terminates, due to the high similarity between these cases. Regarding our example, this would mean that the specialized MFs that were extracted for the first solid circle case would also be assigned to the other two solid circular cases. Since there are six unique shapes in the case base, we would really only have to perform MF Extraction six times in this example. It should also be noted that even though there are six unique shapes in this example, it is possible that different shapes prescribe the same action at time step t, but are simply not similar in their problem component sub-runs.

6.5.1.2 *MCLvN Memory Feature Extraction*

In contrast to the MF Extraction process in MCLv0, the final output of MF Extraction in MCLvN is *not* a set of specialized MFs for each case. Instead, MCLvN computes a set of *raw* MFs that are extracted for each case in the case base. MF Extraction in MCLvN possesses two optimizations that make it more efficient than its counterpart in MCLv0: Memory Singularity Principle (MSP) and Tree Extraction.

The first optimization is the use of the Memory Singularity Principle (MSP), which states that comparison of memories of run components is equivalent to comparison of those run components directly. In other words, there is no need to compare MFs that were created from run components when we can just compare the run components directly. (It is important to note here that MSP only applies to raw MFs, not specialized MFs such as value-back and time-back MFs.) As we already discussed previously, one consequence of the MSP is that *all* raw MFs for a single test case can be extracted from just a single MemTB+ call, rather than $3R$ calls (where R is the number of raw MFs that are ultimately needed to express the memory influences relevant to the test case).
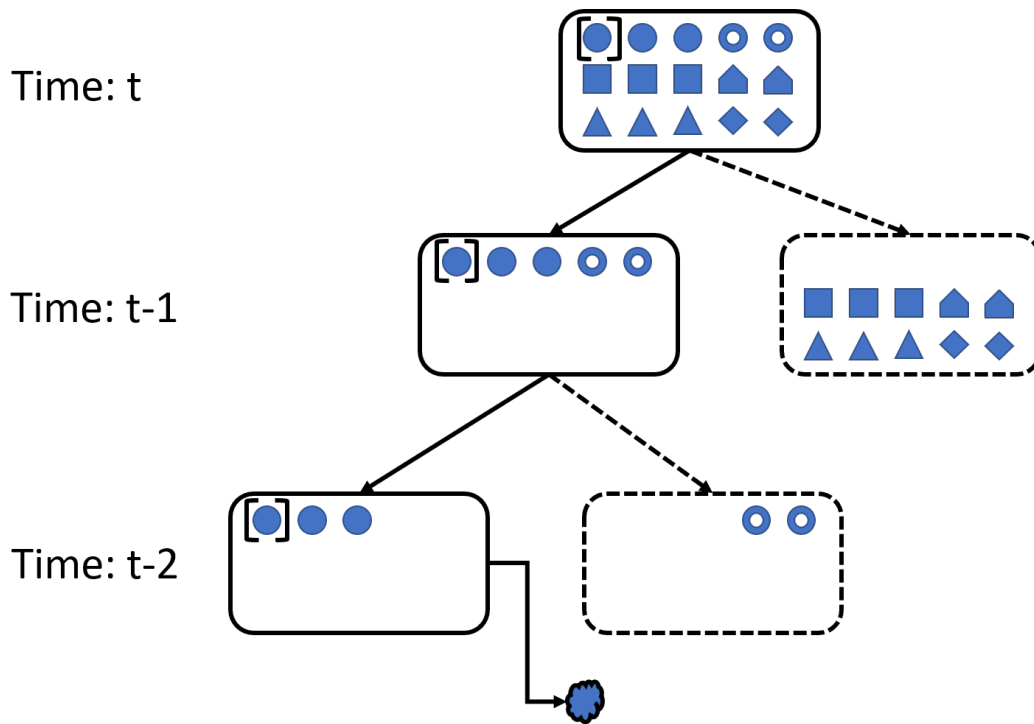
Figure 6.14: MCLvN MF Extraction — Memory Singularity Principle, Part 1

In Figure 6.14, a test case is selected (the solid circular case surrounded by square brackets). Then, MemTB+ filters out cases from the candidates case pool that are not similar to the test case at time t. However, the cases remaining in the candidates case pool still differ on the prescibed action to take at time t, so MemTB+ enters another level of recursion to remove cases from the candidate case pool that differ from the test case at time $t-1$. At this point, all remaining cases in the candidate cases pool agree on the prescribed action for time t and a single raw MF is extracted, as represented by a cloud shape in the figure. This process is similar to that in MCLv0, but in MCLvN, the process continues.
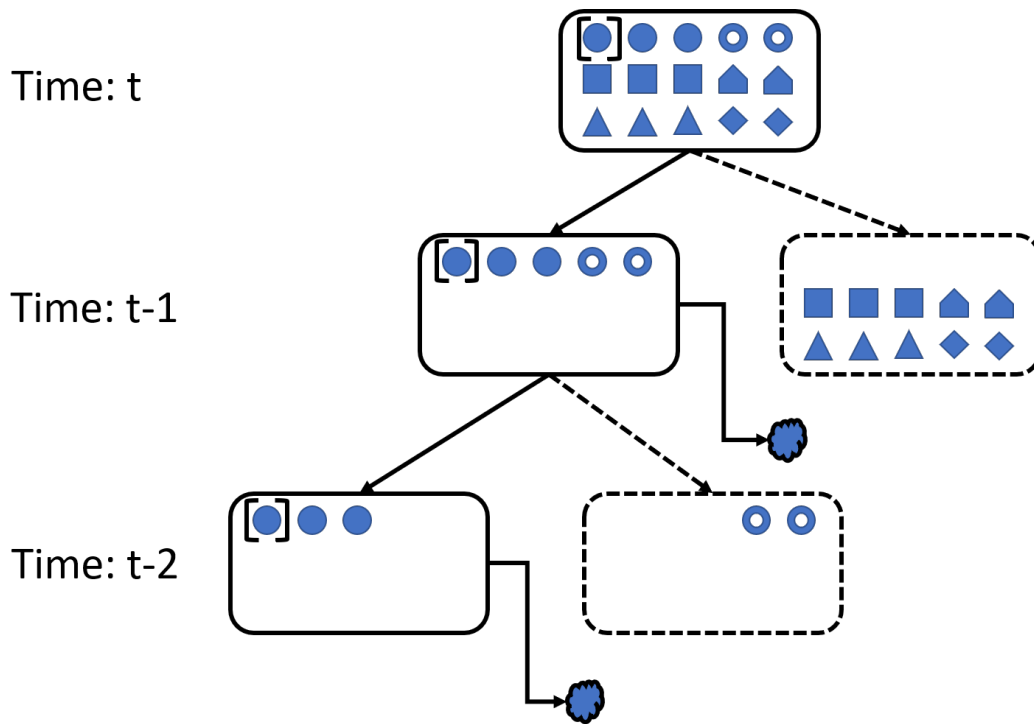
Figure 6.15: MCLvN MF Extraction — Memory Singularity Principle, Part 2

One must keep in mind that the purpose of MFs is to help a learning algorithm distinguish between time steps where one set of actions is required from other time steps where some other set of actions should be performed. For time steps where the same set of actions are performed, the MFs will encode one set of values and for other time steps, the MFs will encode a different set of values. With this in mind, MemTB+ will return control back to the previous level of recursion and do two things:

1. MemTB+ will check all the cases that were rejected at this level of recursion (which we'll call "current level rejected cases" or CLR cases). For every raw MF that was previously extracted for a deeper level of recursion, MemTB+ will compare each of the CLR cases with the test case at the time-back value specified by the raw MF. If all of the CLR cases differ

from the test case for at least one of these time-back values, then MemTB+ terminates at this level of recursion.

2. Otherwise, it is necessary to extract a raw MF that differentiates the test case from the CLR cases, which MemTB+ does here.

In the case where a raw MF is required at time $t-1$ for our example, Figure 6.15 shows the creation of this MF. At this point, MF Extraction finishes and a set of raw MFs has been extracted for the current test case.

	Test	C1	C2	C3
t	*	*	*	*
t-1	*	*	*	=
t-2	*	*	*	*
t-3	*	*	%	*
t-4	*	&	*	&
t-5	*	&	*	&

1. Pool: Test, C1, C2, [C3]
Reject C3

2. Pool: Test, C1, [C2]
Reject C2

3. Pool: Test, [C1]
Reject C1
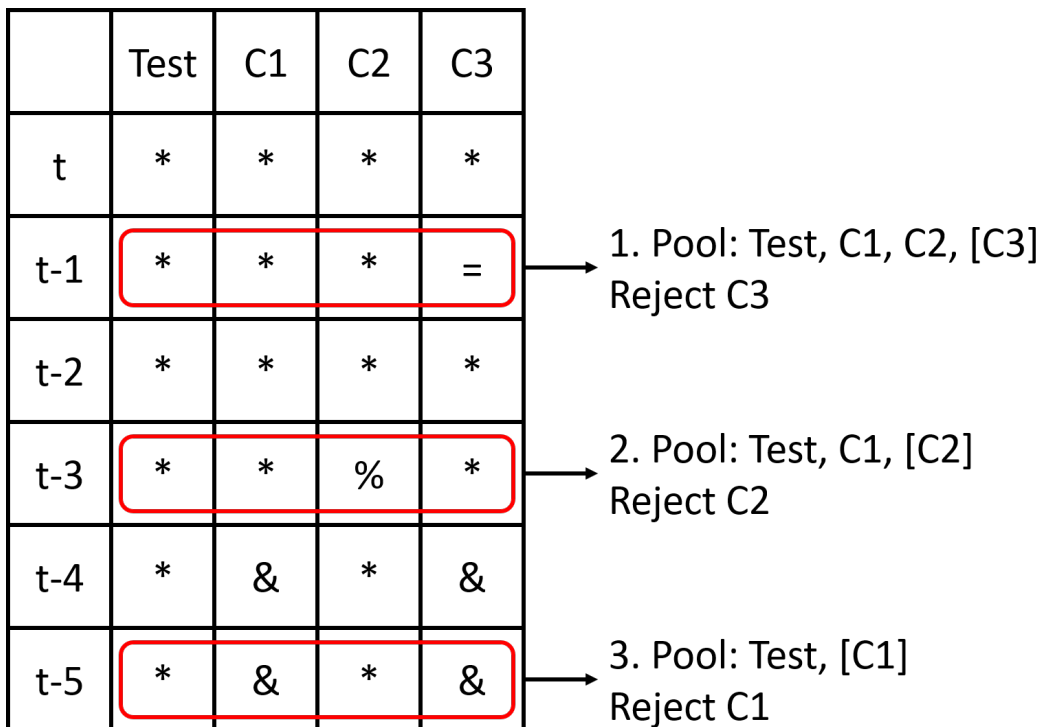
Figure 6.16: MCLvN MF Extraction — MF Pruning by MSP, Part 1

To further demonstrate the power of MSP in pruning unnecessary raw MFs, we provide an example here. Figure 6.16 shows a test case (denoted "Test") and three cases that prescribe different actions

at time t than the test case. The three cases (denoted C1, C2, and C3) must be rejected during MF Extraction. MemTB+ begins by comparing all candidate cases to the test case at time t — neither C1, C2, nor C3 are rejected. However, at time $t-1$, case C3 differs from the test case and is rejected. Case C2 is rejected as being different from the test case at time $t-3$, then Case C1 is rejected as being different from Test at time $t-5$.

Let's assume that at the MemTB+ recursion level for time $t-5$, all remaining candidate cases agree on the action to take for time t. Thus, a raw MF is extracted to differentiate the cases that were rejected at this recursion level for being different from Test at time $t-5$. This is shown in Figure 6.17.

	Test	C1	C2	C3
t	*	*	*	*
t-1	*	*	*	=
t-2	*	*	*	*
t-3	*	*	%	*
t-4	*	&	*	&
t-5	*	&	*	&

6. Pool: Test, C1, C2, [C3]
No MF needed.
C3 already resolved by MF(&, t-5)

5. Pool: Test, C1, [C2]
Create MF(%, t-3)   ⟨ *, t-3 ⟩
Resolve C2

4. Pool: Test, [C1]
Create MF(&, t-5)   ⟨ *, t-5 ⟩
Resolve C1
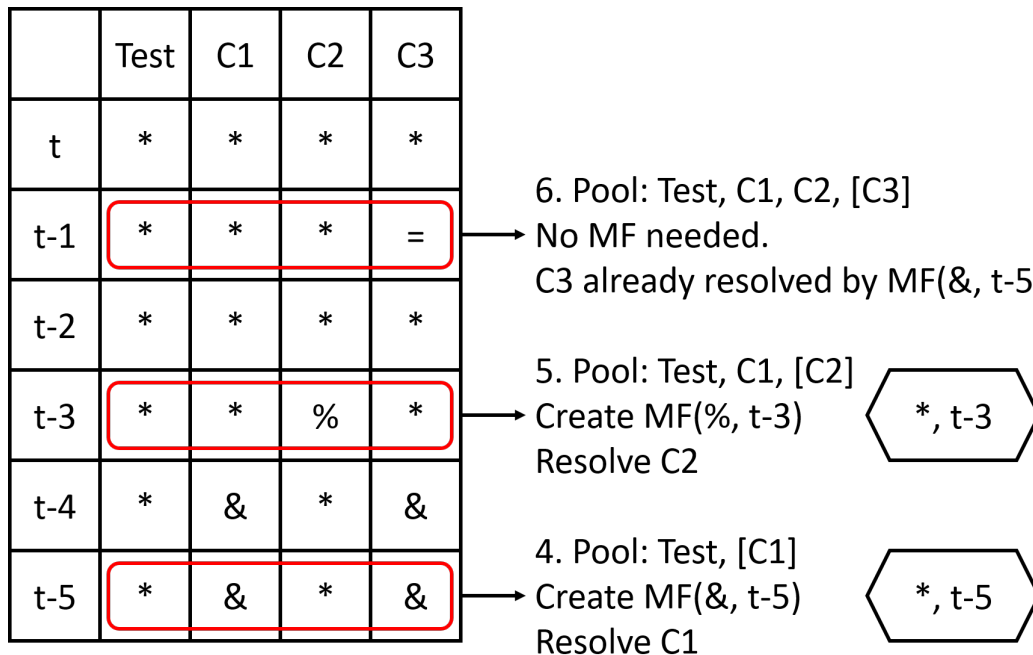
Figure 6.17: MCLvN MF Extraction — MF Pruning by MSP, Part 2

In Figure 6.17, a raw MF encoding the value '*' at time $t-5$ is created (it stores the value for Test at time $t-5$). This raw MF thus "resolves" case C1 by encoding one instance where C1 differs

from Test (because instead of *, C1's run component is & at time $t-5$). Then, control returns to the previous MemTB+ recursive call. When control returns to the MemTB+ recursive call processing comparisons at time $t-3$, case C2 is analyzed to see if its run components differ from at least one of the raw MFs that were previously extracted. At this point, there is only raw MF (value *, time $t-5$). However, case C2 also stores value * at time $t-5$ and is thus similar to Test at time $t-5$. Therefore, it is necessary to extract another raw MF for time $t-3$ to differentiate C2 from Test. The raw MF that is generated stores value '*' and time $t-3$.

From there, control will eventually return to the MemTB+ function call that was processing comparisons at time $t-1$. Case C3, which was rejected from the candidate case pool at this point, is then compared to all previously extracted raw MFs to see if it differs from at least one of them. For the raw MF (value *, time $t-3$), C3 also has value * (like Test) at time $t-3$ and is not resolved. For the raw MF (value *, time $t-5$), C3 differs from test, because it has value & instead of * at time $t-5$. Thus, C3 is resolved. If we assume there are no other cases that were rejected along with C3 at this recursive call, then we can say that all rejected cases have been resolved for this time step and thus no raw MFs have to be extracted.

As a result of leveraging MSP, now MCLvN only needs to call MemTB+ once for each case in the case base. Figure 6.18 shows the application of MF Extraction in MCLvN for a new test case (a hollow circle, the one with square brackets around it). As we can see in the figure, this test case shares one raw MF in common with the solid circle test case from Figure 6.15 (the top raw MF generated from the first split in the tree at $t-1$). However, the second raw MF generated for the hollow circle test case is different. In each of these examples, as soon as a case is rejected from the candidate cases pool, it is no longer considered, but we can further optimize the process of MF Extraction if we don't do this.

The second optimization used by MCLvN is called "Tree Extraction". Tree Extraction treats the
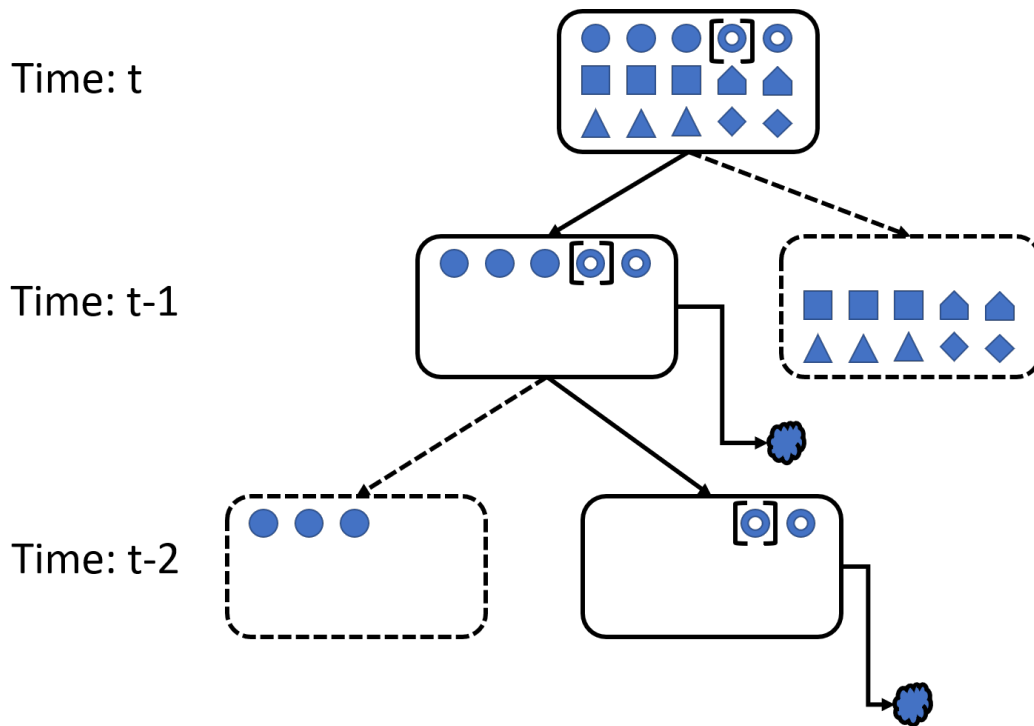
Figure 6.18: MCLvN MF Extraction — Memory Singularity Principle, New Case

MF Extraction process like a postfix tree traversal (where a node in the tree processes its children before it processes itself). Figure 6.19 shows the first step in tree extraction, in combination with MSP.

In Figure 6.19, the first solid circular case is the test case. When cases are filtered at time t, the non-circular cases are rejected and set aside. Then, the cases are compared at time $t - 1$ and the hollow circular cases are rejected from the candidate cases pool. Raw MF 1 is then extracted and control returns to the $t - 1$ MemTB+ function call. Here, tree extraction processes the cases that were rejected (the hollow circular cases). It does so by picking a new test case (which is arbitrarily the first hollow circular case) and a new candidate cases pool is formed. All cases in this new candidate cases pool agree on the prescribed case for time t, so a raw MF is extracted (raw MF 2). Control then returns to the $t - 1$ MemTB+ function call. With all cases (retained and rejected)

204

having been processed, MemTB+ (according to MSP) compares all CLR (current level rejected) cases with raw MFs 1 and 2. If not all CLR cases are resolved, then raw MF 3 is created.
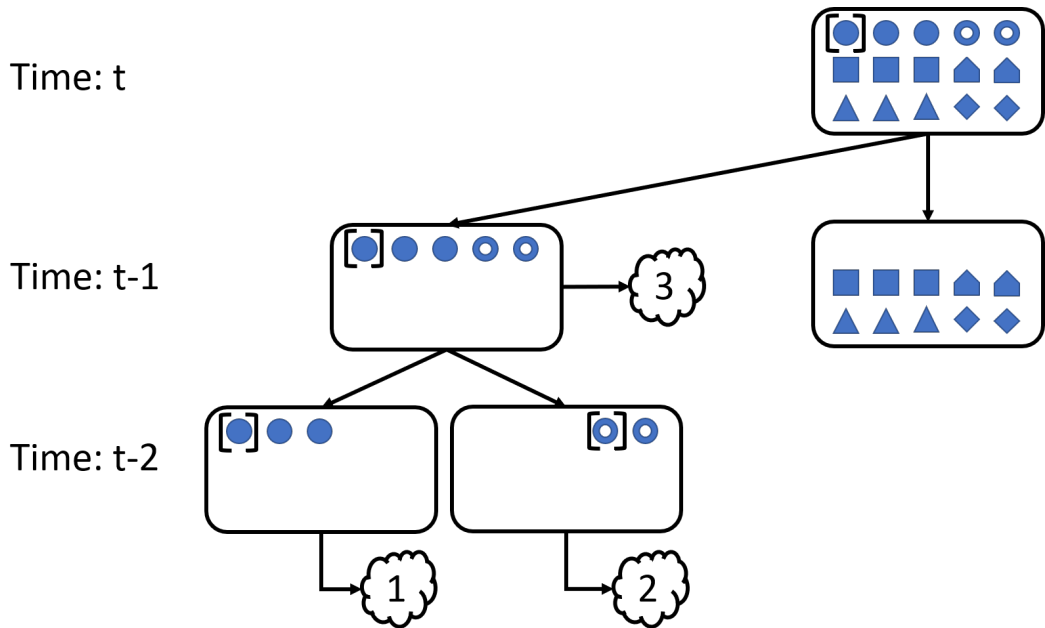
Figure 6.19: MCLvN MF Extraction — Tree Extraction, Part 1

It is important to note here that if we had not been using tree extraction, then the raw MFs for the hollow circular test case would have to be computed starting from the root of the "tree" and compared to all other cases (e.g. the squares, houses, triangles, and diamonds) (see Figure 6.17).

We assume that the hollow circle cases are dissimilar to the square/house/triangle/diamond cases, given that they are similar to the solid circle cases and given that the solid circle cases are dissimilar to the square/house/triangle/diamond cases. Under this assumption, it is not necessary to compare the hollow circle cases to the non-circular cases, which is what enables tree extraction.
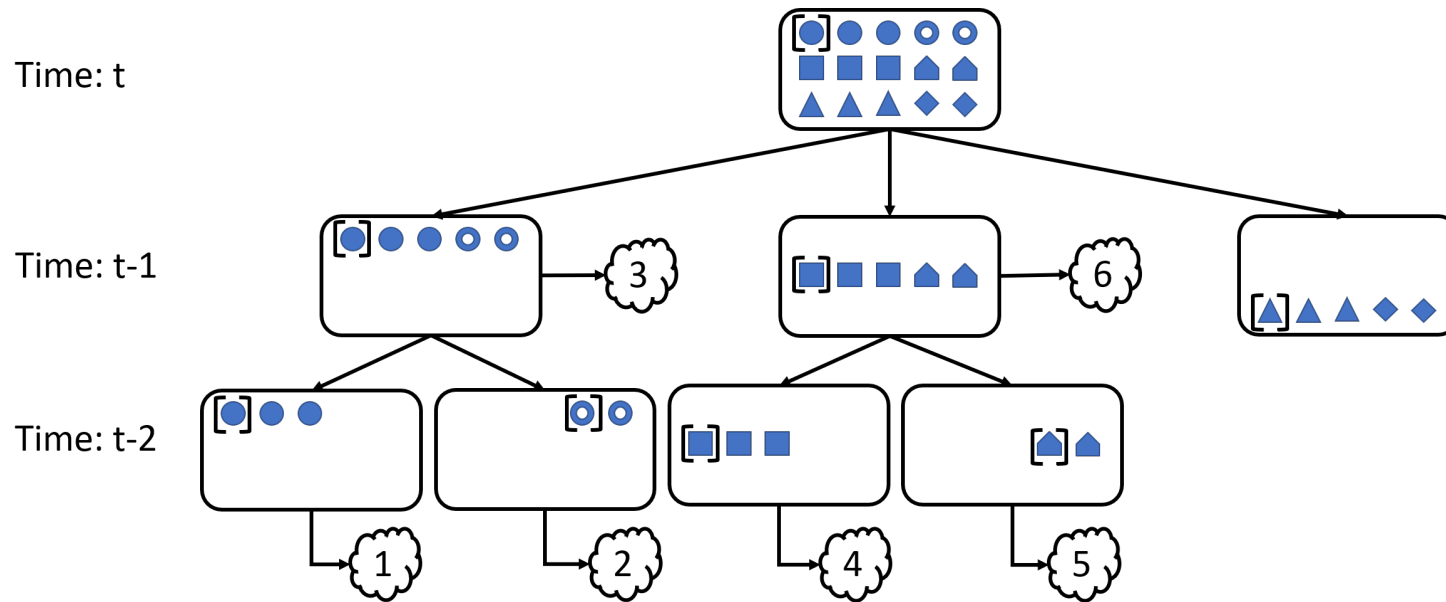
Figure 6.20: MCLvN MF Extraction — Tree Extraction, Part 2
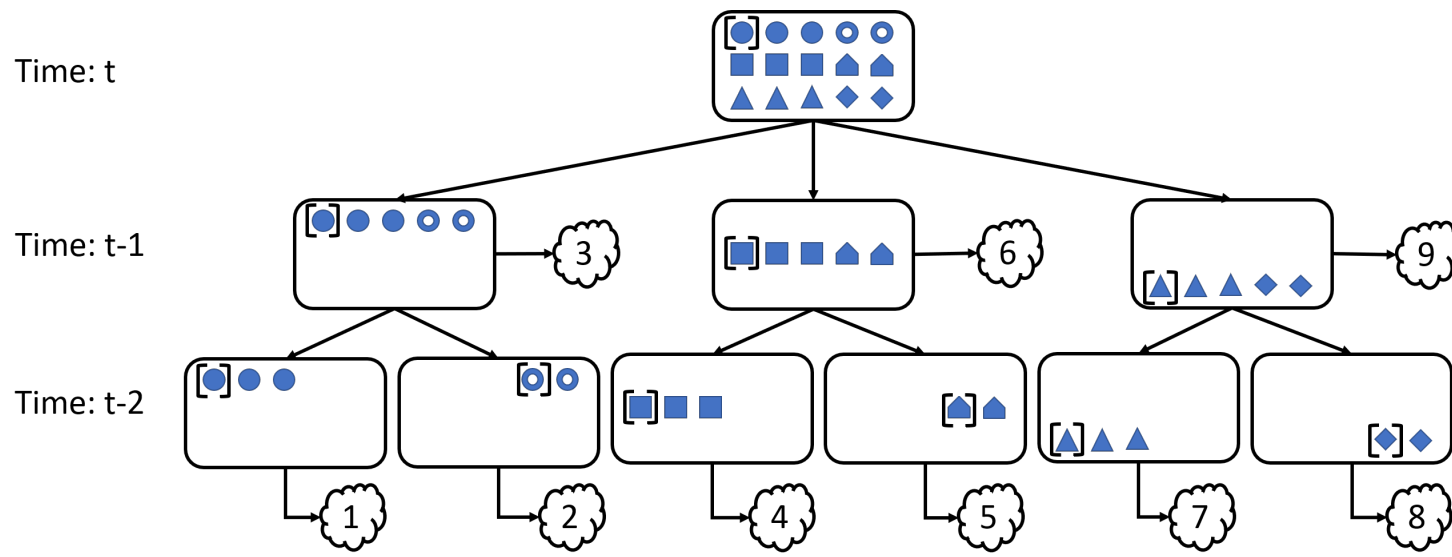
Figure 6.21: MCLvN MF Extraction — Tree Extraction, Part 3

Control returns to the MemTB+ recursive call for time t. Figure 6.20 shows the continuation of tree extraction operations. In Figure 6.20, the rejected cases (the non-circular cases) are consolidated into a new candidate cases pool and the first square-shaped case is arbitrarily assigned as the new test case. Cases are then compared to the new test case and all the triangle and diamond-shaped cases are removed from the pool and set aside. Similar to how the circular cases were processed, a raw MF (raw MF 4) is created for the square cases, raw MF 5 is created for the house-shaped cases, and raw MF 6 is created for the square/house-shaped cases (assuming that circular, triangular, and diamond-shaped cases are not resolved by the extracted raw MFs accumulated thus far). Control then returns to the MemTB+ recursive call for time t.

In a similar manner to the square cases, the triangle and diamond cases are analyzed, as seen in Figure 6.21. The triangle case is arbitrarily chosen as the new test case. The diamond cases are rejected from the candidate cases pool and set aside. Raw MF 7 is created for the triangle cases. Then, the diamond case is chosen as the new test case. No cases are removed from the pool of diamond cases, so raw MF 8 is generated. Then, control returns to the parent recursive call for MemTB+ at time $t - 1$ and raw MF 9 is created.

Figure 6.22: MCLvN MF Extraction — Tree Extraction, End Result

The end result for tree extraction in our example is shown in Figure 6.22. Next to each case is the set of raw MFs that were extracted for it. When a raw MF is generated, it is automatically assigned to all cases that are in the candidate cases pool at that time.

We emphasize here that for each case, only *raw MFs* have been extracted. It is up to MF Re-

finement to determine which set of specialized MFs appropriately express the memory influences codified by these raw MFs.

6.6   Improved Memory Feature Refinement

This section describes the improved MF Refinement stage of MCLvN. MF Extraction found the memory influences that were relevant for each case. MF Refinement finds the best representations for memory influences that affect the behavior as a whole. However, the assumption that memory influences affect the behavior at every time step can be false. Often, memory influences are context-specific, meaning that a memory influence may only be relevant within certain contexts or situations.

Thus, the task for MF Refinement in MCLvN is as follows: Given a list of raw MF sets, which were extracted for each case in the previous stage, generate a set of context definitions that describe situations where specific memory influences are relevant. This is different from the MF Refinement stage in MCLv0, which received as input sets of *specialized* MFs for each case and which only generated a single set of specialized MFs as its output.

Contexts, as discussed in Chapter 2, are structures that define expectations and appropriate behavior for a given type of situation. The advantage of using contexts in modeling memory influences for observed behavior is that contexts can help MCL detect rare memory influences. Figure 6.23 shows a hypothetical example of the frequencies of various MFs in a case base. If we arbitrarily say that MFs must appear in at least 60% of the cases in the case base to be retained in the final refined MF set, then only MFs 1, 2, 3, and 5 would be retained.
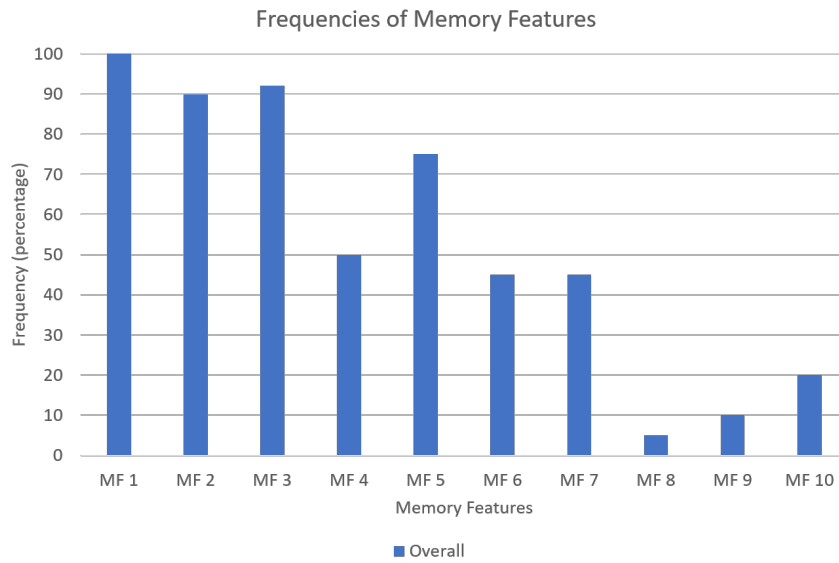
Figure 6.23: MF Refinement without Context
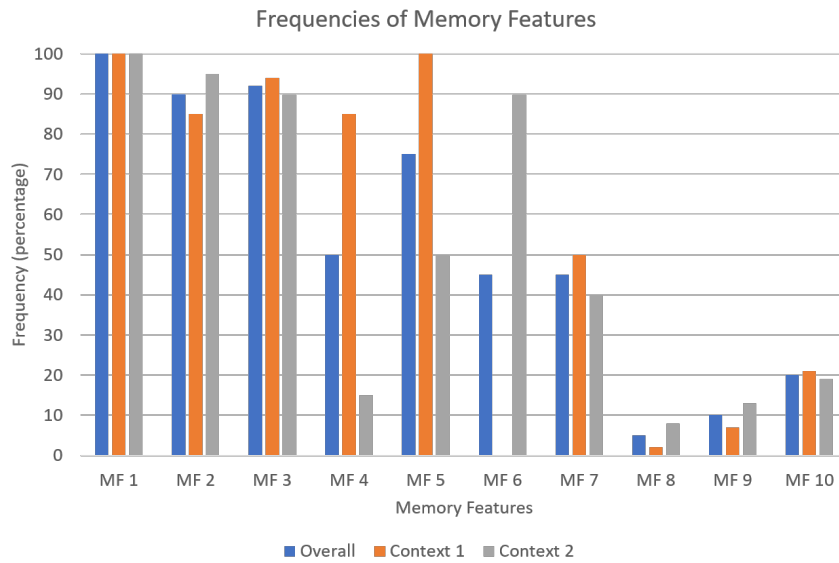
Figure 6.24: MF Refinement with Context

Figure 6.24 shows the frequencies of MFs in our example 1) over all cases in the case base, 2) over

all cases within Context 1, and 3) over all cases within Context 2. According to our 60% threshold, MF 4 appears in only 50% of all cases in the case base, but it appears in 85% of cases in Context 1. Because of its high relevance in Context 1, it should not be filtered out. MFs 6 and 7 both appear in 45% of the cases in the case base. However, while MF 6 has a 90% relevance in Context 2 and should not be filtered out, MF 7 is does not exceed a frequency over 60% in either context and should be filtered out. Thus, MFs 1-5 are relevant to Context 1, and MFs 1-3, and 6 are relevant to Context 2. MFs 4-6 are relevant for exactly one context in our example.

Given the significance of context in modeling memory influences on observed behavior, we now have to address two issues:

1. How do we define context within the scope of memory?

2. How do we learn the representations of the appropriate contexts?

Contexts can be defined manually by a developer or discovered automatically by a system such as COPAC [82]. A context, as defined in [82] is "Any recognized situation defined by a set of similar practices that can approximate the set of actions, procedures, and expectations employed by an expert within the situation."

In this research, we make a distinction between *behavioral contexts*, which follow the definition in [82], and *memory contexts*, which we define as follows:

> Any recognized situation defined by a set of similar memory influences that represent the relationships between past observations and present actions by an observed actor within the situation.

In our definition, observations pertain to anything that is knowable by an external observer (e.g.

perceptions of the environment, actions taken by the observed actor) and relationships between past and present are constants based on the passage of time, as defined by our MFs.
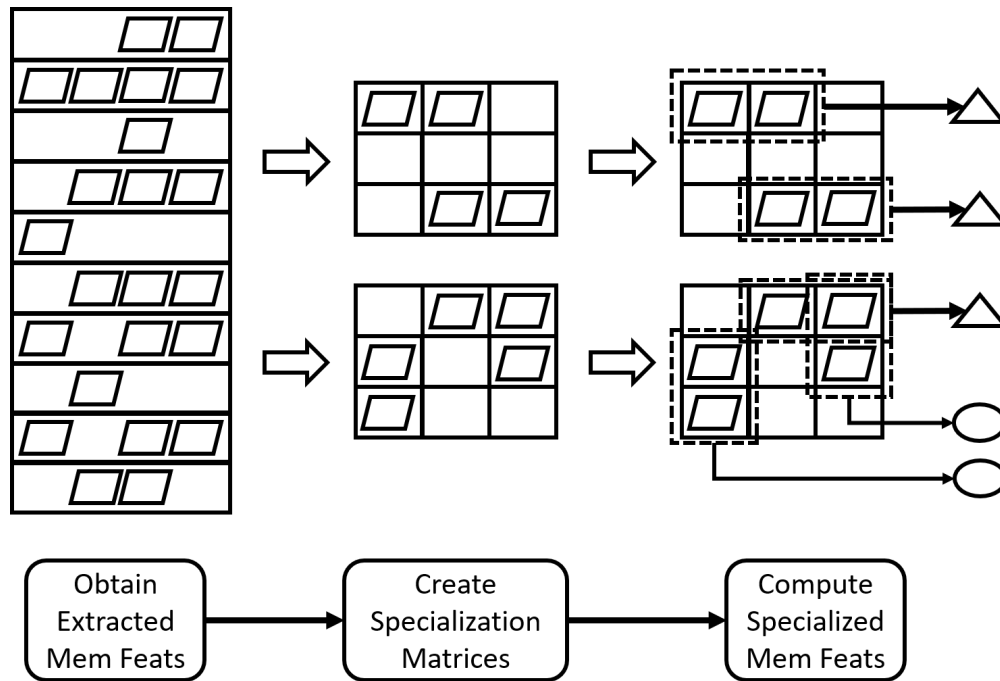
Figure 6.25: MF Refinement Overview — Part 1

Similar to COPAC, MCL will learn which contexts exist, which contexts they can transition to, and which parts of the input traces are mapped to those contexts, but MCL will *not* learn which behaviors should be executed nor which conditions will trigger a change in context; this knowledge is meant to be learned by a subsequent LfO algorithm. In contrast to COPAC, MCL will also learn which specialized MFs are relevant to each memory context.

Figures 6.25 and 6.26 present an overview of how specialized MFs are generated for various context definitions in the MF Refinement stage of MCLvN. The steps in MF Refinement, according to the figure, are as follows:

1. *Obtain Extracted Mem Feats*: The input to the MF Refinement stage is the list of raw MF sets that were extracted for each case. In Figure 6.25, cases are long rectangles and their raw MFs are denoted by parallelograms.

2. *Create Specialization Matrices*: To compute which specialized MFs are most appropriate for accounting for the memory influences encoded by the raw MFs, we initialize one matrix per feature, its cells denoting how many cases possessed a raw MF with a particular value and time-back component.

3. *Compute Specialized Mem Feats*: This step uses the specialization matrices to create specialized MFs (value-back denoted by triangles and time-back denoted by ovals). Each specialized MF accounts for one or more types of raw MFs.

4. *Assign Specialized Mem Feats to Each Case*: The raw MFs for each case are replaced by one or more of the specialized MFs computed in the previous step. Each specialized MF may account for one or more of a case's raw MFs, so there may fewer specialized MFs assigned to each case than there are raw MFs.

5. *Create Memory Contexts*: In this research, memory contexts are contexts centered around specific specialized MF sets. The memory contexts are represented by long hexagons and each case is mapped to exactly one of the memory contexts.

6. *Filter Memory Contexts. Get Context Transitions.*: The proportion of cases in the case base that belong to a given memory context is compared to a threshold. If the memory context's coverage of the case base does not exceed the threshold, it is deleted and its cases are assigned to a default context. The transitions between all remaining memory contexts are computed to create a context map.

Figure 6.26: MF Refinement Overview — Part 2

Load next case's
raw MF set.

Has each raw MF
in the set been
added to a matrix?

No → Load next raw MF. Get
the feature *f*, time-
back *tb*, and value *v* of
the raw MF.

Increment the count
for the specialization
matrix of feat *f* at
cell (v, *tb*).

Initialize empty lists for: memory
contexts, context map, master
specialized MFs. For each feature,
create empty specialization matrix.

Yes

Load next feature's spec.
matrix. Compute the sum
of each row/column.

If time-back (column) has
highest sum, create new value-
back MF for that time-back.

If value (row) has highest
sum, create new time-
back MF for that value.

No

Have all cases' raw MF sets been
added to the spec. matrices?

Get highest
row/column
sums. Are both
sums zero?

No

Add new MF to master specialized MFs
list. Set cells in highest row (or column)
to zero. Update row/column sums.

Yes

Have spec. MFs been computed
for each spec. matrix?

No

Yes

Yes

Load next case.
Initialize case's spec.
MF set to empty set.

Have all master specialized   *No*   Load next master specialized MF.
MFs been compared to   →   If it matches at least one of the
case's raw MFs?   case's raw MFs, add it to the case.

Has each case been assigned
specialized MFs?

No

Yes

Yes

Is there a memory
context with the
same specialized MF
set as the case?

No → Create new memory context.
Set its specialized MF set to
that of the case. Add this
context to the context map.

Have all memory contexts
been checked for coverage?

No

Add a transition to this case's
memory context from the
previous case's memory context
in the context map.

Yes

Add this case to that memory context.

Yes

Return memory context list and
context map.

Load next memory
context. Does its
coverage of the casebase
exceed CXT_THRESH?

Yes

No

Delete memory context. Assign all its cases to the default
context. Adjust context transitions in context map.

Figure 6.27: MF Refinement — Flowchart

Figure 6.27 shows a flowchart for the different procedures in the MF Refinement stage of MCLvN. We cover these procedures in more detail here. In Figure 6.27, the start state has a dotted border and the terminating state has a bold solid border. Each major operation in the flowchart begins with a node on the leftmost side. If a given operation has been done, control proceeds downward to the next operation. Otherwise, the appropriate procedure is employed to complete the operation.

The first three operations in MF Refinement have to do with computing an optimal set of specialized MFs to account for the extract raw MFs across all cases. These operations are:

1. Initialize structures for a master set of specialized MFs (spec. MFs for short), list of memory contexts, context map, and specialization matrices.

2. Add the raw MFs of every case to the specialization matrices.

3. Use the specialization matrices to compute specialized MFs that summarize the raw MFs therein.

First, we discuss the data structures that are initialized in the first operation of MF Refinement in MCLvN:

- *Memory Contexts*: Each memory context will contain the list of cases that belong to that context and the list of specialized MFs that are relevant for that context.

- *Context Map*: The context map defines which transitions between memory contexts were observed in an actor's behavior.

- *Specialization Matrices*: One spec. matrix tracks the number of raw MFs (across all cases) that have particular combinations of values and time-backs for a given feature. There is one spec. matrix created for each feature.

216

- *Master specialized MFs*: This is the list of specialized MFs that are relevant to the entire case base. However, not all of them are necessarily relevant to each memory context.

In the second operation of MF Refinement, the specialization matrices for each feature are updated with information from all raw MFs from all cases. Each specialization matrix is essentially a two dimensional array. For each raw MF of each case, the feature name, feature value, and time-back (number of time steps before the current time step) are obtained from the raw MF. Then, the specialization matrix belonging to feature f is updated – the cell corresponding to the feature value and time-back of the raw MF stores the number of cases with this same raw MF, so we increment it by one.

In the third operation of MF Refinement, the specialization matrices are used to compute the smallest set of specialized MFs that account for all raw MFs in the matrix. The following steps are used for each feature's specialization matrix:

1. Compute the sum of counts for each row (feature value) and column (time-back) in the matrix. If all sums are zero, then terminate this procedure.

2. Obtain the highest sum. If the highest sum belongs to a row (feature value), then create a time-back MF parameterized with the feature value corresponding to the row. This time-back MF accounts for all raw MFs with the same feature value.

3. If the highest sum belongs to a column (time-back), then create a value-back MF parameterized with the time-back corresponding to the column. This value-back MF accounts for all raw MFs with the same time-back.

4. If there's a tie, break the tie using the original row/column sums and choosing the row/column with the highest original sum.

217

5. Zero out the cells in the winning row/column. Add the new specialized MF to the list of master specialized MFs.

We now show an example of this procedure. In Figure 6.28, we see a hypothetical specialization matrix for feature F. Feature F can take on values a, b, c, and d. Raw MFs for this feature have time-back values of 1, 2, 3, 7, or 8. The number of cases with raw MFs for each combination of feature value and time-back are recorded in the cells. For example, the number of cases with a raw MF for feature F, value a, and time-back 1 is 8. In this matrix, the sum of each row and column is computed. The row sums are stored in the columns labeled C_v and O_v. The column sums are stored in the rows labeled C_t and O_t. The C sums will be updated every time a specialized MF is created. The O sums will not change and will be used to break ties.

	t-1	t-2	t-3	t-7	t-8	C_v	O_v
F=a	8	10	17	9	6	50	50
F=b	5	15				20	20
F=c	12		3	11	14	40	40
F=d	15	5				20	20
C_t	40	30	20	20	20		
O_t	40	30	20	20	20		

MFs
None

Figure 6.28: MF Specialization Example — Part 1

	t-1	t-2	t-3	t-7	t-8	C_v	O_v
F=a	8	10	17	9	6	50	50
F=b	5	15				20	20
F=c	12		3	11	14	40	40
F=d	15	5				20	20
C_t	40	30	20	20	20		
O_t	40	30	20	20	20		

MFs
Time-Back: F=a

Figure 6.29: MF Specialization Example — Part 2

	t-1	t-2	t-3	t-7	t-8	C_v	O_v
F=a	*	*	*	*	*	0	50
F=b	5	15				20	20
F=c	12		3	11	14	40	40
F=d	15	5				20	20
C_t	32	20	3	11	14		
O_t	40	30	20	20	20		

MFs
Time-Back: F=a
Time-Back: F=c

Figure 6.30: MF Specialization Example — Part 3

In Figure 6.29, the first row corresponding to value F=a is selected with a red circular border, because it has the highest sum out of all the rows and columns. Thus, a time-back MF parameterized

219

with feature value F=a is created.

In Figure 6.30, the cell counts in the first row are zeroed out and the row/column sums (the C sums) have been updated. Now, the row corresponding to value F=c has the highest sum. Thus, a time-back MF parameterized with feature value F=c is created.

In Figure 6.31, the cell counts in the third row have been zeroed out and the row/column sums have been updated. Now, the sums for the first column, second column, second row, and last row are tied for the highest sum (20). However, the first column has the highest original O sum, so it is the winning column. A value-back MF with time-back parameter $t - 1$.

	t-1	t-2	t-3	t-7	t-8	C_v	O_v
F=a	*	*	*	*	*	0	50
F=b	5	15				20	20
F=c	*		*	*	*	0	40
F=d	15	5				20	20
C_t	20	20	0	0	0		
O_t	40	30	20	20	20		

MFs
Time-Back: F=a
Time-Back: F=c
Value-Back: t-1

Figure 6.31: MF Specialization Example — Part 4

In Figure 6.32, the second column has the highest sum, so a value-back MF with time-back parameter $t - 2$ is created. After this, all row and column sums are zero and the computation of specialized MFs for this feature matrix is complete. The end result is shown in Figure 6.33.

	t-1	t-2	t-3	t-7	t-8	C_v	O_v
F=a	*	*	*	*	*	0	50
F=b	*	15				15	20
F=c	*		*	*	*	0	40
F=d	*	5				5	20
C_t	0	20	0	0	0		
O_t	40	30	20	20	20		

MFs
Time-Back: F=a
Time-Back: F=c
Value-Back: t-1
Value-Back: t-2

Figure 6.32: MF Specialization Example — Part 5

	t-1	t-2	t-3	t-7	t-8	C_v	O_v
F=a	8	10	17	9	6	50	50
F=b	5	15				20	20
F=c	12		3	11	14	40	40
F=d	15	5				20	20
C_t	40	30	20	20	20		
O_t	40	30	20	20	20		

MFs
Time-Back: F=a
Time-Back: F=c
Value-Back: t-1
Value-Back: t-2

Figure 6.33: MF Specialization Example — End Result

The computation of specialized MFs corresponds to the states *Create Specialization Matrices* and *Compute Specialized Mem Feats* in the overview graphic in Figure 6.25. The end result of this

first half of MF Refinement is a master set of specialized MFs, which summarize the memory influences for all raw MFs in the entire case base. However, we may not want all these specialized MFs to be retained in the final output, which is where context comes in.

For the second half of the MF Refinement, shown by the states in Figure 6.26, the memory contexts and context map are computed. First, the raw MF sets for each case are replaced by an appropriate set of specialized MFs. Then, these specialized MF sets are used to define various memory contexts. Then, instead of filtering individual MFs, we filter out *contexts*.

The work in [81] made the distinction between *defined contexts* and *executed contexts*. Defined contexts are the parameters that define a behavior in a given situation. Executed contexts are instances of the defined context and may vary slightly from one another. For example, a defined context could dictate that a car driver "drives slowly" on neighborhood roads, but possible executed contexts could a car driver driving at 15 mph in the morning on West St, but later driving at night at 20 mph on Palm St.

The last two operations in the flowchart in Figure 6.27 define the precise procedures for generating memory contexts, given the master set of specialized MFs:

1. For each case in the case base, check to see which master specialized MFs "match" its raw MFs — all such specialized MFs comprise the specialized MF set for the case. A raw MF matches a value-back MF if the feature name and time-back parameter are the same. A raw MF matches a time-back MF if the feature name and feature value parameter are the same.

2. Once the case's specialized MF set is computed, its specialized MF set is compared to those of all preexisting memory contexts. If there is a memory context with the same specialized MF set, the case is added to that memory context. Otherwise, a new memory context with the case's specialized MF set is created and added to the list of memory contexts.

Once the memory contexts have been created, it's possible that some of the memory contexts may have been generated as a result of random or stochastic behavior. Such memory contexts should be removed. MCL verifies the utility of each memory context by computing the proportion of cases in the case base that are assigned to this memory context and comparing that proportion to a manually defined CXT-THRESH threshold. If the proportion is high enough, the memory context is retained. Otherwise, all cases assigned that memory context are assigned to an empty catch-all "default context" and the memory context is deleted. Once all memory contexts have been verified, the observed transitions between all memory contexts are computed in order to generate the context map.

The reason why the application of a filtering threshold to contexts instead of individual MFs is advantageous is because this approach makes it easier to filter out MFs that are generated as a result of random behavior. MCL has no mechanism for determining whether a behavior is random (and thus cannot benefit from memory) or just highly complex. Thus, MCL will attempt to extract MFs to explain every variation in observed behavior. Not all these MFs will be useful for subsequent LfO.

If a behavior is random, but it is executed for a long period in time, then certain raw MFs may appear in several cases and appear to be a legitimate memory influence. However, it is much less likely for these raw MFs to repeat themselves simultaneously. In other words, a group of MFs (like those encapsulated in a memory context) are much less likely to appear repeatedly (in several cases) than just a single MF is.

The other difference between MCLv0 and MCLvN is that MCLvN accomplishes MF specialization in the Refinement stage instead of the Extraction stage. This is advantageous because MCLvN can compute specialized MFs that express memory influences over the whole behavior rather than computing specialized MFs that benefit just a single case (as is the case in MCLv0 Extraction).

223

However, MCL's output is meant to work with a wide variety of LfO algorithms, even those that do not use context, and thus an additional step is recommended to generate just a single refined set of specialized MFs that describe the majority of memory influences over an entire behavior (instead of just those specific to a given context).

One could just take the superset of all memory contexts' specialized MF sets, but this may result in a specialized MF set that is rather large in size. Instead, we employ the following procedure to get a single refined MF set, similar to the output from MCLv0's Refinement stage. Given the list of memory contexts that have a case base coverage that exceeds the CXT-THRESH threshold, do the following:

1. For each specialized MF in the memory context, compute how many cases in the case base use this specialized MF (regardless of each case's context assignment).

2. If the specialized MF's case base coverage exceeds a manually defined threshold MF-THRESH, then include that specialized MF in the final refined set.

When we use both CXT-THRESH and MF-THRESH thresholds to generate a final MF set for a context-free LfO algorithm, then it is advised to set CXT-THRESH to a value that is no higher than MF-THRESH because the MFs in a memory context cannot be included in the set if the memory context itself is filtered out. To enforce this constraint, we will refer to CXT-THRESH AND MF-THRESH as "Low Filtering (LF) Threshold" and "High Filtering (HF) Threshold", respectively.

In the situation where memory-enhanced traces are being generated for a context-base LfO algorithm, the superset of the memory contexts' specialized MFs will be added to the traces because each context can dictate which MFs it employs during learning. We now discuss Trace Enhancement in MCLvN.

6.7 Improved Trace Enhancement

The improvements to the Trace Enhancement phase of MCLvN are minimal. The major improvement in this phase is that it has been modified in the prototype to allow for the development of future MFs beyond those described in this dissertation. In MCLv0, the Trace Enhancement phase had custom code for computing the values of VMFs and TMFs. However, in MCLvN, the Trace Enhancement phase allows the MFs to compute their own memory values for a trace, given the trace values of other relevant features. In this way, the Trace Enhancement part of MCLvN does not need to be modified when a new MF type is created because that new MF will be responsible for computing its own values for a trace.

We also note here that MFs are generated by MCL by using discretized versions of traces and their values for each time step are computed using the discretized traces. However, Trace Enhancement incorporates these MFs (and their computed values for each time step) into the original traces, which are not discretized.

6.8 Comprehensive Approach Summary

In this chapter, we discussed changes to our baseline conceptual approach (MCLv0) that were designed to create a more comprehensive conceptual approach (MCLvN). We discussed the specific limitations of MCLv0 and provided an overview of how specific components of MCLvN address these limitations. Then, we discussed each phase of MCLvN in detail: Preprocessing, Memory Feature Extraction, Memory Feature Refinement, and Trace Enhancement. In the next chapter, we discuss a prototype implementing MCLvN.

CHAPTER 7: PROTOTYPE IMPLEMENTATION

In this chapter, we describe a prototype implementation of our conceptual approach described in this chapter. To differentiate between the conceptual approach and the prototype that implements this approach, we call the approach just "MCL" (Memory Composition Learning algorithm) and we call the prototype "MCLS" (Memory Composition Learning System). The purpose of this prototype is to:

- Assess our conceptual approach and prove that our results are not the product of black magic.

- Investigate the needs of applying our approach in varied domains, as will be discussed in Chapter 8.

- Validate the hypothesis of this dissertation, as described in Chapter 3.

First, we describe the format of the inputs and outputs of MCLS. Then, we describe the framework we used to implement MCLS, including the organization and interaction of code files and their invocation protocols.

7.1   System Inputs and Outputs

This section describes the format of the inputs and outputs of MCLS. In its simplest form, MCLS receives as input behavioral traces written to file. MCLS processes these *original traces* by learning the memory influences within these traces and then incorporating them into *memory-enhanced traces*, which are then written to file.

Trace information is written in two files. The first file is a *header file*, which records information

about the nature of the *features*, and the second file is a *data file*, which records the values of these features over time.

The header file is formatted as follows. If there are F features, then the header file will have F lines. Each line consists of the name of one feature (which shall not have any spaces in it) and the feature's *type*. A feature can either have the "numeric" type (if it encodes continuous values) or "nominal" (if it encodes discrete values). It may also have additional information about the feature for the purposes of discretization (e.g. center and range values). The file delimeter (the character that separates elements on a single line) for the header file is a comma ','.

Table 7.1 shows the contents of an example header file in a car driving domain with continuously-valued features *Speed* and *DistToStop* and discrete-valued features *TrafficLightColor* and *DriverIsYawning*. We present these contents in tabular view.

Table 7.1: Example header file contents

Speed	numeric
DistToStop	numeric
TrafficLightColor	nominal
DriverIsYawning	nominal

The data file encodes the values of these features over time for one run. If the recorded run has T time steps, then the data file will consist of T lines. On each line, there will be F values, one for each feature for that time step. The order of these values corresponds to the order of the features in the header file. The delimeter for the data file is a comma ','.

Table 7.2 shows the contents of an example data file for the car driving domain. The first line encodes a Speed value of 4.3, a DistToStop value of 102.5, a TrafficLightColor value of 1.0, and a

227

DriverIsYawning value of 0.0.

Table 7.2: Example data file contents

4.3	102.5	1.0	0.0
4.4	101.6	2.0	0.0
4.47	100.7	2.0	1.0
4.51	99.8	2.0	1.0

Two things should be noted. First, units are not specified. Speed could be in meters per second or miles per hour. At the developer's choice, such information could be encoded in another file, but such information is not pertinent to the functionality of MCLS. The important thing is that all values for Speed have the same units, whatever they are. Second, the values for discrete features are represented with floating point numbers. This makes it easier for MCLS to process traces internally. The real-world meaning behind the numbers is not pertinent to the functionality of MCLS, though it could be encoded in another file by the developer. For clarity of this example, a TrafficLightColor value is 1.0 or 2.0 if the light is red or green, respectively, and a DriverIsYawning value is 0.0 or 1.0 if the driver's yawning status is False or True, respectively. We also note that a comma will separate the elements on each line in the file, but the tabular view of the data file is presented in Table 7.2 for convenience and easy viewing by the reader.

The separation of feature information and run information into two files serves two purposes:

1. For multiple traces of observed behavior in the same domain, a single header file can encode feature information for all input traces. Each trace will simply require a unique data file.

2. Certain feature information is required for processing by the Weka machine learning framework [28] and this file format suits such needs.

228

MCLS converts the original traces into memory-enhanced traces. Both types of traces are formatted in the same manner as described above. However, the memory-enhanced traces will have additional "memory features" encoding memory influences. Thus, a unique header file will be generated for each output memory-enhanced trace.

A file written in XML is used to specify various hyperparameters for MCLS. The most pertinent hyperparameters are as follows:

- Current Problem Threshold (CPT): A value between 0 and 1 that determines the minimum similarity two cases must have at the current time step to be considered "similar". This is used in Temporal Backtracking [20].

- Past Problem Threshold (PPT): A value between 0 and 1 that determines the minimum similarity two cases' perceptions (i.e. problem components) must have in time steps prior to the current time step in order to be considered "similar". This is also used in TB [20].

- Past Action Threshold (PAT): A value between 0 and 1 that determines the minimum similarity two cases' actions (i.e. solution components) must have in time steps prior to the current time step in order to be considered "similar". This is also used in TB [20].

- Inverse Resolution (IR): A boolean that determines whether MCLS will use the Memory Singularity Principle (MSP) to check if memory influences for rejected cases are accounted for in earlier time steps. This hyperparameter was set to False for all experiments to avoid confounding influence of this mechanism, but the code infrastructure is present for use in future work.

- Low Filtering Threshold (LF): A value between 0 and 1 that determines the minimum proportion of cases in the case base that a memory context be relevant to in order to contribute MFs to the final refined MF set.

229

- High Filtering Threshold (HF): A value between 0 and 1 that determines the minimum proportion of cases in the case base that a MF from a retained memory context must be appear in to be retained in the final refined MF set.

We next describe the code base for MCLS.

7.2   Code Implementation

MCLS was written in the Python programming language, using the Python 3 standard. Python was selected as the programming language for this research because of its flexible syntax and extensive library support, which allows for rapid prototyping and easier high-level design. An object-oriented approach was taken to implement MCLS. It is organized into the following packages:

- *args*: This package handles code to store MCLS parameters. These parameters are set by the user in one of several XML files.

- *cbr*: This package handles code that implements custom functionality for Case Based Reasoning (CBR). For example, the code implementing the case and case base aspects of MCLS are located here.

- *extractors*: This package handles code for the Memory Feature Extraction phase of MCL. The modules in this package are interchangeable and the module to be used will be specified by the learner class, which is set by the user.

- *learners*: This package handles code for various MCLS implementations. A *learner* class selects the components for each phase of MCL and invokes their functionality. In other words, a learner defines which code is used to execute the different phases of MCL.

- *mem*: This package handles code pertaining to Memory Features.

- *misc*: This package stores code, such as low-level data structures and common data processing operations, that is usable by all other packages.

- *refiners*: This package handles code for the Memory Feature Refinement phase of MCL. The modules in this package are also interchangeable and the appropriate module is specified by the learner class, which is set by the user.

- *regenerators*: This package handles code for the Trace Enhancement phase of MCL.

- *run*: This package stores all code that initiates MCLS operations. Each file in this package has a "main" method where program execution begins.

- *tempback*: This package handles code that implements Temporal Backtracking or Memory Temporal Backtracking.

- *traces*: This package stores code for reading from and writing to trace files (e.g. header files, data files). It also stores data structures for representing information read from these files for use by other MCLS components.

- *weka*: This package stores code for preparing input files for the Weka framework and interpreting its output.

A full documentation of the code base is outside the scope of this chapter and even this dissertation. Nonetheless, we provide a concrete view of MCLS through UML (Unified Modeling Language). Figure 7.1 shows a class diagram for MCLS to visually represent the interaction between code files. Figure 7.2 shows a sequence diagram to represent the operations over time within MCLS.

Figure 7.1: Class UML diagram for MCLS

232

The "Main" class in Figure 7.1 is not really a class, but it is shown because it is the starting point of execution in MCLS. It first invokes the ArgsManager class to read the following parameters from a XML file:

- Input and output directories.

- The header file and data files to process.

- The learner module to use. (The learner module itself specifies which MF extractor/refiner modules it uses.)

Next, the "Main" class has access to a Learner instance. This is actually an interface which is implemented by one of two Python classes:

- *LearnerMinimal*: This module implements the most minimal version of MCL (MCLv0).

- *LearnerGraphSim*: This module implements the comprehensive MCLvN approach.

The "DiscretizerMain" is a separate process for discretizing traces that will be used as input to the MCLS module. The Common class contains functionality usable by any class.

The MF Extraction module in MCLv0 relies on a "Temporal Backtracking agent" as an intermediary class between the MCL components and the CBR components. Basic Memory Temporal Backtracking (encapsulated in the MemoryTempBack class in Figure 7.1) extracts specialized MFs. It interacts with the Casebase class, which manages the CBR aspects of MCLS.
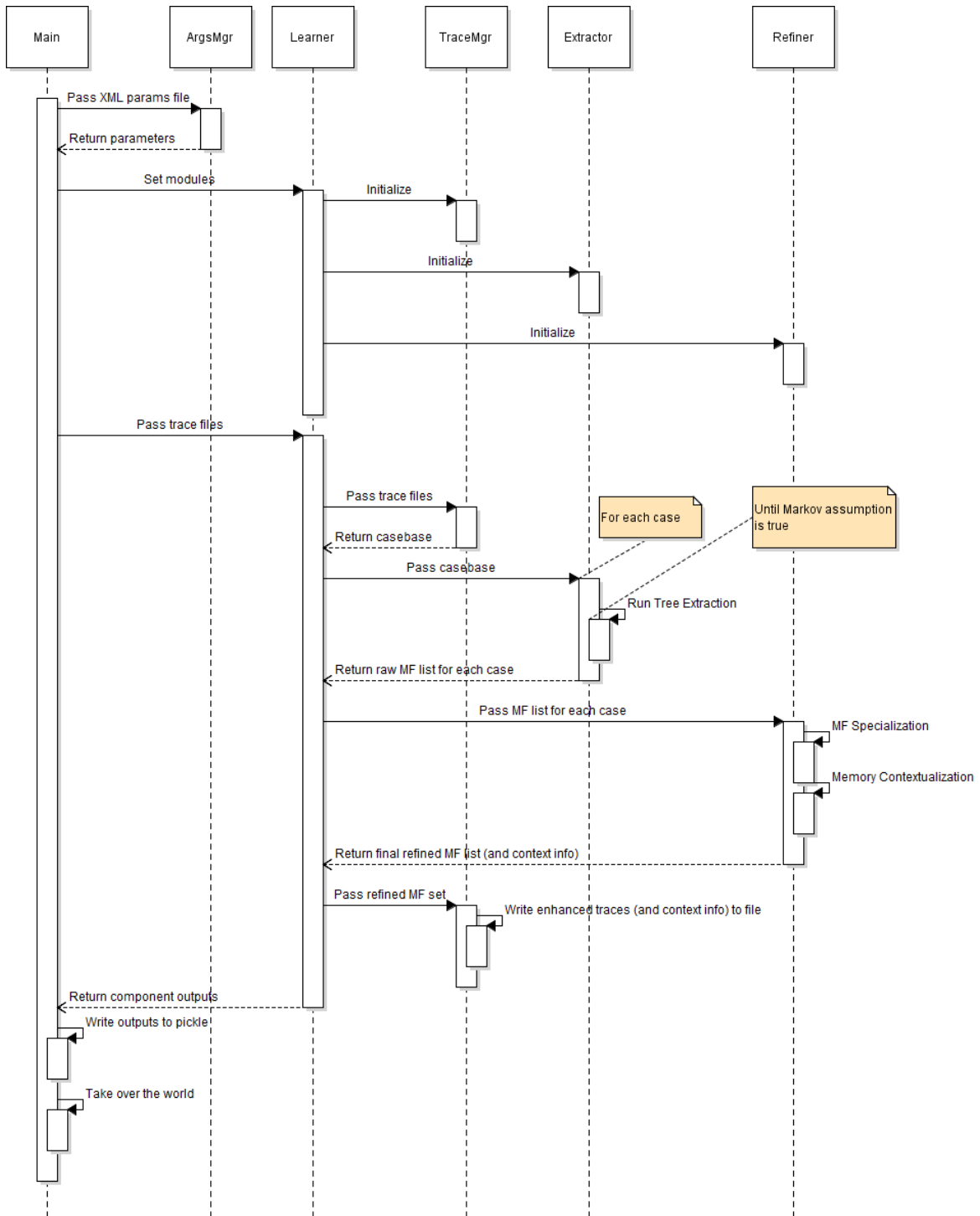
Figure 7.2: Sequence UML diagram for MCLS

For MCLvN, LearnerGraphSim interacts with the MemTraceMgr class for reading and discretizing traces read from file and incorporating memory features into traces. Its RefinerMatrix module interacts with a MemMatrix class (for MF specialization), a MemSetMgr class (to process sets of MFs), and DefCxt and ExeCxt modules (for contextualization).

Figure 7.2 shows the operations MCLS undergoes when learning a memory model from input traces. The specific operations depend on the learner module used, but the general flow is the same:

1. Read arguments from XML file in the ArgsMgr class. Launch MCLS from the command prompt or terminal and pass this XML file as an argument.

2. Set up the Learner modules.

3. Pass the trace files for the Learner to process. Do the following:

   - Convert the trace information into a case base.

   - Extract a list of MFs for each case in the case base.

   - Refine the MFs list into a single refined MF set.

   - Create memory-enhanced traces by adding MFs from the refined set.

4. Save outputs of all Learner sub-components to a pickle file (specific to Python).

7.3   Prototype Summary

In this section, we discussed matters pertinent to an actual prototype system, called MCLS, which implements the conceptual approach described in Chapter 6. First, we discussed the inputs and outputs of MCLS. Then, we provided a high-level view of the code implementation. In the next chapter, we discuss how we evaluated MCLS to investigate our hypothesis.

CHAPTER 8: EXPERIMENTS AND RESULTS

This chapter describes the evaluation procedure for assessing the validity of the hypothesis in Chapter 3 and presents the results of the evaluation. We utilized two simulated domains for this purpose: vacuum cleaner and lawn mower. The same vacuum cleaner domain that was described in Section 5.5.1 was used in our evaluation of MCLS. It is a simple domain used as a proof-of-concept. The lawn mower domain is a custom domain of our own creation to facilitate the execution of memory-influenced behaviors in humans and is a more realistic domain.

This chapter is organized as follows:

- Section 8.1 describes the experiments and criteria for success that we used to evaluate MCL.

- Section 8.2 describes the experiments and results for the vacuum cleaner domain.

- Section 8.3 describes the experiments and results for the lawn mower domain.

- Section 8.4 summarizes the findings of our experimentation and its impact on the dissertation research.

8.1   Description of Experiments

This section describes how we will evaluate the final MCLS prototype to test our hypothesis. We first describe the criteria for success for the final MCL algorithm and MCLS prototype. Then, we describe each assessment in detail and how they will provide evidence that validates the hypothesis.

8.1.1 Success Criteria

The success criteria dictating whether the dissertation research has matured to completion are as follows:

1. MCL must be able to measurably model memory-based behaviors. The adequacy of a learned MF set will be determined by the degree to which a one-to-one mapping between perception/memory space and an action space can be achieved or else explained away by stochastic effects.

2. MCL must be able to a) improve the performance of a subsequent LfO-Level-2-bound learning algorithm and b) perform comparably with other memory-based LfO algorithms.

3. MCL must be able to operate in discrete, continuous, deterministic, and stochastic domains while satisfying all other success criteria.

In the evaluation of MCLv0 in Section 5.5, criterion 1 was only addressed with qualitative means, criterion 2b was not addressed, and criterion 3 was not addressed (because the vacuum cleaner domain only involves eight binary variables). However, the addition of the lawn mower domain addresses criterion 3. The other criteria are addressed in this chapter.

8.1.2 Specific Tests

We propose two tests to assess the MCL approach for fulfilling the dissertation hypothesis:

- *Function Test*: This test assesses whether MCL is functioning as intended and will allow for evaluation of success criterion 1. This test will be conducted in the vacuum cleaner domain because the domain is simple enough to know whether MCL's output is correct without

evaluating machine learning performance. MCL will learn memory features for various cleaner agents. Then, a human-interpretable meaning behind the generated memory features should be produced and analyzed both qualitatively and quantitatively for correctness. This will ensure that the modifications to MCL did not hinder any core functionalities of the prototype.

- *Design Test*: This test assesses the efficacy of MCL in its intended setting to evaluate success criteria 2 and 3. This test will be conducted in both the vacuum cleaner and lawn mower domains. Here, MCL will generated memory-enhanced traces and then Level 2-bound LfO algorithms will learn from these traces. The learning performance of these algorithms on the memory-enhanced traces will be compared to that of memory-based LfO.

We now describe our experimental setup for these tests.

8.1.3 Experimental Setup

This section describes the inputs and outputs for each experiment. The inputs to MCLS are a single trace or set of traces that record the behavior of a single observed entity. The outputs of MCLS are memory-enhanced traces, one for each input trace, and contextual information. The contents of the memory-enhanced traces are the same as the input traces except for the possible addition of one or more MFs. Regarding contextual information, a list of executed contexts and defined contexts is produced, which can be used as input to context-based learning.

The outputs of MCL are then used as inputs to machine learning algorithms. We use the following supervised learning algorithms:

- Bayes Network: This is a probabilistic representation of procedural knowledge that is im-

plemented in the Weka framework [28].

- J.48 Decision Tree: This is a fast learning algorithm that is implemented in the Weka framework [28].

- Multilayer Perceptron: This is a standard neural network learner that is implemented in the Weka framework [28].

None of these algorithms are memory-based learners. We compute the learning performance of each of these algorithms with the original traces and with the memory-enhanced traces generated by MCLS to see if the learning performance improved. Learning performance is evaluated using the F1 measure.

Finally, we compare the learning performance of the above algorithms coupled with MCL with that of the memory-based Temporal Backtracking algorithm [20].

8.2   Vacuum Cleaner Experiments

This section describes our experiments and results from the vacuum cleaner domain. First, we briefly describe the vacuum cleaner domain that was described in Section 5.5.1. Next, we summarize our evaluation of MCLv0 from Section 5.5. Then, we describe our results of MCLvN (implemented in prototype MCLS) in the vacuum cleaner domain and how those results compare to the results for MCLv0.

8.2.1   Vacuum Cleaner Domain Description — Recap

The vacuum cleaner domain [58] is a grid-like world where a 2x2 vacuum cleaner agent moves up, down, left, and right bumping into obstacles and cleaning dirt patches. The cleaner agent perceives

the environment through eight binary sensors, two per direction. One sensor tracks the closest object (obstacle or dirt) in that direction. The other sensor tracks the distance (close or far) of the tracked object. (An object is considered "close" if it is next to the cleaner agent.)

There are several cleaner agents that exhibit behaviors that are deterministic or non-deterministic and belonging to LfO Level 1, 2, or 3. These agents are listed in Table 8.1.

Table 8.1: Vacuum Cleaner Agents

LfO Level	Deterministic	Non-deterministic
1	Fixed Sequence (SEQ)	N/A
2	Wall Follower (WF)	Random (RD)
3	Zig Zag (ZZ)	Straight Line (SL)

The WF, RD, and SL agents have "smart" variations (SWF, SRD, and SSL) that operate in the same way as the non-smart counterparts, but they prefer moving toward dirt. Each agent has one trace that is 1000 time steps long for each of the 7 possible maps.

8.2.2   Results — MCL Version 0

The reader is referred to Section 5.5.2 for detailed results from the evaluation of MCLv0. However, we briefly summarize the results here. We generated memory-enhanced traces for all vacuum cleaner agents with MCLv0. Then, we recorded the F1-measure (averaged over all seven traces for a single agent) for the J.48 decision tree, Bayes net, and Multilayer Perceptron algorithms both when using the original traces and when using the memory-enhanced traces.

For the Level 1 SEQ agent, all learning algorithms saw an appreciable increase in the F1 score

240

when using the memory-enhanced traces. The same pattern was observed for the Level 3 SL, SSL, and ZZ agents. However, no such increase is seen for the Level 2 RD, SRD, WF, and SWF agents, but no decrease is seen either; this is the expected result because Level 2 agents do not use memory at all. Thus, learning performance saw improvements for agents that require either memory (for the Level 3 agents) or internal state (for the Level 1 agent).

The learned MFs for each agent are as follows:

- For the Level 1 SEQ agent, 5 value-back and 2 time-back MFs are learned to track where in its 20-move sequence the agent is at.

- For the deterministic Level 2 WF and SWF agents, no MFs were learned, which is desired. However, for the non-deterministic RD and SRD agents, many extraneous MFs were learned (when none are required) because MCLv0 does not have a mechanism for detecting random behavior that cannot benefit from MFs.

- For the Level 3 SL, SSL, and ZZ agents, an appropriate set of MFs were learned, but based on intuition, some of the MFs were not necessary.

Therefore, there is little room for improvement regarding improvements in learning performance. However, improvements could be made in the computational efficiency of MF extraction and conciseness of learned MF sets for the agents.

8.2.3 Results — MCL Version N

This section describes the results for MCLvN for the vacuum cleaner domain. We divide this into two sub-sections. The first sub-section analyzes the learned MFs as part of the function test. The

241

second sub-section describes the improvements in learning performance. These tests ensure that MCLvN performs just as well if not better than MCLv0.

8.2.3.1 Analysis of Learned Memory Features

This section will describe the MFs learned by MCLvN for each agent.

- *SEQ*: Fixed Sequence (Level 1, Deterministic)

  - Time-Back MFs: none

  - Value-Back MFs: `vAction(1),vAction(2),vAction(3),vAction(5)`

  - Analysis: All MFs learned for this agent pertain to the Action feature, which is the only relevant feature for a Level 1 LfO agent. Thus, all learned MFs are helpful.

- *RD*: Random (Level 2, Non-deterministic)

  - Time-Back MFs: none

  - Value-Back MFs: none

  - Analysis: A Level 2 agent does not use memory at all. MCLvN learned no MFs for the Random agent, which is desirable.

- *SRD*: Smart Random (Level 2, Non-deterministic)

  - Time-Back MFs: none

  - Value-Back MFs: none

  - Analysis: MCLvN learned no MFs for the Smart Random agent, which is desirable because the Level 2 Smart Random agent does not use memory at all.

- *SSL*: Smart Straight Line (Level 3, Non-deterministic)

  - Time-Back MFs: `tAction(`*Left*`)`, `tAction(`*Right*`)`, `tAction(`*Up*`)`, `tAction(`*Down*`)`

  - Value-Back MFs: `vAction(1)`

  - Analysis: All MFs learned for this agent pertained to the Action feature. This agent needs to refer to the action it took in the previous time step. The value-back MF conveys this information sufficiently, so the time-back MFs can be considered extraneous.

- *SWF*: Smart Wall Follower (Level 2, Deterministic)

  - Time-Back MFs: none

  - Value-Back MFs: none

  - Analysis: MCLvN learned no MFs for the Smart Wall Follower agent, which is desirable because the Level 2 Smart Wall Follower agent does not use memory at all.

- *SL*: Straight Line (Level 3, Non-deterministic)

  - Time-Back MFs: `tAction(`*Left*`)`, `tAction(`*Right*`)`, `tAction(`*Up*`)`, `tAction(`*Down*`)`

  - Value-Back MFs: `vAction(1)`

  - Analysis: All MFs learned for this agent pertained to the Action feature. This agent, similar to its smart variant, needs to refer to the action it took in the previous time step. The value-back MF conveys this information sufficiently, so the time-back MFs can be considered extraneous.

- *WF*: Wall Follower (Level 2, Deterministic)

  - Time-Back MFs: none

– Value-Back MFs:, none

– Analysis: MCLvN learned no MFs for the Wall Follower agent, which is desirable because the Level 2 Wall Follower agent does not use memory at all.

- *ZZ*: Zig Zag (Level 3, Deterministic)

  – Time-Back MFs: `tAction(`*`Left`*`)`, `tAction(`*`Right`*`)`, `tAction(`*`Up`*`)`, `tAction(`*`Down`*`)`

  – Value-Back MFs: `vAction(1)`

  – Analysis: The Zig Zag agent must refer to its previous action when moving in a straight line horizontally, but it must refer to the last vertical action it took when it bumps into a wall and must choose a vertical direction. Thus, all MFs, which pertain to the Action feature, are useful.

Table 8.2: MF Counts for Vacuum Cleaner

Agent	MCLv0 VMFs	MCLv0 TMFs	MCLvN VMFs	MCLvN TMFs
SEQ	5	2	4	0
RD	14	14	0	0
SRD	15	12	0	0
WF	0	0	0	0
SWF	0	0	0	0
SL	1	4	1	4
SSL	2	5	1	4
ZZ	2	4	1	4

Table 8.2 shows the counts of value-back MFs (VMFs) and time-back (TMFs) for each agent, both for MCLv0 and MCLvN. As we can see from Table 8.2, the most noticeable improvement in

the MF sets from MCLv0 to MCLvN is seen in the Level 2 Random and Smart Random agents. In MCLv0, many individual MFs were learned in an attempt to rationalize the non-deterministic choices of the random agents, resulting in a large final refined MF set. In MCLvN, *sets* of MFs (i.e. memory contexts) were analyzed at a time for the random agents, and these memory contexts did not occur frequently enough to contribute any MFs to the final refined MF set, which is desirable because the random agents do not use memory at all.

Some extraneous MFs were removed from the Level 1 Fixed Sequence agent and the Level 3 Zig Zag and Smart Straightline agents, but the MF sets remained largely the same from MCLv0 to MCLvN. Thus, the memory influences on the agents were made more concise in some cases by MCLvN without a loss in any crucial MFs.

8.2.3.2   Analysis of Learning Performance

This section describes the learning performance of various learning algorithms when using memory-enhanced traces generated by MCLv0 versus those generated by MCLvN. We evaluate the F1-score assigned to the learning performance of each learning algorithm for each agent. Figures 8.1, 8.2, and 8.3 show the F1 scores for the Bayes Network, J.48 Decision Tree, and Multilayer Perception learning algorithms, respectively.

The objective of this evaluation was to ensure that no mechanism in MCLvN hindered the original memory learning capabilities of MCLv0. In other words, we wanted to ensure that MCLvN performed at least as well as MCLv0 in the original vacuum cleaner domain. There was little room for improvement in the performance of MCLv0 in terms of F1-score, so we sought to ensure that there was no significant decrease in learning performance when using memory-enhanced traces generated by MCLvN. As noted in Section 5.5.2, results for Bayes Net and Multilayer Perceptron were generated from a single trial and no statistical analysis is performed.
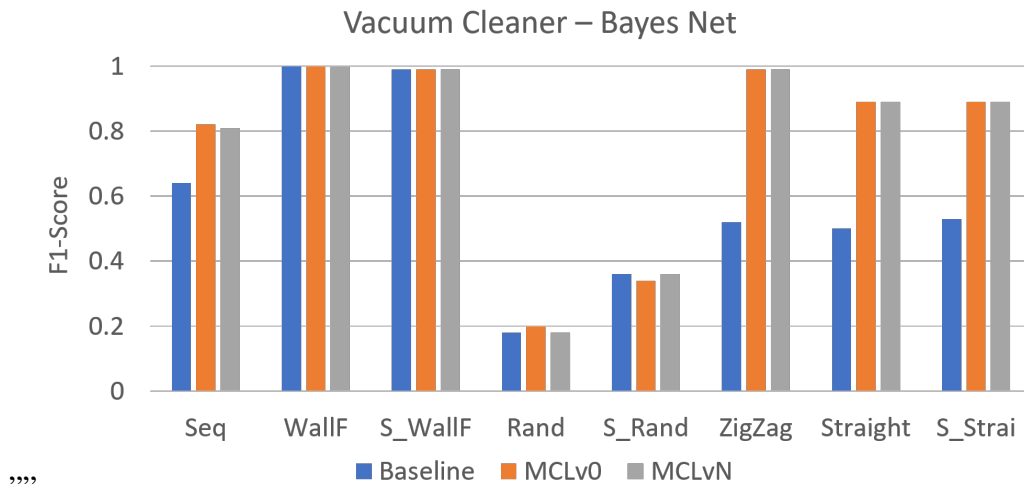
Figure 8.1: Vacuum Cleaner Bayes Network Learning Performance
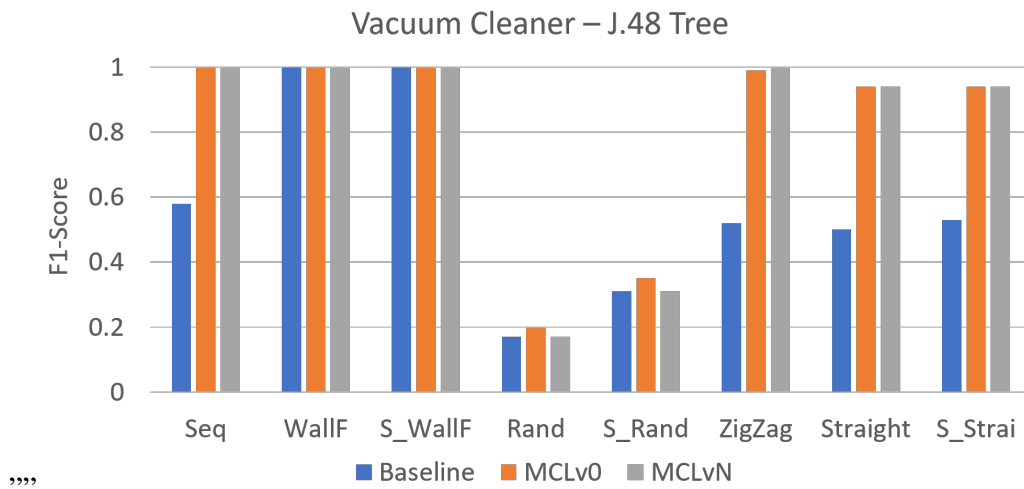
Figure 8.2: Vacuum Cleaner J.48 Decision Tree Learning Performance
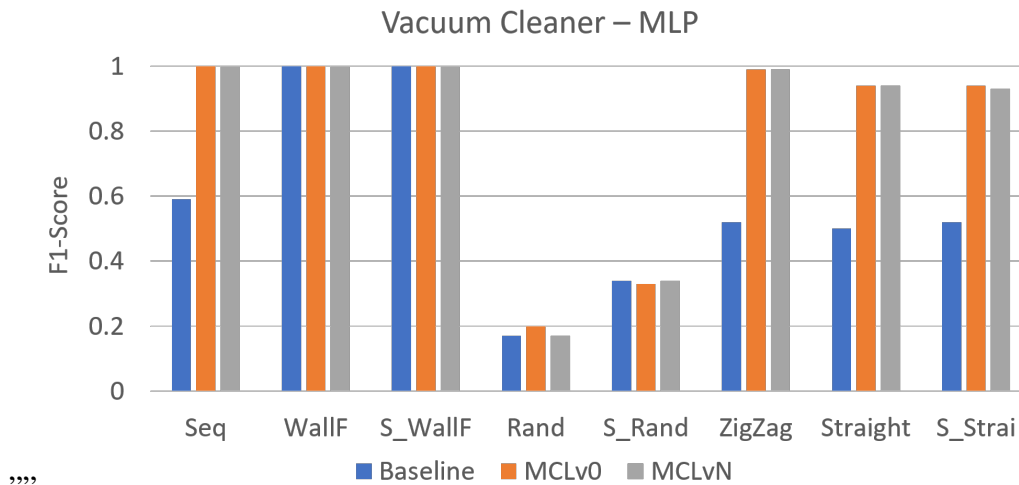
Figure 8.3: Vacuum Cleaner Multilayer Perceptron Learning Performance

In general, the trends observed for each of our three learning algorithms are the same. The F1-scores corresponding to the MCLvN traces are very similar to those for the MCLv0 traces, indicating that MCLvN's effect on improvements in learning are at least as good as MCLv0.

One may notice that in certain cases, there are very small decreases in F1-score for MCLvN when compared to MCLv0. In Figure 8.1, this can be seen for the Fixed Sequence (Seq) and Random (Rand) agents. For both of these agents, the decrease in F1-score for MCLvN is likely a decrease in overfitting, which is desirable. In the case of the Fixed Sequence agent, MCLv0 learned four MFs pertaining to the perception of the environment, which we know has no bearing on the agent's actual behavior. These MFs are thus artifacts, which may have artificially increased the F1-score for MCLv0. In the case of the Random agent, MCLv0 learned 28 MFs (all extraneous) which had the effect of increasing the learning performance slightly due to overfitting. MCLvN, which learned no MFs, did not invoke such overfitting. We thus conclude that MCLvN passes this functional evaluation in the vacuum cleaner domain.

8.3   Lawn Mower Experiments

This section describes our experiments and results from the lawn mower domain. First, we describe the lawn mower simulation we created for this research and the memory-influenced behaviors that are executed within this domain. Then, we describe the experiments we ran in this domain. Finally, we discuss results.

8.3.1   Lawn Mower Simulation Domain

This section describes the lawn mower simulation (LMS). Specifically, we discuss the motivation and mechanics behind LMS as well as the memory-influenced behaviors we hoped to induce in this domain.

The LMS domain was created in order to facilitate several memory-influenced human behaviors that we hoped to model with MCL. In order to ensure complete control over the testing environment (so as to guarantee that humans indeed used memory), we created a custom simulation from scratch in Unity. LMS was chosen as a tactical simulation testbed that was specifically designed for facilitating non-trivial and continuously-valued memory-influenced behaviors in human subjects. It was created to provide a variety of memory-influenced behaviors that MCL could learn and to serve as an indicator of definitive success or failure by MCLS, regarding our hypothesis.

In the LMS, a human controls a lawn mower in a simulation with the objective of mowing as many grass patches as possible while avoiding obstacles. The lawn mower is controlled via a Logitech Force 3D joystick, where the X and Y axes control horizontal and vertical movement, respectively. The joystick was selected as our input device for this domain for two reasons. First, the joystick provides a continuous value in the range -1 to +1 along each axis, depending on how fast a human desires to move the lawn mower. Continuous outputs were desired for this domain. Second, the

joystick is an relatively easy input device to use and yet is unwieldy enough to produce noise in a humans's movements, which increases the difficulty of the learning task for LfO.

There were 13 scenarios in the LMS, each of which focused on specific new memory-based entity (or a combination of such entities). The details behind each scenario are discussed in Appendix D. Typically, the scenario is preceded by a screen with text describing any new entities that are being introduced by that level, followed by another screen describing the minimum requirements for achieving the requirements of the scenario (e.g. "Get 60+ grass points and hit the exit in 60 seconds."). Then, the scenario begins.

Figure 8.4: Lawn Mower Simulation Game Play
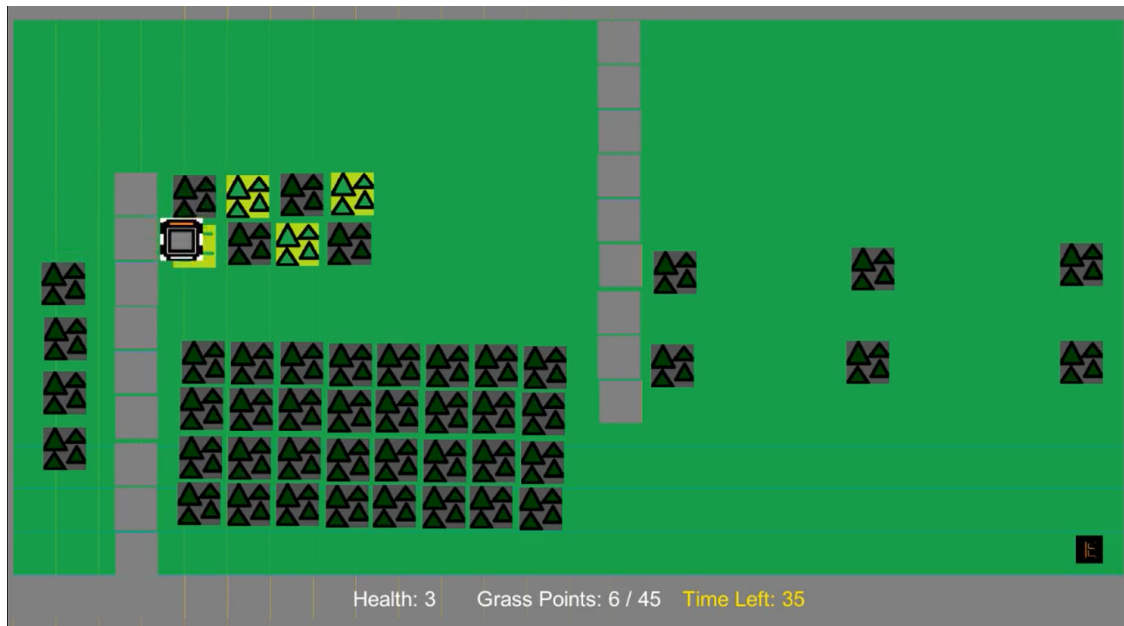
Figure 8.4 shows a screenshot of a human controlling the lawn mower in the LMS. The human controls a lawn mower (a gray square with tiny black wheels and an orange front bumper). The gray squares are impassable barriers. Grass patches are green squares with green triangles. The exit (initially black) activates (turns orange) when the human has mowed enough grass patches — the

249

human will finish the scenario upon arriving at the exit after it activates. The human's health, grass total (number obtained and number needed), and time remaining are displayed on the bottom of the screen. After the scenario ends, the human is shown the result of their performance (pass/fail). If the human mows enough grass patches and gets to the exit within the time limit, they move on to the next scenario. If the human runs out of time or their health points are reduced to zero, they must repeat the scenario. The human is given up to four attempts to meet the requirements of a scenario. On the human's last attempt, the human's health point total increases to 100 so that they can exhibit the full range of memory-influenced behaviors in the scenario without being subject to the health point constraint, but if they still fail to achieve the scenario requirements, they are transitioned to the next scenario.

The environment area in each scenario is the same size and the lawn mower always starts in the bottom left of the environment. Each scenario is on average 60 seconds long and the human begins with three health points each time (except on their last attempt). The purpose of the time limit is the ensure that the human is in constant motion and exhibiting mostly tactical behavior (rather than deliberative strategic behavior). The health points allow the human to make some mistakes (as is expected in human behavior in general), but it forces the human to behave mostly competently. The required number of grass patches to mow in a given scenario is set to a value that forces the human to move through most of the environment and encounter the various memory-based entities in the scenario. For scenarios that are more difficult, the required number of grass patches to mow is set to a value that is less than the maximum number of grass patches in order to make it easier for the human to fulfill the scenario requirements. The simulation is deterministic and the humans's joystick inputs for each time step are recorded to an external file.

The various memory-based entities in the simulation can be summarized in Table 8.3. Each scenario is designed around a single entity (or group of entities) such that it should be impossible for the human to successfully complete the scenario without demonstrating the desired memory-based

250

behavior in reaction to the entity in question. The environmental setup for each scenario for the LMS are described in Appendix D.

Table 8.3: Lawn Mower simulation entities

Entity Type	Boon	Hazard
Consumable	Grass	N/A
Persistent	Dark Grass	Rock
Memory Static	Gold Pot	Ground Hog
Memory Dynamic	Leprechaun	Snake
Memory Phased	Fairy	Sprinkler

In Table 8.3, the simulation entities are divided into "boons" (which the human actively seeks to collect) and "hazards" (which the human actively seeks to avoid). From there, the entities are divided into one of five categories:

1. Consumable — An entity that is visible until consumed.

2. Persistent — These entities are persistently visible.

3. Memory Static — These entities are temporarily visible, then potentially occluded (hidden from view) within a grass patch until the lawn mower collides with them.

4. Memory Dynamic — These entities move back-and-forth horizontally or vertically and become invisible when moving through a grass patch.

5. Memory Phased — These entities have massive areas of effect that trigger in reaction to the lawn mower's presence. Their behavior goes through multiple phases.

251

We now discuss each simulation entity in detail and which memory-influenced behavior (if any) it requires the human to perform.

The grass patch is the main commodity of the LMS. When the lawn mower moves over a grass patch, the grass patch takes on a "mowed appearance" so that the human doesn't try to mow it again; for all intents and purposes, the grass patch becomes "invisible".

The persistently visible entities are dark grass patches and rocks. Dark grass patches, like regular grass patches, can be mowed once for one point. However, a dark grass patch's mowed appearance is the same as its uncut appearance. This forces the human to remember which dark grass patches have already been mowed. The rocks are persistently visible and will decrease the lawn mower's health by one if the human tries to mow over them. The rocks are also indestructible.

The "memory static" entities (ground hogs and gold pots) are non-moving entities that are only temporarily visible to the human. These entities become visible to the human when the human comes within proximity. This is called "peeking". Then, if the entity is in a grass patch, it will become invisible to the human after a short time. This is called "hiding". If the lawn mower collides with a hidden memory static object, then the object becomes visible permanently. The LMS has two memory static objects: ground hogs and gold pots. The ground hog and gold pot are virtually the same, except that the former reduces the lawn mower's health by one upon a collision while the second restores one health by one (unless the lawn mower is already at maximum health). Also, gold pots are only usable one time. Memory is required because the human must remember that he or she saw the entity in that location and either actively go toward it or avoid it.

The "memory dynamic" entities (snakes and Leprechauns) are moving entities that are invisible when moving through grass patches. In the LMS, snakes cause the lawn mower to lose one health point and the Leprechauns restore one health point if the lawn mower is below maximum health. The snakes are indestructible (to avoid animal abuse) and may damage the lawn mower multiple

times, but Leprechauns will disappear after colliding with the lawn mower. Memory is required for addressing memory dynamic entities because when the entity becomes invisible, the human must not only remember that the entity is there in the grass, but the human must remember the entity's direction of movement and anticipate where the entity has moved over time. The entity will not change speed, but it may change direction if it collides with a barrier while it is hidden by grass.
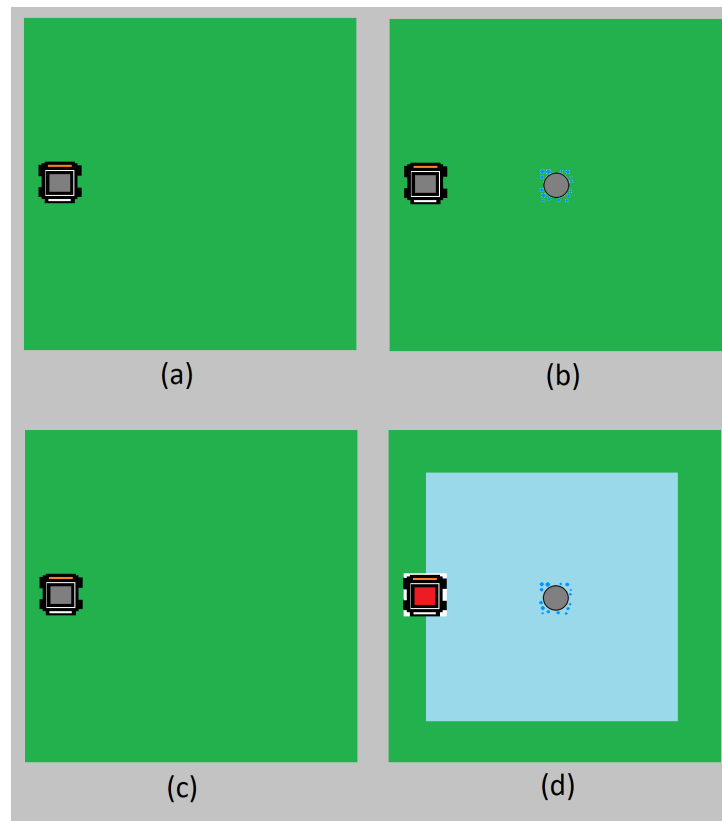
Figure 8.5: Sprinkler phases in Lawn Mower Simulation.

The "memory phased" entities (sprinklers and fairies) go through multiple "phases" in response to the lawn mower coming within proximity to them. This is illustrated in Figure 8.5 for the sprinkler. Initially, a sprinkler is in the "unspawned" phase (part a of the figure). When the lawn mower comes within proximity, the sprinkler enters the "peeking" phase and reveals itself to the

human (part b of the figure). After a short while, the entity turns invisible and enters the "standby" phase (part c of the figure). Then, the entity enters the "spray" phase (part d of the figure) and affects the lawn mower if the lawn mower is in the spray area. If it's a sprinkler, the lawn mower loses one health point if the lawn mower is in the spray area during this phase. If it's a fairy, the lawn mower regains all lost health if the lawn mower is in the spray area during this phase. Finally, the entity enters the "cool down" phase where it turns invisible again and remains so for a brief time period before entering the unspawned phase and again detecting if the lawn mower is within proximity. (Also, colliding with the sprinkler or fairy itself has no effect.) Memory is required to handle memory phased entities because the unspawned, standby, and cool down phases all look identical (the entity is invisible) and yet each requires a different response from the human.

8.3.2   Test Procedures

For each person who experienced the LMS scenarios, we generated a trace of their behavior for each of the 13 scenarios. For each trace, we tracked the state of the world using pie wedges, adapted from their use in [69]. We divided the world into 8 pie wedges centered around the lawn mower and relative to north. For a given entity type (e.g. rock, grass patch, etc.), we recorded the distance to the closest visible entity of that type for each pie wedge (if there was any such entity in that wedge). This is illustrated in Figure 8.6 — black lines denote pie wedge boundaries and red lines denote distances to the closest entity in the wedge (the entity type is "grass patch" in the figure). For sprinklers and fairies, we also recorded whether the entity was in the "peeking" phase or the "spray" phase (the sprinklers and fairies are invisible for all other phases). We recorded pie wedge distances for the following entity types: barrier, grass, dark grass, rock, gold pot, ground hog, Leprechaun, snake, sprinkler, fairy. We also recorded the distance to the exit, the bearing angle of the mower to the exit, and whether the exit was active. This is a total of 99 perception features. However, it should be noted that most levels only used a small subset of the entity types

254

and so most of the features in the traces held their default values for the duration of the trace.
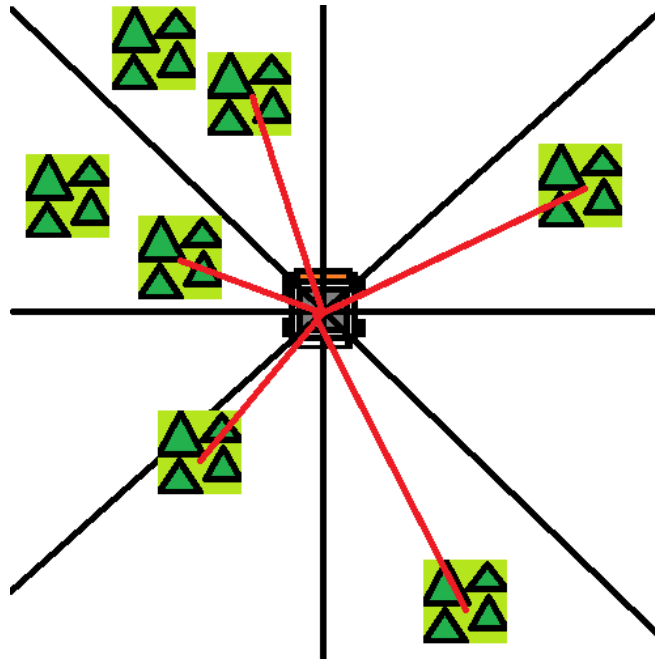
Figure 8.6: Pie wedges.

MCL was designed for applications with a single output feature, so we combined the joystick axis values generated during gameplay into a single output feature as follows:

1. Choose a value D denoting the number of discrete values the joystick can take for a single axis.

2. For each possible combination of discretized joystick inputs along both axes, assign a nominal value. There will be $D * D$ possible values.

3. For each time step in a human's trace, discretize their joystick inputs for both axes, then assign to the output variable the value corresponding to the combination of discretized joystick inputs that was observed.

For example, if we want each joystick axis to have $D = 5$ possible values, then there will be $D * D = 25$ possible values for the output feature that combines the joystick input values. Figure 8.7 shows how the XY coordinate space of possible joystick inputs are assigned to regions that represent combinations of discretized XY values. Each axis is divided into $D = 5$ possible spaces. There are 25 possible regions, corresponding to 25 possible values for the single output feature.
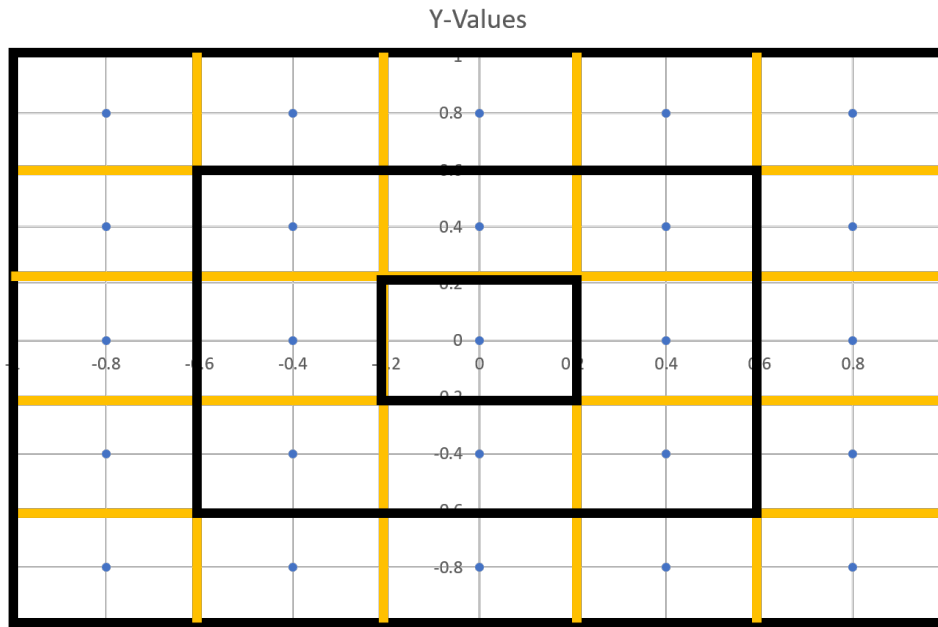
Figure 8.7: Combination of joystick inputs into one feature

In the figure, each region indicates a general direction for the joystick. The center region indicates relatively little movement of the joystick. The inner ring indicates eight possible directions the joystick could be going at a moderate speed. The outer ring indicates 16 possible directions the joystick could be going at high speed (three per direction, plus one per corner). This joystick output is what must be predicted by machine learning algorithms seeking to learn a human's policy in the LMS domain.

8.3.3 Experiments

For this domain, we collected trace data from 11 persons. We generated one trace for each of the 13 scenarios for a given person, resulting in a total of 143 traces.

The humans we collected data from are anonymous, identified only by the first one or two letters of the nicknames they chose for themselves when they participated in the study. Those identifiers are: AJ, AK, AW, CA, CH, D, G, L, P, T, W.

Table 8.4: Abbreviations for Lawn Mower Scenarios

Abbrevation	Scenario	Appearance Order
DG	Dark Grass Level	2
F1	Fairy Level 1	12
F2	Fairy Level 2	13
GP	Gold Pot Level	6
G1	Grass Test Level	4
G2	Grass Train Level	1
GH	Ground Hog Level	5
L	Leprechaun Level	9
R	Rock Level	3
Sn1	Snake Level 1	7
Sn2	Snake Level 2	8
Sp1	Sprinkler Level 1	10
Sp2	Sprinkler Level 2	11

Furthermore, we use abbreviated identifiers for the different scenarios in the LMS, which are shown in Table 8.4. The design of each scenario is covered in Appendix D. It should be noted that the G1 scenario (Grass Test Level) involves grass, rocks, and dark grass, not just grass.

Based on outputs by MCL, four types of memory-enhanced traces were generated from each base-line trace using various PPT/PAT thresholds, which are discussed in Chapter 7:

1. *State-Based*: A state-based memory-enhanced trace is created when MCL's PPT and PAT thresholds are 0.90 and 0.00, respectively. Only MFs based on perception features can be generated.

2. *Action-Based*: An action-based memory-enhanced trace is created when MCL's PPT and PAT thresholds are 0.00 and 0.90, respectively. Only MFs based on the Action feature can be generated.

3. *State/Action-Based*: A state/action-based memory-enhanced trace is created when MCL's PPT and PAT thresholds are both 0.90. MFs based on both perception and action can be generated.

4. *Hybrid*: A hybrid memory-enhanced trace is a combination of the state-based and action-based traces. It combines both the perception MFs from the state-based trace and the action MFs from the action-based trace.

The three PPT/PAT parameter sets used to generate state-based, action-based, and state/action-based traces were adopted from Floyd's work [20]. These parameter sets were geared toward learning behaviors that are dependent only on perception, only on action, or on both perception and action. However, in real-time domains, such as the lawn mower domain, it may be better to learn MFs based on perception features independently from MFs based on action features. There are two reasons for this: perception/action imbalance, and nominal/numeric feature imbalance.

First, the number of features that describe an agent's perception may differ drastically from the number of features that describe the agent's actions in the environment. MCL alternates between

creating perception MFs and action MFs. Within a given group of features (perception or action features), the different features essentially "compete" to be the feature upon which the new MF is based. Therefore, the memory influences associated with a specific perception (or action) feature may be discovered more slowly if there are many perception (or action) features. Thus, if there are many more perception features than there are action features (or vice versa), then the memory influences associated with perception may be learned by MCL more slowly than memory influences associated with action features (or vice versa). The consequence of this imbalance is that MCL may attempt to learn additional (and potentially extraneous) MFs from action features to account for memory influences for perception features that have yet to be learned. This leads to inaccuracy in modeling an observed entity's memory.

Second, learning MFs may suffer from nominal/numeric feature imbalance. Nominal features have a fixed number of possible values. Two nominal values are either completely the same or completely different. In contrast, numeric features have an infinite number of possible values, but these values can be partially similar (e.g. 2.0 and 3.0 are more similar to each other than 2.0 and 100.0).

In a real time environment, numeric feature values will in general change gradually from time step to time step. An example of this is distance – the distance between an agent and an object of interest will gradually decrease as the agent moves toward it, time step to time step. In contrast, nominal feature values can change drastically from time step to time step because any change in the feature's value will make it completely dissimilar from previous values. Because it is easier for nominal features to experience significant change over time than it is for numeric features, MFs will more readily be created for nominal features during MCL's Extraction process.

Both perception/action imbalance and nominal/numeric imbalance are present in the lawn mower domain, where there are 99 perception features and 1 action feature, and 19 nominal perception

features and 80 numeric perception features.

The first measure we took to address these imbalances was to evaluate learning performance on hybrid memory-enhanced traces (as defined previously). This addresses the first imbalance by allowing perception MFs and action MFs to be learned independently from each other. This also addresses the second imbalance, because the singular action feature is a nominal feature, in contrast to the perception features, which are mostly numeric.

The second measure we took to address the perception/action imbalance was to down-sample our traces from 60 Hz to 10 Hz. This allows numeric features to change more drastically from time step to time step, because time steps will now have approximately 0.1 seconds between them instead of only about 0.016 seconds between them.

For each PPT/PAT parameter set, we tested the following high/low filtering thresholds for MF Refinement:

- HF=0.10, LF=0.05

- HF=0.10, LF=0.025

- HF=0.10, LF=0.01

- HF=0.05, LF=0.05

- HF=0.05, LF=0.025

- HF=0.05, LF=0.01

- HF=0.025, LF=0.025

- HF=0.025, LF=0.01

The filtering thresholds determine which MFs from the Extraction phase are retained in the final refined MF set that is used to create the memory-enhanced traces. For each HF/LF parameter set, we computed the number of cases in the case base that were allocated to a context other than the default context. The results for the parameter set that minimized the default context size were used for a given trace.

8.3.4   Results — MCL Version N

This section describes the results of MCLS (MCLvN) for LMS. With the merits of MCLvN having been verified in the vacuum cleaner domain, we now now proceed to compare machine learning performance on memoryless (baseline) traces compared to that for memory-enhanced traces generated by MCLvN.

For each human subject, we analyze three types of results:

1. *Memory Statistics*: We analyze the optimal HF/LF parameters and the categorization of the case base regarding memory.

2. *Memory Models*: We analyze the MFs that were learned by MCL for each trace.

3. *Learning Performance*: We analyze the learning performance of various algorithms for the traces associated with each level.

Regarding the Memory Statistics results, we compute the following statistics for each trace:

- The optimal High Filter (HF) and Low Filter (LF) parameters that minimized the size of the default context for the trace.

261

- The number of memory sets/contexts (Set Count) that contributed to the final refined MF set, after being filtered during MCL's refinement process. This number can be low or high, as long as all memory influences are comprehensively expressed.

- The number of cases (Mem Cases) that did not fall under the default context, and the number of cases that did (Default Cases). We want the former metric to be high and the latter one to be low.

- The number of MFs (MF Count) that were learned for the trace. This number can be low or high, as long as all memory influences are comprehensively expressed.

Regarding learning performance, we computed two results:

1. F1-scores, computed via 10-fold cross validation on the entire trace.

2. F1-scores, computed via training on 66% of the trace and testing on the remaining 33%.

For our results, we present two caveats:

1. The stochastic learner Bayes Net may offer different learning results when initialized with a different seed. The results presented here are the result of single trial learning, wherein a single learning result is computed. Thus, the results averaged over multiple runs may differ slightly from those presented in this section.

2. Qualitatively, we analyze differences in F1 scores on an absolute basis rather than through statistical testing. The behaviors of a human entity in the same situation may differ from run to run. However, it is not empirically possible to "wipe" a person's memory between runs, and so the behavior in subsequent runs may be affected by a person's memory of a scenario from a prior attempt. Thus, results are only computed for a person's final attempt

on a given scenario. Any claim that one result is "better" than another is based on an absolute comparison.

First, we analyze the results for each human individually. Then, we offer insights into common trends across all humans overall for the lawn mower domain.

8.3.4.1 Human AJ

Tables 8.5, 8.6, and 8.7 show the optimal parameters and memory categorization of the case base for action-based, state-based, and action/state-based traces, respectively.

Table 8.5: Action-Based Memory Parameters – Human AJ

Person	Scenario	HF	LF	Set Count	Mem Cases	Default Cases	MF Count
AJ	DG	0.05	0.05	30	210	81	2
AJ	F1	0.05	0.05	35	187	65	2
AJ	F2	0.05	0.05	37	206	79	3
AJ	G1	0.05	0.05	46	247	87	2
AJ	G2	0.05	0.05	40	240	172	4
AJ	GH	0.05	0.05	47	227	121	2
AJ	GP	0.025	0.025	42	263	73	3
AJ	L	0.05	0.05	38	273	107	2
AJ	R	0.05	0.05	23	171	66	1
AJ	Sn1	0.05	0.05	56	231	153	4
AJ	Sn2	0.05	0.05	24	190	52	3
AJ	Sp1	0.05	0.05	100	445	283	3
AJ	Sp2	0.05	0.05	32	247	81	4

From Table 8.5, we can see that for the majority of the LMS scenarios, the optimal HF/LF param-

eters action-based memory-enhanced traces (PPT=0.00, PAT=0.90) for Human AJ were 0.05 and 0.05. The exception was the Gold Pot (GP) scenario, which gained the smallest default context size for for HF=0.025, LF=0.025. In general, about one third of the case base for each scenario fell under the default context. At most, four MFs were learned for a trace, but the number of unique memory sets that contributed to the final refined set ranged from 23 to 100.

Table 8.6: State-Based Memory Parameters – Human AJ

Person	Scenario	HF	LF	Set Count	Mem Cases	Default Cases	MF Count
AJ	DG	0.025	0.025	17	248	43	1
AJ	F1	0.05	0.05	11	231	21	3
AJ	F2	0.05	0.05	25	211	74	5
AJ	G1	0.05	0.05	20	278	56	1
AJ	G2	0.025	0.01	1	412	0	0
AJ	GH	0.05	0.05	13	314	34	1
AJ	GP	0.05	0.05	25	256	80	1
AJ	L	0.05	0.05	47	236	144	3
AJ	R	0.05	0.05	13	197	40	0
AJ	Sn1	0.05	0.05	24	273	111	2
AJ	Sn2	0.05	0.05	13	205	37	2
AJ	Sp1	0.05	0.05	88	407	321	0
AJ	Sp2	0.05	0.05	7	304	24	0

For state-based memory-enhanced traces (see Table 8.6), there were in general fewer memory sets and smaller default contexts. Fewer MFs were learned on average for each scenario. Again, the optimal HF/LF parameters were 0.05 and 0.05 for most scenarios, with the exception of the G2 (Grass Train Level) and DG (Dark Grass) scenarios, which did better under less constrictive filtering thresholds. In cases where no MFs were learned, human AJ's behavior was likely deemed too random to infer meaningful MFs for.

Table 8.7: State/Action-Based Memory Parameters – Human AJ

Person	Scenario	HF	LF	Set Count	Mem Cases	Default Cases	MF Count
AJ	DG	0.025	0.01	30	267	24	6
AJ	F1	0.025	0.01	35	218	34	5
AJ	F2	0.025	0.01	37	257	28	6
AJ	G1	0.025	0.01	46	281	53	4
AJ	G2	0.025	0.01	40	361	51	9
AJ	GH	0.025	0.01	47	292	56	6
AJ	GP	0.025	0.01	42	281	55	4
AJ	L	0.025	0.01	38	344	36	8
AJ	R	0.025	0.01	23	223	14	8
AJ	Sn1	0.025	0.01	56	337	47	9
AJ	Sn2	0.025	0.01	24	217	25	5
AJ	Sp1	0.025	0.01	100	553	175	6
AJ	Sp2	0.025	0.01	32	298	30	6

For state/action-based memory-enhanced traces (see Table 8.7), an interesting trend is that the least constrictive HF/LF parameters (HF=0.025, LF=0.01) yielded the best results. Except for the Sp1 (Sprinkler 1) scenario, the default context sizes for the scenarios are relatively small. Furthermore, the number of MFs learned for each scenario are on average higher than the MF counts for the state-based and action-based traces.

The following MFs were learned for the action-based memory-enhanced traces for Human AJ.

- *DG*: v-Action(1), v-Action(2)

- *F1*: v-Action(1), v-Action(2)

- *F2*: v-Action(1), v-Action(2), v-Action(3)

- *G1*: v-Action(1), v-Action(2)

- *G2*: t-Action(S), t-Action(Se), v-Action(1), v-Action(2)

- *GH*: v-Action(1), v-Action(2)

- *GP*: v-Action(1), v-Action(2), v-Action(3)

- *L*: v-Action(1), v-Action(2)

- *R*: v-Action(1)

- *Sn1*: t-Action(w), t-Action(C), v-Action(1), v-Action(2)

- *Sn2*: v-Action(1), v-Action(2), v-Action(3)

- *Sp1*: v-Action(1), v-Action(2), v-Action(3)

- *Sp2*: t-Action(sW), t-Action(sw), v-Action(1), v-Action(2)

In general, the mostly commonly learned MFs were value-back MFs for Action at time-back values of 1-3. For the G2 and Sp2 scenario, time-back MFs for downward joystick positions were learned and for the Sn1 scenario, a time-back MF for the motionless joystick position was learned. This indicates that Human AJ's behavior was largely reactive when only focusing on their choice of joystick position, as evidenced by most memory influences being confined to a few time steps back in memory. However, it appears that downward and motionless joystick positions may have been instrumental in memory-based behavior for a couple scenarios.

The following MFs were learned for the state-based memory-enhanced traces for Human AJ.

- *DG*: t-DarkGrassW4Dist(1000.0)

- *F1*: t-FairyW4Dist(1000.0), t-RockW4Dist(4.100), v-RockW7Dist(29)

- *F2*: t-FairyW2Dist(1000.0), t-FairyW4Dist(1000.0), t-RockW7Dist(1000.0), v-FairyW5Dist(98), v-RockW3Dist(111)

- *G1*: t-DarkGrassW4Dist(1000.0)

- *G2*: none

- *GH*: t-GrassW0Dist(1000.0)

- *GP*: t-GoldPotW2Dist(1000.0)

- *L*: t-GrassW0Dist(1000.0), t-GrassW4Dist(1000.0), t-SnakeW7Dist(1000.0)

- *R*: none

- *Sn1*: t-GrassW3Dist(1000.0), t-SnakeW0Dist(1000.0)

- *Sn2*: t-GrassW0Dist(1000.0), v-GrassW2Dist(17)

- *Sp1*: none

- *Sp2*: none

For Human AJ, we can see that Wedge 4 (which covers the area to the human's southwest) and to a lesser degree Wedges 0 and 2 (which cover the areas to the human's northeast and southeast, respectively) are prominent in several learned MFs. Furthermore, the majority of learned MFs are time-back MFs tracking the value 1000.0 (which indicates that there is no entity of a given type in that wedge). This indicates that these MFs were learned to account for memory of when a wedge was last unoccupied by a given entity type.

The following MFs were learned for the state/action-based memory-enhanced traces for Human AJ.

- *DG*: t-Action(w), t-Action(E), t-Action(S), v-Action(1), v-Action(2), v-Action(3)

- *F1*: t-Action(C), t-Action(e), t-Action(N), v-Action(1), v-Action(2)

- *F2*: t-Action(C), t-Action(E), t-Action(nE), v-Action(1), v-Action(2), v-Action(3)

- *G1*: v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *G2*: t-Action(E), t-Action(nE), t-Action(S), t-Action(Nw), t-Action(N), t-Action(Ne), t-Action(Se), v-Action(1), v-Action(2)

- *GH*: t-Action(E), t-Action(S), t-Action(N), v-Action(1), v-Action(2), v-Action(3)

- *GP*: v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *L*: t-Action(C), t-Action(n), t-Action(N), t-Action(Se), t-Action(se), v-Action(1), v-Action(2), v-Action(3)

- *R*: t-Action(E), t-Action(nE), t-Action(S), t-Action(N), t-Action(Ne), t-Action(Se), v-Action(1), v-Action(2)

- *Sn1*: t-Action(w), t-Action(C), t-Action(ne), t-Action(N), v-Action(1), v-Action(2), v-Action(3), v-Action(4), v-Action(5)

- *Sn2*: t-Action(N), t-Action(Ne), v-Action(1), v-Action(2), v-Action(3)

- *Sp1*: t-Action(C), v-Action(1), v-Action(2), v-Action(3), v-Action(4), v-Action(5)

- *Sp2*: t-Action(nw), t-Action(n), t-Action(sW), t-Action(sw), v-Action(1), v-Action(2)

Most notably, even though MFs for both perception and action features can be generated for state/action-based memory-enhanced traces (PPT=0.90, PAT=0.90), only action MFs were learned for Human AJ for these traces. It is as though allowing tree extraction to create branches based

268

on differences between cases in perception features served mostly to allow more nuanced memory-influences based on the Action feature to be learned, rather than facilitating the creation of perception-based MFs. This is evidenced by the presence of several time-back action MFs that were not present for the action-based traces. A second observation is that for the time-back MFs, the value for "West" (left joystick inputs) is rarely present in the MFs. This result is intuitive, given that the LMS scenarios generally require the human to move from left to right. A third observation is that value-back MFs for Action at time-back values of 1-3 were learned for almost every scenario, which indicates that Human AJ's joystick inputs are typically consistent within a few time steps.
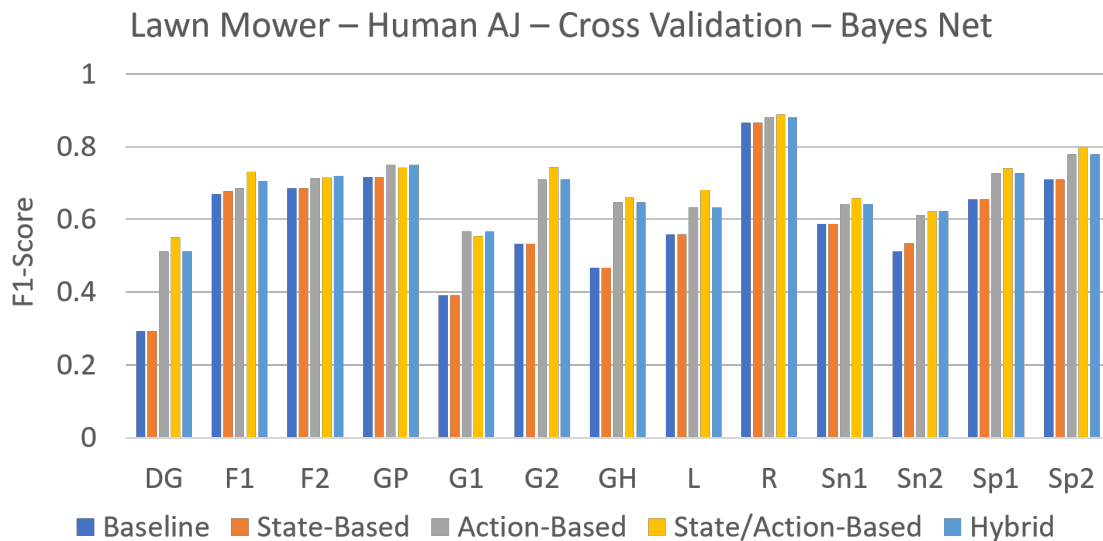
Figure 8.8: LMS Results, Human AJ, Bayes Net, Cross Validation

Figure 8.8 shows the cross validation results for the Bayes Net learning algorithm. For each level, the F1 score for baseline (memory-less) traces is compared to four memory-enhanced trace types: state-based, action-based, state/action-based, and hybrid. Again, hybrid traces simply use the superset of MFs learned for state-based and action-based traces.

There are two trends apparent from the figure. First, the state-based memory-enhanced traces did

269

not result in significant improvement in F1 scores. This result is surprising, given how perception MFs resulted in very small default context sizes in our analysis of Memory Parameters results for state-based traces for Human AJ. However, it should be noted that the addition of perception MFs did not result in lower F1 scores than those for the baseline traces.

The second trend is that the other three memory-enhanced trace types (the ones that allow for Action MFs) generally improved learning performance, especially for the DG, G1, G2, and GH scenarios. More specifically, the state/action-based traces in several cases (e.g. the DG, G2, GH, L, Sn1, Sp2 scenarios) outperformed the hybrid traces, which suggests that tree extraction that compares cases for both perception and action (even if only Action MFs are learned) generally results in better learning performance than simply learning perception and action MFs independently, then combining them. The one exception to this result is seen in the G1 scenario, where hybrid traces resulted in a higher F1 score. However, hybrid traces resulted in F1 scores equivalent to those of the action-based traces.

Figure 8.9 shows the F1 scores that result from the learning algorithm training on 66% of the trace and being evaluated on the remaining 33%. Understandably, the F1 scores are much lower, given that there is less training data available to the learner in this evaluation. We still observe that the memory-enhanced traces result in F1 scores that are no worse than those for the baseline trace. In particular, higher scores are observed for action-based, state/action-based, and hybrid traces, though the increases are less than those observed for the cross validation evaluation.

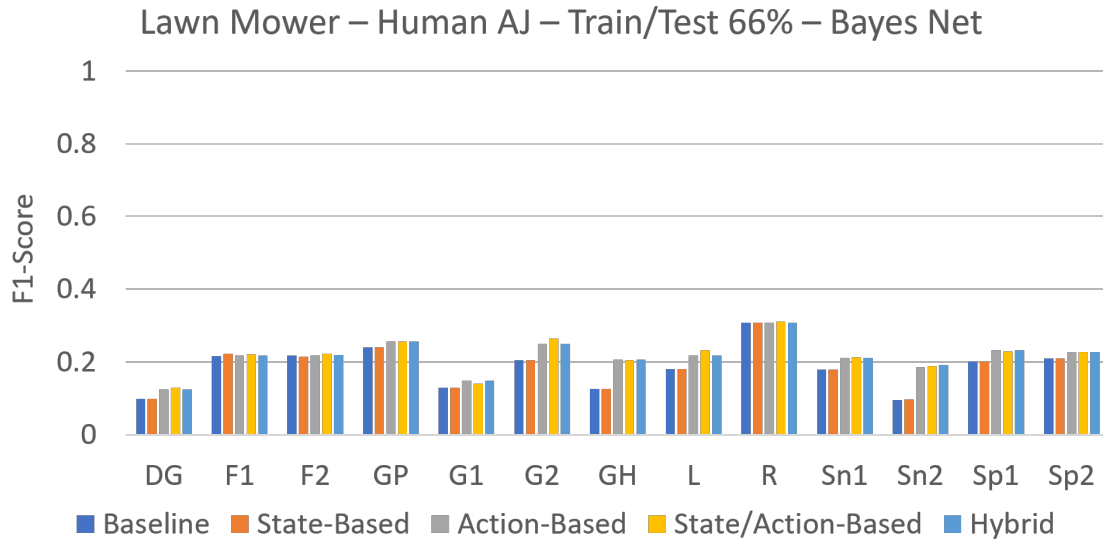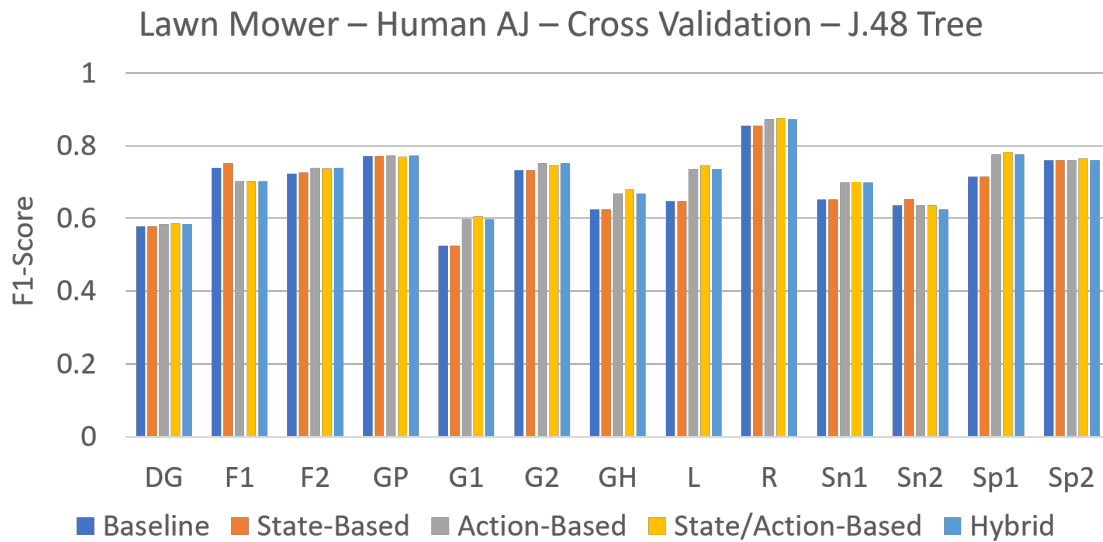Figure 8.9: LMS Results, Human AJ, Bayes Net, Train/Test Split

Figure 8.10: LMS Results, Human AJ, J.48 Tree, Cross Validation

Figure 8.10 shows the results of cross validation for the J.48 Decision Tree algorithm. The differ-

ences between the results for the J.48 Tree learner and those of the Bayes Net learner are twofold. First, the results for the Dark Grass (DG) scenario are the same for all traces for J.48 Tree, even though the Bayes Net learner saw the greatest learning improvements for this scenario when using the memory-enhanced traces. In addition to the learning improvements for the memory-enhanced traces being more muted for J.48 Tree than for Bayes Net, the memory-enhanced traces result in lower F1 scores than those for the baseline traces in the F1 scenario (and for the hybrid trace in the Sn2 scenario). Interestingly, the state-based traces (which allow for only perception MFs) for these scenarios have the highest F1 score out of all the traces (including baseline traces). This means that for certain scenarios, modeling memory from perception features alone may be the appropriate approach.
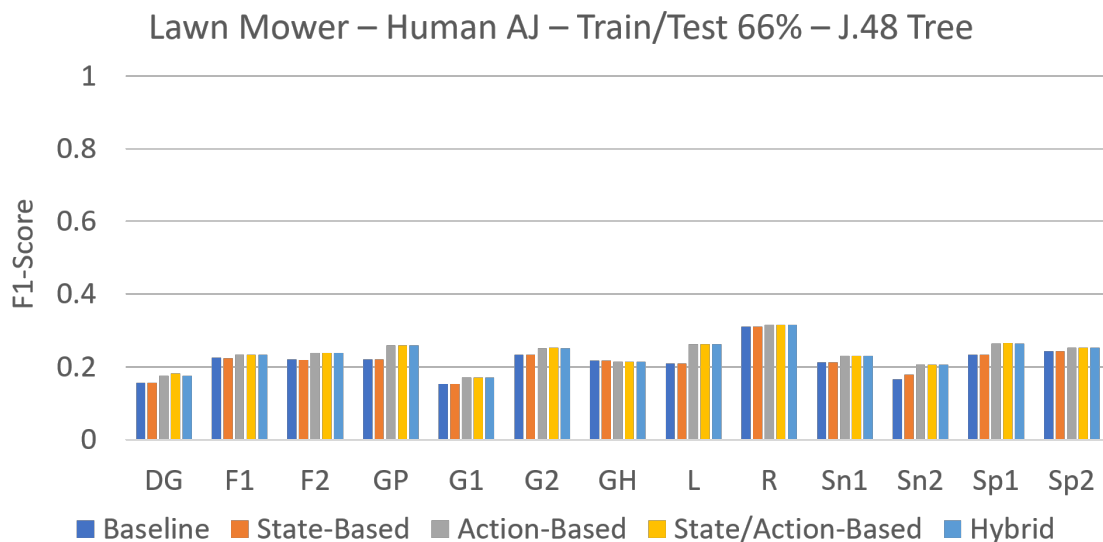

Figure 8.11: LMS Results, Human AJ, J.48 Tree, Train/Test Split

Figure 8.11 shows the train/test evaluation results for the J.48 Tree learner. Even though the F1 scores are lower than those for the cross validation evaluation, the action-based, state/action-based, and hybrid traces always resulted in higher F1 scores than those of the baseline traces, which

suggests that Action MFs offer an advantage to a learner's ability to generalize its procedural knowledge to unseen situations.

8.3.4.2 Human AK

The Memory Parameter analysis for Human AK is shown in Tables 8.8, 8.9, and 8.10 for action-based, state-based, and state/action-based memory-enhanced traces, respectively.

Table 8.8: Action-Based Memory Parameters – Human AK

Person	Scenario	HF	LF	Set Count	Mem Cases	Default Cases	MF Count
AK	DG	0.05	0.05	44	234	106	2
AK	F1	0.05	0.05	55	251	133	4
AK	F2	0.05	0.05	17	153	31	1
AK	G1	0.05	0.05	46	342	91	3
AK	G2	0.05	0.05	87	336	229	3
AK	GH	0.05	0.05	37	237	77	2
AK	GP	0.05	0.05	49	239	124	1
AK	L	0.05	0.05	47	263	97	1
AK	R	0.05	0.05	27	183	53	1
AK	Sn1	0.025	0.025	55	266	104	4
AK	Sn2	0.025	0.025	61	391	124	3
AK	Sp1	0.05	0.05	55	378	193	2
AK	Sp2	0.05	0.05	47	250	110	3

From Table 8.8, we can see that the most common HF/LF parameters were HF=0.05 and LF=0.05, with the Snake scenarios (Sn1 and Sn2) being the exception by using less constrictive thresholds. The number of memory sets used for the different scenarios varied from 17 to 87, but the number of MFs in the final refined set only varied from 1 to 4. We also observe that the default context

273

sizes are relatively large, even accounting for two-thirds of the case base in the G2 scenario (which features only grass).

Table 8.9: State-Based Memory Parameters – Human AK

Person	Scenario	HF	LF	Set Count	Mem Cases	Default Cases	MF Count
AK	DG	0.05	0.05	26	273	67	5
AK	F1	0.05	0.05	24	296	88	0
AK	F2	0.05	0.05	5	171	13	0
AK	G1	0.05	0.05	29	320	113	4
AK	G2	0.025	0.01	1	565	0	0
AK	GH	0.05	0.05	15	260	54	1
AK	GP	0.025	0.025	20	305	58	0
AK	L	0.05	0.05	20	290	70	0
AK	R	0.05	0.05	13	198	38	0
AK	Sn1	0.05	0.05	28	229	141	2
AK	Sn2	0.05	0.05	58	306	209	1
AK	Sp1	0.05	0.05	26	428	143	2
AK	Sp2	0.05	0.05	15	313	47	3

For Table 8.9, we observe that fewer MFs in general were learned for state-based traces. The optimal values for the HF/LF thresholds again were generally HF=0.05 and LF=0.05. Interestingly, the scenario with the smallest default context size (G2) for the state-based traces had one of the largest proportions of cases in the default context for action-based traces, which indicates that perception memories may be more important for this particular scenario.

Table 8.10: State/Action-Based Memory Parameters – Human AK

Person	Scenario	HF	LF	Set Count	Mem Cases	Default Cases	MF Count
AK	DG	0.025	0.01	44	293	47	6
AK	F1	0.025	0.01	55	321	63	7
AK	F2	0.025	0.01	17	176	8	4
AK	G1	0.025	0.01	46	379	54	4
AK	G2	0.025	0.01	87	436	129	7
AK	GH	0.025	0.01	37	274	40	4
AK	GP	0.025	0.01	49	318	45	7
AK	L	0.025	0.01	47	298	62	2
AK	R	0.025	0.01	27	213	23	4
AK	Sn1	0.025	0.01	55	306	64	6
AK	Sn2	0.025	0.01	61	423	92	3
AK	Sp1	0.025	0.01	55	495	76	5
AK	Sp2	0.025	0.01	47	316	44	6

For state/action-based traces (see Table 8.10), we again see that the least constrictive HF/LF values yielded the best results. The most number of MFs were generally learned for these traces with relatively small default context sizes. The number of memory sets used is comparable to the numbers for action-based traces.

The following MFs were learned for the action-based traces for Human AK.

- *DG*: v-Action(1), v-Action(2)

- *F1*: t-Action(C), t-Action(e), v-Action(1), v-Action(2)

- *F2*: v-Action(1)

- *G1*: v-Action(1), v-Action(2), v-Action(3)

- *G2*: v-Action(1), v-Action(2), v-Action(3)

- *GH*: v-Action(1), v-Action(2)

- *GP*: v-Action(1)

- *L*: v-Action(1)

- *R*: v-Action(1)

- *Sn1*: t-Action(w), t-Action(C), v-Action(1), v-Action(2)

- *Sn2*: v-Action(1), v-Action(2), v-Action(3)

- *Sp1*: t-Action(C), v-Action(1)

- *Sp2*: t-Action(C), v-Action(1), v-Action(2)

We observe that MCL learned mostly value-back MFs for Human AK for the action-based traces. The time-back parameters for these MFs are constrained to just the most recent three time steps. However, we also observe that the motionless joystick input is the value parameter for the time-back MFs learned for the two Sprinkler scenarios (Sp1 and Sp2). This result is intuitive, given how one must wait in place for a sprinkler to transition from its spray phase to its cooldown phase before safely proceeding to the next part of the scenario. The same time-back MF is also observed for the first Snake scenario (Sn1), which indicates an effort by Human AK to use patient waiting to allow the snake entities to move out of the intended mower trajectory.

The following MFs were learned for the state-based traces for Human AK.

- *DG*: t-DarkGrassW0Dist(1000.0), t-DarkGrassW7Dist(1000.0), t-GrassW7Dist(1000.0), v-DarkGrassW4Dist(188), v-ExitState(17)

- *F1*: none

- *F2*: none

- *G1*: t-DarkGrassW0Dist(1000.0), t-DarkGrassW4Dist(1000.0), t-DarkGrassW7Dist(1000.0), t-RockW4Dist(1000.0)

- *G2*: none

- *GH*: t-GrHogW7Dist(1000.0)

- *GP*: none

- *L*: none

- *R*: none

- *Sn1*: t-ExitState(Off), t-ExitState(On)

- *Sn2*: t-GrassW4Dist(1000.0)

- *Sp1*: t-GrassW4Dist(1000.0), t-GrassW6Dist(1000.0)

- *Sp2*: t-GrassW4Dist(1000.0), v-GrassW3Dist(29), v-SprinklerW4Dist(79)

We observe that several MFs based on the Dark Grass wedge distance features were learned for both the dedicated Dark Grass (DG) level and for the conglomerate Grass Test (G1) level. However, these MFs were all for the maximum distance 1000.0, which indicates that Human AK's behavior depends on which wedges used to be devoid of Dark Grass patches. These memories perhaps informed Human AK as to which wedges contained newer dark grass patches that were likely unmowed.

277

We also observe a couple MFs based on the ExitState feature, which were learned for the DG and Sn1 scenarios. This suggests that the activation of the Exit entity in these scenarios resulted in delayed, but significantly different behavior by Human AK in reaction to the exit.

The following MFs were learned for the state/action-based traces for Human AK.

- *DG*: t-Action(N), t-Action(SE), v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *F1*: t-Action(C), t-Action(e), t-Action(E), t-Action(N), v-Action(1), v-Action(2), v-Action(3)

- *F2*: t-Action(C), t-Action(E), t-Action(N), v-Action(1)

- *G1*: v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *G2*: t-Action(S), t-Action(NE), t-Action(SE), v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *GH*: t-Action(nE), v-Action(1), v-Action(2), v-Action(3)

- *GP*: t-Action(ne), t-Action(N), t-Action(Ne), t-Action(NE), v-Action(1), v-Action(2), v-Action(3)

- *L*: v-Action(1), v-Action(2)

- *R*: t-Action(Ne), t-Action(NE), v-Action(1), v-Action(2)

- *Sn1*: t-Action(w), t-Action(C), v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *Sn2*: v-Action(1), v-Action(2), v-Action(3)

- *Sp1*: t-Action(C), t-Action(e), v-Action(1), v-Action(2), v-Action(3)

- *Sp2*: t-Action(C), t-Action(e), v-Action(1), v-Action(2), v-Action(3), v-Action(4)

Similar to Human AJ, Human AK's behavior resulted in MCL learning no MFs based on perception for the state/action-based memory-enhanced traces. Instead, several time-back MFs for Action were learned. Again, we observe the absence of left or "West" joystick inputs in the learned MFs, and we observe the strong presence of value-back Action MFs with time-back parameters confined mostly to the previous 3-4 time steps.
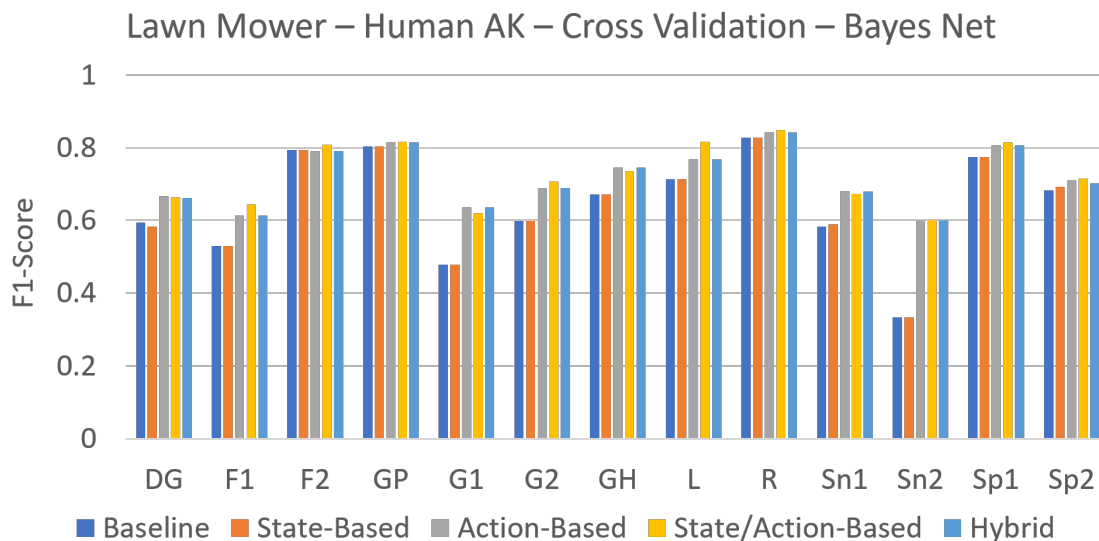

Figure 8.12: LMS Results, Human AK, Bayes Net, Cross Validation

Figure 8.12 shows the cross validation F1 scores for the Bayes Net learner. Except for the DG scenario (for which a relatively larger number of MFs were learned), the state-based traces resulted in F1 scores equivalent to those of the baseline traces. In the majority of the scenarios, the action-based, state/action-based, and hybrid traces resulted in higher F1 scores than those for the baseline traces. (The exception being the second Fairy scenario, F2, where the F1 scores were slightly worse for action-based and hybrid traces.)

For a few scenarios, the state/action-based trace F1 scores were higher than those the hybrid traces (e.g. F1, G2, L), and they generally resulted in F1 scores that were no less than those of the hybrid
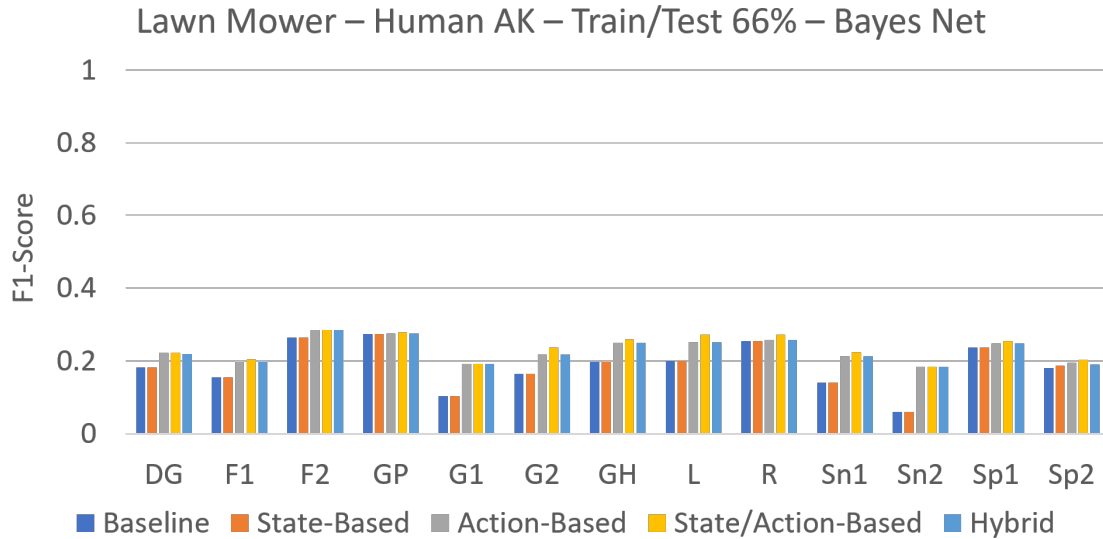
traces across all scenarios.

Figure 8.13: LMS Results, Human AK, Bayes Net, Train/Test Split

In Figure 8.13, we observe that the memory-enhanced traces of all types except state-based saw the greatest increases in F1-score for the G1 and Sn1 scenarios, which was also the case in the cross validation evaluation. This provides evidence that generalizability of machine learning can be improved with the addition of MFs for agent actions.

Figure 8.14 shows the results of cross validation for the J.48 Decision Tree. We observe that there is not much improvement in F1 score for the memory-enhanced traces, and that there is even a decrease in score for the DG (Dark Grass), F1 (Fairy 1), and GP (Gold Pot) scenarios. This could either be caused by the behavior by Human AK being mostly reactive (and thus the learned MFs were extraneous and harmful for learning) or that the learned MFs were not comprehensive enough to fully capture the memory influences on Human AK's behavior. These results are interesting, in light of the positive results for the Bayes Net learner.
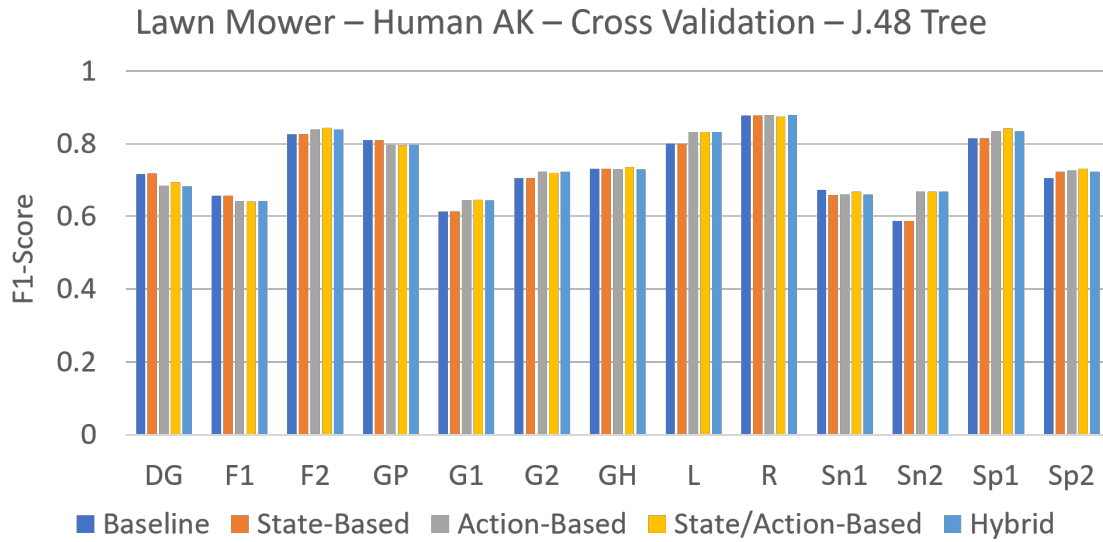
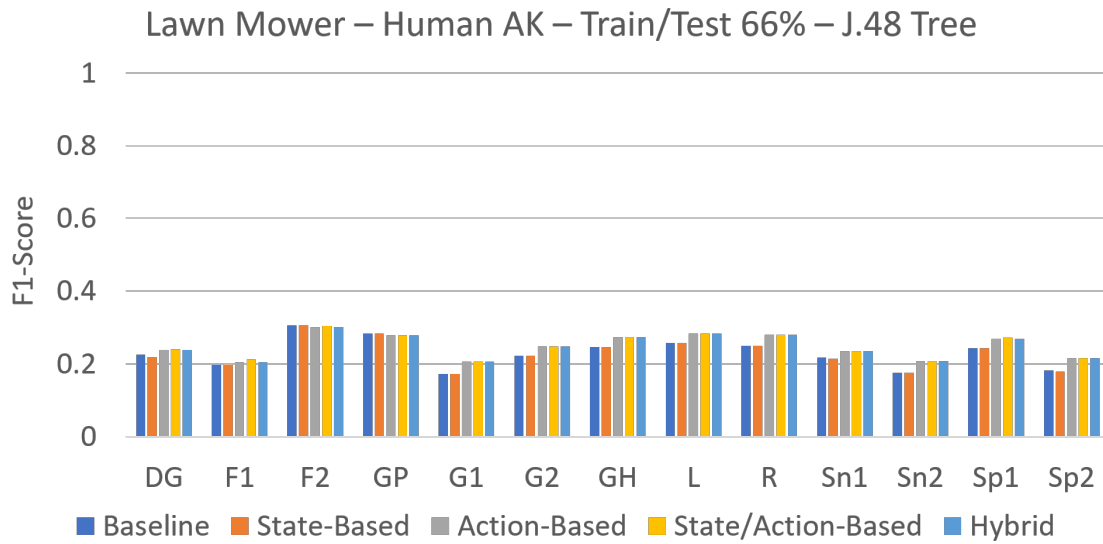Figure 8.14: LMS Results, Human AK, J.48 Tree, Cross Validation

Figure 8.15: LMS Results, Human AK, J.48 Tree, Train/Test Split

Figure 8.15 shows the results of the train/test split evaluation for J.48 Tree. Here, we see F1 score

increases for the memory-enhanced traces for several scenarios. In particular, the increases in the DG and F1 scenarios, though slight, are in direct contrast to the decrease in score that was observed for the cross validation evaluation.

8.3.4.3   Human AW

Tables 8.11, 8.12, and 8.13 show the Memory Parameters statistics for action-based, state-based, and action/state-based memory-enhanced traces, respectively.

Table 8.11: Action-Based Memory Parameters – Human AW

Person	Scenario	HF	LF	Set Count	Mem Cases	Default Cases	MF Count
AW	DG	0.05	0.05	24	224	64	2
AW	F1	0.05	0.05	49	207	97	2
AW	F2	0.05	0.05	29	149	43	2
AW	G1	0.05	0.05	54	351	121	2
AW	G2	0.05	0.05	45	285	124	3
AW	GH	0.05	0.05	42	260	111	2
AW	GP	0.05	0.05	47	277	114	4
AW	L	0.05	0.05	62	225	190	2
AW	R	0.05	0.05	16	248	23	3
AW	Sn1	0.05	0.05	52	352	170	2
AW	Sn2	0.05	0.05	46	263	90	2
AW	Sp1	0.05	0.05	42	315	117	4
AW	Sp2	0.05	0.05	48	272	113	2

In all scenarios, the optimal values for HF and LF thresholds were HF=0.05 and LF=0.05. The number of learned MFs for each scenario was 2-4 MFs. The smallest default context size was seen in the Rock (R) scenario, which is intuitive because there are no occluded entities in that scenario.

282

Table 8.12: State-Based Memory Parameters – Human AW

Person	Scenario	HF	LF	Set Count	Mem Cases	Default Cases	MF Count
AW	DG	0.05	0.05	45	155	133	1
AW	F1	0.05	0.05	36	189	115	2
AW	F2	0.05	0.05	11	162	30	1
AW	G1	0.025	0.025	62	303	169	2
AW	G2	0.025	0.01	1	409	0	0
AW	GH	0.05	0.05	18	311	60	0
AW	GP	0.05	0.05	32	295	96	1
AW	L	0.05	0.05	22	271	144	0
AW	R	0.05	0.05	14	234	37	0
AW	Sn1	0.025	0.025	48	379	143	2
AW	Sn2	0.05	0.05	20	260	93	1
AW	Sp1	0.025	0.025	74	261	171	1
AW	Sp2	0.05	0.05	12	316	69	2

For state-based traces (see Table 8.12), the HF/LF parameters were optimally around 0.05 on average, which a few exceptions, most notably the Grass Training (G2) scenario. For the G2 scenario, no MFs were learned and the default context was non-existent, which suggests completely reactive behavior for that scenario. A similar trend is seen for the Rock (R) scenario, which a small default context size and no learned MFs. This is intuitive, given that the Rock scenario has no occluded entities and thus a small memory requirement on part of the human. Even though the Leprechaun scenario resulted in no learned MFs, the default context size is high, which indicates that there are memory influences that MCL simply had difficulty formalizing in a coherent MF set.

Table 8.13: State/Action-Based Memory Parameters – Human AW

Person	Scenario	HF	LF	Set Count	Mem Cases	Default Cases	MF Count
AW	DG	0.025	0.01	24	269	19	5
AW	F1	0.025	0.01	49	249	55	5
AW	F2	0.025	0.01	29	174	18	5
AW	G1	0.025	0.01	54	401	71	3
AW	G2	0.025	0.01	45	345	64	7
AW	GH	0.025	0.01	42	329	42	9
AW	GP	0.025	0.01	49	347	44	8
AW	L	0.025	0.01	62	337	78	8
AW	R	0.025	0.01	16	256	15	4
AW	Sn1	0.025	0.01	52	455	67	8
AW	Sn2	0.025	0.01	46	297	56	5
AW	Sp1	0.025	0.01	42	379	53	8
AW	Sp2	0.025	0.01	48	336	49	6

Table 8.13 features very non-constrictive HF/LF parameters, relatively high MF counts, and relatively small default context sizes. These trends were also evident for Humans AJ and AK.

These were the MFs learned for the action-based memory-enhanced traces for Human AW.

- *DG*: v-Action(1), v-Action(2)

- *F1*: v-Action(1), v-Action(2)

- *F2*: v-Action(1), v-Action(2)

- *G1*: v-Action(1), v-Action(2)

- *G2*: v-Action(1), v-Action(2), v-Action(3)

- *GH*: v-Action(1), v-Action(2)

- *GP*: t-Action(n), t-Action(N), v-Action(1), v-Action(2)

- *L*: v-Action(1), v-Action(2)

- *R*: v-Action(1), v-Action(2), v-Action(3)

- *Sn1*: v-Action(1), v-Action(2)

- *Sn2*: v-Action(1), v-Action(2)

- *Sp1*: t-Action(S), t-Action(Se), v-Action(1), v-Action(2)

- *Sp2*: v-Action(1), v-Action(2)

With the exception of the the single time-back MF for an upward joystick input in the GP (Gold Pot) scenario, mostly value-back MFs parameterized with time-back values less than 3 were learned.

These were the MFs learned for the state-based memory-enhanced traces for Human AW.

- *DG*: t-DarkGrassW4Dist(1000.0)

- *F1*: t-FairyW1Dist(1000.0), t-GrassW0Dist(1000.0)

- *F2*: t-GrassW1Dist(1000.0)

- *G1*: t-GrassW2Dist(1000.0), t-GrassW3Dist(1000.0)

- *G2*: none

- *GH*: none

- *GP*: t-GoldPotW4Dist(1000.0)

- *L*: none

- *R*: none

- *Sn1*: t-GrassW0Dist(1000.0), t-SnakeW0Dist(1000.0)

- *Sn2*: t-GrassW0Dist(1000.0)

- *Sp1*: t-SprinklerW0Dist(1000.0)

- *Sp2*: t-GrassW1Dist(1000.0), t-GrassW5Dist(1000.0)

All learned MFs were based on wedge distance features and were parameterized with the maximum distance value 1000.0. Also, the most common wedges were Wedges 0-1 (northeast regions) and Wedges 3-4 (southern regions), which indicates that vertical perception has a significant memory influence on Human AW's behavior.

These were the MFs learned for the state/action-based memory-enhanced traces for Human AW.

- *DG*: t-Action(nE), t-Action(NE), v-Action(1), v-Action(2), v-Action(3)

- *F1*: t-Action(nE), v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *F2*: t-Action(E), t-Action(nE), t-Action(NE), v-Action(1), v-Action(2)

- *G1*: v-Action(1), v-Action(2), v-Action(3)

- *G2*: t-Action(n), t-Action(N), t-Action(SE), t-Action(sE), v-Action(1), v-Action(2), v-Action(3)

- *GH*: t-Action(Sw), t-Action(S), t-Action(Nw), t-Action(N), t-Action(NE), t-Action(Se), t-Action(SE), v-Action(1), v-Action(2)

- *GP*: t-Action(C), t-Action(n), t-Action(S), t-Action(N), v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *L*: t-Action(C), t-Action(n), t-Action(ne), t-Action(S), t-Action(N), t-Action(s), v-Action(1), v-Action(2)

- *R*: v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *Sn1*: t-Action(E), t-Action(nE), t-Action(N), t-Action(Ne), t-Action(NE), v-Action(1), v-Action(2), v-Action(3)

- *Sn2*: t-Action(N), v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *Sp1*: t-Action(S), t-Action(Se), t-Action(SE), v-Action(1), v-Action(2), v-Action(3), v-Action(4), v-Action(5)

- *Sp2*: t-Action(e), t-Action(nE), t-Action(sW), v-Action(1), v-Action(2), v-Action(3)

Similar to Humans AK and AJ, several time-back MFs based on the Action feature were learned for the state/action-based traces, but no perception MFs were learned. However, in contrast to AK and AJ, Human AW's behavior resulted in MCL learning MFs for several "slow" joystick inputs that were not recorded purely in the extreme positions for the four cardinal directions N, E, S, W. Examples include t-Action(s), t-Action(nE), t-Action(Se).

Figure 8.16 shows cross validation results for Bayes Net. In all scenarios (especially DG, F1, and G1), the action-based, state/action-based, and hybrid memory-enhanced traces resulted in increased F1 scores over the baseline. State-based traces saw little difference in F1 score from those of the baseline traces. These same trends are seen in Figure 8.17 for the train/test split evaluation.

Figure 8.18 shows cross validation results for the J.48 Decision Tree. Increases in F1-score are seen for some scenarios, though not as pronounced as those for Bayes Net. Decreases in F1 score for the memory-enhanced traces (except state-based) are observed for the F2 (Fairy 2) and R (Rock) scenarios. However, such decreases are not evident in Figure 8.19 for the train/test split evaluation.
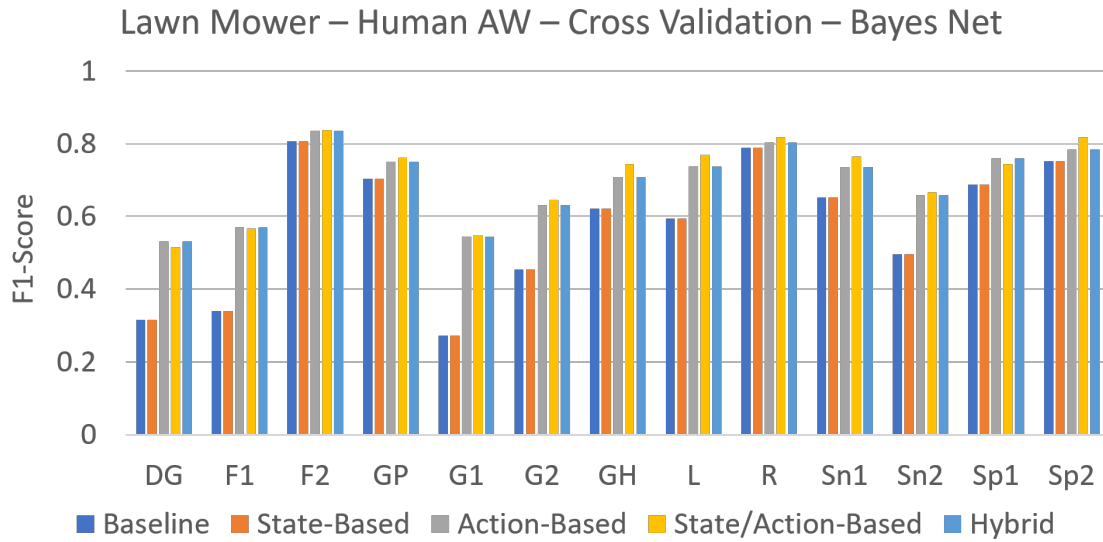
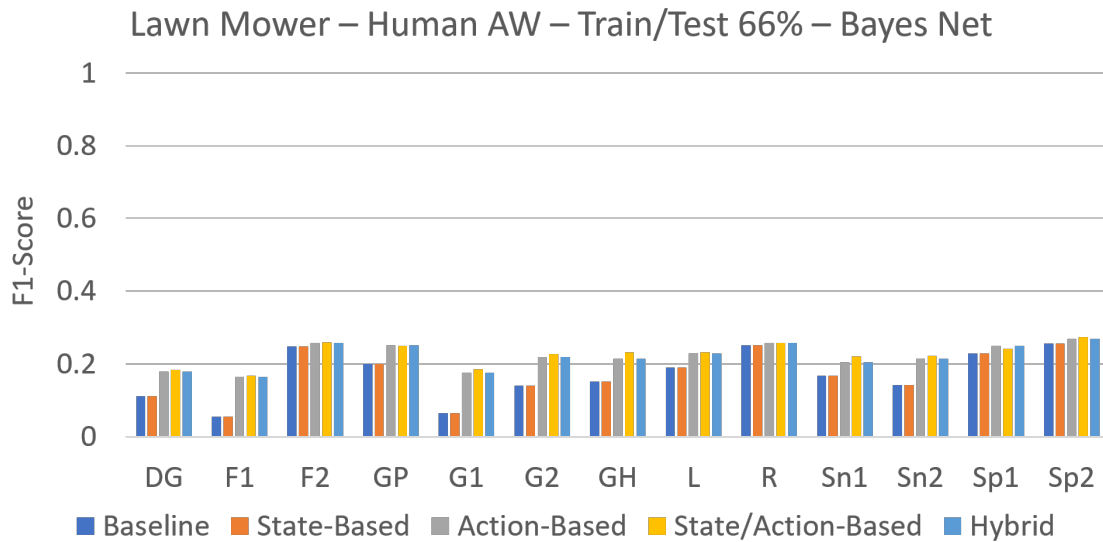Figure 8.16: LMS Results, Human AW, Bayes Net, Cross Validation

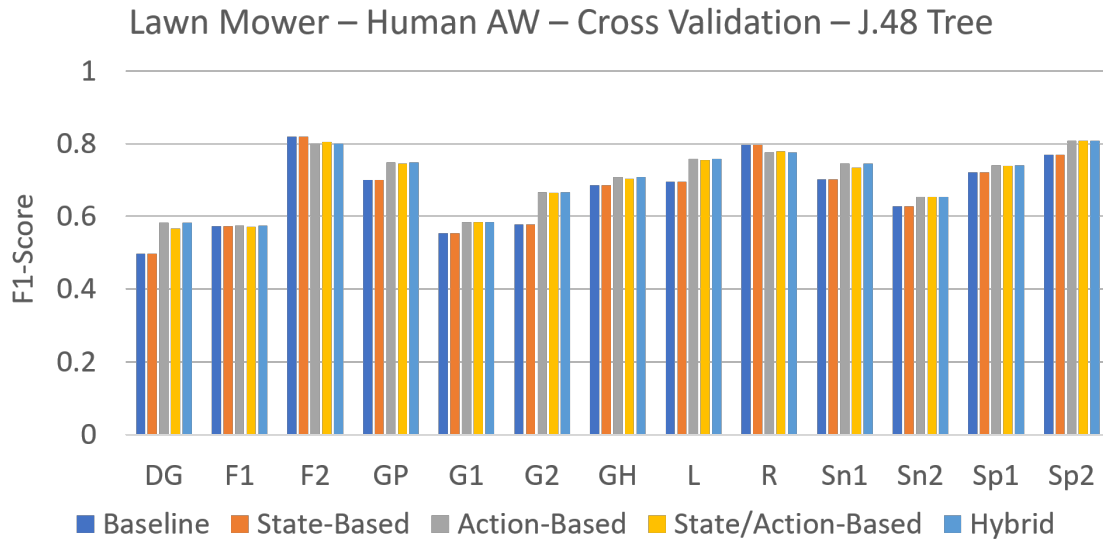Figure 8.17: LMS Results, Human AW, Bayes Net, Train/Test Split

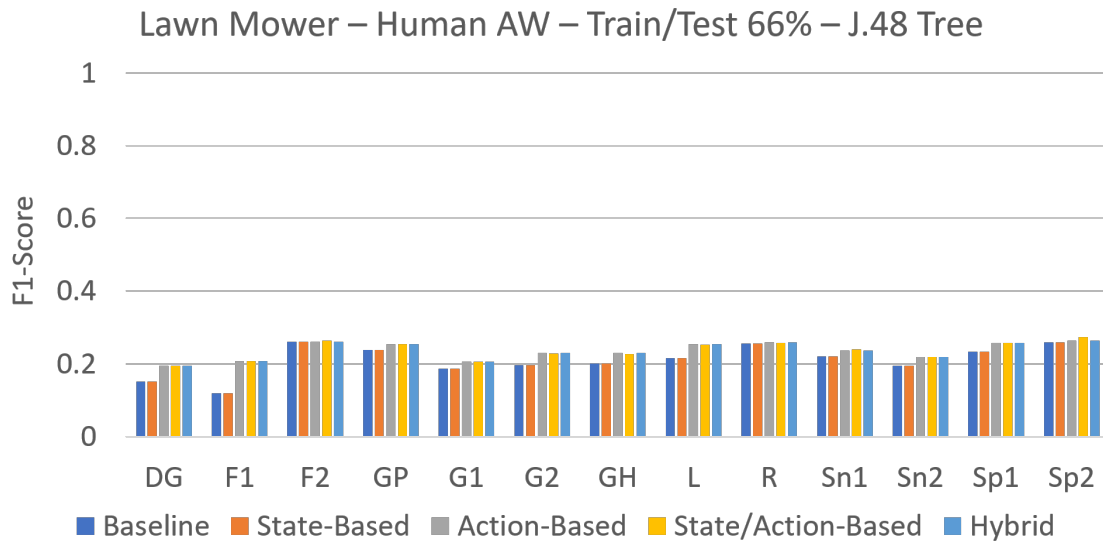Figure 8.18: LMS Results, Human AW, J.48 Tree, Cross Validation

Figure 8.19: LMS Results, Human AW, J.48 Tree, Train/Test Split

8.3.4.4   Human CA

Tables 8.14, 8.15, and 8.16 show Memory Parameter statistics for action-based, state-based, and state/action-based memory-enhanced traces, respectively.

Table 8.14: Action-Based Memory Parameters – Human CA

Person	Scenario	HF	LF	Set Count	Mem Cases	Default Cases	MF Count
CA	DG	0.05	0.05	55	281	144	3
CA	F1	0.025	0.025	32	248	62	2
CA	F2	0.025	0.025	20	176	31	2
CA	G1	0.025	0.025	73	456	142	3
CA	G2	0.05	0.05	62	352	193	2
CA	GH	0.05	0.05	58	272	178	1
CA	GP	0.05	0.05	34	333	117	2
CA	L	0.05	0.05	46	290	109	2
CA	R	0.025	0.025	32	198	59	2
CA	Sn1	0.05	0.05	80	386	213	3
CA	Sn2	0.05	0.05	61	368	125	3
CA	Sp1	0.05	0.05	72	546	267	3
CA	Sp2	0.05	0.05	37	288	91	3

From Table 8.14, we observe a few more scenarios that had the best results with less constrictive thresholds than HF=0.05 and LF=0.05, as compared to the previous human participants. We observe a relatively small number of MFs, but relatively large default contexts.

290

Table 8.15: State-Based Memory Parameters – Human CA

Person	Scenario	HF	LF	Set Count	Mem Cases	Default Cases	MF Count
CA	DG	0.05	0.05	23	345	80	4
CA	F1	0.05	0.05	25	235	75	1
CA	F2	0.05	0.05	9	182	25	2
CA	G1	0.05	0.05	102	282	316	2
CA	G2	0.025	0.025	16	485	60	0
CA	GH	0.05	0.05	10	399	51	1
CA	GP	0.05	0.05	23	367	83	0
CA	L	0.05	0.05	48	256	143	4
CA	R	0.05	0.05	10	221	36	0
CA	Sn1	0.05	0.05	62	332	267	1
CA	Sn2	0.05	0.05	53	270	223	0
CA	Sp1	0.025	0.025	121	333	480	0
CA	Sp2	0.05	0.05	12	339	40	3

From Table 8.15 for state-based traces, we observe optimal results for mostly HF=0.05 and LF=0.05 thresholds. Even though there are a few scenarios with no learned MFs, the L (Leprechaun) and DG (Dark Grass) scenarios have four. The Sp1 scenario statistics are particularly interesting because the number of cases in this scenario are many, and the default context is larger than all other memory-based contexts combined.

Table 8.16: State/Action-Based Memory Parameters – Human CA

Person	Scenario	HF	LF	Set Count	Mem Cases	Default Cases	MF Count
CA	DG	0.025	0.01	55	349	76	7
CA	F1	0.025	0.01	32	273	37	5
CA	F2	0.025	0.01	20	191	16	3
CA	G1	0.025	0.01	73	483	115	4
CA	G2	0.025	0.01	62	455	90	8
CA	GH	0.025	0.01	58	369	81	9
CA	GP	0.025	0.01	38	400	50	6
CA	L	0.025	0.01	46	347	52	6
CA	R	0.025	0.01	32	232	25	5
CA	Sn1	0.025	0.01	80	478	121	6
CA	Sn2	0.025	0.01	61	405	88	4
CA	Sp1	0.025	0.01	72	693	120	7
CA	Sp2	0.025	0.01	37	331	48	7

Table 8.16 shows statistics for state/action-based traces for Human CA. Similar to previous humans, we observe the least constrictive HF/LF thresholds producing the smallest default contexts (which are small in size) and a greater number of MFs across all scenarios.

The following MFs were learned for the action-based memory-enhanced traces for Human CA.

- *DG*: v-Action(1), v-Action(2), v-Action(3)

- *F1*: v-Action(1), v-Action(2)

- *F2*: v-Action(1), v-Action(2)

- *G1*: v-Action(1), v-Action(2), v-Action(3)

- *G2*: v-Action(1), v-Action(2)

- *GH*: v-Action(1)

- *GP*: v-Action(1), v-Action(2)

- *L*: v-Action(1), v-Action(2)

- *R*: v-Action(1), v-Action(2)

- *Sn1*: v-Action(1), v-Action(2), v-Action(3)

- *Sn2*: v-Action(1), v-Action(2), v-Action(3)

- *Sp1*: t-Action(SE), v-Action(1), v-Action(2)

- *Sp2*: t-Action(N), t-Action(Ne), v-Action(1)

Only value-back MFs with time-back parameter values less than 4 are learned (with the exception of the time-back MF in the Sp1 (Sprinkler 1) scenario).

The following MFs were learned for the state-based memory-enhanced traces for Human CA.

- *DG*: t-DarkGrassW4Dist(1000.0), t-DarkGrassW5Dist(1000.0), t-DarkGrassW7Dist(1000.0), t-GrassW7Dist(1000.0)

- *F1*: t-RockW4Dist(1000.0)

- *F2*: t-FairyW6Dist(1000.0), v-RockW7Dist(14)

- *G1*: t-DarkGrassW0Dist(1000.0), t-DarkGrassW6Dist(1000.0)

- *G2*: none

- *GH*: t-GrHogW0Dist(1000.0)

- *GP*: none

- *L*: t-GrassW4Dist(1000.0), t-RockW4Dist(1000.0), t-RockW5Dist(1000.0), v-GrassW7Dist(22)

- *R*: none

- *Sn1*: t-GrassW0Dist(1000.0)

- *Sn2*: none

- *Sp1*: none

- *Sp2*: t-GrassW4Dist(1000.0), t-SprinklerW4Dist(1000.0), v-GrassW6Dist(18)

The grass distance features were the basis for several MFs across multiple scenarios. As observed for previous humans, Human CA's behavior did not result in many perception MFs, perhaps due to human inconsistency in behavior.

The following MFs were learned for the state/action-based memory-enhanced traces for Human CA.

- *DG*: t-Action(w), t-Action(s), v-Action(1), v-Action(2), v-Action(3), v-Action(4), v-Action(5)

- *F1*: t-Action(S), t-Action(Nw), t-Action(SE), v-Action(1), v-Action(2)

- *F2*: t-Action(N), v-Action(1), v-Action(2)

- *G1*: v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *G2*: t-Action(Nw), t-Action(N), t-Action(Se), t-Action(SE), v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *GH*: t-Action(SW), t-Action(nw), t-Action(n), t-Action(Se), t-Action(SE), v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *GP*: t-Action(n), t-Action(S), t-Action(N), t-Action(Se), v-Action(1), v-Action(2)

- *L*: t-Action(C), t-Action(N), v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *R*: t-Action(S), t-Action(Se), v-Action(1), v-Action(2), v-Action(3)

- *Sn1*: t-Action(S), t-Action(Se), v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *Sn2*: v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *Sp1*: t-Action(SW), t-Action(S), t-Action(SE), t-Action(sE), v-Action(1), v-Action(2), v-Action(3)

- *Sp2*: t-Action(E), t-Action(Nw), t-Action(N), t-Action(Ne), t-Action(sE), v-Action(1), v-Action(2)

For Human CA, the value-back Action MFs are strongly present for time-back parameters close to 1. We also see no perception MFs, but several time-back Action MFs. Strangely, only the L (Leprechaun) scenario features a time-back MF for the motionless joystick input "C", which suggests that remaining still did not heavily influence Human CA's memory, as it was apparent from MCL's output.

Figure 8.20 shows the Bayes Net cross validation results. Unlike the previous human subjects, increases in F1 score are readily seen in all scenarios. There are several instances where the state/action-based trace resulted in a higher F1 score than the hybrid trace, the opposite trend was seen in the DG, F1, and G1 scenarios. The first trend holds in the train/test split evaluation seen in Figure 8.21, while the second trend is less pronounced.
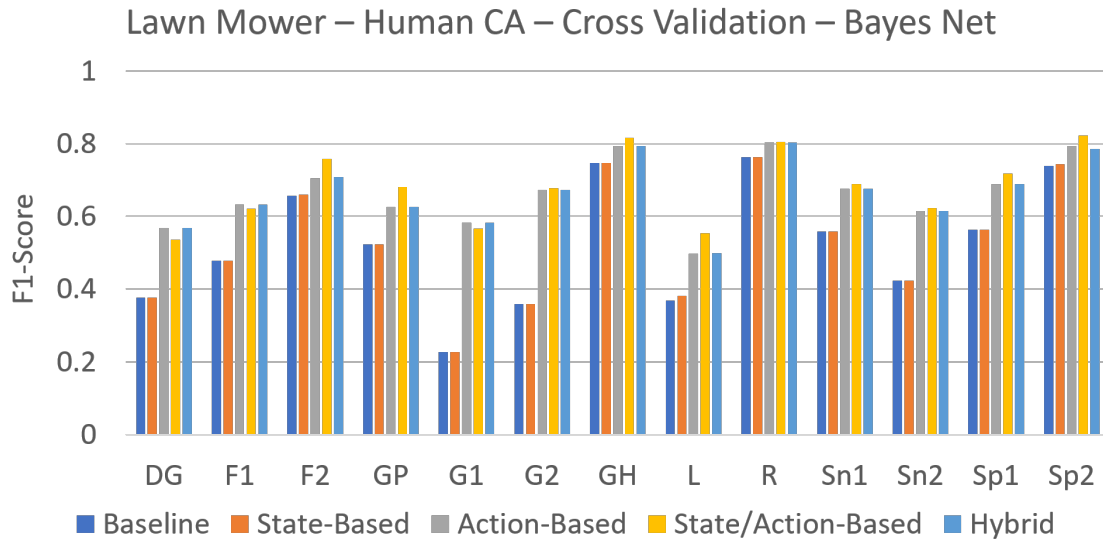
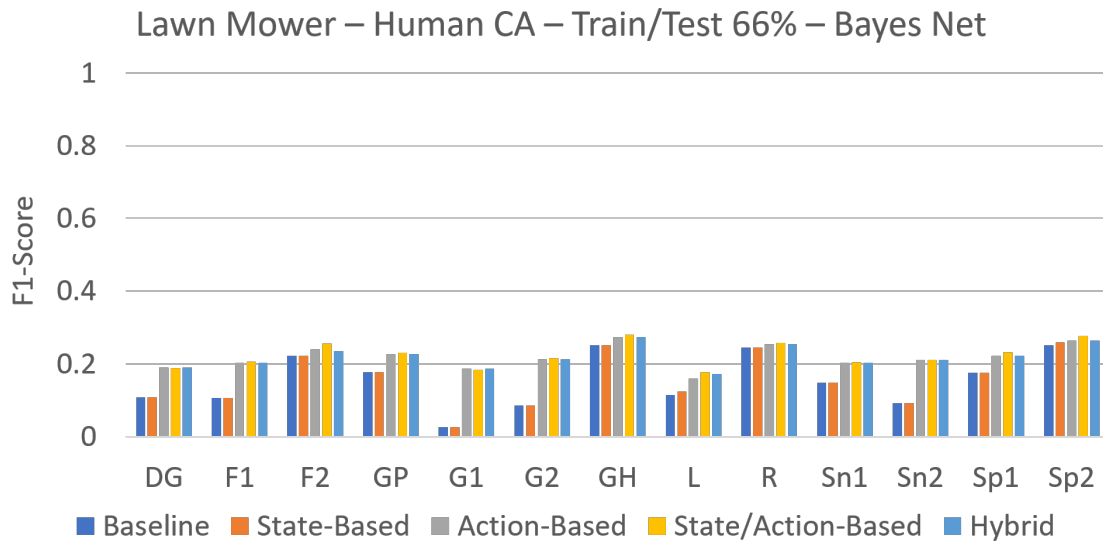Figure 8.20: LMS Results, Human CA, Bayes Net, Cross Validation

Figure 8.21: LMS Results, Human CA, Bayes Net, Train/Test Split
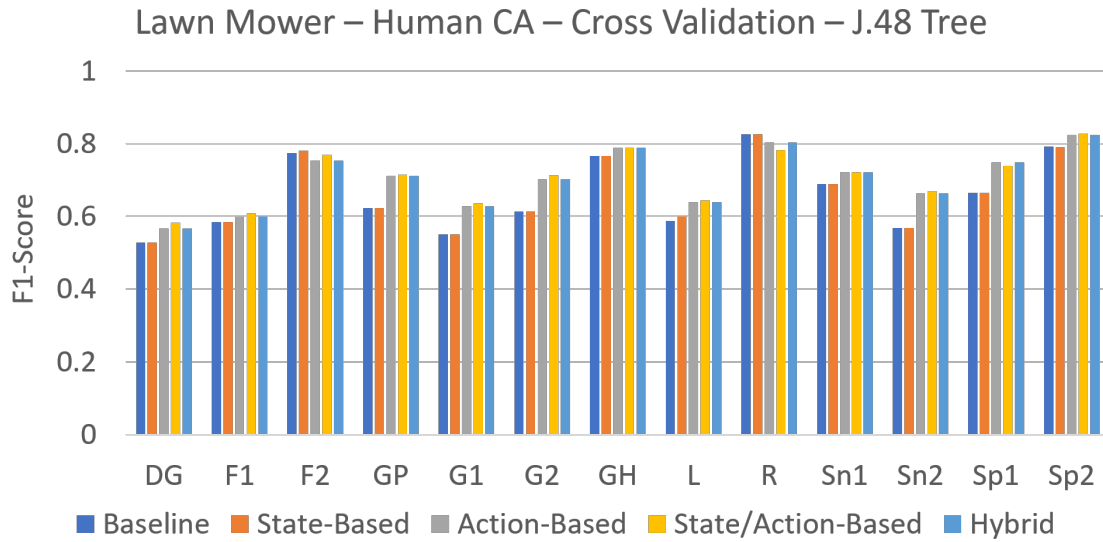
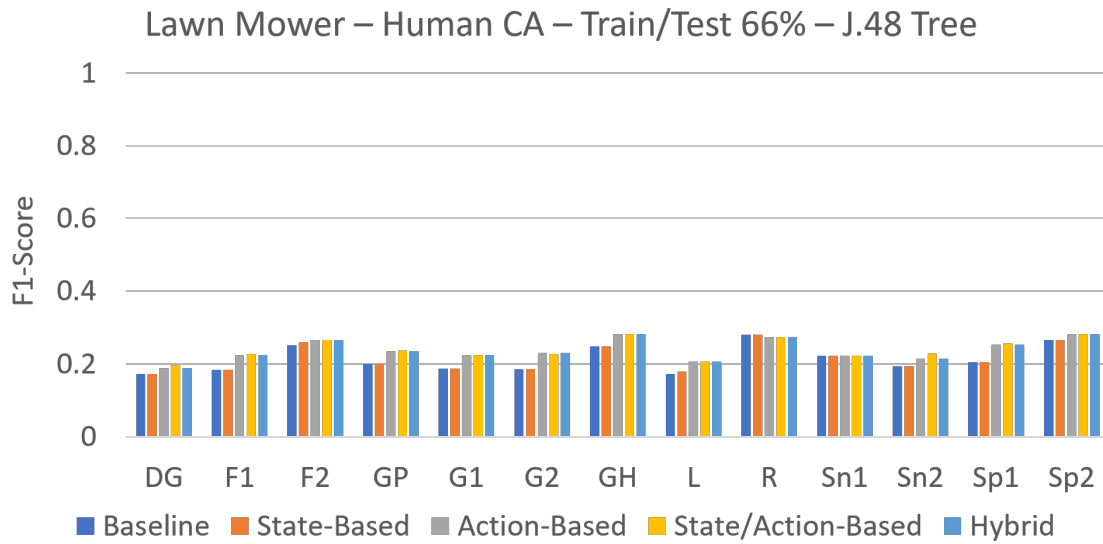Figure 8.22: LMS Results, Human CA, J.48 Tree, Cross Validation

Figure 8.23: LMS Results, Human CA, J.48 Tree, Train/Test Split

Figure 8.22 shows the cross validation results for J.48 Decision Treee. We observe that all memory-

enhanced traces except state-based have increased F1 scores over those of the baseline traces, except in the F2 and R scenarios, where the F1 scores are less than those for the baseline traces. However, this is only true of scenario R in the train/test split evaluation (see Figure 8.23).

8.3.4.5 Human CH

Tables 8.17, 8.18, and 8.19 show Memory Parameter statistics for action-based, state-based, and state/action-based memory-enhanced traces, respectively.

Table 8.17: Action-Based Memory Parameters – Human CH

Person	Scenario	HF	LF	Set Count	Mem Cases	Default Cases	MF Count
CH	DG	0.05	0.05	32	269	77	2
CH	F1	0.05	0.05	43	220	110	2
CH	F2	0.05	0.05	22	190	41	2
CH	G1	0.025	0.025	47	422	121	3
CH	G2	0.05	0.05	61	316	169	3
CH	GH	0.05	0.05	57	275	145	2
CH	GP	0.05	0.05	51	293	103	2
CH	L	0.05	0.05	43	229	118	2
CH	R	0.05	0.05	26	210	42	2
CH	Sn1	0.05	0.05	72	284	148	2
CH	Sn2	0.025	0.025	62	303	130	3
CH	Sp1	0.05	0.05	67	358	201	2
CH	Sp2	0.05	0.05	46	148	95	2

For all but two of the scenarios in Table 8.17, HF=0.05 and LF=0.05 produced the smallest default context size, which is still relatively large in most scenarios. Each scenario also contains at least two MFs that were learned by MCLvN.

298

Table 8.18: State-Based Memory Parameters – Human CH

Person	Scenario	HF	LF	Set Count	Mem Cases	Default Cases	MF Count
CH	DG	0.05	0.05	12	294	52	3
CH	F1	0.025	0.025	33	219	111	1
CH	F2	0.05	0.05	16	186	45	2
CH	G1	0.05	0.05	83	235	308	0
CH	G2	0.025	0.01	2	483	2	0
CH	GH	0.025	0.01	2	416	4	0
CH	GP	0.05	0.05	26	309	87	2
CH	L	0.05	0.05	42	210	137	2
CH	R	0.05	0.05	7	229	23	0
CH	Sn1	0.05	0.05	7	397	35	0
CH	Sn2	0.05	0.05	38	315	118	1
CH	Sp1	0.05	0.05	61	333	226	3
CH	Sp2	0.05	0.05	18	187	56	0

In Table 8.18, the default context size is generally smaller across all scenarios except in the first sprinkler scenario (Sp1) and the Grass Test (G1) leve, which has a default context bigger than all other memory-based contexts combined. Interestingly, the G2 (Grass Train) and GH (Ground Hog) scenarios do not feature any learned MFs and yet have very small default context sizes.

Table 8.19: State/Action-Based Memory Parameters – Human CH

Person	Scenario	HF	LF	Set Count	Mem Cases	Default Cases	MF Count
CH	DG	0.025	0.01	32	312	34	6
CH	F1	0.025	0.01	43	279	51	5
CH	F2	0.025	0.01	22	212	19	5
CH	G1	0.025	0.01	47	470	73	5
CH	G2	0.025	0.01	61	402	83	5
CH	GH	0.025	0.01	57	339	81	5
CH	GP	0.025	0.01	51	340	56	4
CH	L	0.025	0.01	43	301	46	9
CH	R	0.025	0.01	26	230	22	4
CH	Sn1	0.025	0.01	72	341	91	5
CH	Sn2	0.025	0.01	62	340	93	4
CH	Sp1	0.025	0.01	67	459	100	6
CH	Sp2	0.025	0.01	46	198	45	7

The HF/LF thresholds that produced the smallest default context sizes were as non-constraining as possible. Default context sizes were relatively smaller and MF counts were relatively higher than those for the state-based or action-based traces.

These are the MFs learned for each scenario for the action-based memory-enhanced traces for Human CH. All MFs are value-back with time-back parameters less than four.

- *DG*: v-Action(1), v-Action(2)

- *F1*: v-Action(1), v-Action(2)

- *F2*: v-Action(1), v-Action(2)

- *G1*: v-Action(1), v-Action(2), v-Action(3)

- *G2*: v-Action(1), v-Action(2), v-Action(3)

- *GH*: v-Action(1), v-Action(2)

- *GP*: v-Action(1), v-Action(2)

- *L*: v-Action(1), v-Action(2)

- *R*: v-Action(1), v-Action(2)

- *Sn1*: v-Action(1), v-Action(2)

- *Sn2*: v-Action(1), v-Action(2), v-Action(3)

- *Sp1*: v-Action(1), v-Action(2)

- *Sp2*: v-Action(1), v-Action(2)

These are the MFs learned for each scenario for the action-based memory-enhanced traces for Human CH. The value parameter for all MFs is maximum distance 1000.0, except for the Rock wedge MF for the F2 scenario, which tracks the time since the distance was near 0.0. This is a unique MF, indicating that memory of promixity to a rock above the human was significant. No particular wedge appeared to be prominent across all the scenarios.

- *DG*: t-DarkGrassW6Dist(1000.0), t-DarkGrassW7Dist(1000.0), v-GrassW6Dist(25)

- *F1*: t-GrassW0Dist(1000.0)

- *F2*: t-FairyW0Dist(1000.0), t-RockW7Dist(2.7)

- *G1*: none

- *G2*: none

- *GH*: none

- *GP*: t-GoldPotW1Dist(1000.0), t-GrassW4Dist(1000.0)

- *L*: t-GrassW4Dist(1000.0), t-SnakeW0Dist(1000.0)

- *R*: none

- *Sn1*: none

- *Sn2*: t-GrassW7Dist(1000.0)

- *Sp1*: t-GrassW5Dist(1000.0), t-SprinklerW1Dist(1000.0), t-SprinklerW5Dist(1000.0)

- *Sp2*: none

The following MFs were learned for state/action-based MFs. All MFs are based on Action. The value-back MFs track the value of Action as far back as five time steps back. The most prominent joystick input values in the MFs seem to be south or downward inputs.

- *DG*: t-Action(E), t-Action(sE), v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *F1*: t-Action(E), v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *F2*: t-Action(C), v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *G1*: v-Action(1), v-Action(2), v-Action(3), v-Action(4), v-Action(5)

- *G2*: v-Action(1), v-Action(2), v-Action(3), v-Action(4), v-Action(5)

- *GH*: t-Action(N), t-Action(SE), t-Action(sE), v-Action(1), v-Action(2)

- *GP*: t-Action(N), v-Action(1), v-Action(2), v-Action(3)

- *L*: t-Action(C), t-Action(E), t-Action(n), t-Action(S), t-Action(N), t-Action(Ne), t-Action(s), v-Action(1), v-Action(2)

- *R*: t-Action(N), v-Action(1), v-Action(2), v-Action(3)

- *Sn1*: t-Action(S), t-Action(Se), v-Action(1), v-Action(2), v-Action(3)

- *Sn2*: t-Action(SE), v-Action(1), v-Action(2), v-Action(3)

- *Sp1*: t-Action(SW), t-Action(Sw), t-Action(S), t-Action(sE), v-Action(1), v-Action(2)

- *Sp2*: t-Action(SW), t-Action(N), t-Action(sW), v-Action(1), v-Action(2), v-Action(3), v-Action(4)

Figure 8.24 shows cross validation Bayes Net F1 scores. State-based trace F1 scores were equivalent to baseline trace F1 scores for all scenarios. All other memory-enhanced trace F1 scores exceeded the baseline F1 score, with the state/action-based traces having the highest score for the majority of scenarios. The same trend is seen in the train/test split evaluation (see Figure 8.25), although the state/action-based trace F1 score is more tied with those of the other memory-enhanced traces.

Figure 8.26 shows J.48 Tree cross validation F1 scores. Except in the DG and R scenarios, all memory-enhanced trace F1 scores (except for state-based) exceed the baseline trace F1 score.

In Figure 8.27, the memory-enhanced traces (except state-based) have higher F1 scores in all scenarios. However, they have an especially high score in the Sn1 scenario, indicating a possible tendency to mostly use straight line movements. These traces all share two value-back MFs for Action, for time-back values close to 1. Thus, it seems that Human CH's behavior in this scenario was very consistent from time step to time step.
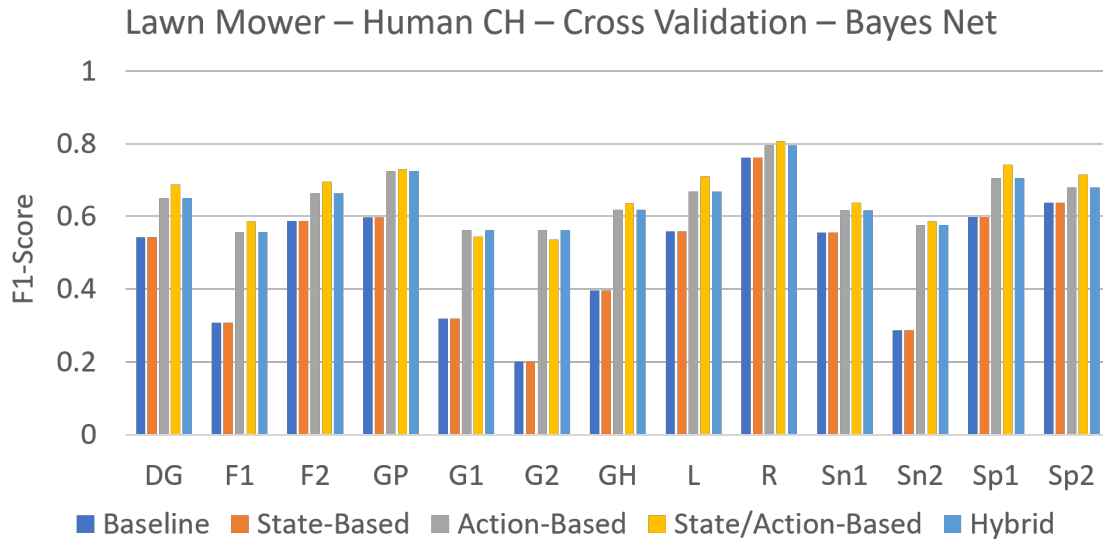
303

Figure 8.24: LMS Results, Human CH, Bayes Net, Cross Validation
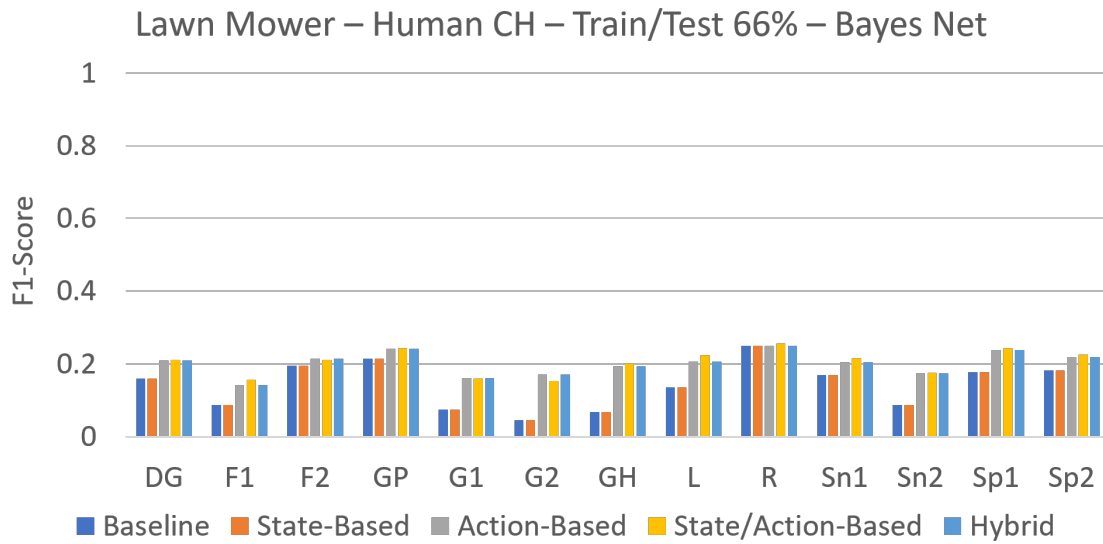

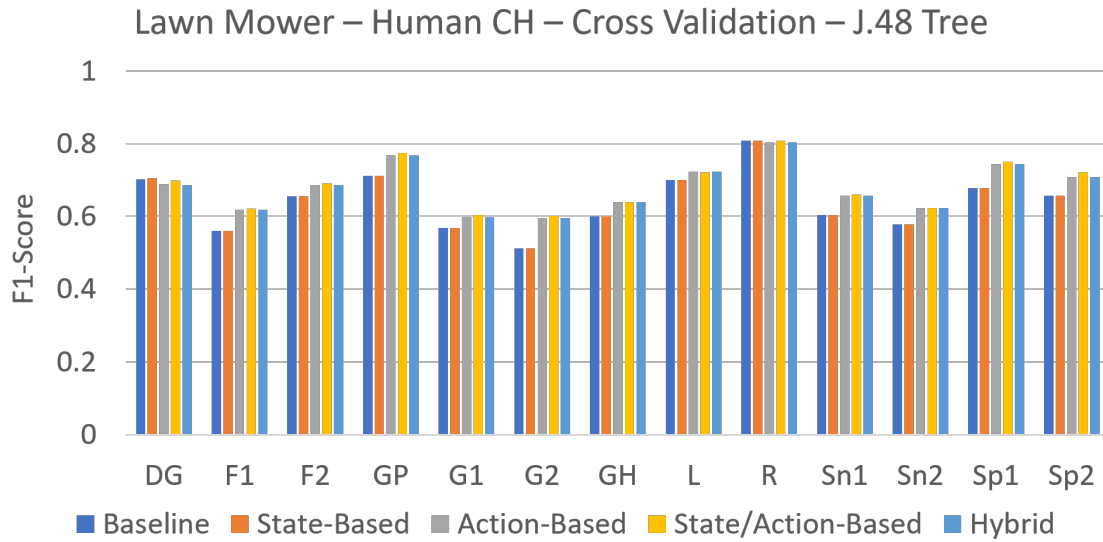Figure 8.25: LMS Results, Human CH, Bayes Net, Train/Test Split

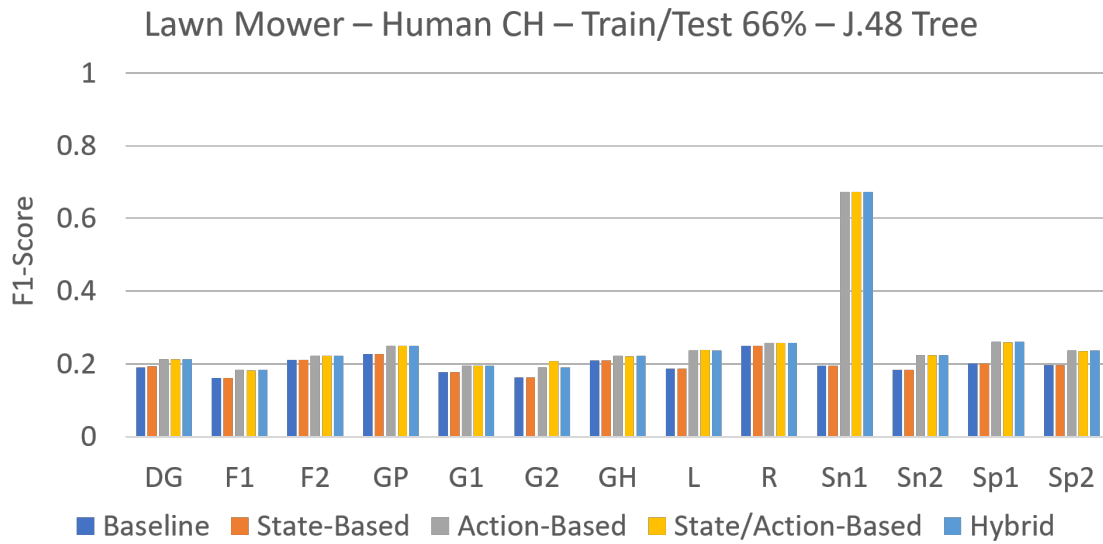Figure 8.26: LMS Results, Human CH, J.48 Tree, Cross Validation

Figure 8.27: LMS Results, Human CH, J.48 Tree, Train/Test Split

8.3.4.6 Human D

Tables 8.20, 8.20, and 8.20 show Memory Parameter statistics for action-based, state-based, and state/action-based memory-enhanced traces, respectively.

Table 8.20: Action-Based Memory Parameters – Human D

Person	Scenario	HF	LF	Set Count	Mem Cases	Default Cases	MF Count
D	DG	0.05	0.05	43	257	97	2
D	F1	0.05	0.05	36	206	58	3
D	F2	0.025	0.025	20	176	33	2
D	G1	0.05	0.05	52	438	144	2
D	G2	0.05	0.05	66	245	238	3
D	GH	0.05	0.05	49	226	94	2
D	GP	0.05	0.05	34	280	91	1
D	L	0.05	0.05	52	262	145	4
D	R	0.05	0.05	39	202	71	2
D	Sn1	0.05	0.05	36	241	55	3
D	Sn2	0.05	0.05	50	282	129	3
D	Sp1	0.05	0.05	52	398	149	2
D	Sp2	0.05	0.05	31	168	43	5

In Table 8.20, all scenarios except one had the smallest default context for HF=0.05 and LF=0.05. However, the G2 scenario featured a default context that encompassed almost all cases. This indicates that Human D's behavior was fairly random in this scenario. In contrast, the default context was the smallest in the Sp2 and F2 scenarios, which both feature phased memory-based entities (fairies and sprinklers).

Table 8.21: State-Based Memory Parameters – Human D

Person	Scenario	HF	LF	Set Count	Mem Cases	Default Cases	MF Count
D	DG	0.05	0.05	89	148	206	1
D	F1	0.05	0.05	56	109	155	0
D	F2	0.05	0.05	24	141	68	0
D	G1	0.025	0.025	160	204	378	0
D	G2	0.025	0.01	1	483	0	0
D	GH	0.05	0.05	33	239	81	0
D	GP	0.05	0.05	52	163	208	0
D	L	0.05	0.05	79	162	245	0
D	R	0.05	0.05	6	246	27	0
D	Sn1	0.05	0.05	3	282	14	0
D	Sn2	0.05	0.05	48	235	176	3
D	Sp1	0.05	0.05	102	236	311	0
D	Sp2	0.05	0.05	25	148	63	2

Table 8.21 again features HF=0.05 and LF=0.05 as the most common thresholds used. Very few MFs were learned for the state-based traces. In contrast to previous humans, the default context sizes are relatively large for the state-based traces for this human, which is further evidence of more random/erratic behavior.

Table 8.22: State/Action-Based Memory Parameters – Human D

Person	Scenario	HF	LF	Set Count	Mem Cases	Default Cases	MF Count
D	DG	0.025	0.01	43	297	57	4
D	F1	0.025	0.01	34	231	33	5
D	F2	0.025	0.01	20	186	23	3
D	G1	0.025	0.01	52	497	85	4
D	G2	0.025	0.01	66	396	87	8
D	GH	0.025	0.01	49	263	57	6
D	GP	0.025	0.01	38	324	47	4
D	L	0.025	0.01	52	331	76	5
D	R	0.025	0.01	39	236	37	7
D	Sn1	0.025	0.01	36	259	37	4
D	Sn2	0.025	0.01	50	340	71	7
D	Sp1	0.025	0.01	53	469	78	6
D	Sp2	0.025	0.01	31	180	31	7

Table 8.22 shows statistics for state/action-based traces. Here, the default contexts are relatively small and the HF/LF parameters are non-constrictive. A modest number of MFs are learned for each scenario.

The following MFs were learned for Human D's action-based traces. Most MFs were value-back Action MFs, but there were a few time-back MFs for extreme upward (north or N) and rightward (east or E) joystick inputs.

- *DG*: v-Action(1), v-Action(2)

- *F1*: v-Action(1), v-Action(2), v-Action(3)

- *F2*: v-Action(1), v-Action(2)

308

- *G1*: v-Action(1), v-Action(2)

- *G2*: t-Action(N), v-Action(1), v-Action(2)

- *GH*: v-Action(1), v-Action(2)

- *GP*: v-Action(1)

- *L*: t-Action(C), t-Action(E), v-Action(1), v-Action(2)

- *R*: v-Action(1), v-Action(2)

- *Sn1*: v-Action(1), v-Action(2), v-Action(3)

- *Sn2*: v-Action(1), v-Action(2), v-Action(3)

- *Sp1*: v-Action(1), v-Action(2)

- *Sp2*: t-Action(nW), t-Action(NW), v-Action(1), v-Action(2), v-Action(3)

Human D's learned MFs for the state-based traces are shown below. The sheer absence of MFs, compared with the previous humans, is interesting. However, the role of the Grass distance feature in Wedges 2-3 seemed to be significant for the second Snake (Sn2) and second Sprinkler (Sp2) scenarios. This may be due to the interaction of with grass in the southeastern direction in conjunction with interaction with the snakes/sprinklers while Human D was mowing.

- *DG*: t-DarkGrassW1Dist(1000.0)

- *F1*: none

- *F2*: none

- *G1*: none

- *G2*: none

- *GH*: none

- *GP*: none

- *L*: none

- *R*: none

- *Sn1*: none

- *Sn2*: t-GrassW3Dist(3.5), t-GrassW3Dist(3.7), t-SnakeW4Dist(1000.0)

- *Sp1*: none

- *Sp2*: t-GrassW2Dist(1000.0), t-GrassW3Dist(1000.0)

The following MFs were learned for the state/action-based memory-enhanced traces for Human D. Mostly value-back MFs for Action were learned, but additional time-back MFs for Action were also learned. The value parameter of "North" (upward joystick input) seemed to be prominent among the time-back MFs.

- *DG*: v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *F1*: t-Action(NE), v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *F2*: t-Action(NE), v-Action(1), v-Action(2)

- *G1*: v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *G2*: t-Action(S), t-Action(N), t-Action(Ne), v-Action(1), v-Action(2), v-Action(3), v-Action(4), v-Action(5)

- *GH*: t-Action(S), t-Action(Nw), t-Action(N), v-Action(1), v-Action(2), v-Action(3)

- *GP*: t-Action(C), t-Action(N), t-Action(s), v-Action(1)

- *L*: t-Action(C), t-Action(E), t-Action(n), v-Action(1), v-Action(2)

- *R*: t-Action(Nw), t-Action(N), t-Action(SE), t-Action(s), v-Action(1), v-Action(2), v-Action(3)

- *Sn1*: v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *Sn2*: t-Action(SW), t-Action(E), t-Action(Se), t-Action(sE), v-Action(1), v-Action(2), v-Action(3)

- *Sp1*: t-Action(w), t-Action(C), t-Action(N), v-Action(1), v-Action(2), v-Action(3)

- *Sp2*: t-Action(nW), t-Action(NW), t-Action(N), t-Action(NE), v-Action(1), v-Action(2), v-Action(3)
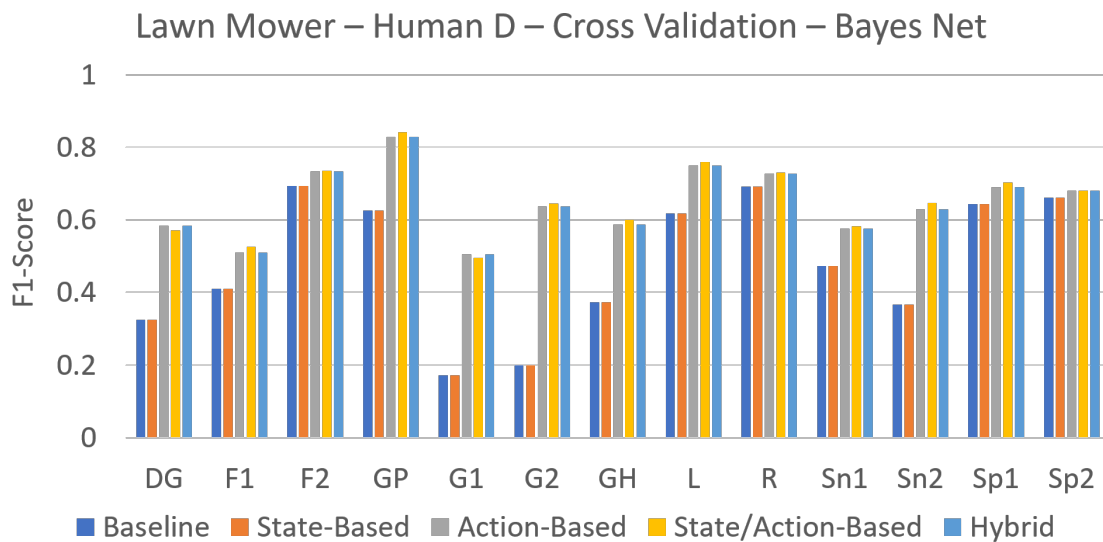
Figure 8.28: LMS Results, Human D, Bayes Net, Cross Validation

Figure 8.28 shows the Bayes Net cross validation results for Human D. The biggest increases in F1 score from the memory-enhanced traces are in the G1 and G2 scenarios. A large increase in score is also present in the DG scenario. All these scenarios involve grass or dark grass patches. Supposedly, one could simply remember where they have traveled rather than relying on memory of mowed grass patch locations to ensure that all grass patches get mowed. This strategy could more readily manifest itself in the form of memory of one's actions rather than memory of one's perceptions. However, the F1 scores for the baseline traces were low to begin with, which indicates possibly chaotic behavior in reaction to the relatively open environment in these scenarios.

Interestingly, a large increase in F1 score for the memory-enhanced traces is also observed for the Sn2 scenario, which is interesting because one must react to the inferred positions of snakes and thus memory of perception is important. The same F1 score increases that were observed in the cross validation evaluation are also evident in the train/test split evaluation seen in Figure 8.29.
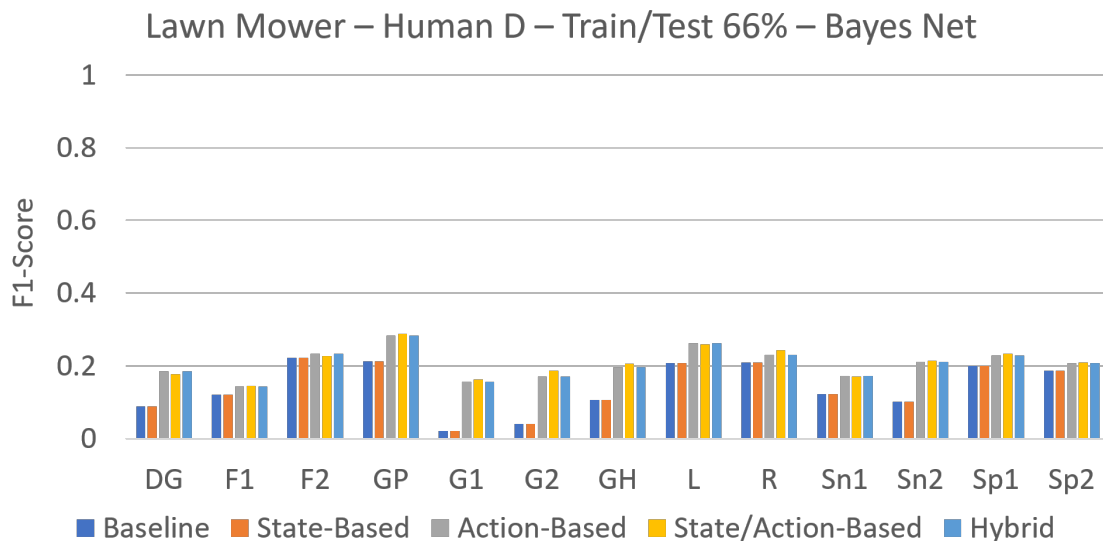
Figure 8.29: LMS Results, Human D, Bayes Net, Train/Test Split
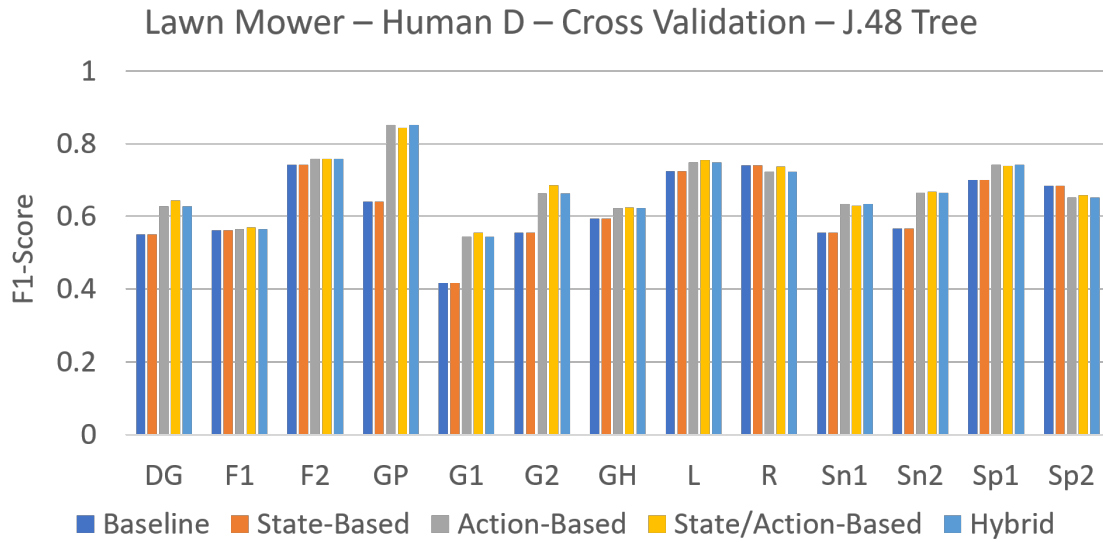
Figure 8.30: LMS Results, Human D, J.48 Tree, Cross Validation
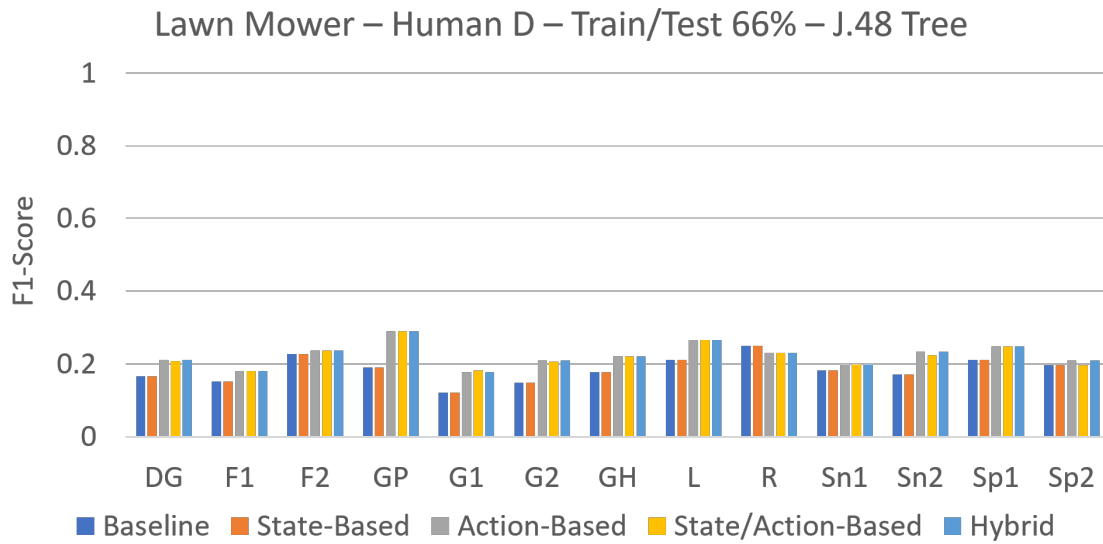
Figure 8.31: LMS Results, Human D, J.48 Tree, Train/Test Split

Figure 8.30 shows the J.48 Decision Tree cross validation results for Human D. In contrast to

the Bayes Net results, the largest F1 score increases are seen in the GP (Gold Pot) scenario. If the human moves fast enough, one can collide with the gold pot health restoratives before they disappear, thus reducing the memory requirement in that scenario. The increases in F1 score for the DG, G1, and G2 scenarios are also present, but to a lesser degree than they were for the Bayes Net results. Similar trends are also seen in the train/test split evaluation, as seen in Figure 8.31.

8.3.4.7   Human G

Tables 8.23, 8.24, and 8.25 show the Memory Parameter statistics for action-based, state-based, and state/action-based memory-enhanced traces, respectively.

Table 8.23: Action-Based Memory Parameters – Human G

Person	Scenario	HF	LF	Set Count	Mem Cases	Default Cases	MF Count
G	DG	0.05	0.05	44	184	89	4
G	F1	0.025	0.025	40	218	66	3
G	F2	0.05	0.05	43	156	101	4
G	G1	0.05	0.05	41	332	127	3
G	G2	0.05	0.05	68	363	185	3
G	GH	0.05	0.05	52	176	190	3
G	GP	0.05	0.05	36	272	99	5
G	L	0.025	0.025	43	209	90	4
G	R	0.05	0.05	40	192	83	2
G	Sn1	0.05	0.05	62	238	138	4
G	Sn2	0.05	0.05	55	206	94	2
G	Sp1	0.05	0.05	27	192	94	1
G	Sp2	0.05	0.05	54	218	115	3

Table 8.23 shows that the number of MFs learned for each scenario ranges from one to four. The

default context sizes are relatively large.

Table 8.24: State-Based Memory Parameters – Human G

Person	Scenario	HF	LF	Set Count	Mem Cases	Default Cases	MF Count
G	DG	0.05	0.05	6	251	22	0
G	F1	0.05	0.05	34	198	86	2
G	F2	0.025	0.025	6	248	9	2
G	G1	0.05	0.05	63	263	196	2
G	G2	0.025	0.01	1	548	0	0
G	GH	0.025	0.01	1	366	0	0
G	GP	0.05	0.05	24	264	107	1
G	L	0.05	0.05	26	216	83	2
G	R	0.025	0.025	10	263	12	7
G	Sn1	0.05	0.05	19	308	68	3
G	Sn2	0.05	0.05	22	234	66	5
G	Sp1	0.05	0.05	2	275	11	0
G	Sp2	0.05	0.05	19	220	113	2

For state-based traces (see Table 8.24), we see that the number of memory sets is less than that for action-based traces. In some cases, the default context size is very small. However, it is interesting to note that scenario R (Rock) has the most MFs, even though this scenario has no occluded entities. Perhaps the behavior of Human G relied on memory of previous positions relative to certain entities.

Table 8.25: State/Action-Based Memory Parameters – Human G

Person	Scenario	HF	LF	Set Count	Mem Cases	Default Cases	MF Count
G	DG	0.025	0.01	44	241	32	10
G	F1	0.025	0.01	40	247	37	5
G	F2	0.025	0.01	43	222	35	11
G	G1	0.025	0.01	41	387	72	7
G	G2	0.025	0.01	68	443	105	6
G	GH	0.025	0.01	52	324	42	8
G	GP	0.025	0.01	36	338	33	7
G	L	0.025	0.01	43	269	30	8
G	R	0.025	0.01	40	239	36	6
G	Sn1	0.025	0.01	62	310	66	7
G	Sn2	0.025	0.01	55	250	50	5
G	Sp1	0.025	0.01	27	268	18	9
G	Sp2	0.025	0.01	54	272	61	6

Table 8.25 shows statistics for state/action-based traces. The default context sizes are small, MF counts are relatively high, the number of memory sets used is comparable to that for action-based traces, and HF/LF thresholds are as low as possible.

The following MFs were learned for the action-based traces. Value-back MFs for Action for the previous 1-3 time steps were the most prominent MF. However, a time-back MFs based on upward or "northward" joystick inputs were also learned.

- *DG*: t-Action(Nw), t-Action(N), v-Action(1), v-Action(2)

- *F1*: v-Action(1), v-Action(2), v-Action(3)

- *F2*: t-Action(SE), t-Action(sE), v-Action(1), v-Action(2)

- *G1*: v-Action(1), v-Action(2), v-Action(3)

- *G2*: v-Action(1), v-Action(2), v-Action(3)

- *GH*: t-Action(n), v-Action(1), v-Action(2)

- *GP*: t-Action(C), t-Action(e), t-Action(E), v-Action(1), v-Action(2)

- *L*: t-Action(N), t-Action(Ne), v-Action(1), v-Action(2)

- *R*: v-Action(1), v-Action(2)

- *Sn1*: t-Action(C), v-Action(1), v-Action(2), v-Action(3)

- *Sn2*: v-Action(1), v-Action(2)

- *Sp1*: v-Action(1)

- *Sp2*: t-Action(N), v-Action(1), v-Action(2)

The following MFs were learned for state-based traces for Human G.

- *DG*: none

- *F1*: t-RockW3Dist(1000.0), t-RockW4Dist(1000.0)

- *F2*: t-GrassW7Dist(1.0), t-GrassW7Dist(1000.0)

- *G1*: t-DarkGrassW3Dist(1000.0), t-RockW5Dist(1000.0)

- *G2*: none

- *GH*: none

- *GP*: t-GoldPotW5Dist(1000.0)

- *L*: t-GrassW3Dist(1000.0), t-SnakeW3Dist(1000.0)

- *R*: t-RockW3Dist(1.8), t-RockW3Dist(2.0), t-RockW3Dist(2.1), t-RockW3Dist(2.300), v-GrassW0Dist(25), v-RockW3Dist(19), v-RockW3Dist(21)

- *Sn1*: t-GrassW0Dist(1000.0), t-SnakeW0Dist(1000.0), t-SnakeW0Dist(9.200)

- *Sn2*: t-ExitState(Off), t-GrassW4Dist(1000.0), t-SnakeW0Dist(1000.0), t-SnakeW0Dist(8.1), t-SnakeW7Dist(1000.0)

- *Sp1*: none

- *Sp2*: t-GrassW0Dist(1000.0), t-GrassW0Dist(3.800)

What is unique about Human G, as compared to the previous humans, is that several time-back MFs are learned for wedge distance features that are not parameterized with the maximum distance 1000.0 Examples include the time-back MFs for the Wedge 3 Rock distance feature, for values 2.0, 2.1, and 2.3. Other examples include the snake distance feature for the Sn2 scenario and the grass distance feature for scenario F2. Additionally, a few value-back MFs were also learned for certain perception features for times as far back as 25 time steps prior to the present.

The following MFs were learned for the state/action-based traces for Human G.

- *DG*: t-Action(W), t-Action(C), t-Action(E), t-Action(nW), t-Action(ne), t-Action(nE), t-Action(Nw), t-Action(N), v-Action(1), v-Action(2)

- *F1*: v-Action(1), v-Action(2), v-Action(3), v-Action(4), v-Action(5)

- *F2*: t-Action(C), t-Action(e), t-Action(E), t-Action(nE), t-Action(SE), t-Action(s), t-Action(sE), v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *G1*: t-Action(E), t-Action(sE), v-Action(1), v-Action(2), v-Action(3), v-Action(4), v-Action(5)

- *G2*: t-Action(S), t-Action(N), v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *GH*: t-Action(C), t-Action(e), t-Action(E), t-Action(n), t-Action(s), v-Action(1), v-Action(2), v-Action(3)

- *GP*: t-Action(C), t-Action(e), t-Action(E), t-Action(N), v-Action(1), v-Action(2), v-Action(3)

- *L*: t-Action(C), t-Action(e), t-Action(E), t-Action(N), t-Action(Ne), t-Action(Se), v-Action(1), v-Action(2)

- *R*: t-Action(E), t-Action(Nw), t-Action(N), v-Action(1), v-Action(2), v-Action(3)

- *Sn1*: t-Action(C), t-Action(E), t-Action(n), t-Action(Se), v-Action(1), v-Action(2), v-Action(3)

- *Sn2*: t-Action(W), t-Action(E), t-Action(NE), v-Action(1), v-Action(2)

- *Sp1*: t-Action(W), t-Action(E), t-Action(nW), t-Action(nE), t-Action(SE), t-Action(s), t-Action(sE), v-Action(1), v-Action(2)

- *Sp2*: t-Action(n), t-Action(N), v-Action(1), v-Action(2), v-Action(3), v-Action(4)

Similar to the previous humans, no perception MFs were learned, but several time-back and value-back MFs were learned. The value-back MFs tracking Action seem to be limited to five time steps prior to the present. No particular direction emerges and an incredibly prominent joystick value for the time-back Action MFs.
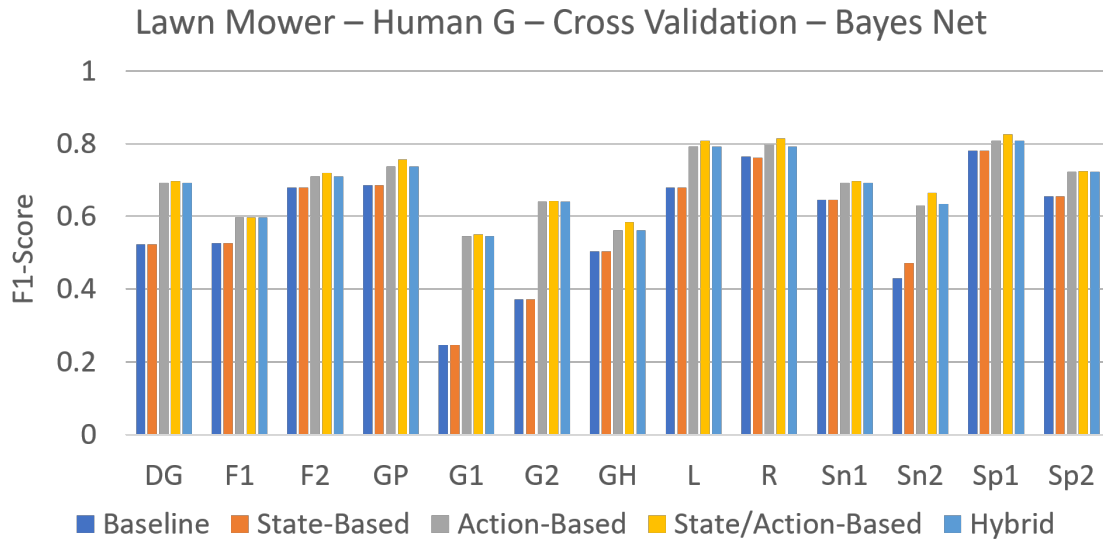
Figure 8.32: LMS Results, Human G, Bayes Net, Cross Validation
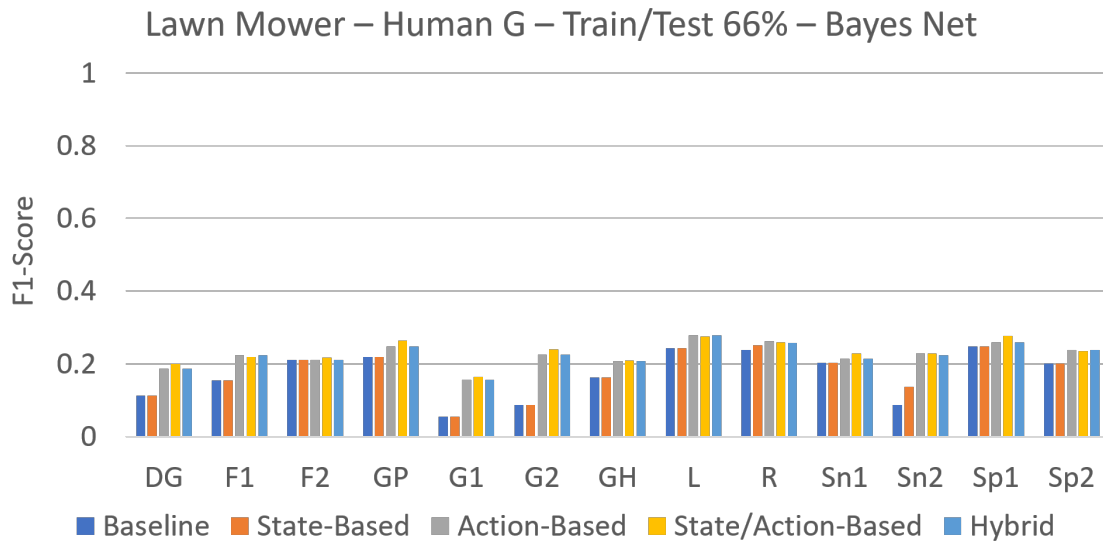
Figure 8.33: LMS Results, Human G, Bayes Net, Train/Test Split

Figure 8.32 shows Bayes Net cross validation F1 scores. We can see that there is one instance

where the state-based trace does outperform the baseline trace in terms of F1 score (see scenario Sn2), but it otherwise has equivalent F1 scores to those of baseline traces. The other memory-enhanced traces have higher F1 scores than the baseline trace, with the state/action-based trace having the highest score in scenarios F2, GP, GH, R, Sn2, and Sp1 (and having an F1 score that is approximately equal to that of the hybrid trace for the other scenarios). Similar trends are observed in the train/test split evaluation, as seen in Figure 8.33. The relatively low F1 scores for the the G1 and G2 scenarios for baseline traces may indicate chaotic behavior in reaction to the open environment in these scenarios.

Figure 8.34 shows the J.48 Tree cross validation results for Human G. For the DG, GP, and G2 scenarios, the memory-enhanced traces (except state-based) have a notably higher F1 score than that for the baseline trace. However, three interesting trends are observed for Human G. In the GH (Ground Hog) scenario, the action-based and hybrid traces have lower F1 scores than the baseline. The state-based and state/action-based traces only manage to equal the baseline trace in F1 score. In the Sn2 scenario, the state-based and state/action-based traces again have approximately the same F1 score, which exceeds those for the action-based, hybrid, and baseline traces. And finally, the Sp1 scenario shows the state-based memory-enhanced trace having the highest F1 score out of all the memory-enhanced traces, which also is roughly equivalent to the baseline trace's F1 score. Thus, we have three examples where the state-based memory-enhanced trace can outperform the other types of memory-enhanced traces. However, this trend is not as readily apparent in the train/test split evaluation, as seen in Figure 8.35.
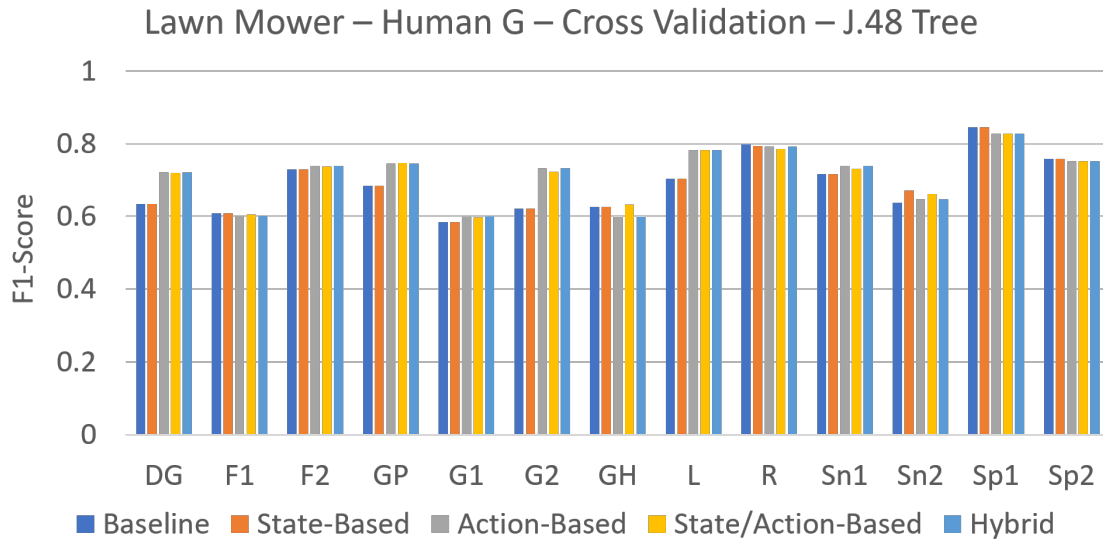
Figure 8.34: LMS Results, Human G, J.48 Tree, Cross Validation
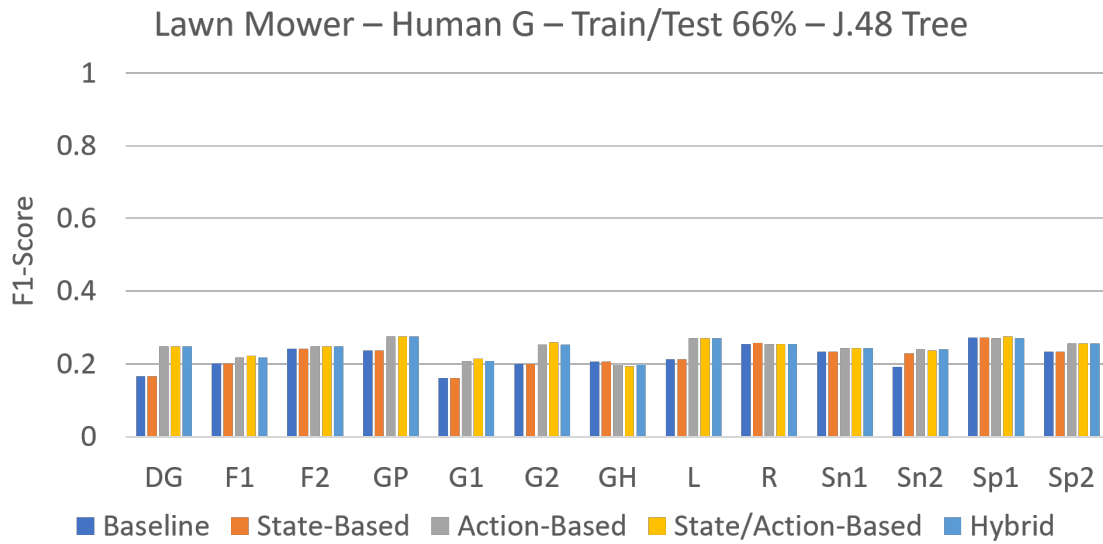
Figure 8.35: LMS Results, Human G, J.48 Tree, Train/Test Split

8.3.4.8   Human L

Tables 8.26, 8.27, and 8.28 show Memory Parameter statistics for action-based, state-based, and state/action-based memory-enhanced traces for Human L, respectively.

Table 8.26: Action-Based Memory Parameters – Human L

Person	Scenario	HF	LF	Set Count	Mem Cases	Default Cases	MF Count
L	DG	0.025	0.025	59	308	137	3
L	F1	0.05	0.05	43	219	94	4
L	F2	0.05	0.05	27	156	63	1
L	G1	0.05	0.05	46	418	134	3
L	G2	0.05	0.05	79	287	305	5
L	GH	0.05	0.05	27	207	63	2
L	GP	0.05	0.05	40	304	117	2
L	L	0.05	0.05	82	226	183	2
L	R	0.05	0.05	19	210	40	1
L	Sn1	0.05	0.05	47	221	94	2
L	Sn2	0.05	0.05	39	258	105	3
L	Sp1	0.05	0.05	73	345	213	3
L	Sp2	0.05	0.05	40	131	94	3

Table 8.26 shows that HF=0.05 and LF=0.05 are the most common filtering thresholds for the action-based traces. The number of MFs learned for each scenario range from one to five. Default context sizes are relatively large in size.

Table 8.27: State-Based Memory Parameters – Human L

Person	Scenario	HF	LF	Set Count	Mem Cases	Default Cases	MF Count
L	DG	0.05	0.05	96	173	272	0
L	F1	0.05	0.05	63	153	160	1
L	F2	0.05	0.05	17	160	59	0
L	G1	0.05	0.05	77	295	257	0
L	G2	0.05	0.05	25	483	109	8
L	GH	0.025	0.01	1	270	0	0
L	GP	0.05	0.05	47	224	197	4
L	L	0.05	0.05	50	214	195	1
L	R	0.025	0.025	4	245	5	1
L	Sn1	0.05	0.05	28	237	78	1
L	Sn2	0.05	0.05	56	207	156	0
L	Sp1	0.05	0.05	36	350	208	1
L	Sp2	0.05	0.05	13	180	45	0

Table 8.27 shows that HF=0.05 and LF=0.05 are the most common filtering thresholds for state-based traces. Interestingly, the lowest HF/LF values are employed for the GH (Ground Hog) scenario, even though the default context has size 0 and no MFs are learned for that scenario.

Table 8.28: State/Action-Based Memory Parameters – Human L

Person	Scenario	HF	LF	Set Count	Mem Cases	Default Cases	MF Count
L	DG	0.025	0.01	59	359	86	6
L	F1	0.025	0.01	46	257	56	5
L	F2	0.025	0.01	30	195	24	5
L	G1	0.025	0.01	46	493	59	5
L	G2	0.025	0.01	79	484	108	10
L	GH	0.025	0.01	27	246	24	7
L	GP	0.025	0.01	40	371	50	5
L	L	0.025	0.01	82	318	91	7
L	R	0.025	0.01	19	236	14	5
L	Sn1	0.025	0.01	47	261	54	5
L	Sn2	0.025	0.01	39	327	36	7
L	Sp1	0.025	0.01	74	436	122	6
L	Sp2	0.025	0.01	40	193	32	8

Table 8.28 possesses the same trends that were observed for the previous humans for state/action-based traces. The HF and LF levels that resulted in the smallest default context size for the scenario were employed. The most number of MFs are learned for these traces.

The following MFs were learned for the action-based memory-enhanced traces for Human L. The majority of them are value-back Action MFs with time-back parameters less than four.

- *DG*: v-Action(1), v-Action(2), v-Action(3)

- *F1*: t-Action(w), t-Action(C), v-Action(1), v-Action(2)

- *F2*: v-Action(1)

- *G1*: v-Action(1), v-Action(2), v-Action(3)

- *G2*: t-Action(S), t-Action(N), v-Action(1), v-Action(2), v-Action(3)

- *GH*: v-Action(1), v-Action(2)

- *GP*: v-Action(1), v-Action(2)

- *L*: v-Action(1), v-Action(2)

- *R*: v-Action(1)

- *Sn1*: v-Action(1), v-Action(2)

- *Sn2*: v-Action(1), v-Action(2), v-Action(3)

- *Sp1*: t-Action(S), t-Action(s), v-Action(1)

- *Sp2*: t-Action(N), t-Action(Ne), v-Action(1)

The following MFs were learned for the state-based traces for Human L. The G2 scenario is notable for its many MFs, all pertaining to the grass distance feature for Wedge 2 (north east) or Wedge 5 (south west). The time-back MFs track the time since close proximity with a grass patch. The value-back MFs track the distance to grass patches approximately 2 seconds prior to the present.

- *DG*: none

- *F1*: t-FairyW3Dist(1000.0)

- *F2*: none

- *G1*: none

- *G2*: t-GrassW2Dist(5.300), t-GrassW5Dist(4.4), t-GrassW5Dist(4.5), v-GrassW2Dist(15), v-GrassW2Dist(16), v-GrassW2Dist(17), v-GrassW2Dist(24), v-GrassW2Dist(27)

- *GH*: none

- *GP*: t-GoldPotW0Dist(1000.0), t-GrassW4Dist(1000.0), t-GrassW6Dist(1000.0), v-RockW6Dist(24)

- *L*: t-RockW4Dist(1.900)

- *R*: t-GrassW0Dist(1000.0)

- *Sn1*: t-SnakeW0Dist(1000.0)

- *Sn2*: none

- *Sp1*: t-GrassW0Dist(1000.0)

- *Sp2*: none

The following MFs were learned for Human L's state/action-based memory-enhanced traces. A combination of value-back and time-back MFs based on the Action were learned, but as with the previous humans, no perception MFs were learned. It is also notable that the time-back MFs are generally parameterized with joystick values with full-tilt (e.g. N instead of n, S instead of s).

- *DG*: t-Action(S), t-Action(N), v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *F1*: t-Action(w), t-Action(C), t-Action(E), v-Action(1), v-Action(2)

- *F2*: t-Action(C), t-Action(E), v-RockW7Dist(28), v-Action(1), v-Action(2)

- *G1*: v-Action(1), v-Action(2), v-Action(3), v-Action(4), v-Action(5)

- *G2*: t-Action(Sw), t-Action(S), t-Action(N), t-Action(Ne), t-Action(Se), v-Action(1), v-Action(2), v-Action(3), v-Action(4), v-Action(5)

- *GH*: t-Action(E), t-Action(n), t-Action(N), t-Action(sE), v-Action(1), v-Action(2), v-Action(3)

- *GP*: v-Action(1), v-Action(2), v-Action(3), v-Action(4), v-Action(5)

- *L*: t-Action(n), t-Action(S), t-Action(N), v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *R*: t-Action(E), t-Action(S), t-Action(Se), t-Action(sE), v-Action(1)

- *Sn1*: t-Action(E), t-Action(sE), v-Action(1), v-Action(2), v-Action(3)

- *Sn2*: t-Action(S), t-Action(N), v-Action(1), v-Action(2), v-Action(3), v-Action(4), v-Action(5)

- *Sp1*: t-Action(C), t-Action(e), t-Action(S), t-Action(s), v-Action(1), v-Action(2)

- *Sp2*: t-Action(C), t-Action(E), t-Action(N), t-Action(Ne), t-Action(sw), t-Action(s), v-Action(1), v-Action(2)
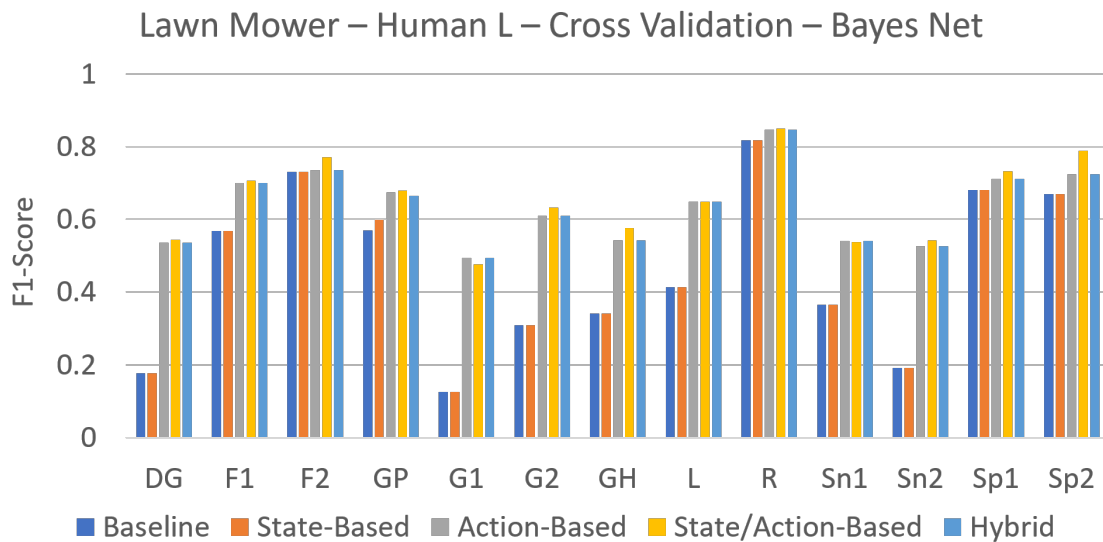
Figure 8.36: LMS Results, Human L, Bayes Net, Cross Validation

Figure 8.36 shows the Bayes Net cross validation F1 scores. The action-based, state/action-based,

and hybrid traces see F1 scores that are higher than that of the baseline traces for every scenario.

A similar trend is generally visible in the train/test split evaluation, as seen in Figure 8.37.
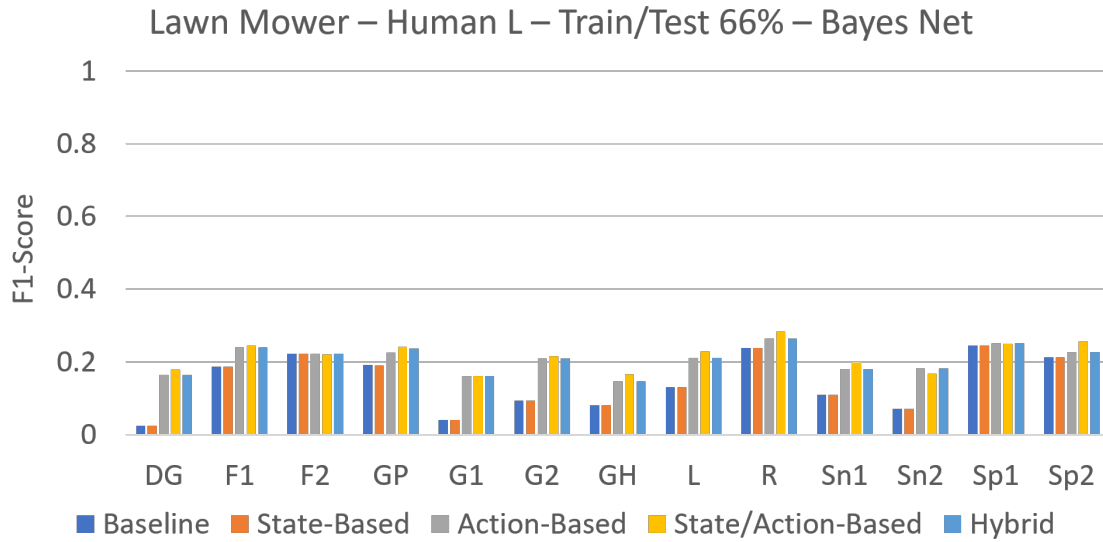
Figure 8.37: LMS Results, Human L, Bayes Net, Train/Test Split
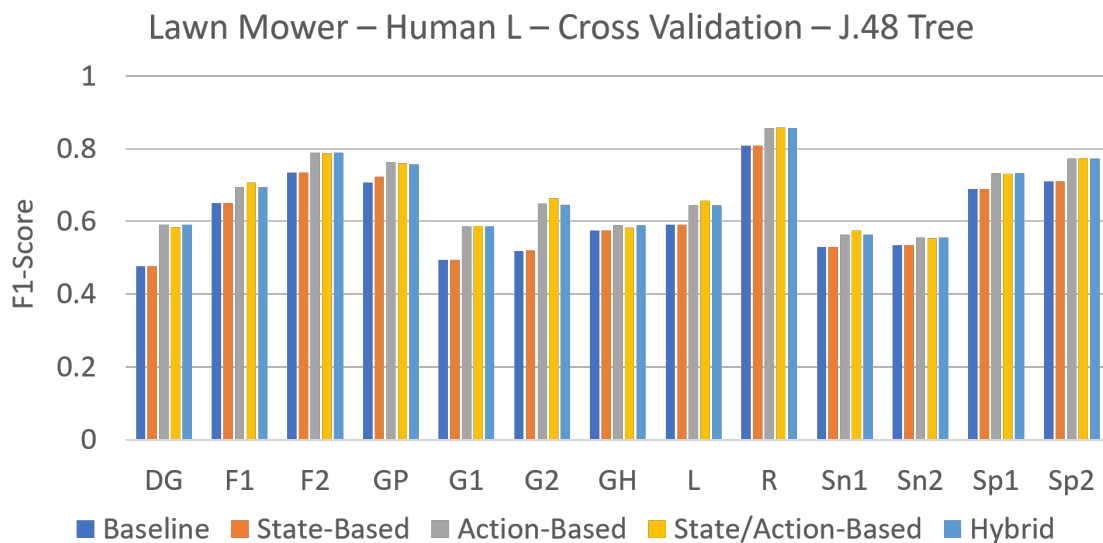
Figure 8.38: LMS Results, Human L, J.48 Tree, Cross Validation

The same trends we observed for the Bayes Net results are present for J.48 Tree results in both the cross validation evaluation (see Figure 8.38) and the train/test split evaluation (see Figure 8.39) with one exception. In scenario L (Leprechaun), the baseline and state-based trace F1 scores in the train/test split evaluation are very high compared to F1 scores for those traces in all other scenarios for that evaluation. What's interesting is that the memory-enhanced traces (action-based, state/action-based, and hybrid) did not achieve such high scores, despite having the same feature values. Thus, the MFs in those traces were responsible for diminishing the F1 score for those traces. It is possible that Human L's behavior during the test portion of the evaluation was largely different from that for the training portion.
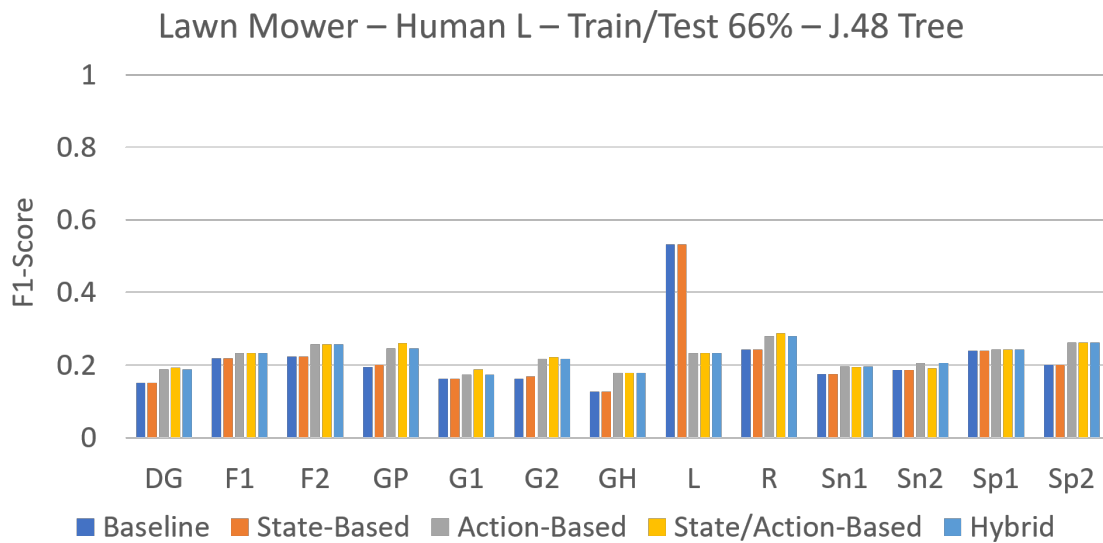
Figure 8.39: LMS Results, Human L, J.48 Tree, Train/Test Split

8.3.4.9   Human P

Tables 8.29, 8.30, and 8.31 show Memory Parameter statistics for Human P's action-based, state-based, and state/action-based memory-enhanced traces, respectively.

Table 8.29: Action-Based Memory Parameters – Human P

Person	Scenario	HF	LF	Set Count	Mem Cases	Default Cases	MF Count
P	DG	0.05	0.05	36	242	81	4
P	F1	0.025	0.025	42	310	80	3
P	F2	0.05	0.05	26	216	65	2
P	G1	0.025	0.025	63	421	135	3
P	G2	0.05	0.05	61	365	179	3
P	GH	0.05	0.05	45	234	90	1
P	GP	0.05	0.05	36	273	95	2
P	L	0.05	0.05	32	274	73	2
P	R	0.05	0.05	22	212	44	2
P	Sn1	0.05	0.05	48	283	96	3
P	Sn2	0.05	0.05	50	328	140	2
P	Sp1	0.05	0.05	27	318	76	4
P	Sp2	0.05	0.05	38	297	76	3

In Table 8.29, we observe typical HF/LF parameter values for action-based traces. The MF counts for each scenario are also relatively low. However, the default context sizes are also relatively small when compared to the sizes observed for the previous humans.

Table 8.30: State-Based Memory Parameters – Human P

Person	Scenario	HF	LF	Set Count	Mem Cases	Default Cases	MF Count
P	DG	0.05	0.05	83	153	170	0
P	F1	0.05	0.05	52	208	182	1
P	F2	0.05	0.05	51	134	147	0
P	G1	0.05	0.05	61	317	239	1
P	G2	0.025	0.01	13	523	21	11
P	GH	0.05	0.05	32	214	110	1
P	GP	0.05	0.05	54	231	137	0
P	L	0.05	0.05	33	251	96	2
P	R	0.05	0.05	10	230	26	0
P	Sn1	0.05	0.05	42	228	151	0
P	Sn2	0.05	0.05	26	354	114	1
P	Sp1	0.05	0.05	80	216	178	2
P	Sp2	0.05	0.05	17	307	66	4

Table 8.30 also shows typical HF/LF parameter values for state-based traces. The MF counts are also low, as was also observed for the previous humans. However the variance in the default context size is high among the scenarios, ranging from 21 cases (scenario G2) to 239 cases (scenario G1).

Table 8.31: State/Action-Based Memory Parameters – Human P

Person	Scenario	HF	LF	Set Count	Mem Cases	Default Cases	MF Count
P	DG	0.025	0.01	36	286	37	6
P	F1	0.025	0.01	42	343	47	4
P	F2	0.025	0.01	27	258	23	6
P	G1	0.025	0.01	63	459	97	4
P	G2	0.025	0.01	61	458	86	5
P	GH	0.025	0.01	45	273	51	3
P	GP	0.025	0.01	36	328	40	6
P	L	0.025	0.01	32	312	35	5
P	R	0.025	0.01	22	235	21	3
P	Sn1	0.025	0.01	48	322	57	5
P	Sn2	0.025	0.01	50	402	66	6
P	Sp1	0.025	0.01	29	359	35	7
P	Sp2	0.025	0.01	38	330	43	4

Table 8.31 shows the typical trends for state/action-based traces. The HF/LF parameter values are low. Default context sizes are small. MF counts are higher than those for the state-based or action-based traces.

The MFs learned for Human P's action-based traces are listed below.

- *DG*: t-Action(SE), t-Action(sE), v-Action(1), v-Action(2)

- *F1*: v-Action(1), v-Action(2), v-Action(3)

- *F2*: v-Action(1), v-Action(2)

- *G1*: v-Action(1), v-Action(2), v-Action(3)

- *G2*: v-Action(1), v-Action(2), v-Action(3)

- *GH*: v-Action(1)

- *GP*: v-Action(1), v-Action(2)

- *L*: v-Action(1), v-Action(2)

- *R*: v-Action(1), v-Action(2)

- *Sn1*: v-Action(1), v-Action(2), v-Action(3)

- *Sn2*: v-Action(1), v-Action(2)

- *Sp1*: t-Action(SE), t-Action(sE), v-Action(1), v-Action(2)

- *Sp2*: v-Action(1), v-Action(2), v-Action(3)

In addition to the typical value-back Action MFs that have been observed in the action-based traces of the previous humans, we observe that the southeast (down/left) joystick inputs are the value parameters of four time-back Action MFs (see scenarios DG and Sp1).

The following MFs were learned for the state-based traces for Human P:

- *DG*: none

- *F1*: t-FairyW2State(Wait)

- *F2*: none

- *G1*: t-RockW4Dist(1000.0)

- *G2*: t-GrassW1Dist(3.7), t-GrassW1Dist(3.900), t-GrassW6Dist(2.900), t-GrassW6Dist(3.0), t-GrassW6Dist(3.1), t-GrassW6Dist(3.6), t-GrassW6Dist(3.800), v-GrassW1Dist(19), v-GrassW1Dist(22), v-GrassW1Dist(23), v-GrassW1Dist(24)

- *GH*: t-GrHogW2Dist(0.8)

- *GP*: none

- *L*: t-GrassW4Dist(1000.0), t-GrassW6Dist(1000.0)

- *R*: none

- *Sn1*: none

- *Sn2*: t-GrassW4Dist(1000.0)

- *Sp1*: t-GrassW7Dist(1000.0), t-SprinklerW7Dist(1000.0)

- *Sp2*: t-GrassW0Dist(5.0), t-SprinklerW2Dist(1000.0), t-SprinklerW5Dist(1000.0), t-SprinklerW7Dist(3.5)

Several MFs in the G2 are used to track grass patches in Wedges 1 and 6 (northeast and northwest, respectively). We can see that it has taken several MFs to track essentially the same time-back parameter (e.g. t-GrassW6Dist(3.1), t-GrassW6Dist(3.6), t-GrassW6Dist(3.800) all track the time since the distance to a grass patch in Wedge 6 was around 3). This suggests that discretization may have been too fine-grained for this behavior, necessitating multiple MFs to convey a single memory influences.

The following MFs were learned for Human P's state/action-based traces.

- *DG*: t-Action(SE), t-Action(sE), v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *F1*: v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *F2*: t-Action(E), t-Action(nE), v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *G1*: v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *G2*: t-Action(S), v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *GH*: t-Action(S), v-Action(1), v-Action(2)

- *GP*: t-Action(Sw), t-Action(S), t-Action(Se), v-Action(1), v-Action(2), v-Action(3)

- *L*: t-Action(N), v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *R*: v-Action(1), v-Action(2), v-Action(3)

- *Sn1*: t-Action(S), v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *Sn2*: t-Action(E), t-Action(nE), t-Action(SE), v-Action(1), v-Action(2), v-Action(3)

- *Sp1*: t-Action(SW), t-Action(Se), t-Action(SE), t-Action(sE), v-Action(1), v-Action(2), v-Action(3)

- *Sp2*: v-Action(1), v-Action(2), v-Action(3), v-Action(4)

Similar to what was observed for previous humans' state/action-based traces, no perception MFs were learned and a combination of value-back MFs (with small time-back parameters) and time-back MFs were learned.

Figure 8.40 shows the Bayes Net cross validation F1 scores. In every scenario, the action-based, state/action-based, and hybrid traces had F1 scores that exceeded those for the baseline traces. In about half of those scenarios, the state/action-based trace had the highest F1 score and the hybrid trace (tied with the action-based trace) had the highest score in the other half of those scenarios. Similar trends are observed in the train/test split evaluation, as seen in Figure 8.41. In Figure 8.41, we also note that F1 scores for for the Sn1 scenario were almost 0, but that scores for action-based, state/action-based, and hybrid traces were much higher. This could indicate a switch from behavior based on perception memory influences to that based on action memory influences partway through the scenario.
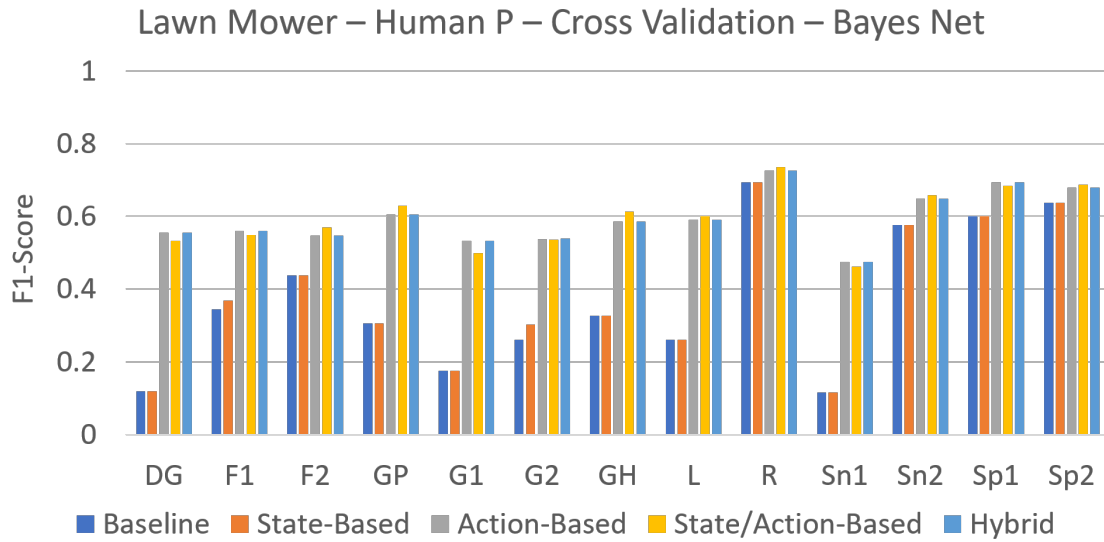
Figure 8.40: LMS Results, Human P, Bayes Net, Cross Validation
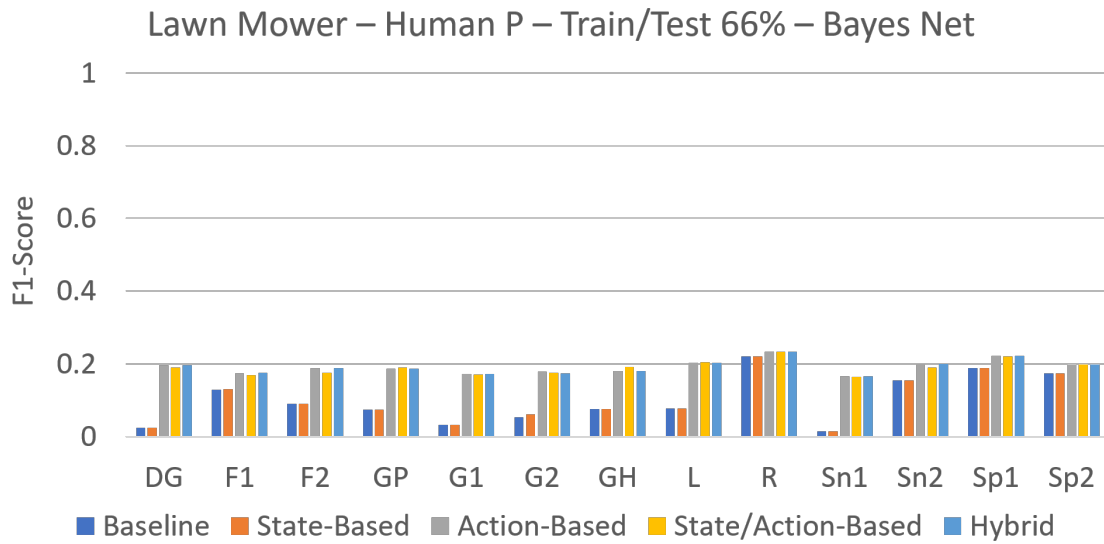
Figure 8.41: LMS Results, Human P, Bayes Net, Train/Test Split
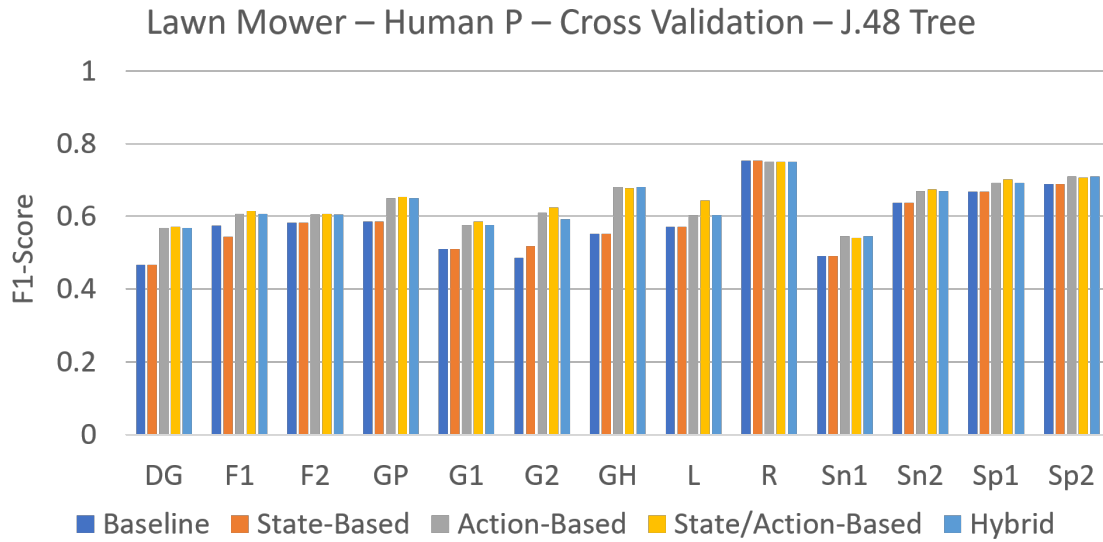
Figure 8.42: LMS Results, Human P, J.48 Tree, Cross Validation
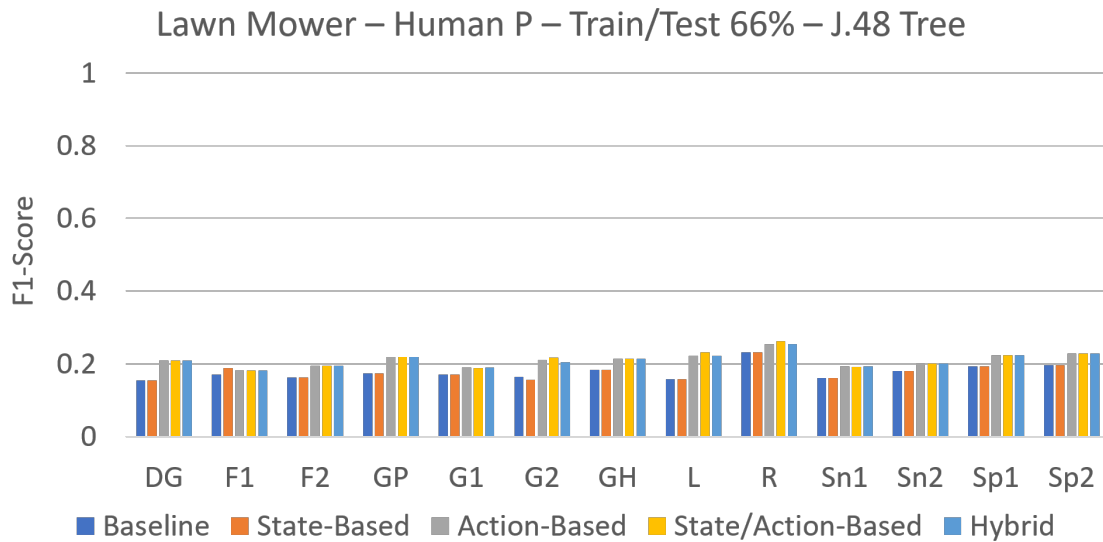
Figure 8.43: LMS Results, Human P, J.48 Tree, Train/Test Split

Figure 8.42 shows the J.48 Decision Tree cross validation results. Except for scenario R (where

all traces had roughly the same F1 score), the memory-enhanced traces (except state-based) had F1 scores that exceeded the baseline trace F1 scores across all scenarios. This trend is generally present in the train/test split evaluation, except for the F1 scenario where the state-based trace actually has the highest F1 score by a small margin (see Figure 8.43).

8.3.4.10 Human T

Tables 8.32, 8.33, and 8.34 show Memory Parameter statistics for action-based, state-based, and state/action-based memory-enhanced traces for Human T, respectively.

Table 8.32: Action-Based Memory Parameters – Human T

Person	Scenario	HF	LF	Set Count	Mem Cases	Default Cases	MF Count
T	DG	0.05	0.05	22	183	56	1
T	F1	0.05	0.05	40	213	80	2
T	F2	0.05	0.05	27	195	54	4
T	G1	0.05	0.05	43	335	110	2
T	G2	0.05	0.05	77	262	238	4
T	GH	0.05	0.05	55	212	116	2
T	GP	0.05	0.05	57	275	130	2
T	L	0.05	0.05	78	267	186	3
T	R	0.05	0.05	29	202	59	2
T	Sn1	0.05	0.05	60	279	231	2
T	Sn2	0.05	0.05	50	312	139	3
T	Sp1	0.05	0.05	54	368	225	2
T	Sp2	0.05	0.05	59	262	166	5

In Table 8.32, we see typical HF=0.05 and LF=0.05 threshold values. The MF counts for each scenario are also low and the default context sizes are relatively high.

Table 8.33: State-Based Memory Parameters – Human T

Person	Scenario	HF	LF	Set Count	Mem Cases	Default Cases	MF Count
T	DG	0.05	0.05	23	171	68	1
T	F1	0.05	0.05	46	163	130	0
T	F2	0.05	0.05	23	169	80	2
T	G1	0.025	0.025	64	275	170	0
T	G2	0.05	0.05	17	425	75	3
T	GH	0.05	0.05	32	255	73	0
T	GP	0.05	0.05	31	283	122	0
T	L	0.05	0.05	59	251	202	6
T	R	0.05	0.05	12	223	38	0
T	Sn1	0.05	0.05	57	245	265	0
T	Sn2	0.05	0.05	44	282	169	2
T	Sp1	0.05	0.05	59	384	209	2
T	Sp2	0.05	0.05	43	205	223	8

In Table 8.33, the HF and LF thresholds are usually set to 0.05 to get the smallest default context. However, the default context sizes are still relatively large, even though they have been smaller for other humans. Also, the MF counts range from 0 all the way to 8.

Table 8.34: State/Action-Based Memory Parameters – Human T

Person	Scenario	HF	LF	Set Count	Mem Cases	Default Cases	MF Count
T	DG	0.025	0.01	22	223	16	6
T	F1	0.025	0.01	38	256	37	5
T	F2	0.025	0.01	27	225	24	5
T	G1	0.025	0.01	43	367	78	4
T	G2	0.025	0.01	77	399	101	8
T	GH	0.025	0.01	55	272	56	6
T	GP	0.025	0.01	57	338	67	5
T	L	0.025	0.01	78	361	92	6
T	R	0.025	0.01	29	240	21	6
T	Sn1	0.025	0.01	60	423	87	9
T	Sn2	0.025	0.01	50	388	63	6
T	Sp1	0.025	0.01	58	516	77	7
T	Sp2	0.025	0.01	59	347	81	9

Table 8.34 shows the statistics for state/action-based traces. The smallest default context sizes are achieved with the lowest possible HF and LF alues. The MF counts are high and the default context sizes are generally low.

The following MFs were learned for the action-based memory-enhanced traces for Human T. Value-back MFs for Action at time-back values of 1 and 2 were the most common. Some time-back Action MFs were learned, especially for the the motionless (C) and downward (south or S) joystick inputs.

- *DG*: v-Action(1)

- *F1*: v-Action(1), v-Action(2)

- *F2*: t-Action(w), t-Action(C), v-Action(1), v-Action(2)

- *G1*: v-Action(1), v-Action(2)

- *G2*: t-Action(S), v-Action(1), v-Action(2), v-Action(3)

- *GH*: v-Action(1), v-Action(2)

- *GP*: v-Action(1), v-Action(2)

- *L*: t-Action(C), v-Action(1), v-Action(2)

- *R*: v-Action(1), v-Action(2)

- *Sn1*: v-Action(1), v-Action(2)

- *Sn2*: v-Action(1), v-Action(2), v-Action(3)

- *Sp1*: t-Action(C), v-Action(1)

- *Sp2*: t-Action(w), t-Action(C), t-Action(s), v-Action(1), v-Action(2)

The following MFs were learned for Human T's state-based memory-enhanced traces.

- *DG*: t-GrassW2Dist(1000.0)

- *F1*: none

- *F2*: t-FairyW4Dist(1000.0), t-GrassW4Dist(1.3)

- *G1*: none

- *G2*: t-GrassW6Dist(2.2), t-GrassW6Dist(2.400), v-GrassW1Dist(71)

- *GH*: none

- *GP*: none

- *L*: t-GrassW0Dist(1000.0), t-GrassW4Dist(1000.0), t-GrassW6Dist(1000.0), t-RockW6Dist(1000.0), t-SnakeW2Dist(8.6), t-SnakeW2Dist(8.8)

- *R*: none

- *Sn1*: none

- *Sn2*: t-SnakeW4Dist(1000.0), v-GrassW4Dist(19)

- *Sp1*: t-GrassW5Dist(1000.0), t-GrassW5Dist(6.100)

- *Sp2*: t-GrassW4Dist(1000.0), t-GrassW5Dist(1000.0), t-GrassW5Dist(0.700), t-GrassW7Dist(4.0), t-SprinklerW0Dist(5.0), t-SprinklerW3Dist(1000.0), v-GrassW1Dist(76), v-SprinklerW0Dist(77)

Several MFs based on wedge distance features were parameterized with the maximum distance value 1000.0, but some were parameterized with smaller values. A couple value-back MFs (one in the G2 scenario, two in the Sp2 scenario) were also created, tracking the distance to objects above the lawn mower from approximately seven seconds prior to the present time step.

The following MFs were learned for Human T's state/action-based traces.

- *DG*: t-Action(E), t-Action(Nw), t-Action(N), t-Action(Ne), v-Action(1), v-Action(2)

- *F1*: t-Action(C), t-Action(e), v-Action(1), v-Action(2), v-Action(3)

- *F2*: t-Action(w), t-Action(C), t-Action(E), v-Action(1), v-Action(2)

- *G1*: v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *G2*: t-Action(S), t-Action(N), t-Action(se), v-Action(1), v-Action(2), v-Action(3), v-Action(4), v-Action(5)

343

- *GH*: t-Action(E), t-Action(N), v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *GP*: t-Action(E), t-Action(N), v-Action(1), v-Action(2), v-Action(3)

- *L*: t-Action(C), t-Action(e), t-Action(Se), v-Action(1), v-Action(2), v-Action(3)

- *R*: t-Action(Nw), t-Action(N), t-Action(NE), v-Action(1), v-Action(2), v-Action(3)

- *Sn1*: t-Action(C), t-Action(e), t-Action(E), t-Action(nE), t-Action(se), v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *Sn2*: t-Action(C), t-Action(Nw), t-Action(N), v-Action(1), v-Action(2), v-Action(3)

- *Sp1*: t-Action(w), t-Action(C), t-Action(e), t-Action(n), v-Action(1), v-Action(2), v-Action(3)

- *Sp2*: t-Action(w), t-Action(C), t-Action(nw), t-Action(n), t-Action(s), v-Action(1), v-Action(2), v-Action(3), v-Action(4)

No perception MFs are learned. Instead several value-back and time-back MFs are learned for each scenario.

Figure 8.44 shows the Bayes Net cross validation F1 scores. Except for state-based traces, the memory-enhanced traces had higher F1 scores than the baseline trace, especially in the G1 and G2 scenarios (Grass Test and Grass Train levels). This trend is also evident in the train/test split evaluation, seen in Figure 8.45.

Figure 8.46, which shows the J.48 Decision Tree cross validation F1 scores, shows the same trends as those observed for the Bayes Net results, except for the Sn1 (Snake 1) scenario where the baseline and state-based traces have a tied F1 score that exceeds those for the other memory-enhanced traces. However, that second trend is not seen in the train/test split evaluation, as seen in Figure 8.47.
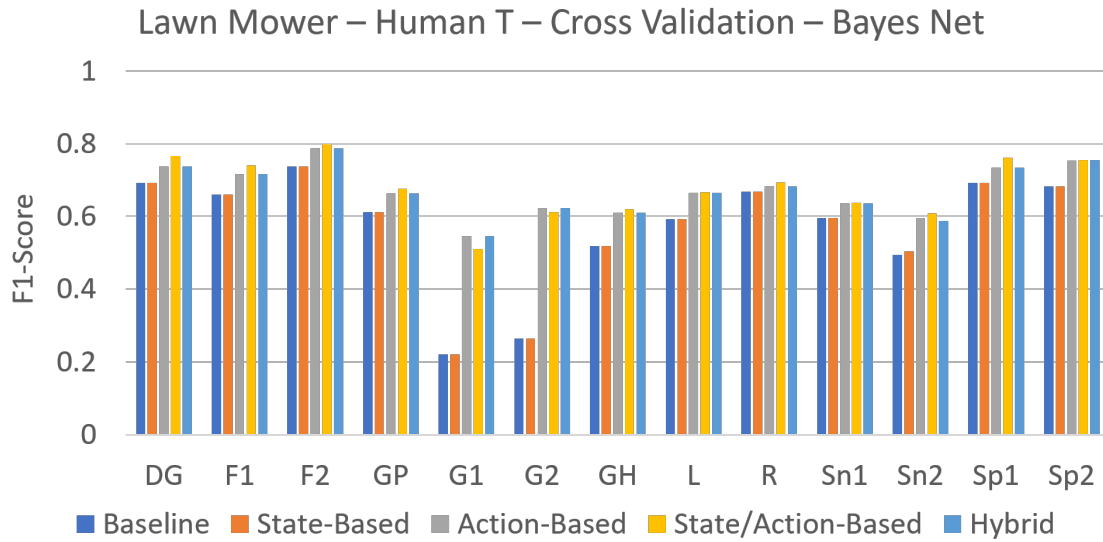
Figure 8.44: LMS Results, Human T, Bayes Net, Cross Validation
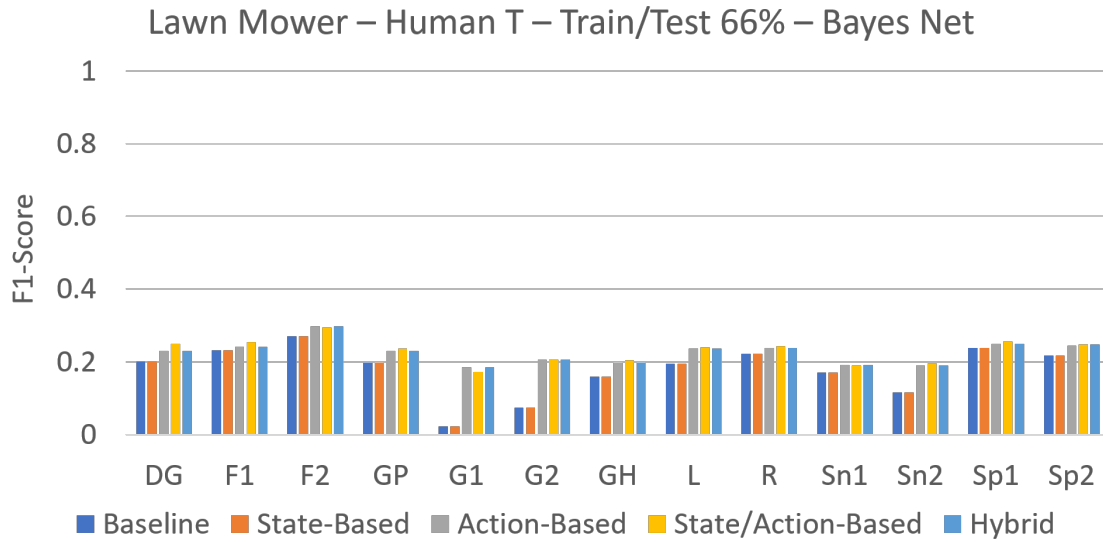
Figure 8.45: LMS Results, Human T, Bayes Net, Train/Test Split
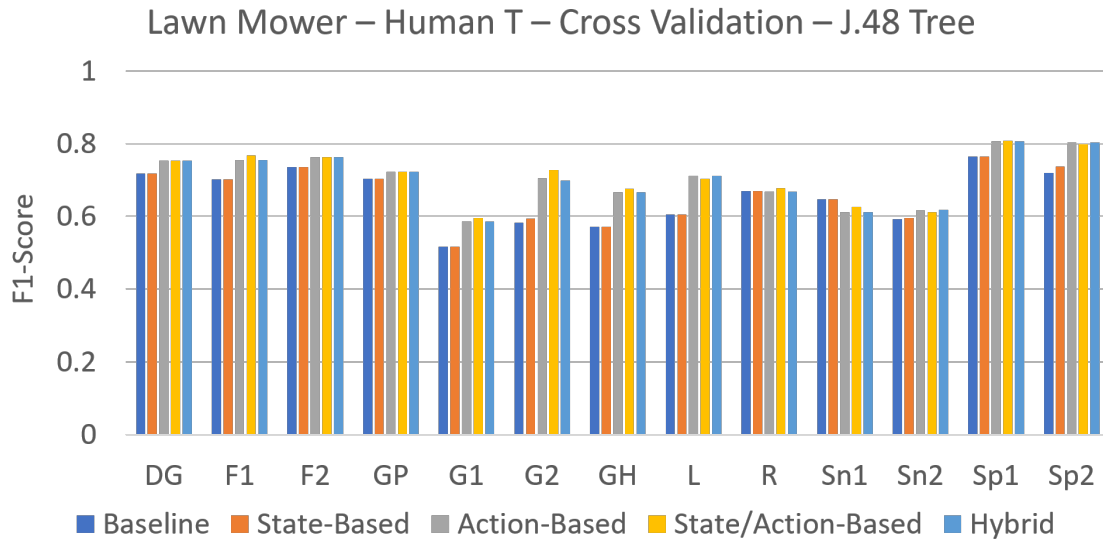
Figure 8.46: LMS Results, Human T, J.48 Tree, Cross Validation
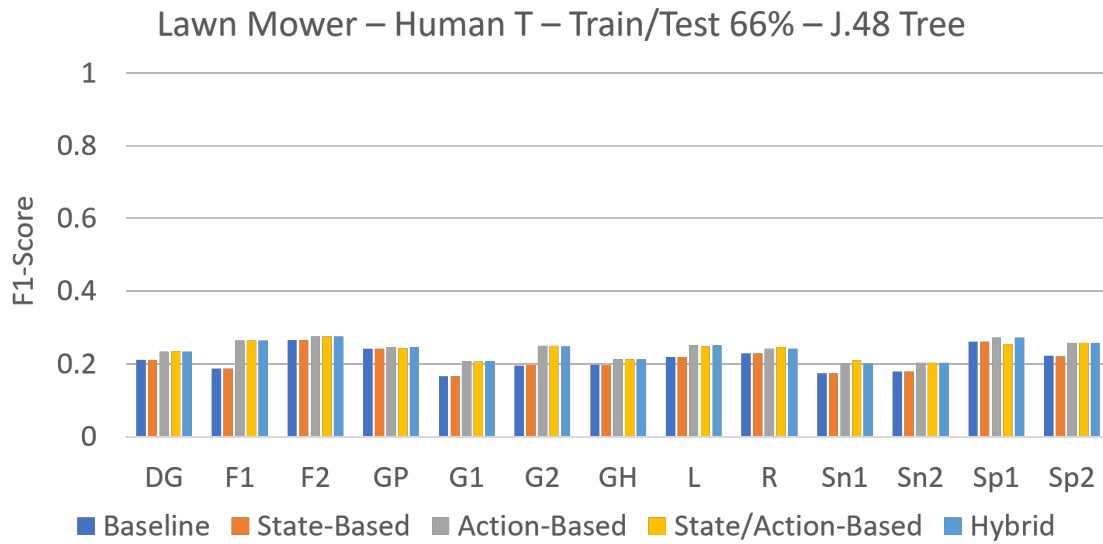
Figure 8.47: LMS Results, Human T, J.48 Tree, Train/Test Split

8.3.4.11   Human W

Tables 8.35, 8.36, and 8.37 show the Memory Parameter statistics for action-based, state-based, and state/action-based memory-enhanced traces for Human W, respectively.

Table 8.35: Action-Based Memory Parameters – Human W

Person	Scenario	HF	LF	Set Count	Mem Cases	Default Cases	MF Count
W	DG	0.05	0.05	67	294	153	3
W	F1	0.05	0.05	59	284	116	3
W	F2	0.05	0.05	33	169	69	2
W	G1	0.05	0.05	70	402	165	3
W	G2	0.05	0.05	61	388	175	4
W	GH	0.05	0.05	55	260	95	2
W	GP	0.05	0.05	48	322	128	4
W	L	0.05	0.05	44	204	128	3
W	R	0.025	0.025	32	231	57	2
W	Sn1	0.025	0.025	43	236	79	3
W	Sn2	0.05	0.05	55	231	128	2
W	Sp1	0.05	0.05	69	342	300	2
W	Sp2	0.05	0.05	64	285	114	4

In Table 8.35, the HF=0.05 and LF=0.05 are the most common threshold values. The default context sizes are still relatively high. The MF counts range from 2 to 4.

Table 8.36: State-Based Memory Parameters – Human W

Person	Scenario	HF	LF	Set Count	Mem Cases	Default Cases	MF Count
W	DG	0.05	0.05	30	308	139	4
W	F1	0.05	0.05	40	257	143	3
W	F2	0.05	0.05	20	165	73	2
W	G1	0.05	0.05	66	270	297	0
W	G2	0.025	0.025	15	515	48	0
W	GH	0.05	0.05	5	334	21	1
W	GP	0.05	0.05	33	323	127	0
W	L	0.05	0.05	20	268	64	2
W	R	0.05	0.05	13	223	65	0
W	Sn1	0.05	0.05	20	247	68	2
W	Sn2	0.05	0.05	24	260	99	0
W	Sp1	0.05	0.05	30	444	198	2
W	Sp2	0.05	0.05	27	274	125	6

Table 8.36 shows the statistics for state-based traces. The MF counts range from 0 to 6. Default context sizes vary from 21 cases(GH scenario) to 297 cases (G1 scenario). HF=0.05 and LF=0.05 are the most common threshold values.

Table 8.37: State/Action-Based Memory Parameters – Human W

Person	Scenario	HF	LF	Set Count	Mem Cases	Default Cases	MF Count
W	DG	0.025	0.01	67	367	80	7
W	F1	0.025	0.01	59	336	64	6
W	F2	0.025	0.01	33	211	27	9
W	G1	0.025	0.01	70	455	112	4
W	G2	0.025	0.01	61	467	96	7
W	GH	0.025	0.01	55	290	65	3
W	GP	0.025	0.01	48	396	54	7
W	L	0.025	0.01	44	278	54	9
W	R	0.025	0.01	32	259	29	4
W	Sn1	0.025	0.01	43	270	45	5
W	Sn2	0.025	0.01	55	290	69	6
W	Sp1	0.025	0.01	69	546	96	8
W	Sp2	0.025	0.01	64	327	72	5

Table 8.37 shows that HF and LF assume the lowest values possible. The default context sizes are small and the MF counts are higher than those for action-based and state-based traces for Human W.

The following MFs were learned for the action-based memory-enhanced traces for Human W. The majority of them are value-back Action MFs with time-back parameter values less than four.

- *DG*: v-Action(1), v-Action(2), v-Action(3)

- *F1*: v-Action(1), v-Action(2), v-Action(3)

- *F2*: v-Action(1), v-Action(2)

- *G1*: v-Action(1), v-Action(2), v-Action(3)

- *G2*: t-Action(N), v-Action(1), v-Action(2), v-Action(3)

- *GH*: v-Action(1), v-Action(2)

- *GP*: t-Action(e), t-Action(E), v-Action(1), v-Action(2)

- *L*: t-Action(N), v-Action(1), v-Action(2)

- *R*: v-Action(1), v-Action(2)

- *Sn1*: v-Action(1), v-Action(2), v-Action(3)

- *Sn2*: v-Action(1), v-Action(2)

- *Sp1*: t-Action(C), v-Action(1)

- *Sp2*: t-Action(N), v-Action(1), v-Action(2), v-Action(3)

The following MFs were learned for the state-based memory-enhanced traces for Human W. Several time-back MFs based on wedge distance features used maximum distance 1000.0 for the value parameter, but some used smaller numbers.

- *DG*: t-DarkGrassW3Dist(3.300), t-DarkGrassW3Dist(3.5), t-DarkGrassW4Dist(1000.0), t-DarkGrassW7Dist(1000.0)

- *F1*: t-FairyW3State(Peek), t-FairyW4Dist(1000.0), t-RockW4Dist(1000.0)

- *F2*: t-FairyW3Dist(1000.0), t-GrassW7Dist(1000.0)

- *G1*: none

- *G2*: none

- *GH*: t-GrassW2Dist(8.6)

- *GP*: none

- *L*: t-GrassW3Dist(1000.0), t-RockW4Dist(2.5)

- *R*: none

- *Sn1*: t-SnakeW0Dist(1000.0), t-BarrierW4Dist(4.4)

- *Sn2*: none

- *Sp1*: t-GrassW5Dist(1000.0), t-SprinklerW5Dist(1000.0)

- *Sp2*: t-BarrierW0Dist(3.300), t-GrassW3Dist(1000.0), t-GrassW4Dist(1000.0), t-SprinklerW2Dist(1000.0),
  t-SprinklerW4State(Spray), t-SprinklerW5Dist(1000.0)

The following MFs were learned for the state/action-based memory-enhanced traces for Human W.
As observed with the previous humans, no perception MFs were learned, but rather a combination
of value-back and time-back Action MFs.

- *DG*: t-Action(C), t-Action(e), t-Action(E), v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *F1*: t-Action(e), t-Action(E), v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *F2*: t-Action(C), t-Action(e), t-Action(E), t-Action(n), t-Action(ne), t-Action(nE), t-Action(NE),
  v-Action(1), v-Action(2)

- *G1*: v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *G2*: t-Action(S), t-Action(N), t-Action(Se), t-Action(SE), v-Action(1), v-Action(2), v-Action(3)

- *GH*: t-Action(N), v-Action(1), v-Action(2)

351

- *GP*: t-Action(e), t-Action(E), t-Action(n), t-Action(N), v-Action(1), v-Action(2), v-Action(3)

- *L*: t-Action(w), t-Action(C), t-Action(e), t-Action(E), t-Action(n), t-Action(N), v-Action(1), v-Action(2), v-Action(3)

- *R*: t-Action(NE), v-Action(1), v-Action(2), v-Action(3)

- *Sn1*: t-Action(S), v-Action(1), v-Action(2), v-Action(3), v-Action(4)

- *Sn2*: t-Action(C), t-Action(e), t-Action(E), v-Action(1), v-Action(2), v-Action(3)

- *Sp1*: t-Action(C), t-Action(n), t-Action(nE), t-Action(N), t-Action(Ne), t-Action(s), v-Action(1), v-Action(2)

- *Sp2*: t-Action(N), v-Action(1), v-Action(2), v-Action(3), v-Action(4)

Figure 8.48 show the Bayes Net cross validation F1 scores for Human W's traces. We observe that state-based traces had F1 scores that are approximately equivalent to those of the baseline traces, but the other memory-enhanced traces had higher F1 scores for all scenarios. This was also observed in the train/test split evaluation, as seen in Figure 8.49. However, the F1 scores for the GP (Gold Pot) scenario for Human W were unusually high, which is perhaps evidence of Human W behaving extremely consistently throughout the trace.

Figure 8.50 and Figure 8.51 show J.48 Decision Tree F1 scores for cross validation and train/test split evaluations, respectively. The action-based, state/action-based, and hybrid traces generally had higher F1 scores than the baseline traces, with the exception being the Sn1 scenario (where the scores were slightly worse for both evaluations) and the F2 scenario in the train/test split evaluation (where the scores were roughly the same). Another interesting observation is that the state-based memory-enhanced trace had the highest cross validation F1 score in the Sp2 scenario, which also exceeded the baseline trace F1 score.
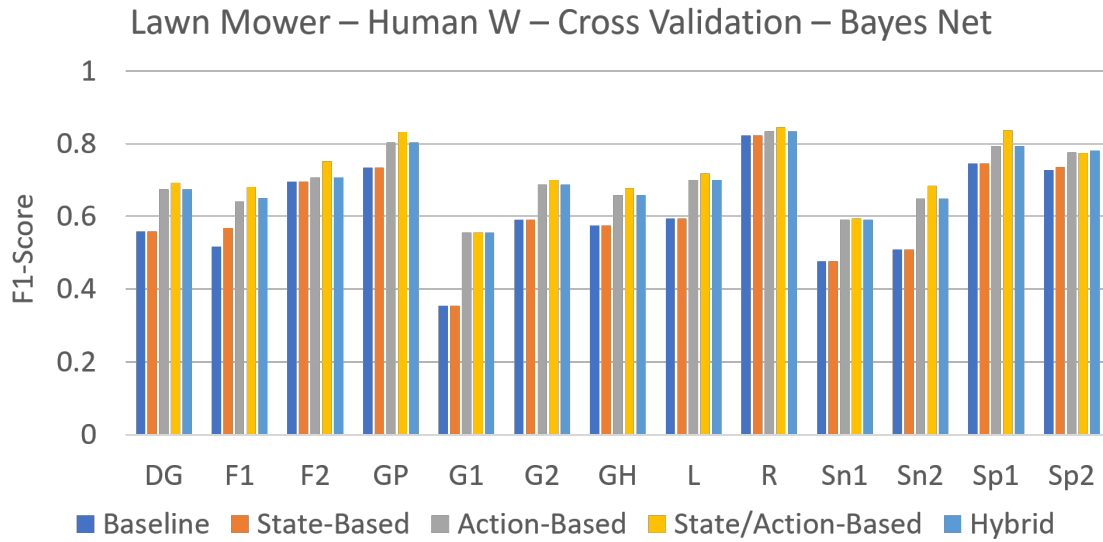
Figure 8.48: LMS Results, Human W, Bayes Net, Cross Validation
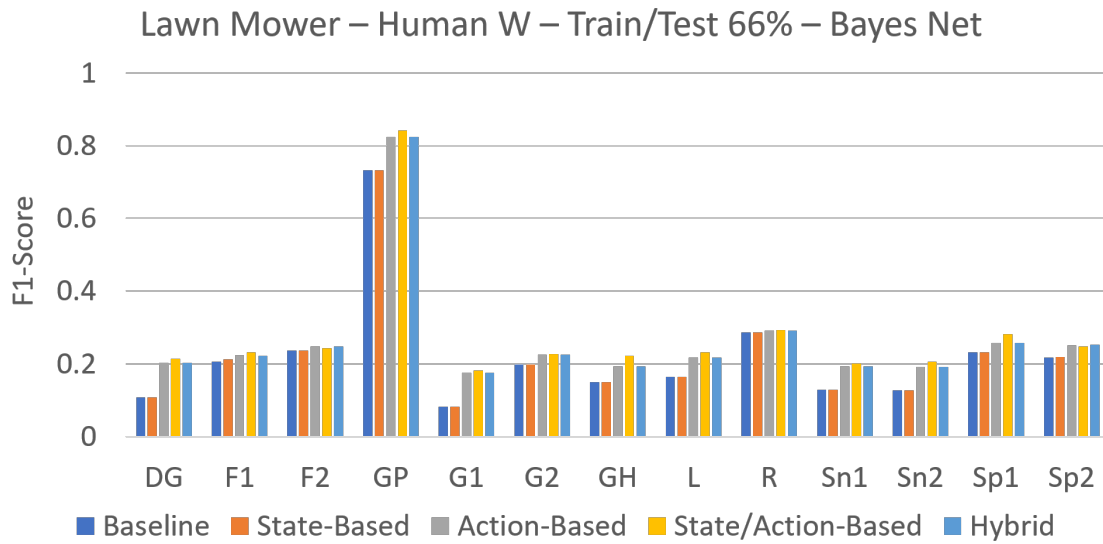
Figure 8.49: LMS Results, Human W, Bayes Net, Train/Test Split
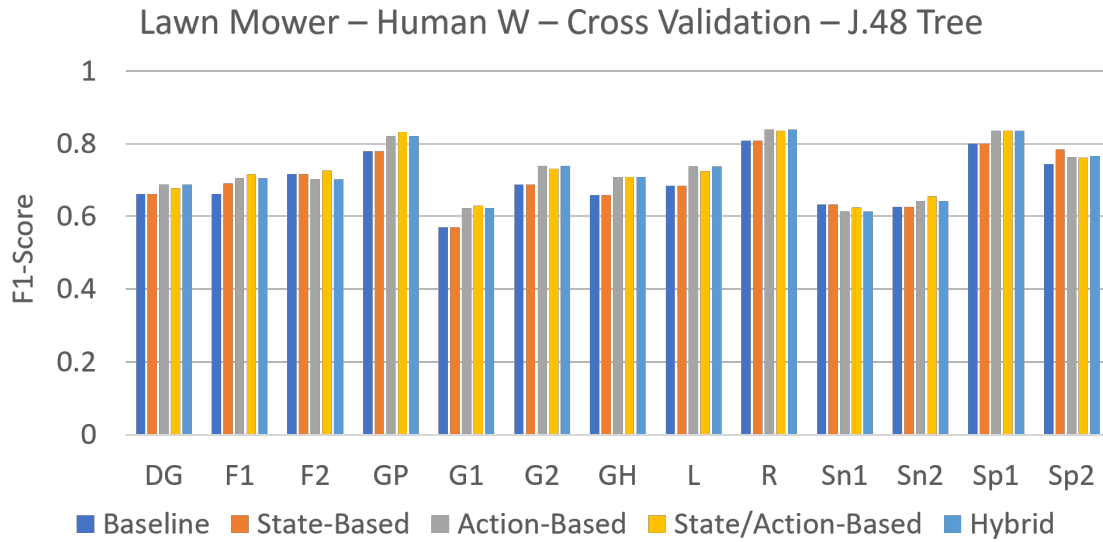
Figure 8.50: LMS Results, Human W, J.48 Tree, Cross Validation
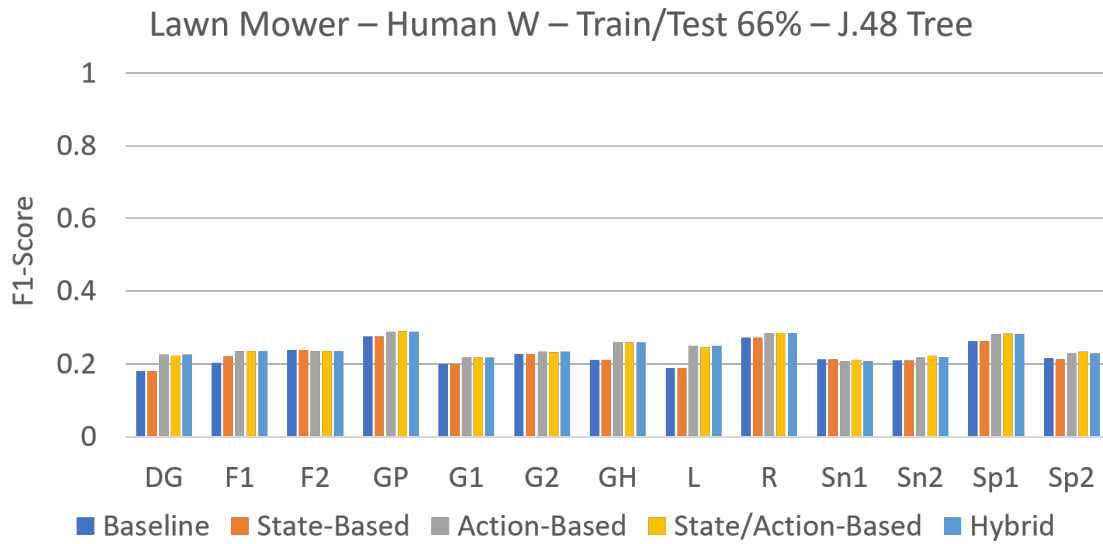
Figure 8.51: LMS Results, Human W, J.48 Tree, Train/Test Split

8.3.4.12   Comparison with Temporal Backtracking

This section compares the results of using supervised learning with memory-enhanced traces generated by MCLvN with those of standard Temporal Backtracking [20]. For each human and scenario, we compare the F1 scores generated by the following algorithms, using 10-fold cross validation:

- J.48 Tree, trained on the state/action-based (STAC) memory-enhanced trace from MCL (denoted "Stac_CV-J48").

- Bayes Net (BN), trained on the state/action-based (STAC) memory-enhanced trace from MCL (denoted "Stac_CV-BayesNet").

- Temporal Backtracking (TB), trained on the original baseline trace (denoted "Base-Tb").

- TB, trained on the STAC memory-enhanced trace (denoted "Stac-Tb"). We wanted to see if TB could benefit from the MFs learned by MCL.

Figures 8.52, 8.53, 8.54, 8.55, 8.56, 8.57, 8.58, 8.59, 8.60, 8.61, and 8.62 show the results of this comparison for Humans AJ, AK, AW, CA, CH, D, G, L, P, T, and W, respectively.
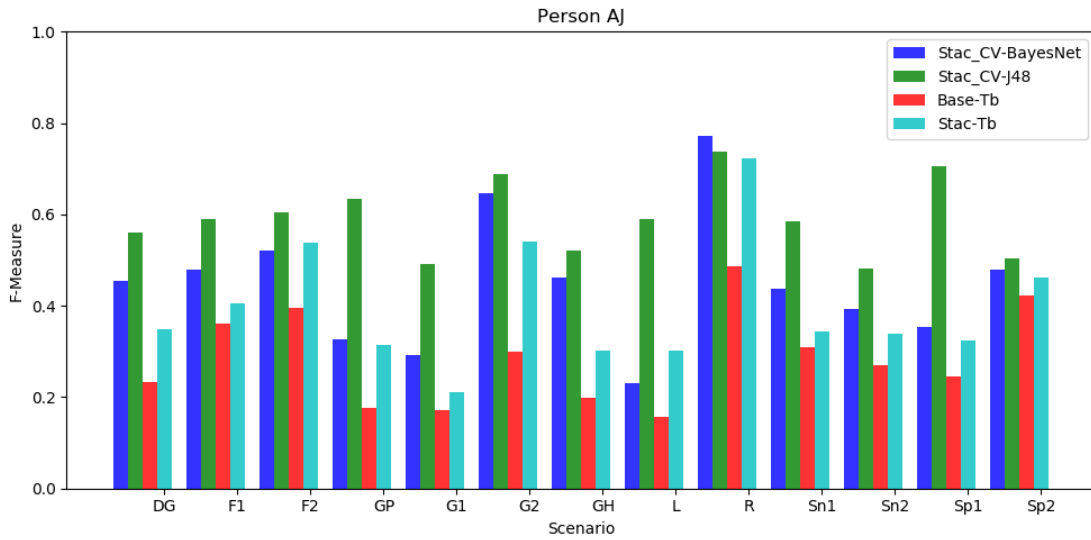
Figure 8.52: LMS Results, Human AJ, Cross Validation, Cross-Methods Comparison

In Figure 8.52, we observe that in all scenarios except the Rock (R) scenario for Human AJ, the J.48 learner with STAC traces had the best performance. The second observation we see is that the F1 score for TB on the original traces generally had the worst performance for all scenarios. The third observation we see is that TB's F1 score when using the STAC traces is always above that of TB on the baseline traces. In certain instances, TB with STAC traces will outperform the Bayes Net learner (with STAC traces), namely in the Fairy 2 (F2) and Leprechaun (L) scenarios, but it generally did not achieve F1 scores as high as those for the Bayes Net and J.48 tree learners.
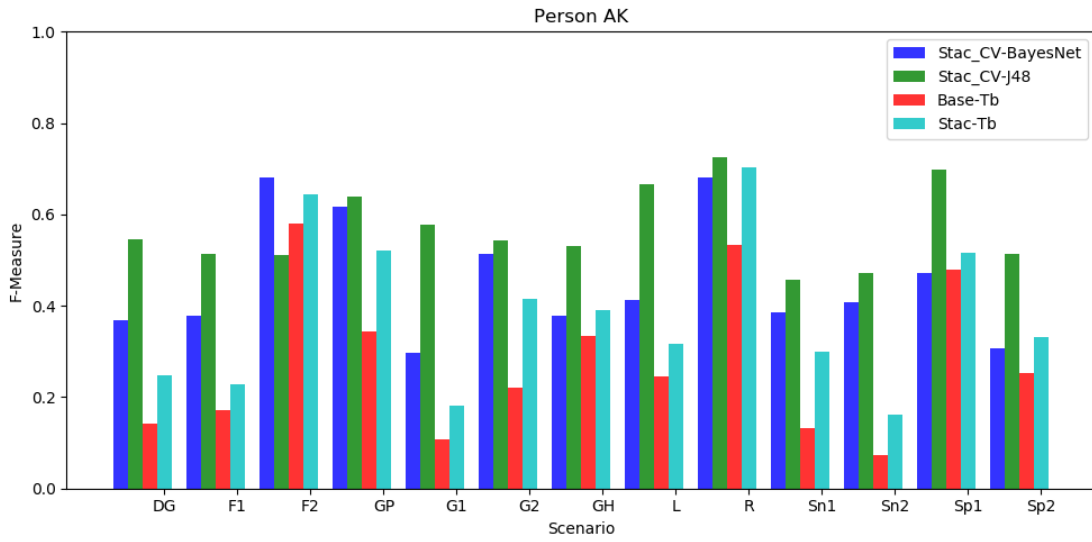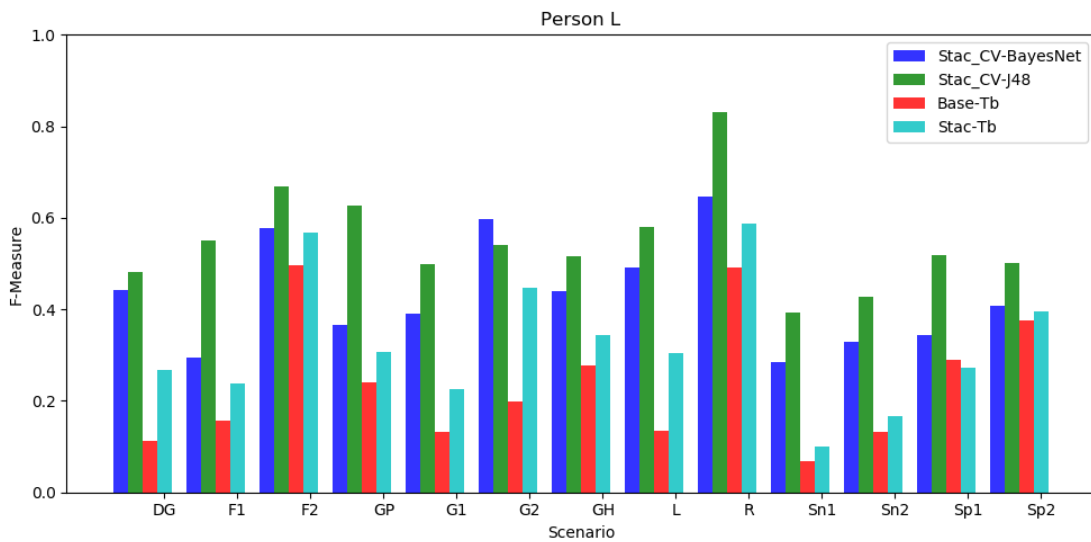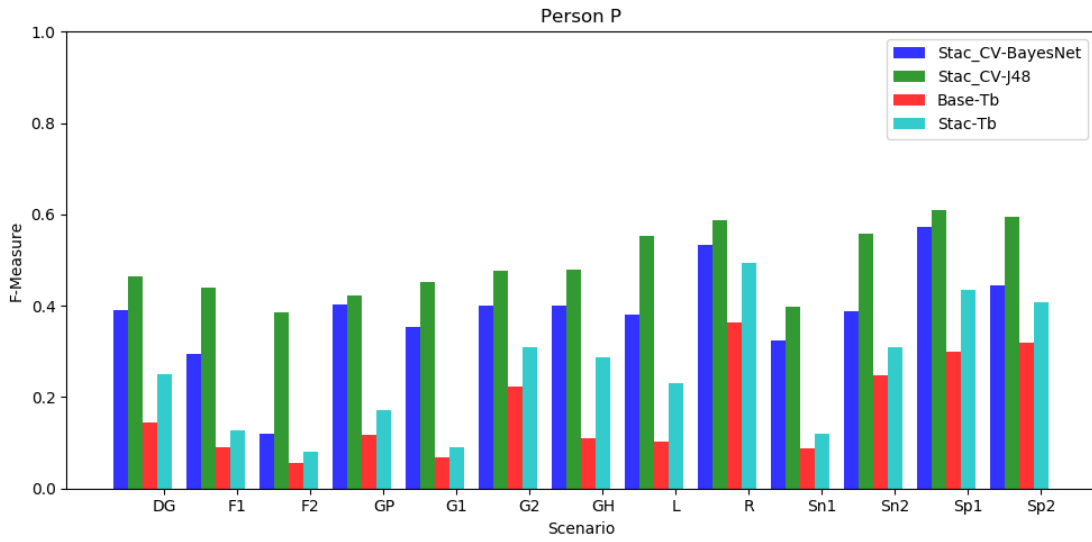
Figure 8.53: LMS Results, Human AK, Cross Validation, Cross-Methods Comparison
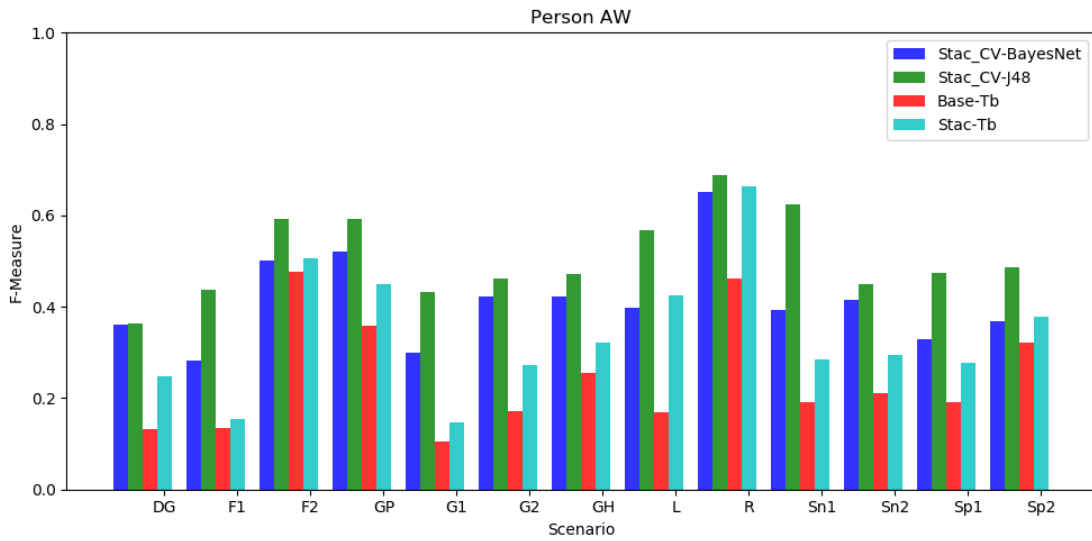
Figure 8.54: LMS Results, Human AW, Cross Validation, Cross-Methods Comparison

In Figure 8.53, we see the results for Human AK. We observe that the F1 scores for TB with STAC

traces are higher than those for TB with baseline traces across all scenarios. In the F1, GH, R, Sp1, and Sp2 scenarios, the F1 score for TB (STAC traces) can outperform either Bayes Net or J.48 tree, but never both for the same scenario. In all but one scenario (F1), J.48 tree with STAC traces has the highest F1 score.

In Figure 8.54, we see the results for Human AW. We observe that J.48 tree has the highest F1 score in all scenarios. Bayes Net generally has the next highest score, but in the F2, L, and R scenarios, TB with STAC traces has a higher score. TB with baseline traces has the lowest score in all scenarios.

In Figure 8.55, we see the results for Human CA. Except for the G2 scenario, J.48 tree has the highest F1 score. In the F2 and Sp2 scenarios, TB with STAC traces has a higher F1 score than that of the Bayes Net learner, but the Bayes Net learner has the higher score in the other scenarios. Once again, TB with baseline traces has the lowest F1 score in all scenarios.
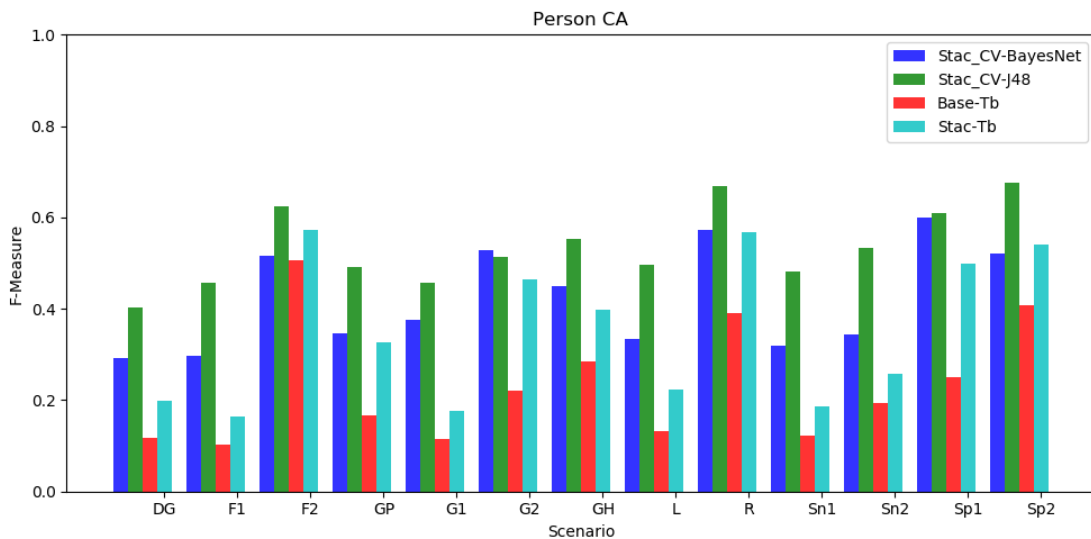
Figure 8.55: LMS Results, Human CA, Cross Validation, Cross-Methods Comparison
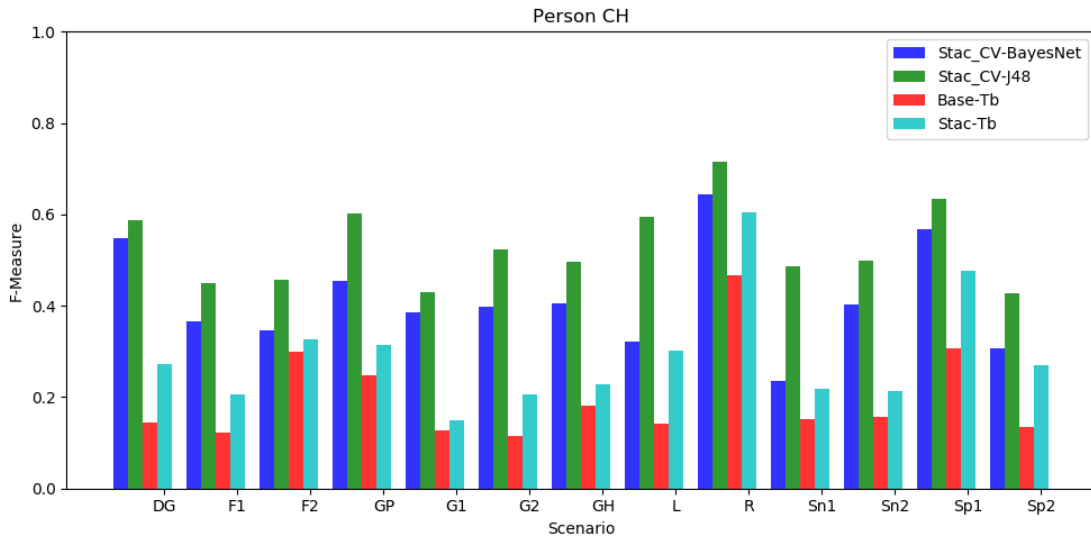
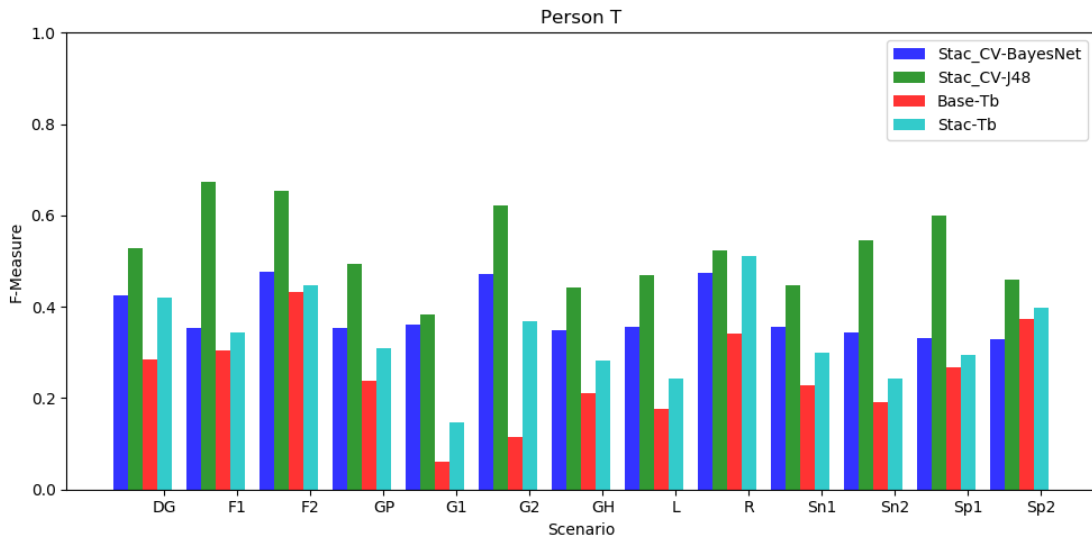Figure 8.56: LMS Results, Human CH, Cross Validation, Cross-Methods Comparison

In Figure 8.56, we see the results for Human CH. In all scenarios, J.48 tree has the highest F1 score, followed by Bayes Net, followed by TB with STAC traces, followed lastly by TB on the baseline traces.

In Figure 8.57, we see the results for Human D. Except in the Rock (R) scenario, J.48 tree has the highest F1 score. The Bayes Net learner has the second highest F1 score in all scenarios except F2 and GP. TB with STAC traces always has a higher F1 score than that of TB with baseline traces. Notably, TB with baseline traces has a roughly equivalent score to the Bayes Net learner in the F2 scenario.
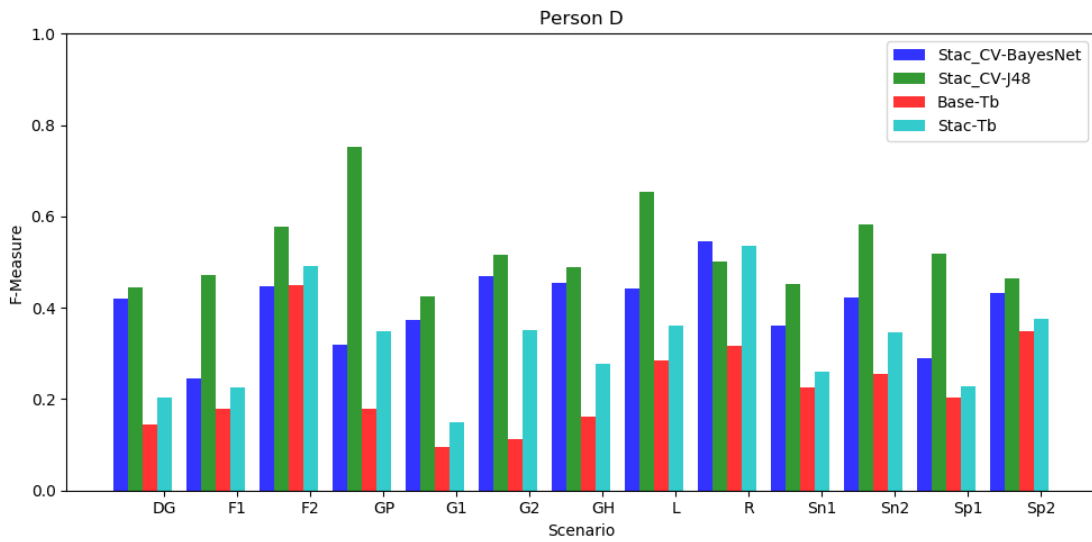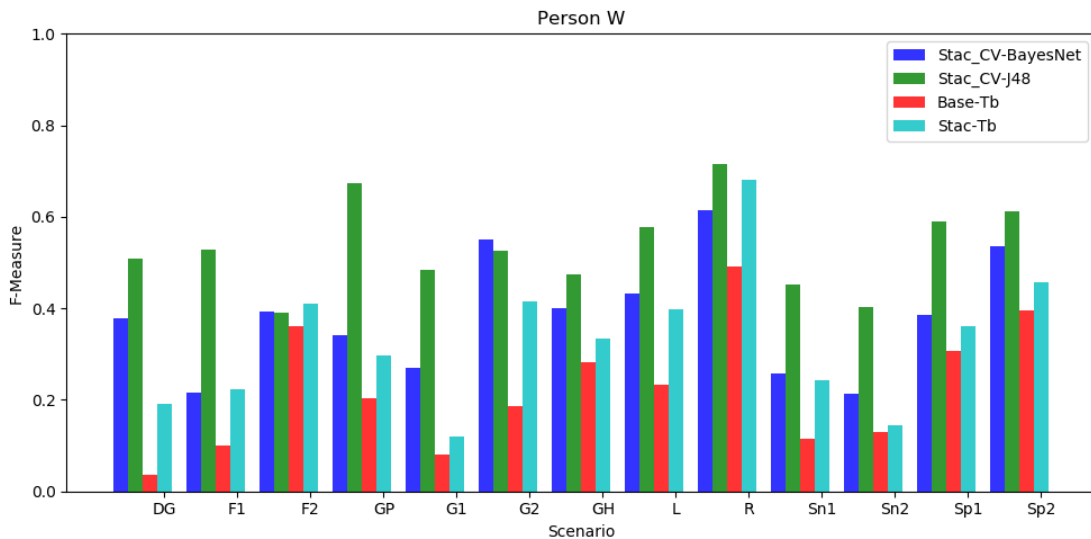
Figure 8.57: LMS Results, Human D, Cross Validation, Cross-Methods Comparison
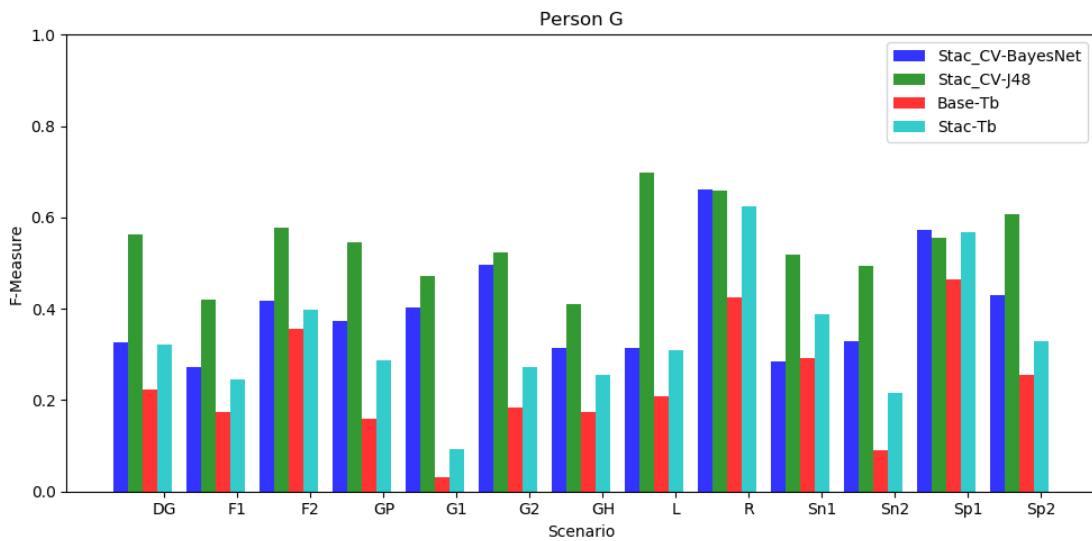
Figure 8.58: LMS Results, Human G, Cross Validation, Cross-Methods Comparison

Figure 8.58 shows the results for Human G. One notable result is seen in the Sp1 scenario, where

TB with the STAC trace has a F1 score that is roughly equivalent to those of the Bayes Net and J.48 tree learners. In the Sn1 scenario, TB with STAC outperforms Bayes Net, but not J.48. It performs similarly to Bayes Net in the DG, F2, and L scenarios. However, the typical trend is that TB F1 scores (both with STAC and baseline traces) are generally lower than those for the J.48 and Bayes Net learners.

Figure 8.59 shows the results for Human L. We observe the common trend of TB with STAC traces having a higher F1 score than TB with baseline traces; we also observe the trend of the J.48 tree and Bayes Net learners generally having higher F1 scores than TB with STAC, though TB with STAC performs similarly to Bayes Net in the Sp2 and F2 scenarios.

Figure 8.59: LMS Results, Human L, Cross Validation, Cross-Methods Comparison

Figure 8.60: LMS Results, Human P, Cross Validation, Cross-Methods Comparison

Figure 8.61: LMS Results, Human T, Cross Validation, Cross-Methods Comparison

Figure 8.60 shows the results for Human P. Similar to Human CH, the J.48 tree has the highest F1

score, followed by Bayes Net, followed by TB with STAC traces, followed by TB with baseline traces across all scenarios.

Figure 8.61 shows the results for Human T. The scenarios in which the results deviate from the common trends seen for the other humans include scenario R, where TB with the STAC trace outperforms Bayes Net and performs similarly to J.48 tree, and scenario Sp2 where Bayes Net has the lowest F1 score (even lower than TB with the baseline trace).

Figure 8.62 shows the results for Human W. For the F1, F2, and R scenarios, TB with STAC traces has a higher F1 score than the Bayes Net learner. For all other scenarios, TB with STAC traces has a lower F1 score than Bayes Net, though it has a higher F1 score than TB with the baseline traces. Also, J.48 tree has the highest F1 score for all scenarios except scenario G2.

Figure 8.62: LMS Results, Human W, Cross Validation, Cross-Methods Comparison
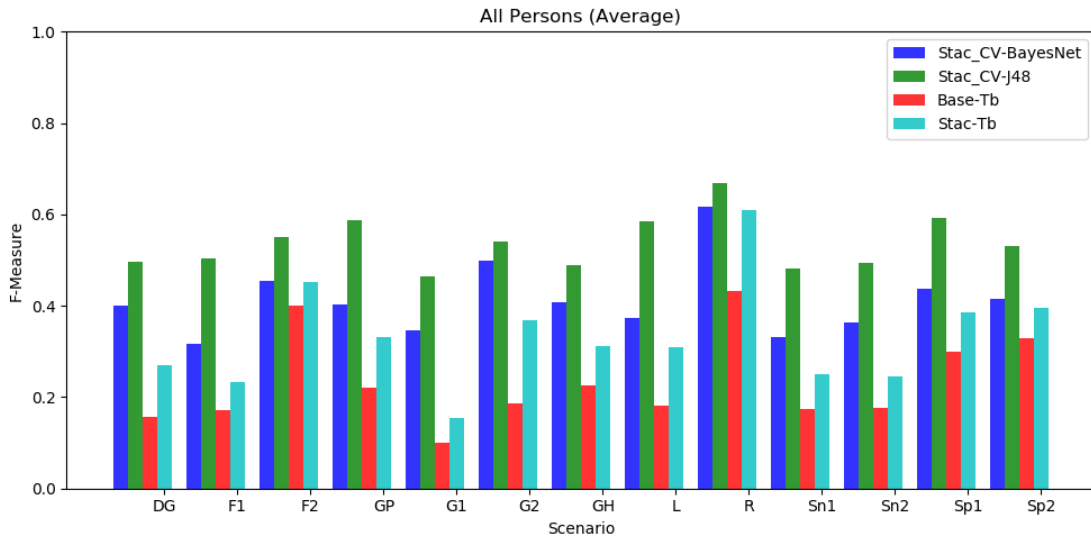
Figure 8.63: LMS Results, Human W, Cross Validation, Cross-Methods Comparison

Figure 8.63 shows the F1 scores for each learner, averaged over all humans for each scenario. These overall results exemplify the general trends that we have seen across learning methods, which are summarized here:

1. In general, the J.48 tree learner seemed to achieve the highest performance when using the STAC memory-enhanced traces.

2. Temporal Backtracking, when using the original/baseline traces, did not achieve F1 scores as high as those of the learners which used the STAC traces generated by MCL.

3. Temporal Backtracking sees an increase in F1 score when using the STAC traces instead of the original/baseline traces. These F1 scores typically do not exceed those for the J.48 tree and Bayes Net learners, which are reactive learners. However, TB with STAC traces can sometimes achieve performance as good as that of the Bayes Net learner in certain scenarios.

We have thus shown that MCL advances the state of the art, allowing standard supervised learning to use memory-enhanced traces to achieve learning performance that is both better than its own when using the original memoryless traces and that is comparable to (and generally better than) Temporal Backtracking, a memory-based CBR LfO algorithm. We now discuss overall results of MCLvN for the lawn mower domain.

8.3.4.13 Overall Results

This section presents the overall results for the learning performance F1 scores, averaged over all humans. We also discuss common trends that were observed over all humans.

The following is a summary of the memory-based behaviors that were facilitated by each scenario in LMS.

- Dark Grass (DG): In this level, dark grass patches look the same both mowed and uncut. The human must remember which patches were already mowed.

- Fairy 1/2 (F1 and F2): In these levels, fairies restore all health, but follow a finite state machine. To utilize a fairy, the human must remember which part of the behavior sequence the fairy is in.

- Gold Pot (GP): In this level, a gold pot restores one health point. It reveals itself once, then may be hidden by grass. The human must remember where gold pots were sighted.

- Grass Test (G1): This level combines grass, dark grass, and rocks in one level.

- Grass Train (G2): This level features many grass patches, which each can be mowed once. No significant memory is required.

- Ground Hog (GH): Ground hogs damage the lawn mower. They reveal themselves, then hide under grass. The human must remember ground hog sightings and avoid those locations.

- Leprechaun (L): Leprechauns can move. They restore one health point, but are invisible when moving through grass. The human must remember the Leprechauns' direction and last sighting to predict/infer their location.

- Rock (R): Rocks damage the lawn mower upon contact. They are always visible and do not require significant memory.

- Snake 1/2 (Sn1 and Sn2): Snakes can move. They damage the lawn mower and are invisible when moving through grass. The human must remember the snakes' direction and last sighting to predict/infer their location.

- Sprinkler 1/2 (Sp1 and Sp2): Sprinklers damage the lawn mower within an area of effect, but they follow a finite state machine. To safely proceed past a sprinkler, the human must remember which part of its behavioral sequence a sprinkler is in.

Table 8.38 shows the LMS scenarios, from lowest baseline F1 score to highest baseline F1 score. The Rock (R) scenario has the highest score. It also has no memory-based entities – its primary purpose was to serve as a soft tutorial showing the human that rocks cause damage. The Grass Train (G2) scenario has the lowest score. Even though it has no explicit memory-based entities (just grass), it is potentially one of the longest scenarios and has potentially the one of the largest number of entities – it essentially serves as a soft tutorial and initial practice session for controlling the lawn mower with a joystick.

Table 8.38: Baseline F1 scores

BN Scenario	BN F1	J48 Scenario	J48 F1
G2	0.27	G2	0.54
GH	0.38	DG	0.59
DG	0.41	GH	0.60
Sn2	0.42	Sn2	0.60
F1	0.49	F1	0.62
G1	0.50	G1	0.63
Sn1	0.51	Sn1	0.63
L	0.54	L	0.66
GP	0.63	GP	0.70
F2	0.68	F2	0.73
Sp1	0.68	Sp1	0.73
Sp2	0.69	Sp2	0.73
R	0.77	R	0.79

Dark Grass (DG), Ground Hog (GH), and Snake 2 (Sn2) are the scenarios with the next lowest baseline F1 scores. The DG scenario is unique in that the entities requiring memory change state, but remain visible. In contrast, the ground hogs in GH and the snakes in Sn2 change visibility. In other words, dark grass patches appear to remain the same and memory of their actual state is required, but ground hogs (and snakes) appear to cease to exist and memory of their actual location is required. The latter type of memory is also used for other memory-based entities.
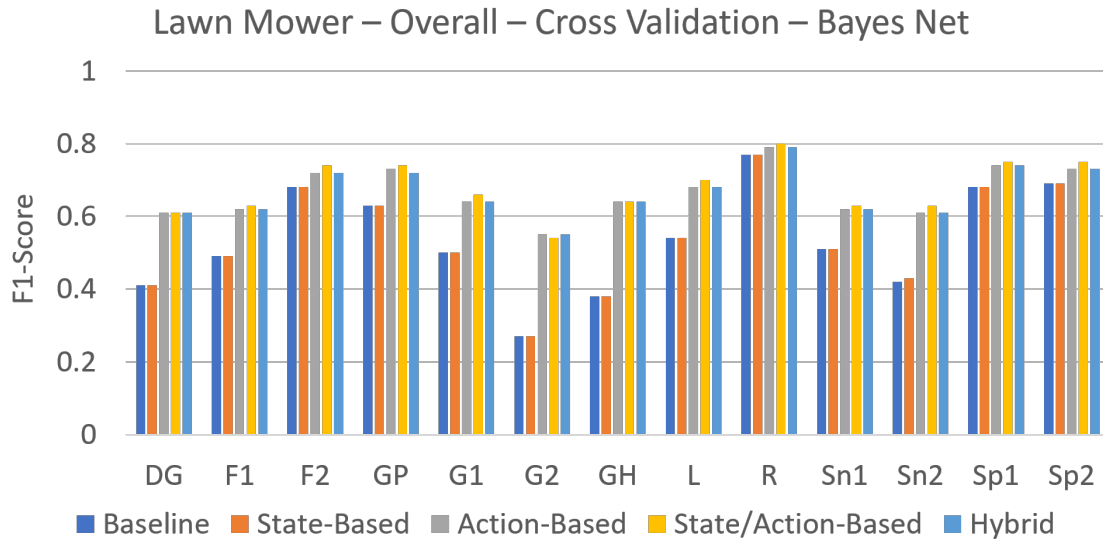
Figure 8.64: LMS Results, Human Overall, Bayes Net, Cross Validation
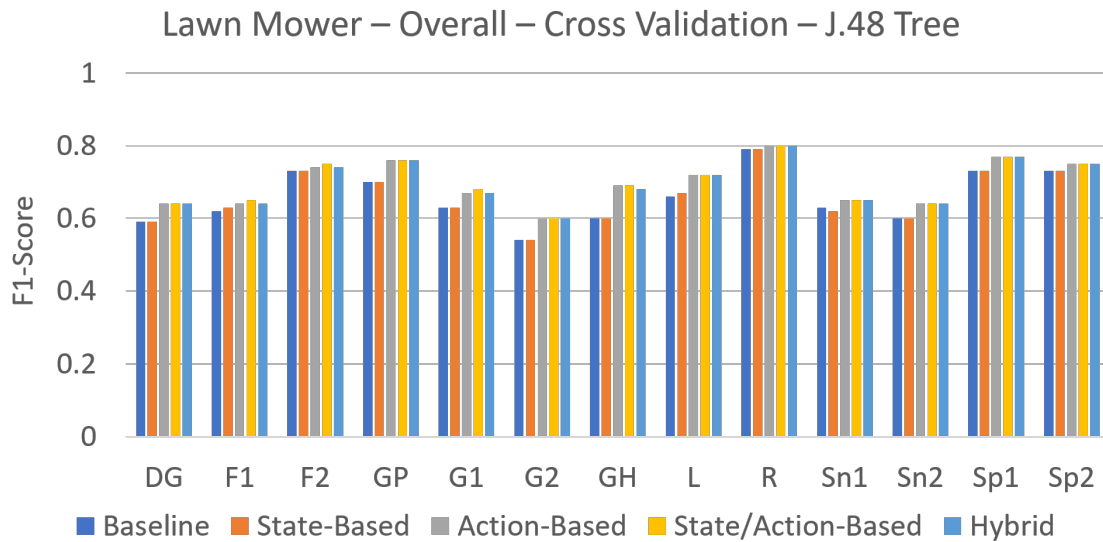
Figure 8.65: LMS Results, Human Overall, J.48 Tree, Cross Validation

Figure 8.64 shows the average F1 scores for each scenario for the baseline traces and the four types

of memory-enhanced traces, under the cross validation evaluation of the Bayes Net learner. Figure 8.65 shows the average F1 scores for each scenario for the cross validation evaluation of the J.48 Decision Tree learner.

We note the following observations:

1. In all scenarios, the F1 score for the state-based memory-enhanced trace is roughly equivalent to that of the baseline trace. (Again, this analysis is made using absolute comparisons, not statistical analysis.)

2. The action-based, state/action-based, and hybrid memory-enhanced traces, based on an absolute comparison, on average have higher F1 scores than the baseline trace F1 score in the majority of the scenarios. (The increase is very small for the Rock scenario.) The F1 score increases are more pronounced in the Bayes Net results.

3. If the scenarios are ordered by F1 score on a memory-enhanced trace (excluding state-based trace), then the order will be approximately the same as the scenario order for F1 scores on baseline traces.

4. In several scenarios, the state/action-based traces have the highest F1 score out of all the memory-enhanced traces for a given scenario.
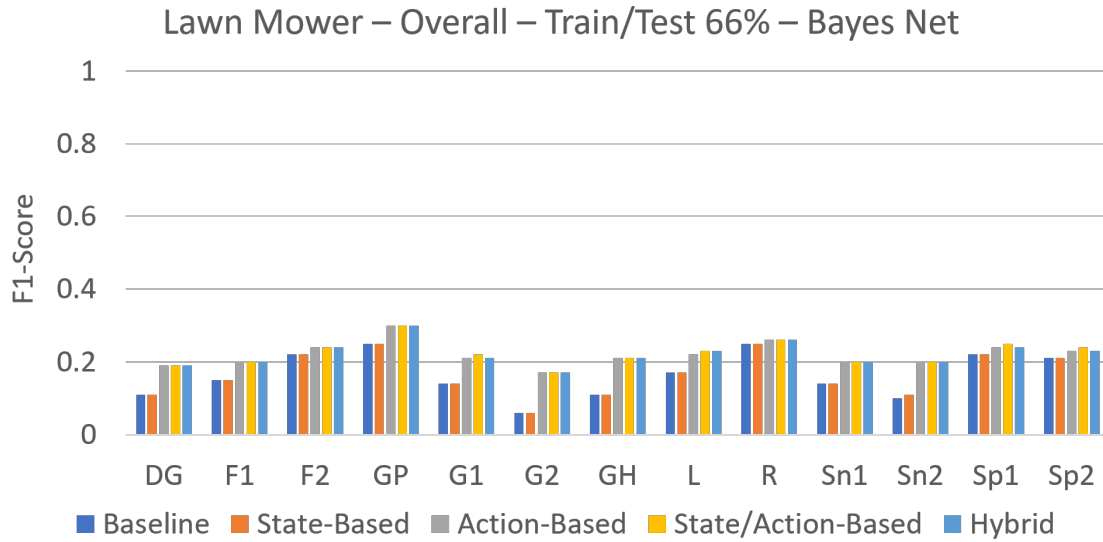
Figure 8.66: LMS Results, Human Overall, Bayes Net, Train/Test Split
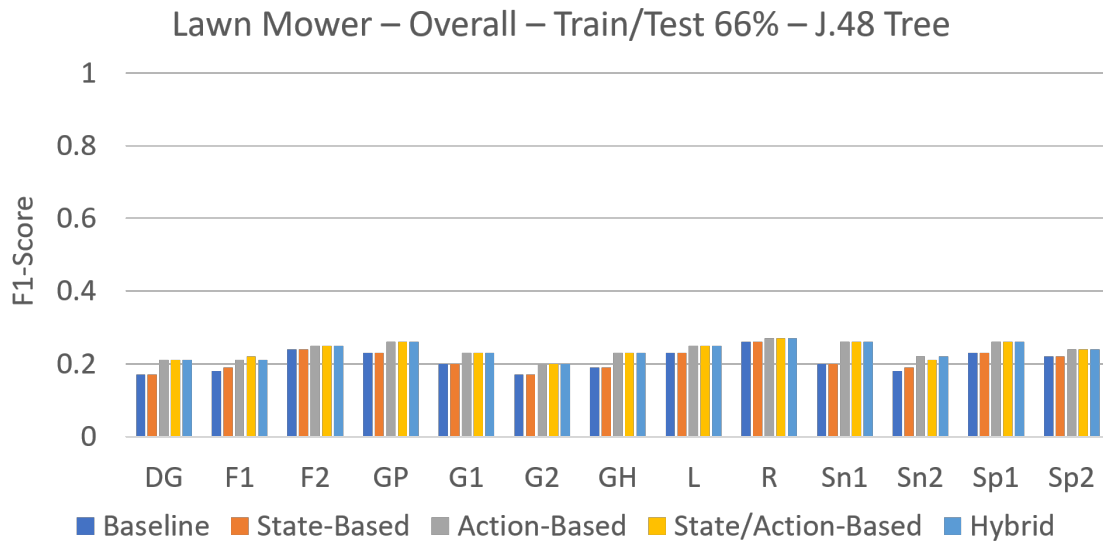
Figure 8.67: LMS Results, Human Overall, J.48 Tree, Train/Test Split

For our discussion, we reiterate the caveats to our results. First, results for stochastic learners

(e.g. Bayes Net) are computed for only a single trial instead of averaged over multiple runs. Second, learning results for a given human in a given scenario are only computed for the human's last attempt on the scenario, and so no statistical analysis is performed. Comparisons between results are based on an absolute comparison only. We also note here that even though we present and analyze results that are aggregated across all humans, MCL is geared toward modeling the memory influences for a specific observed actor, not the memory influences across actors for a given task. Furthermore, our use of only 11 human subjects in the LMS domain does not allow for strong statistical power. To perform an effective power analysis or to prove statistical significance for a claim that MCL improves learning performance for all humans in general for the lawn mower domain, more human subject testing is recommended.

Figure 8.66 shows the average F1 scores for each scenario for the train/test split evaluation of the Bayes Net learner. Figure 8.67 shows the average F1 scores for for each scenario for the train/test split evaluation of the J.48 Decision Tree learner. From these graphs, we note the following observations:

1. The memory-enhanced traces (except the state-based variant) still result in improved F1 scores across all scenarios.

2. The improvements in F1 score are more pronounced in the Bayes Net results.

3. The scenarios, ordered by F1 score on memory-enhanced traces, will be ordered largely in the same way as they would be if they were ordered by baseline F1 score.

During our analysis of the results for the individual humans, the following trends were observed:

1. For action-based memory-enhanced traces, the number of MFs learned by MCL is relatively low and the default context size is relatively high.

371

2. For state-based memory-enhanced traces, the number of MFs learned by MCL is even lower than that for action-based memory-enhanced traces.

3. For state/action-based memory-enhanced traces, the default context size is generally small. This is achieved through very low High Filter (HF) and Low Filter (LF) thresholds. The MF count is relatively high in general.

4. The learned MFs for action-based memory-enhanced traces will mostly be value-back.

5. The learned MFs for state-based memory-enhanced traces will typically take one of two forms. Either it will track the time since a distance was last at its maximum value or it will track the time since a distance was nearly zero. Nominal features will very rarely be the basis of learned perception MFs. This trend could be an artifact of the lawn mower domain in that the mere presence (or absence) of an entity may have been enough to inform human behavior, rather than precisely how far away the entity was.

6. The learned MFs for state/action-based memory-enhanced traces will be mostly if not entirely based on the Action feature. The value-back MFs learned will track the most recent Action values. The time-back MFs may or may not be strongly associated with particular directional inputs of the joystick, depending on the human's behavior.

The most interesting observation from these results is that MCL prefers not to create MFs based on perception for state/action-based memory-enhanced traces, and yet these traces learn more MFs than action-based memory-enhanced traces. The reason for this is likely because of the perception/action and nominal/numeric imbalances mentioned previously. There are many perception features and only one action feature. Thus, MCL's tree extraction will quickly create raw MFs for the Action feature, but it will be slower to create raw MFs for particular perception features because there are so many of them to choose from. Furthermore, most of the perception features

are numeric, so they many not experience significant enough change to warrant the creation of raw MFs until several time steps have occurred. In contrast, nominal features, such as the Action feature, will generate a raw MF if their value changes at all, even in a single time step. Therefore, memory influences are more readily resolved through Action MFs.

We reduced the sampling rate of our traces to 10 Hz and evaluated hybrid traces (which combine perception MFs from state-based traces and action MFs from action-based traces) to mitigate these imbalances. However, the state/action-based memory-enhanced traces generally outperformed hybrid traces.

Thus, there are three take-aways from the lawn mower domain experiments with regards to the impact on this investigation:

1. The strong emphasis on learning MFs based on a human's choice of Action indicates the importance of including memory mechanisms that store "past intent" or facilitate the continuation of immediate activities in progress. In other words, the burden on LfO is greatly reduced when a mechanism (such as MFs) store information about immediate.

2. As seen by the high performance of state/action-based memory enhanced traces over hybrid traces, despite the observed feature imbalance between nominal/numeric and perception/action features, the consideration of perceptions and actions in tandem for the formation of appropriate MFs is key to effective learning, even if one set of features do not contribute directly to the formation of MFs based on those features themselves, but rather indirectly to the formation of MFs based on other features.

3. Memory is unobservable, but we do see evidence of a positive impact on LfO as a result of including memory, which was the goal of our research. Our memory-enhanced traces generally resulted in improved learning performance both when coupled with memoryless

supervised learning and when compared to the memory-based learner Temporal Backtracking.

8.4   Discussion

This chapter described the experiments for two domains, vacuum cleaner and lawn mower, which were conducted to investigate the hypothesis in Chapter 3. We evaluated our approach, MCLvN, as it was defined in Chapter 6, pertaining to its ability to model memory that can be used to improve LfO.

In the vacuum cleaner domain, we ran the same set of experiments that we conducted to test MCLv0 and we found that MCLvN achieved learning performance that was as effective as that of MCLv0 regarding learning performance and more concise in its representation of memory influences through the removal of certain extraneous MFs.

In the lawn mower domain, we evaluated MCL on human data, which contains inconsistent behavior and continuously-valued features in a real-time environment. We included various memory-based entities in several scenarios to facilitate memory-based behaviors in our human subjects. For this domain, we tested four memory-enhanced trace types: action-based, state-based, state/action-based, and hybrid. Our results indicate that state/action-based memory-enhanced traces are best suited for the lawn mower domain and we observed increased learning performance for the majority of the lawn mower scenarios across all humans.

Thus, we have empirically demonstrated the feasibility of our approach in fulfilling the objective of our investigated hypothesis. MCL can automatically learn and model the memory influences on observed human behavior and present its learned memory features as additional input for improving LfO performance.

CHAPTER 9: CONCLUSIONS AND FUTURE WORK

This chapter summarizes the research undertaken in this dissertation as well as its impact on the field and future research. It is organized as follows:

- Section 9.1 summarizes the scope of the research undertaken in this dissertation, its motivation, and main contributions.

- Section 9.2 discusses the difficulties that were encountered during this research and how they were addressed.

- Section 9.3 summarizes the main findings of this research.

- Section 9.4 outlines possible directions for future research stemming from the findings in this dissertation.

9.1 Summary

The research in this dissertation was motivated by the many impactful steps made by Learning from Observation (LfO) toward automatically modeling human behavior. Models of human behavior can be used to create artificially intelligent agents that can replace the human operator, analyze human behavior for abnormalities, or automate low-level decision-making for human-directed tasks. These models of human behavior can be constructed manually through knowledge engineering or they can be created automatically through machine learning, specifically LfO.

LfO, as discussed in Chapter 2, has the potential to more quickly and accurately transfer the knowledge of how to perform a tactical task from an observed expert agent to a learning agent solely from external observation. This notion of learning follows the intuition of how humans learn from each

other by watching how others perform a task rather than through explicit verbal articulation of specific procedures. LfO has been applied to many practical applications in simulated and physical agents. However, the problem of failing to properly account for the effects of internal state influences, such as memory of past events, still remained [58]. The call for LfO-specific algorithms that deviated from the family of standard supervised learning techniques was the ultimate motivation behind this research.

Various approaches to accounting for memory in LfO already existed. In particular, the intuition behind Floyd's work [20] seemed like an appealing manner by which entire runs of observed performance (e.g. memory of *everything* that has occurred) could be converted into usable procedural knowledge without any training. This research originally germinated from an introspective analysis of the strengths and weaknesses of Temporal Backtracking [20]. Temporal Backtracking was an advantageous memory-based learning algorithm because it discriminates between different cases in a case base by analyzing where they differ in their history (memory) without making any assumptions about where those discrepancies in cases' run histories were located. TB goes as far back in time as necessary to resolve differences in memory between cases recommending different actions. However, the notion that resulted from this introspection was how TB would be much more efficient if it could only remember certain aspects of an observed actor's performance, such as the last action taken or the time since the last observed instances of certain feature values.

Thus, the idea behind this dissertation research was created from the idea that TB only compares cases at time steps prior to the present time step because there is some sort of memory influence, but if the cause of that memory influence could be memorized, then TB would not have to analyze that part of a case's run history. Thus, TB could be used to find the memory influences in an observed actor's performance (and never again have to infer that influence). If TB could infer every memory influence, as would be evidenced by TB being immediately resolved in its first iteration, then it would be possible to combine these memory influences with learning algorithms

that would otherwise not use information from memory. These ideas were discussed in Chapter 4.

The validation of the hypothesis described in Chapter 3 formed the scope of the investigation in this dissertation. The hypothesis we sought to validate is listed here:

"The internal state of an observed entity exhibiting a behavior possibly reliant on memory of past events can be modeled through a set of memory features that approximate the influence of memory on the entity's decisions."

Put more concisely, we hypothesize that a preprocessing step can learn a set of memory features that summarize the influences of memory on observed behavior and improve LfO. Thus, Memory Composition Learning (MCL) was created to adapt TB to learning memory influences, as discussed in Chapter 5. The first step was to learn memory influences that "resolved" a single case via an extraction process modeled after a series of modified TB operations (called Memory TB). Then, the results for each case would be combined in a refinement process to find an appropriate manner of expressing memory influences that affect the observed behavior as a whole over all cases; such memory influences were expressed as Memory Features (MFs), which comprise a "memory model" of the observed entity. In Chapter 5, this approach was validated in a simple simulated vacuum cleaner domain (also used in [58]) such that memoryless learning algorithms could use the learned MFs to improve learning performance for behaviors requiring memory.

However, the minimalist approach, as discussed in Chapter 6, had several limitations that hindered the application of MCL to realistic human data. These limitations were 1) ignoring context, 2) inefficient MF extraction, 3) inability to handle continuous data, 4) extraneous memory representations, 5) not accounting for stochastic effects. Human data is context-based, large-scale, continuous, complex, and inconsistent/random/stochastic. Thus, the remainder of the dissertation addressed these aspects of human behavior in learning a memory model through MCL.

A comprehensive approach to MCL, labeled MCLvN, was implemented. It had the following improvements, as described in Chapter 6:

1. To account for context, a new contextualization mechanism was added to MCL's refinement stage. It was inspired by the approach to contextualization used in COPAC [81], but it takes certain liberties due to the inherent difference between behavioral contexts and memory contexts.

2. To account for inefficiencies in MF extraction, MCL was modified to deviate from TB's need to address each case separately. Instead, the case base was treated like a tree-based graph such that the memory influences from all cases in the case base could be extracted all at once.

3. To account for continuous data, a discretization module was added to MCL to convert continuous data into discretized data. The trace enhancement step was also modified to incorporate learned MFs back into continuous traces.

4. To address the simplicity of the basic value-back and time-back MFs, the MF refinement stage was modified to more intelligently use these MFs to express extracted memory influences. First, the specialization of raw memory influences as value-back or time-back was transferred from the extraction stage to the refinement stage. Second, the refinement stage was modified to use "combination counts" as part of a greedy approach to iteratively choose specialized MFs that comprehensively express the raw memory influences that were extracted for each case in the case base.

5. To address the effects of stochastic, random, and inconsistent behavior, the MF refinement stage of MCL was modified to analyze *sets* of extracted MFs within memory contexts instead of analyzing MFs individually. The intuition behind this change is that it is easy for

378

individual MFs to appear repeatedly in random behavior, but it is much harder for these MFs to repeatedly appear together as a set.

In Chapter 8, we outlined a series of experiments to assess the effectiveness of MCLvN in addressing human behavior in a new lawn mower simulation (LMS). In the LMS, humans were tasked with controlling a lawn mower in a simulation to mow as many grass patches as possible while interacting with various simulation entities with memory-based behaviors. Our results showed that a human's behavior is strongly correlated with memory of their previous actions, as if the human's actions occur as a continuation of past intent. The end result of including this information in the learned MFs was an increase in learning performance for the majority of scenarios in the LMS for our human subjects.

9.2   Issues Encountered

Several unexpected challenges were encountered during the investigation of the dissertation hypothesis. A brief summary is presented here.

9.2.1   Test Domain Selection

After the validation of the MCLv0 approach in the vacuum cleaner domain, the first problem for the investigation was finding a suitable test domain for the final MCL implementation. The selected test domain had to fulfill three criteria:

1. The test domain had to involve human data.

2. The test domain had to involve continuously-valued features.

3. The test domain had to include memory-based behaviors.

There were several test domains at our disposal that fulfilled the first two criteria, but fulfilling the third criterion was difficult. To guarantee that memory was being used, we not only needed to know precisely when a human would be employing memory, but we had to know how memory would be used and what it would be influenced by, so that we could test for correctness in MCL. We ultimately decided that a custom simulation was required to be able to control which memory-based behaviors would be used and when.

The idea of a lawn mower came from a desire to use a test domain similar to the vacuum cleaner domain, but more advanced and catered to the needs of the research. Similar to a vacuum cleaner, a lawn mower is viewed top-down and it moves about an environment encountering consumable objects and impassable obstacles. However, grass objects can used to obscure objects, such that a human controlling the mower would have to remember where those objects are and react appropriately. The decision to use a custom simulation was a wise choice in that we had complete control over the memory-based entities that a human would interact with and we could modify and add whatever was required for the research. However, our choice produced two issues:

1. The use of an external game engine (i.e. Unity) allowed us to automate various aspects of simulation development, but such aspects of the game engine were largely inaccessible for modification.

2. The concept of a lawn mower provided intuitive support behind some of the memory-based behaviors in the simulation because they were derived from real human experience. However, remaining completely faithful to the realm of what would be possible for a real lawn mower would have limited the types of memory-based behaviors that could be tested.

The first issue was mitigated through extensive research into the documentation behind Unity. The

second issue was addressed by allowing the lawn mower simulation to define its own rules, entities, and mechanisms. The objective was not to recreate a faithful rendition of a lawn mowing experience, but rather to develop an environment where memory-based behaviors could be exhibited.

9.2.2 Instructing Human Subjects

As the test domain was being developed, another problem that arose was how to teach a human subject how to operate the lawn mower and how to react to the entities in the environment. Possible solutions included simply demonstrating the simulation mechanics to a new human subject, or incorporating mechanisms into the simulation itself that would help the human subject figure it out themselves. The latter choice was selected for the sake of expediency during testing. This was accomplished through the incorporation of "soft tutorials" in the simulation, or mechanisms that would not allow a human to progress unless they demonstrated understanding of how to perform the desired behavior. In the LMS, soft tutorials were incorporated into the layout of the different scenarios combined with a timer and health points. In short, a human could not pass a scenario unless they performed the desired memory-influenced behavior correctly. Additionally, description screens explaining in text how new memory-based entities worked were also provided.

However, the following unintended consequences ensued as a result of this decision:

1. Many human subjects simply choose not to read the directions, either due to impatience or the assumption that they could figure it out on their own. The result was that several participants failed their first attempt at a scenario and had to redo the scenario.

2. If a human redid a scenario, their behavior was likely affected by what they remembered from their previous attempts, possibly confounding how they used memory during the attempt during which they were successful.

381

3. The possibility of "failing" was a possible source of stress for the human participants, which may have affected their behavior. On one hand, such stress is desired in that it may cause the human to behave slightly erratically, an inherent part of human behavior that we wanted MCL to overcome. On the other hand, the human's focus was inevitably on "passing" the scenario rather than demonstrating the memory-influenced behaviors. The simulation mechanics were designed to make it impossible for a human to pass a scenario without demonstrating the desired behavior, but sometimes a human would deliberately encounter hazards on their way to the scenario's exit point if they had sufficient health points. This was especially evident during the attempt where a human was given infinite health (after they had exhausted their attempts and were essentially given a consolation prize for being unable to complete the scenario successfully).

Another ill-advised decision during the investigation was the decision to solicit human participants for the lawn mower study before validating MCL's capability for processing that data beforehand. On one hand, a test domain should not be catered to the capabilities of the object of evaluation because failure should be a possible outcome. However, a test domain that does not test the expected attributes of the object of evaluation cannot render a definitive verdict of success or failure. In particular, even though the concepts behind the memory-based entities in the LMS were sound, their placement in a scenario can affect to what degree a human uses memory when interacting with them. Further testing could have ensured that the scenario layouts were optimal.

9.2.3 Incorporating Context

Regarding the development of MCL itself, there were several issues that had to be addressed. The first issue was how to incorporate context into MCL, which would comprise a completely new module. The motivation behind including context was to provide a mechanism for capturing rare

memory influences while filtering raw MFs that are the product of random behavior. The Context-Based Reasoning (CxBR) paradigm was the best choice, given its basis in tactical behavior and the heavy precedent it had within the LfO literature. With this determination, we had to address two questions:

1. How should contexts be defined within the scope of modeling memory from observation?

2. How should such contexts be learned?

9.2.3.1 Defining Memory Contexts

To answer the first question, we defined a *memory* context as

> Any recognized situation defined by a set of similar memory influences that represent the relationships between past observations and present actions by an observed actor within the situation.

Essentially, a memory context is "a situation wherein a common set of memory influences are relevant to a behavior". As long as the memory influences belonging to a given context are relevant to an agent a given point in time, the agent can be said to be in that context.

To answer the second question, we turned to COPAC, the first known context discovery algorithm for the CxBR paradigm. COPAC first partitions a trace into executed contexts, which are instances of context activation. Then, these instances are clustered to create context definitions. However, the contexts that were discovered by COPAC were defined by similarity of behavior within a given situation, not by which memory influences were at work. Furthermore, the executed contexts are created by finding points in a trace where behaviors change significantly in response to environmental stimuli.

383

We first tried to partition a trace based on drastic changes in the raw MF sets that were extracted for two cases representing adjacent time steps. However, the problem with this approach was that raw MFs vary greatly, and many raw MFs may actually be manifestations of the same specialized MF, which represents the true memory influence we wish to capture. Thus, it made sense to partition traces in places where specialized MF sets that were relevant to adjacent time steps were significantly different.

However, choosing which specialized MF type (value-back or time-back) was the best for representing a raw MF had only been superficially addressed in MCLv0. Basically, the best specialized MF type was chosen based on which type reduced the number of Memory TB iterations in the MF Extraction phase, but if there was a tie, then both types were chosen in order to err on the side of including too many MFs instead of too few. Testing both types of MFs was computationally expensive, which inspired the Memory Singularity Principle (MSP). MSP extracts raw MFs by comparing trace elements (raw MFs) directly instead of creating specialized MFs for comparison. However, since MSP only works on raw MFs, the decision was made to delay the generation of appropriate specialized MFs to the MF Refinement stage.

9.2.3.2   Capturing Correlated Memory Influence

A parallel problem that was being addressed at the time was that of dealing with memory influences that are correlated with other memory influences. For example, the issue of defining the value of one feature immediately after the observance of some other value of another feature seemingly required more complex MFs than just value-back or time-back MFs. Two possible solutions emerged – compound MFs and MF sets. Compound MFs explicitly combine atomic MFs (e.g. time-back or value-back) so that the compound MF only activates when its component MFs both activate. However, an easier solution was that of MF sets – only include the MFs in the MF set if all of

384

them are relevant. The intuition behind this decision was that an MF is only useful if all MFs upon which it depends are present. Interestingly, this had the effect of effectively filtering MFs that were the result of random behavior, because randomly generated MFs don't occur together repeatedly.

The concept of a MF set also impacted the contextualization process in MCL. Because memory-based contexts are inherently different from contexts, as defined in CxBR or the work in COPAC, it became permissible for memory-based contexts to be defined and discovered in a slightly different way. Thus, we created a working context structure: a context and a MF set are one and the same. The ramifications of this decision were as follows:

1. MF sets may only be relevant for an instant. Thus, the temporal nature of behavioral contexts from CxBR no longer applied to our memory contexts.

2. MF sets lose effectiveness when a subset of the MFs are missing. Thus, MF sets are either completely similarly or completely different. This eliminated the need for the context comparison mechanisms that were used by COPAC.

3. Context thus became intertwined with the MF Refinement module inherently and no longer required a separate module. Instead, it became a conceptual construct by which we could understand and articulate the transformation of memory influences over time.

With the definition for memory contexts that was used in this research, there are two positive effects of context for memory modeling in MCLvN:

1. The evaluation of MFs as a group (i.e. memory context) instead of individually made it less likely for MFs to be retained if they were generated as a result of random behavior instead of true memory influences. This benefit applies to all LfO algorithms, even those that are context-free.

385

2. For context-based LfO, contexts allow LfO for more specialized situations to focus on a small set of MFs that are relevant for the given situation, thus reducing the learning burden for each individual context by ignoring MFs not relevant to the context. Furthermore, the use of contexts can help identify which specialized MFs are common across multiple *contexts* rather that just those that appear in a sheer number of *time steps*. In our experiments, we use the F1 score instead of accuracy to evaluate whether all actions, no matter how rare, can be predicted correctly, instead of just the most common action. In the same way, contexts can show which MFs are common across multiple situations, rather just those that appear in the most common memory-based situation.

9.2.4   Considerations for Continuous Data

In order to handle continuous data, we had two choices. We could either redesign the components of MCLv0 to handle both continuous and discrete data or we could process data to be usable in discretized form by MCL's components as they currently were. In order words, we could either change MCL to properly process continuous data or we could change continuous data to be processed by MCL. We chose the latter option, as a simple discretization module would allow us to modify MCL to address the other needs of the research without regard to data types. However, this caused some issues:

1. Discretization requires tuning hyperparameters of its own. Improper discretization could have impacted the results of MCL. Garbage in, garbage out.

2. Discretization affects the complexity of the memory modeling task and it affects the degree to which memory can be modeled. On one hand, having finely discretized data (e.g. many possible discrete values) allows a wide variety of memory influences to be expressed, but it also increases the burden on MCL because MCL will have to parse through more raw MFs

386

to find appropriate refined MFs. On the other hand, coarsely grained discretized data can be more easily parsed by MCL, but it risks the possibility of losing precision in expressing memory influences.

It might have been better to investigate the development of new MFs that can process continuous data, perhaps by defining a range of values that the MF would account for (instead of just a single discrete value). Furthermore, future work could look into using different distributions for discretizing data and investigating the sensitivity of memory to different discretization hyperparameter settings.

9.2.5 *Memory Feature Extraction Hyperparameter Tuning*

Another issue was how to properly tune MF Extraction hyperparameters. In particular, the PPT and PAT thresholds have profound effect on extraction results, and yet it is unknown what their precise values should be for a given domain. The problem is that if a threshold is too high, it will result in an unnecessarily branching factor during tree extraction, creating too many extraneous raw MFs. However, if the threshold is too low, then crucial raw MFs may not be generated. To tackle this issue, we employed two solutions:

- Instead of exhaustively testing all possible PPT/PAT threshold values, we only tested the three used in Floyd's research [20]. He tested PPT=0.0/PAT=0.9, PPT=0.9/PAT=0.0, and PPT=0.9, PAT=0.9 to address action-based, state-based, and state/action-based behaviors.

- If we err on the side of having PPT/PAT thresholds that are too high, we can filter out extraneous MFs through MF Refinement filtering thresholds. Basically, we have a Low Filtering (LF) threshold, which determines which memory contexts are allowed to contribute to the final refined set. However, the High Filtering (HF) threshold determines which MFs

387

from these sets are included in the final refined set. If an MF belongs to a MF set that passes the LF threshold, but itself does not pass the HF threshold, then it is not included. Even though this goes against the principle that a MF set/memory context is only effective if all its component MFs are used, it is a compromise that allows more flexibility in setting appropriate PPT/PAT thresholds.

9.2.6 Feature Imbalance

Finally, the last issue that we encountered was feature imbalance, which comes in two forms: perception/action imbalance and nominal/numeric imbalance. In short, if there are more perception features than action features (or vice versa), then it may be more difficult to generate perception MFs than action MFs. This would make MCL potentially overcompensate by learning extraneous action MFs to try to explain memory influences from perception MFs. Regarding nominal/numeric imbalance, it takes more time in a real time environment for numeric features to change in value sufficiently to warrant the creation of a raw MF during MF Extraction, but any change in a nominal feature's value will generate a raw MF. Therefore, it is more likely for nominal features to generate raw MFs than numeric features.

To mitigate these imbalances, we downsampled our traces in the LMS domain to allow numeric features to change more in value from time step to time step. Additionally, we experimented with hybrid traces, which allow perception and action MFs to be learned separately in two different MCL operations before combining them into a single memory-enhanced hybrid trace.

However, these countermeasures were only partially effective. Down-sampling a trace risks losing important information about memory from time steps that are removed from a trace. Furthermore, hybrid traces do not allow perception and action MFs to be generated together via the MSP or to be joined together in MF sets, which are the working definition for memory contexts. Future work

could investigate the possibility of creating raw MFs for multiple perception features, to overcome the perception/action imbalance.

9.3   Conclusions

The research described in this dissertation looked at a possible means of solving the general problem of incorporating the influences of memory into the LfO task. The specific problem addressed by this research was investigating how to learn a memory model of observed behavior that avoids limitations of prior approaches to memory-based learning. The novel technique proposed by this research is MCL, a memory modeling technique that learns a set of MFs that are tailored to specific observed actors. Such MFs summarize pertinent information from an actor's history that are important for future decisions. These MFs are usable by other standard supervised learning algorithms to allow them to learn memory-based Level 3 behaviors when they would otherwise be limited to learning Level 2 reactive behaviors.

This research is novel in that it differs from several prior approaches to memory-based learning. This research differs from lazy learning approaches to memory-based learning in that a model of an actor's internal state is maintained over time. In lazy learning approaches, such as the work in [20], the effects of memory must be inferred for every decision instead of maintaining an internal state. This research differs from probabilistic approaches to memory-based learning in that the possible internal state values are not predefined. In contrast to works that limit the influence of memory to a finite number of time steps in the past [76] or that predefine the number of possible internal states [58], MCL makes no assumption about the limits or possible values of the internal state, but rather composes the internal state (as influenced by memory) from a minimal set of MFs that are needed to resolve influences of memory that are observed in an actor's behavior. Finally, this research differs from connectionist approaches to machine learning, such as MANNs [64] or LSTMs [26].

Connectionist approaches ultimately consolidate memory influences into a subsymbolic structure that is indecipherable outside of the network. In contrast, MCL learns MFs that are more human-understandable and which can be coupled with other supervised learning algorithms for enhanced performance. The MFs learned by MCL could even be combined with these memory-based LfO algorithms by lessening the burden of learning from memory.

This research was difficult for several reasons. The main challenge was that it was not possible to compare the MFs learned by MCL to any ground truth, because the cognitive processes of humans are unobservable. This makes memory modeling an unsupervised learning problem. The validity of the technique thus had to be evaluated through human inspection of MFs and through improvements in learning performance. The second challenge was that the effectiveness of MCL had to be evaluated independent of the learning algorithm that learned from the memory-enhanced traces it produced. Thus, a general-purpose memory model had to be created by MCL that was usable by a variety of learning algorithms. The final challenge came from the human behaviors themselves, which are inherently inconsistent in their use of memory. Thus, it could not be assumed that memory influences remained constant throughout an actor's performance.

However, the impact and benefit of this research is twofold. First, the benefit to LfO research is that we have proposed a technique for consolidating memory influences into a concise set of MFs comprising the internal state (as influenced by memory) rather than having to save the entire history of an agent's run or having to predefine the possible internal state values or structure. Second, the benefit to machine learning in general is that MCL can be combined with any standard supervised learning algorithm (without modifying the algorithm itself) because the learned MFs can be treated like any other feature. Because the MFs summarize the relevant information from memory, including these MFs restores the Markov assumption that all pertinent information needed for a decision at a given time step is present in the feature values for that time step. This will allow supervised learning algorithms, which rely on the Markov assumption (also known as the i.i.d.

assumption) to learn all three levels of LfO instead of being limited to just Levels 1-2.

In conclusion, the hypothesis from Chapter 3 has been investigated and validated by this dissertation. The main contribution of this research is the novel MCL algorithm. MCL automatically learns the memory influences of observed behavior and codifies them with a set of MFs, which can create memory-enhanced traces that possess relevant information from memory for each time step. This research expands the set of behaviors that can be learned by machine learning techniques and provides a means toward shedding insight into the effects of unobservable phenomena (e.g. memory) on human behavior.

9.4   Future Research

Even though the research in this dissertation was sufficient for validating the hypothesis, there are still several ways to improve or extend this research.

9.4.1   Algorithmic Improvements

This section describes modifications to MCL that can be investigated by future research to see if such modifications are more effective in modeling memory influences on observed behavior.

9.4.1.1   Combining Extraction and Refinement

Both MCLv0 and MCLvN follow a extract-refine-enhance procedure, but there is no reason why future iterations of MCL could not deviate from this. Thus, the first modification would be to combine the extraction and refinement stages of MCL. In Chapter 6, we mentioned that the extraction process in MCLvN was analogous to traversing a tree-based graph. However, it might be insight-

391

ful for future research to actually create a tree-based graph structure, where each location in the tree where a split occurs represents a memory influence that separates some cases in the case base from other cases with a different memory influence at that point in their run. With such a tree like structure, the relationship between the different locations in the tree with splits could be used to inform which specialized MFs are created.
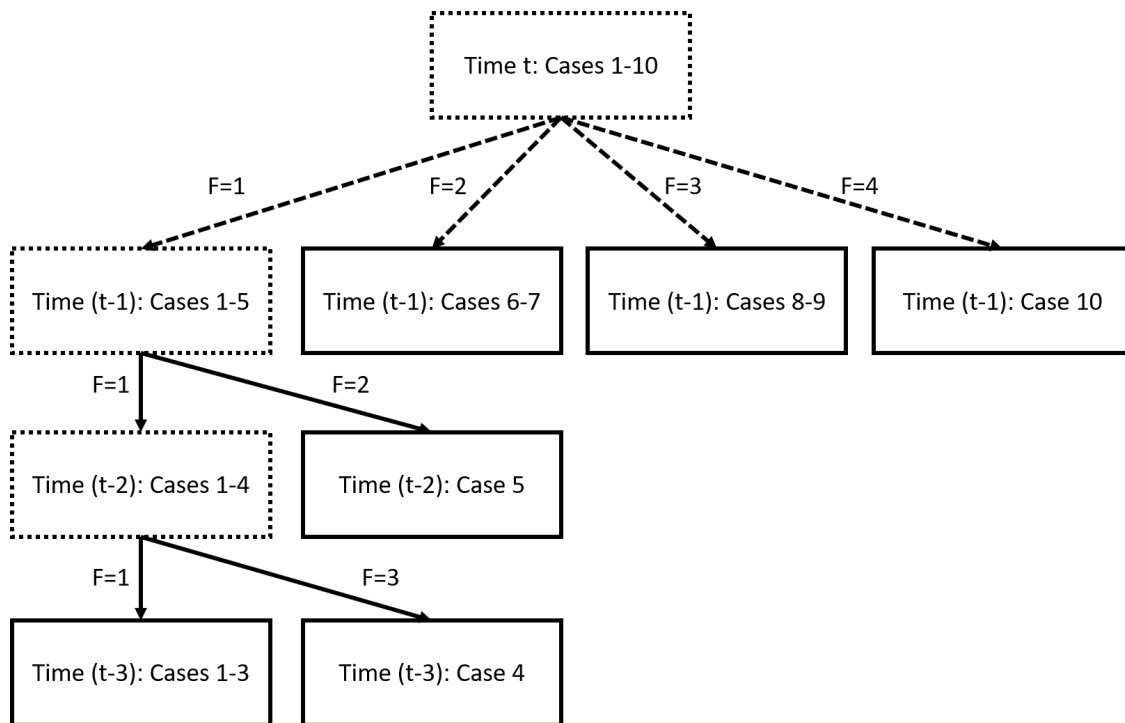
Figure 9.1: Analyzing Memory Features with Tree Structure

An example of this tree like process is in Figure 9.1. In the figure, we begin with 10 cases that have not been differentiated from each other. Suppose these cases only track a single feature F which has four possible values. At time $t-1$, we can see that there are four groups formed, one for each possible value of F. These groups are formed by cases 1-5, 6-7, 8-9, and 10, respectively. At times $t-2$ and $t-3$, cases 1-5 (from the first group) are further split based on the value of feature F. From this simple example, we can see that a value-back MF $vF(1)$ (the value of F at time $t-1$)

392

can be formed because there are multiple tree splits at the first node, denoted by the dashed arrows from the root node of the tree. We can also see that a time-back MF $tF(F = 1)$ (the time back to the last occurrence of $F = 1$) should be formed, based on how many time steps this value is observed, as evidenced by the boxes in the figure with a dashed outline (these are where the splits occur where $F = 1$ is one of the splits).

9.4.1.2   *Alternative Discretization Distributions*

Another possible improvement would be the use of different distributions to use for discretization. We opted to discretize our traces to keep the inner mechanisms for MCL the same for both discrete and continuous data. Our implementation of discretization requires the developer to specify the center values and discretizing increment for each feature, which requires knowledge of the domain. However, it may be more meaningful to discretize the traces using an empirically derived Gaussian distribution. This would involve empirically determining the mean and standard deviation of each feature in the trace. The advantage of using a Gaussian distribution would be that discretization would be an automatic process that is not reliant on domain knowledge. However, the disadvantage is that discretization results would differ between individuals; a value of 1 for a given feature may not be the same for different observed actors. Future research could investigate the efficacy of both approaches to discretization, or other methods of discretization in the literature. Along the lines of processing continuous data with MCL, future work might investigate the efficacy of processing continuous data directly instead of discretizing it. This would require significant alterations to the MCL modules described in this dissertation, but the end product, memory-enhanced traces, would still remain the output of MCL.

9.4.1.3 Improved Memory Feature Specialization

One avenue for potential improvement to MF Refinement would be investigating the use of dynamic programming to select specialized MFs that express a collection of raw MFs. The approach described in this dissertation uses a greedy approach to select the next specialized MF that expresses the most remaining raw MFs. However, it may be more optimal to select two specialized MFs that are less expressive, but more effective together than the specialized MF that would otherwise be selected via the greedy approach.

9.4.1.4 Incorporating Parallelism

The extraction phase in MCLvN analyzes all cases in the case base simultaneously for memory influences, but in MCLv0, all cases are processed mostly independently of each other. Both versions of extraction theoretically have the same result, with MCLvN being much more computationally efficient. However, it may be possible to leverage hardware parallelism to achieve even greater efficiency with a hybrid version of MCLv0 and MCLvN.

9.4.1.5 Feature Discrimination in Memory Feature Extraction

During the extraction phase of MCLvN, one raw MF is created every time there are cases that are removed from the pool of candidate cases that are similar to a test case. This raw MF tracks a single feature, depending on which feature was the most discriminative between cases retained in the pool and cases removed from the pool. However, it's possible that multiple features are exert a memory influence at that time step. Future work could see if MCL's modeling efficacy is increased when creating multiple raw MFs instead of just one. For example, raw MFs could be created for all features with sufficient discriminative power (instead of just the one feature with the highest

394

discriminative power).

Additionally, the "discriminative" power of a feature is computed via a relatively simple metric, which could be replaced by a more complex metric that also accounts for other factors. For example, certain features may be deemed more important inherently by a domain expert. Also, a feature that was already used for a raw MF in an earlier time step may be less important than a discriminative feature that has not yet been incorporated into a raw MF (or vice versa). Additional factors for feature discrimination in order to determine the composition of a raw MF during the extraction phase could be investigated in future work.

9.4.1.6 Alternative Casebase Representations

Future work could also investigate alternative representations for the case base. Temporal Backtracking was case-based, but it may be possible to represent such cases with structures such as matrices or tree-based graphs. Matrices would allow cases to be processed with computationally and space efficient operations, and possibly connectionist methods (such as deep neural networks).

9.4.1.7 Automatic Hyperparameter Tuning

Another subject of future work might involve the investigation of possible mechanisms for tuning MCL's hyperparameters, particularly those involving case comparisons in the extraction phase and those involving the formation of memory-based contexts during contextualization. In essence, these hyperparameters influence how likely it is for cases to be "similar" to each other and how likely it is for memory contexts to used to contribute specialized MFs to the final refined MF set. These parameters were manually tuned in this research, but this process could be done automatically. This would be an important extension to the work done in this dissertation because memory

is unobservable. The desired outputs of MCL cannot be verified by human inspection in very complex domains, so an automatic means for tuning these hyperparameters is necessary to evaluate MCL outputs based on criteria other than final learning performance.

9.4.2 More Complex Memory Features

This research was conducted using a combination of value-back and time-back MFs. However, it's possible that more complex MFs may more concisely and intuitively codify memory influences of observed behavior. In this section, we discuss some possible MF extensions for future research.

One modification to the time-back MF could be to add a "skip" parameter that dictates how many observations of a specific feature value should be ignored. For example, the time-back MF described in this research will only report the time back to the latest observation of a given feature value. However, some memory influences could be expressed as two feature values happening back-to-back (e.g. two "up" actions by the observed actor). It would not be possible to track the time back to the second-to-last "up" action with the time-back MF described in this research. However, we could track it with a modified time-back MF that has a skip parameter set to the value 1; this time-back MF would ignore the latest "up" action and instead report the time back to the second-to-last "up" action observed. This type of MF would be useful for expressing memory influences where the same value is repeatedly observed in a trace (e.g. a car skidding to a stop over a period of a few seconds).

Durative MFs might be a meaningful type of basic MF that tracks how long a specific feature value has been observed. Certain behaviors might be predicated on how long a certain event has transpired. With only time-back and value-back MFs, this memory influence could only be expressed with time-back MFs for every other possible value of the feature, which track essentially how long a given feature has *not* assumed any other value except the one in question. A durative

396

MF may be a more intuitive way to express this memory influence.

Last-value MFs might be another meaningful type of basic MF. This MF tracks the last observed value, excluding a certain default value. For example, in the car driving domain, a driver (hopefully) adheres to the value of the last speed limit sign they observed. The default value for an observed speed limit is N/A, given that a speed limit sign is often not observed. However, a last-value MF would remember the last non-N/A value of the speed limit sign, which is what we care about and what other memory influences might also require.

So far, we have only described basic or "atomic" MFs, but it's possible to have compound MFs whose constituent components are also MFs. One example is memory bigrams. Bigrams are features that are frequently used in natural language processing to track two specific words that occur together. In the same way, it may be useful to track two basic MFs that occur in adjacent time steps with a single memory bigram. Memory bigrams could be used to express memory influences such as "the value of the last action when the last value of feature F was value v". In keeping with the insight for MCLvN refinement that meaningful MFs occur together rather than separately in a given time step, it makes sense that meaningful MFs would also occur together in adjacent time steps. It might also be useful to have memory trigrams (tracking three MFs that occur together in adjacent time steps).

Finally, "memory of memories" or "recursive memories" may be a useful avenue for future investigation. Allowing MFs to track the value of other previously learned MFs, or allowing MFs to be parameterized with the value of other MFs, may allow certain memory influences to be expressed more intuitively. For example, the value of a feature F after the last observance of value V for feature G could be expressed by a compound MF `vF(`t`-tG(V)+1)` or "the value of F in the time step before t that immediately follows the last observance of value V for feature G".

Any future work in the creation of new MFs should maintain a modular design such that MCL

does not need to be modified each time a new MF is created. However, it might be necessary to modify MCL to handle compound MFs and other types of MFs that are parameterized with factors other than time values and feature values.

9.4.3 Behavioral Analysis

One interesting future work from this dissertation research would be to analyze the learned memory models created by MCL. This could serve two purposes. First, memory models can be used to detect abnormal behavior from observation [88]. Basically, in order to determine whether someone appears to be using memory according to one condition versus another, one can create memory models of both conditions and then compare a person's memory usage to both memory models to see which one more closely matches the person's memory usage. This would be useful for determining whether someone is paying attention to key aspects of the environment. Perhaps as a person continues to perform a task, fatigue sets in and a person stops responding to certain environmental stimuli (e.g. memories of past events are no longer affecting the person's decisions). A system that notices that memory usage has changed from what is acceptable could be used to flag dangerous operating conditions for the performing person.

In order to determine whether an observed actor performing in real-time is acting abnormally, it may be useful to create online memory analysis techniques. One way to do this would be to train machine learning classifiers by using MCL. During training, the classifier would learn from MCL which MFs are useful for detecting abnormal behavior. Once trained, the classifier can be used in real-time to detect whether someone's use of memory appears to indicate danger to others (e.g. perhaps the person has stopped paying attention to crucial aspects of the environment or is not following the correct sequence of actions).

Another possible extension of this research would be to investigate aspects of certain tasks that are

398

common between different actors versus those aspects that vary between different actors. MCL learns memory models of observed behavior that are specific to particular observed actors; the learned memory model will only apply to the actor that was observed. However, if multiple memory models are learned for different actors, those memory models could be analyzed to see which MFs are common between actors and which MFs differ. For example, in the car driving domain, one might observe that the majority of drivers respond to the memory of the speed limit sign, but that certain drivers forget about the speed limit sign and start matching the speed of other surrounding cars sooner than other drivers. These insights could be the key to creating generic behavior models that model task-specific procedural knowledge and then adapting them to specific observed actors by only modifying those aspects of the model that are known to vary greatly across different persons.

9.4.4   Informing Transfer Learning

Transfer learning [59] allows a learning task for a given target task in a target domain to capitalize on learning results in a related source domain and task. Transfer learning is useful in situations where labeled data in a task/domain of interest is scarce, but it is more plentiful in a different but related task/domain. For example, a transfer learning algorithm can learn how to detect bunnies in online videos by learning to detect cats in online videos (which are more common). By learning to detect the friendly fuzzy features of felines, a learner can learn to detect the similarly endearing attributes of bunnies in videos.

In the context of transfer learning, as surveyed in [59], a *domain* is the space of inputs available for learning and a *task* is the space outputs that must be determined by learning, given a classification problem. Two domains are different if the feature sets differ or if the distributions of input samples are different. Two tasks are different if the set of classes differs or if the distribution over the

classes is different.

There are three types of transfer learning, briefly summarized from [59] as follows:

- *Inductive transfer learning* allows for learning for one task to benefit learning for a different task, whether or not the source and target domains are the same.

- *Transductive transfer learning* involves the source and target tasks being similar, but the source and target domains being different.

- *Unsupervised transfer learning* involves the improvement of unsupervised learning for a target task via learning for a different source task.

A special case of inductive transfer learning, called "self-taught learning" [61], involves learning for a target task by using *unlabeled* data from a potentially unrelated source domain. Raina et. al. [61] compare self-taught learning with other types of learning as follows:

- *Supervised learning* solves a target task by learning from labeled data from a source domain that is the same as the target domain.

- *Semi-supervised learning* solves a target task by learning from both labeled and unlabeled data from a source domain that is the same as the target domain.

- *Transfer learning* solves a target task by learning a different but related source task for labeled data from a source domain that is related but different from the target domain.

- *Self-taught learning* solves a target task by learning a different source task using unlabeled data from a potentially unrelated source domain.

In the realm of LfO, transfer learning could be applied in two ways:

1. *Application 1*: Given relatively many traces of a *source* person's observed behavior, and given a relatively fewer number of traces of a *target* person's observed behavior, use transfer learning to learn how to perform a task similar to the target person.

2. *Application 2*: Given a person's relatively many traces of observed behavior in one domain, learn how to behave in manner similar to this person in a different but related domain with fewer example traces of this person's behavior.

Regarding the first application of transfer learning to LfO, it may be difficult to obtain many traces or examples of a specific person performing a given task. This is particularly problematic if the task involves behavior that merely requires satisficing, or behaviors that "satisfy" requirements by picking the next available option that "suffices". In other words, a person may elect to pursue a different sequence of actions each time they are presented with the same set of initial conditions. If a person is only observed performing a task once, then it is not possible to learn acceptable variations of their behavior in specific situations.

However, it can be possible to learn reasonable variations in behavior through transfer learning if there are examples of another person performing the same task. The target and source domains are different because different people (who inherently perform differently) are being observed. However, transfer learning is possible because the tasks are the same. This is an example of transductive transfer learning.

Regarding the second application of transfer learning to LfO, it may be possible to model how a specific person operates in different environments. For example, a soldier be trained many times to perform a search-and-rescue operation within an indoor environment. However, an after-action review may be conducted once on the soldier's execution of this same operation within an outdoor forest environment. For a LfO application to model the soldier's behavior in the forest environment (which only occurs once), it may be beneficial to use transfer learning to learn a model of the

soldier's behavior within the indoor environment (for which there are many examples) in order to learn a behavior model for the forest environment. One would expect the soldier to exhibit similar behaviors between both environments. This is an example of inductive transfer learning.

If observations of many different soldiers performing various navigation operations were available (e.g. retreating, supply gathering, moving through a crowd, etc.), then self-taught learning could be executed to learn general human navigation techniques, which would inform LfO for a given soldier for a given operation.

The application of MCL to these instances of using transfer learning to improve LfO depends on how the feature sets are structured. In transfer learning and even self-taught learning, it is assumed that all learning data comes from the same modality (e.g. images, audio, etc.). However, MCL was designed for LfO tasks with domain-specific features.

The potential for MCL to aid in the transfer learning task lies in the fact that the internal state memory models that are created for specific observed actors' behaviors are comprised of composable MFs. It is possible for different actors to share several MFs in common when performing the same task. Thus, the first application of transfer learning to LfO can benefit from MCL as MCL currently exists. This is because the environment and actor's actions share the same set of features — it is only the distribution of values for these features that differs between actors.

However, the ability to use MCL in the second application of transfer learning to LfO depends on how the source and target domains are described. Even though the source task and target task are being performed by the same observed actor, a different set of features may be used to describe the actor's actions (and perceptions of the environment) in the source domain versus the target domain. MCL can still be used in this situation as long as there is at least one feature in common between the domains. This is because MCL forms MFs that codify information for a given feature, and thus MFs that are learned for any features that are used in both domains can be used by transfer

learning. For example, if the source domain uses features A, B, C, and D, and the target domain uses features C, D, E, and F, then transfer learning can use MFs based on features C and D.

The applicability of MCL to the second application of transfer learning to LfO could be extended by including information about which features in the source domain can be treated like which features from the target domain. For example, if certain entities in the source domain are similar to other types of entities in the target domain, then memory influences based on an entity in the source domain can be applied to the corresponding entity in the target domain.

9.4.5   *High-Density Applications*

High-density applications involve numerous feature values that are only meaningful to the degree to which the aggregate collection of their values holds meaning in a more abstract space. For example, in image processing, individual pixel values are not meaningful in most applications, but the detection of objects in such images or activities in steams of images (e.g. video streams) are meaningful. Even though specific objects and actions may be repeated when viewing an image as a whole, individual pixel values are not likely to repeat in a predictable manner.

MCL can only be applied if values repeat often enough to be sufficiently correlated with specific actions taken by an observed actor. This means that feature values should oscillate within a meaningful range (i.e. they should assume the same value multiple times) and coincide with repeated actor actions. Otherwise, MCL will assume the observed behavior is random and no MFs will be learned.

In order for MCL to be used in high-density domains where the above assumptions do not hold, higher-level features will have to be learned by a separate module. This requirement is not specific to MCL. In general, various learning "tasks... require the construction of good internal represen-

tations of the world (or 'features'), which must be invariant to irrelevant variations of the input while, preserving relevant information" [42]. For example, two different people will write the character 'e' in different ways, but but both written instances should still register as the letter 'e' to a computer system [41].
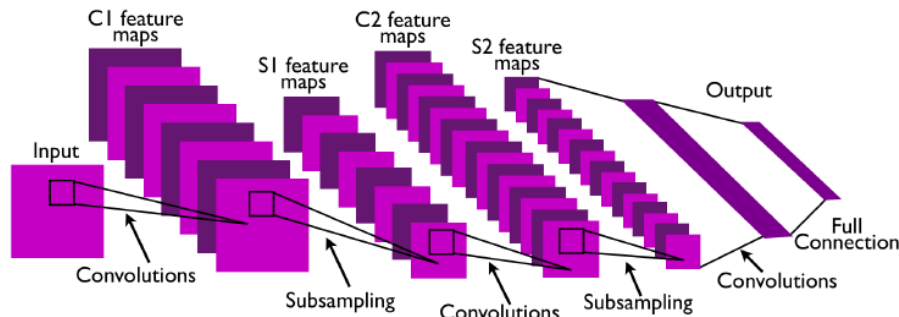
Figure 9.2: Convolutional Network. Reprinted with permission from [42].

Convolutional neural networks (CNNs) [41] can be used to learn higher-level features from lower-level high-density features. Figure 9.2 shows an example CNN that takes as input a single image and outputs a classification decision (e.g. reading an image of a handwritten character and determining which character was written).

CNNs work by alternating between convolutional layers and sub-sampling layers. Convolutional layers transform inputs from the previous layer into higher-level outputs in a series of feature maps. Each output in a feature map represents the network's analysis of regions in the previous layer of the CNN. For example, the first convolutional layer that processes the input image itself essentially looks for a given aspect of an image in every region of the image. This allows the CNN to recognize patterns regardless of where they occur in the image. It also allows the CNN to learn

404

fewer parameters, since the same set of parameters for a given feature map are reused over multiple regions of an input. Sub-sampling layers compute averages over regions of the previous layer in order to reduce the number of inputs that are presented to the next convolutional layer.

The combination of a CNN and MCL could allow MCL to be used in high-density data applications, such as auditory and image processing. The CNN would be used to detect high-level features specified by a developer (such as the presence of cats in videos) and then MCL would used to determine how these features (as detected by the CNN) influenced decisions over time. For example, this could be used in activity detection in videos, wherein certain images of a person moving in sequence could be processed by MCL to learn that memory of certain human poses usually precede other human poses for a given activity. For example, a human usually crouches a bit before jumping and a human usually blinks and inhales before sneezing.

9.4.6   Informing General Decision Making

We define General Decision Making (GDM) informally as as the process of reasoning about current knowledge to achieve some goal. Gonzalez and Dankel [22] use the example of diagnosing problems in a lawn mower (coincidentally) to illustrate the difference between three types of knowledge:

- *General Knowledge* stores prior information about a domain that is applicable to various types of tasks.

- *Specific Knowledge* stores knowledge that is particular to a given problem. This knowledge can be modified as additional information about the problem is derived or learned.

- *Problem-Solving Knowledge* is a set of procedures that determine *how* the general and specific knowledge are used to derive a solution to the problem at hand. Problem-solving knowl-

edge is constant across domains.

The implementation of GDM within computer systems, as noted by Gonzalez and Dankel [22], can either be accomplished through traditional software engineering or through knowledge engineering. Software engineering uses traditional search algorithms to find a solution within some solution space to solve problems. This is essentially a general-purpose approach to solving problems and making decisions in any domain. However, more nuanced and domain-specific problems are better addressed with *knowledge-based* (KB) systems.

A KB system is defined in [22] as a "computerized system that uses knowledge about some domain to arrive at a solution to a problem from that domain. This solution is essentially the same as that concluded by a person knowledgeable about the domain of the problem when confronted with the same problem." KB systems are most suited for capturing *associational expertise*, wherein a blackbox system is understood solely by its effect on an environment [22].

LfO, as noted in Chapter 1, can be considered a means of *automatically* generating a KB system from observation of someone's performance. The blackbox in this example is the observed actor. This dissertation describes how MCL can approximate the influence of memory on the observed actor for more informed LfO. However, the next step is the analysis of memories of the performance a LfO agent (e.g. a KB system) with regards to some goal.

Gonzalez and Dankel [22] describe a case study surrounding a KB system called GenAID (Generator Artificial Intelligence Diagnostics). The purpose of GenAID was to use expert heuristics to analyze data from power generators and determine the presence of possible minor system flaws. These system flaws could then be addressed before they became major flaws that resulted in system downtime, which could cost a power company thousands of dollars per day.

MCL can be used in a KB system such as GenAID to improve GDM over time, regardless of

how the KB system was initially created. If certain decisions by a KB system result in suboptimal outcomes, then MCL can be used to assess which aspects of the KB system's heuristics should be adjusted. Thus, the iterative improvement of KB systems in this fashion becomes a long-term reinforcement learning problem – decisions by the KB system are graded over time as "good" or "not good" (either by some system metric or by a human).

Using GenAID as an example, suppose that GenAID fails to detect certain minor system flaws that ultimately result in a system failure (generator downtime). In order to determine how GenAID should be updated, MCL can be used to analyze the decisions of GenAID over the time period leading up to the system failure (the trace of GenAID's outputs for each perception of the system's state) to see which memory influences were missing from GenAID's analysis. Then, developers can update GenAID (or a modified version of GenAID can update itself) to pay more attention to such aspects of the system.

In this example, MCL has aided in a reinforcement task. Even though MCL was developed to work in tandem with LfO tasks, the output of MCL may be useful for a variety of learning tasks in addition to LfO.

9.4.7 A New Learning from Observation Algorithm

A new class of learning algorithms are needed to address Level 3 memory-influenced behaviors. Standard supervised learners are generally not suited for learning Level 3 memory-influenced behaviors [58]. In response, this dissertation presented MCL, an algorithm that learns information about memory influences (codified by MFs) that can improve LfO algorithms (even standard supervised learners) without modifying the LfO algorithm itself. However, MCL does not accomplish the LfO task by itself – it must be coupled with a preexisting LfO algorithm.

Thus, there are two ways in which MCL could be extended to create a new LfO algorithm. Specifically, this new LfO algorithm would construct the procedural knowledge of the memory contexts that are learned by MCLvN (as described in Chapter 6).

In general terms, MCL learns which memory-influences (codified by MFs in empty memory contexts) frequently occur with specific actions by the observed actor. However, the next logical step is to structure procedural knowledge that says, "Do actions Y when memory influences M are present." In other words, MCL (as described in this dissertation) learns that memory influences M are important in some way (leaving it up to another algorithm to learn how specifically); if MCL were extended, it could learn that memory influences M, when present, require the agent to do actions Y.

Chapter 6 discussed how a context-based LfO algorithm, such as GenCL [19], could benefit from the memory contexts learned by MCL. MCL would automatically learn appropriate memory contexts describing situations where certain memory influences are relevant. Then, the LfO algorithm would learn the action rules and transition rules for each memory context. However, there is a concern that memory contexts are not suited for current context-based LfO algorithms because memory contexts are inherently different from behavioral contexts, as defined in [81]. In the CxBR paradigm [24], procedural knowledge in a context is expected to involve very similar actions. Memory contexts do not necessarily conform to this assumption because the only constant for all actions in a memory context is that all actions can make use of the MFs forming that memory context.

The following procedure loosely describes how to learn *transition rules* within MCL's memory contexts:

1. Load the next memory context m and do the following:

(a) Create *positive* training examples from the values of MFs for m in the time steps belonging to memory context m.

(b) Create *negative* training examples from the values of MFs for m in all time steps *not* belonging to memory context m.

(c) Use symbolic supervised learning to learn the transition rule for m. "If values v are observed for MFs χ, then set memory context m as the active context."

2. If not all memory contexts have been processed, go to Step 1.

The following procedure loosely describes how to learn *action rules* within MCL's memory contexts:

1. Load the next memory context m and do the following:

(a) For each possible action a, do the following:

   i. Set all time steps within memory context m that have action a as a solution component as *positive* training examples. Set all other time steps in m as *negative* training examples.

   ii. Use symbolic supervised learning to learn an rule for action a. "If memory context m is active, then if features F have values V, then perform action a."

(b) Alternatively, use standard supervised learning over all time steps in memory context m.

2. If not all memory contexts have been processed, go to Step 1.

Thus, it is mostly the *transition rules* which would have to be specifically addressed by a new LfO technique integrated with MCL. Memory contexts are defined specifically by memory influences,

so we must ensure that the value of MFs specifically determine which memory context is active. Because MCL attempts to restore the Markov assumption, we assume individual time steps can be treated independently.

Additionally, it may be useful to combine both behavioral contexts and memory contexts by either learning memory contexts within a given behavioral context or learning behavioral contexts within a given memory context. Both types of contexts could even be combined to create a hybrid context definition based on both behaviors within the context and on relevant memory influences.

9.4.8   Integration with Feature Construction

Feature construction [67] is the modification of an *original* feature set into a *new* feature set that either more concisely models an environment or improves prediction accuracy. According to the survey by Sondhi [67], the motivation behind feature construction are the twin problems of feature interaction (removing features that are correlated with each other) and feature irrelevance (removing features that are unhelpful for a prediction task). The latter can be addressed via feature selection (choosing a subset of the original feature set as the new feature set). The former can be addressed by constructing features that are a combination of features from the original set; these new features thus embody any correlative relationships between those component features.

The common thread between feature construction and MCL is that both methods create new features that improve learning performance. The difference between feature construction and MCL is that feature construction generates new features based on the utility of features as distinct discriminators – features that improve information gain. In contrast, MCL constructs features that embody memory influences – relationships of specific feature values with the current time step that are correlated with an observed actor's decisions.

It may be possible to investigate the use of various feature construction methods to improve MCL or to implement alternative approaches to the modeling memory influences. For example, MFs learned by MCL only track a single feature, but feature construction methods could be used to create MFs that model memory influences from multiple features simultaneously.

9.4.9   Proving Memory Coverage

This dissertation investigated the feasibility of the MCL approach toward modeling memory influences for improving LfO. However, a limitation of this work is that there was no guarantee that the learned memory model would improve LfO – it could only provide additional inputs that had potential for improving learning performance. Thus, the degree to which learning improved could be a factor of either MCL or the LfO algorithm itself.

Therefore, a fruitful venture for extending this research would be to develop methods for answering the question, "Given a set of memory feature types (e.g. value-back, time-back, etc.), to what degree can all possible memory-influenced behaviors involving feature set F be expressed?"

To answer this question, one would have to define the set of all possible memory-influenced behaviors for a given feature set and one would have to define all possible expressions of memory-influences for a given memory feature set.

The set of all memory-influenced behaviors could be defined in a brute force manner by enumerating the power set of all features, values, and temporal constants for every time step (up to a maximum constant T). Or, one could define a distribution of relevant internal states according to some empirical data (e.g. a Monte Carlo simulation). The set of all expressions of memory influences, given a memory feature set, could be defined in a similar manner. The question then becomes how to compare the set of possible memory-influenced behaviors with the set of mem-

ory feature expressions. Such an endeavor would likely require an algebraic proof involving the notation of stochastic processes, as in [58].

9.5   Afterword

The formulation of new approaches to learning memory-influenced behaviors from observation is a promising venture toward enabling the creation of AI programs that can both approximate, understand, and benefit from the cognitive intricacies of observed entities. MCL exemplifies just one approach to this problem and perhaps it may inform future improvements or new approaches.

The reader who has persevered to this point of the dissertation (or who just jumped to the end) is encouraged everyday to keep imagining, keep creating, and keep discovering the secrets tucked away by our Maker for us to find. Whatever your journey or pursuit, know that you are loved and that you matter.

APPENDIX A: ANALYSIS OF MEMORY FEATURE EXTRACTION ALGORITHMIC COMPLEXITY

This appendix discusses the run time complexity of the MF Extraction phase of the MCLv0 and MCLvN approaches, discussed in Chapters 5 and 6, respectively.

A.1   MCLv0 Extraction Algorithmic Complexity

We now discuss the algorithmic complexity of the Memory Feature Extraction step in MCLv0, described in Section 5.2.1. Suppose that in a certain domain, MCLv0 must learn from T traces that are at most N time steps in length that record values for F features. (Let us assume that F and T are constants.) This means that the case base created from these traces will have TN cases according to the Memory Feature Extraction (see Algorithm 2 from Section 5.1). In Algorithm 3, Memory Temporal Backtracking (MemTB) is invoked three times for a given case c. Each call to MemTB in the worst case compares every time step in the problem component sub-run of c to that of every other case's problem component sub-run in the case base.

A paraphrased algorithm for the MCLv0 MF Extraction algorithm is seen in Algorithm 12. Here, N cases are processed in procedure `mem-feat-extract-v0`. For each case, a call to `extract-v0` is made (see Algorithm 13), which is shown in Algorithm 13. In the while loop, up to two MFs are added to the test case and case base each turn. In the worst case, MFs are added for each time step in the test case (for a maximum of N time steps). In each of these loop iterations, three calls to the recursive procedure `mem-tb` are made (see Algorithm 14). In each recursive call, the test case is compared to all candidate cases, which is all TN cases in the worst case. The recursive depth can be a maximum of N (the number of time steps in the test case's problem component sub-run).

Algorithm 12 MCLv0 MF Extraction

 1: **procedure** MEM-FEAT-EXTRACT-V0(casebase)
 2:     mem-feat-lists ← (empty-set)
 3:     **for** *test-case* in casebase **do**
 4:         mem-feats ← `extract-v0`(test-case, casebase)

Algorithm 13 MCLv0 Single Case Extraction
___
1: **procedure** EXTRACT-V0(test-case, casebase)
2:     mem-feats ← (empty-set)
3:     **while** True **do**
4:         recurs, raw-mem-feat ← `mem-tb`(test-case, casebase, 0)
5:         **if** recurs $= 0$ **then**
6:             **return** mem-feats
7:         val-test-case, val-casebase ← `add-val-mem-feat`(test-case, casebase)
8:         recurs-val, rmf-val ← `mem-tb`(val-test-case, val-casebase, 0)
9:         tback-test-case, tback-casebase ← `add-tback-mem-feat`(test-case, casebase)
10:       recurs-tback, rmf-tback ← `mem-tb`(tback-test-case, tback-casebase, 0)
11:       **if** recurs-val geq recurs-tback **then**
12:           test-case, casebase ← `add-val-mem-feat`(test-case, casebase)
13:           mem-feats.add(rmf-val.`as-value-back`())
14:       **if** recurs-tback geq recurs-val **then**
15:           test-case, casebase ← `add-tback-mem-feat`(test-case, casebase)
16:           mem-feats.add(rmf-val.`as-time-back`())
___

Algorithm 14 Memory Temporal Backtracking
___
1: **procedure** MEM-TB(test-case, candidate-cases, time-back)
2:     **if** `solutions-agree`(candidate-cases) **then**
3:         best-feat ← `get-best-feat`(candidate-cases, time-back)
4:         **return** raw-mem-feat(best-feat, time-back, test-case.`value`(best-feat))
5:     new-candidates ← (empty-set)
6:     **for** `cur-case` in candidate-cases **do**
7:         sim ← `compare-cases`(*cur-case*, test-case)
8:         **if** sim geq THRESHOLD **then** new-candidates.add(*cur-case*)
9:     **return** `mem-tb`(test-case, new-candidates, time-back $- 1$)
___

The algorithmic complexity for this operation is computed as follows:

1. There are TN cases, each with problem component sub-runs up to N time steps in length.

2. MemTB is invoked each time a new MF is required for a case c. A case c may have up to $2N$ MFs (two per time step), thus requiring at worst N MemTB calls.

3. During each call of MemTB, a recursion depth of N will be achieved in the worst case.

4. During each MemTB recursive call, case c will compared with at most TN cases.

5. When perceptions are compared, all F feature values must be compared. However, MemTB adds one MF to the cases after each iteration, so there will be F, $F + 1$, $F + 2$, etc. feature comparisons per MemTB iteration. However, we assume F is a constant.

For TN cases, there are at most N calls to MemTB per case (one per time step), at most N recursive iterations per MemTB call (one per time step), and at most TN case comparisons per recursive iteration. Thus, the algorithmic complexity for MF Extraction for all cases is $O(N^4)$, if we treat T as a constant.

For certain real-world applications, this runtime may be prohibitively slow. This is the motivation behind the improvements made in MCLvN in the Memory Feature Extraction phase (Step 1a "Extract Memory Features" in Figure 6.1).

A.2   MCLvN Extraction Algorithmic Complexity

In MCLvN, there are still TN cases, but this time all cases are processed with a single call to MemTB+ as a result of the Memory Singularity Principle (MSP). Furthermore, Tree Extraction processes "reject cases" from the point in time when they diverge from a test case, rather than extracting MFs for them from scratch.

In Algorithm 15, the function call to procedure `extract-vN` is invoked only once, for all cases in the case base. This procedure `extract-vN` (see Algorithm 16) is very similar to Memory Temporal Backtracking, except it implements MSP and tree extraction. It first checks if all solutions in the candidates case pool have the same solution, in which case recursion ends. Otherwise, various "case groups" are computed by the procedure `get-case-groups`.

416

Algorithm 15 MCLvN Memory Feature Extraction

1: **procedure** MEM-FEAT-EXTRACT-VN(casebase)
2:     extracted-mem-feats ← (empty-set)
3:     `extract-vN`(casebase, extracted-mem-feats, 0)

Algorithm 16 MCLvN Tree Extraction

1: **procedure** EXTRACT-VN(candidate-cases, mem-feats, time-back)
2:     **if** `solutions-agree`(candidate-cases) **then**
3:         best-feat ← `get-best-feat`(candidate-cases, time-back)
4:         **for** *case* in candidate-cases **do**
5:             test-val ← test-case.`value`(best-feat)
6:             raw-mf ← raw-mem-feat(best-feat, time-back, test-val)
7:             mem-feats.`at`(*case*).`add`(raw-mf)
8:         **return**
9:     case-groups ← `get-case-groups`(candidate-cases)
10:     **for** *group* in case-groups **do**
11:         test-case ← group.`at`(0)
12:         best-feat ← `get-best-feat`(group)
13:         **for** *case* in group **do**
14:             test-val ← test-case.`value`(best-feat)
15:             raw-mf ← raw-mem-feat(best-feat, time-back, test-val)
16:             mem-feats.`at`(*case*).`add`(raw-mf)
17:         `extract-vN`(group, mem-feats, time-back - 1)

Algorithm 17 MCLvN Case Groups Computation

1: **procedure** GET-CASE-GROUPS(candidate-cases)
2:     unmapped-cases ← candidate-cases.`copy`()
3:     case-groups ← (empty-set)
4:     **while** unmapped-cases.`size` > 0 **do**
5:         group ← (empty-set)
6:         test-case ← unmapped-cases[0]
7:         **for** *cur-case* in unmapped-cases **do**
8:             sim ← `compare-cases`(*cur-case*, test-case)
9:             **if** sim > THRESH **then**
10:                 group.`add`(*cur-case*)
11:                 unmapped-cases.`remove`(*cur-case*)
12:         case-groups.`add`(group)
13:     **return** case-groups

In procedure `get-case-groups` (see Algorithm 17), all candidate cases are initially "unmapped". The while loop creates a new group of similar cases. In the worst case, this loop creates a new group for every case, resulting in about $(TN)^2$ case comparisons (the summation of 1 to TN cases). However, if this event occurs, then recursion will end after a single call. On the other extreme, if procedure `get-case-groups` creates just a single group (after only TN case comparisons), then in the worst case, a recursion depth of TN is achieved (where there is only one case group until the beginning of the behavioral trace is reached).

So whether procedure `get-case-groups` has $(TN)^2$ case comparisons and procedure `extract-vN` has a recursion depth of 1 or procedure `get-case-groups` has TN case comparisons and procedure `extract-vN` has a recursion depth of TN (calling procedure `get-case-groups` each time), MF Extraction in MCLvN has a worst-case algorithmic complexity of O(N^2) (where T is a constant), which is significantly better than that of MCLv0.

APPENDIX B: IRB LETTER OF APPROVAL FOR LAWN MOWER STUDY

UNIVERSITY OF CENTRAL FLORIDA

EXEMPTION DETERMINATION

April 3, 2019

Dear Josiah Wong:

On 4/3/2019, the IRB determined the following submission to be human subjects research that is exempt from regulation:

Type of Review:	Initial Study, Category
Title:	Learning Behavioral Memory Representations from Observation in a Simulated Lawn Mower Task
Investigator:	Josiah Wong
IRB ID:	STUDY00000341
Funding:	Name: National Science Foundation (NSF)
Grant ID:	

This determination applies only to the activities described in the IRB submission and does not apply should any changes be made. If changes are made, and there are questions about whether these changes affect the exempt status of the human research, please contact the IRB. When you have completed your research, please submit a Study Closure request so that IRB records will be accurate.

If you have any questions, please contact the UCF IRB at 407-823-2901 or irb@ucf.edu. Please include your project title and IRB number in all correspondence with this office.

Sincerely,

Racine Jacques, Ph.D.
Designated Reviewer

Page 1 of 1

APPENDIX C: DESIGN OF LAWN MOWER SIMULATION

This appendix describes the layouts of each scenario in the lawn mower simulation (LMS). The scenarios are described in the order in which they appeared to the human participants during the collection of human behavior data. These are the scenarios in the LMS that were used to collect human behaviors involving memory. Depending on how many attempts it took a participant to complete each scenario, all scenarios took anywhere from 15 to 30 minutes to complete. Even though the LMS is presented in the guise of a lawn mower simulation, it was not meant to resemble a completely realistic lawn mower experience (as evidenced by the presence of entities based on mythical creatures, such as Leprechauns and fairies). Therefore, certain scenarios have very little grass to cut; the number and placement of grass patches was such that the human subjects, in their pursuit of mowing the required number of grass patches, had maximum exposure to the simulation entities requiring memory-based behaviors. LMS is a test bed designed to facilitate memory-based behaviors in humans. For each scenario, we describe the purpose of the scenario (what we were trying to accomplish), the memory-based behaviors that were required by the human to fulfill the requirements of the scenario, and any difficulties the human may have encountered during game play. Generic details about the LMS are discussed in Section 8.3.1.

Figure C.1 shows the first scenario. Its purpose was to introduce the human to the user interface, joystick controls, and basic lawn mower movement. The human is introduced to a points requirement (e.g. number of grass patches required to enable the exit) and a time limit. The human learns that barriers (gray squares) are impassable and that grass patches can be mowed exactly once for one point. This scenario serves as a baseline to which a human's performance in other scenarios can be compared.
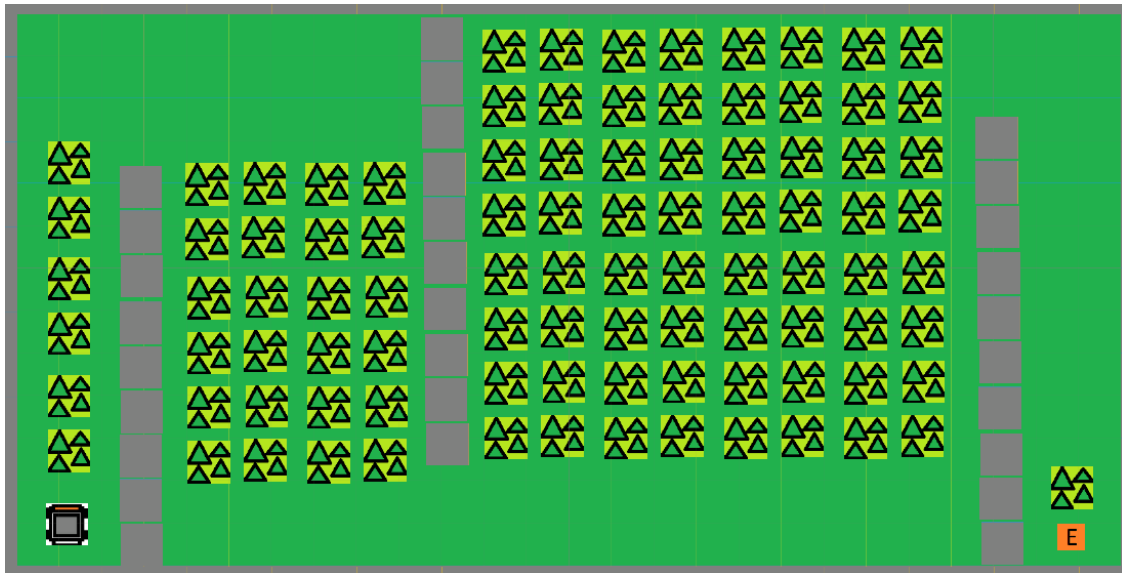
Figure C.1: LMS Scenario 1. Simulation entity introduced: Grass.

There are no explicit memory elements in this scenario. However, a learning agent observing the human's behavior may still require memory in order to remember which direction the human is moving in, similar to how memory was required in the vacuum cleaner domain for the straight line and zig zag agents. The time limit was set such that unless the human is reasonably methodical in their mowing behavior, they will not complete the scenario in time. One strategy that the humans can utilize is mowing two grass patches simultaneously by moving in between them. Thus, a human who employs this strategy will not be aiming directly at a single patch of grass, but rather at a group of grass patches, which is a possible challenge when generating MFs with MCL.

This scenario involves the following memory-based behaviors:

1. Remember which direction the human was moving.

2. If mowing in a zig zag, circular, or random pattern, remember which part of the pattern (if any) was just executed. Execute the next part of the movement pattern.

Figure C.2 shows the second scenario. It introduces the human to dark grass patches. A few regular grass patches are present to demonstrate to the human that dark grass patches, unlike regular grass patches, look the same both mowed and uncut. However, an audible sound is made each time a grass patch or dark grass patch is cut as feedback that a human's exploratory movements are moving in the right direction.
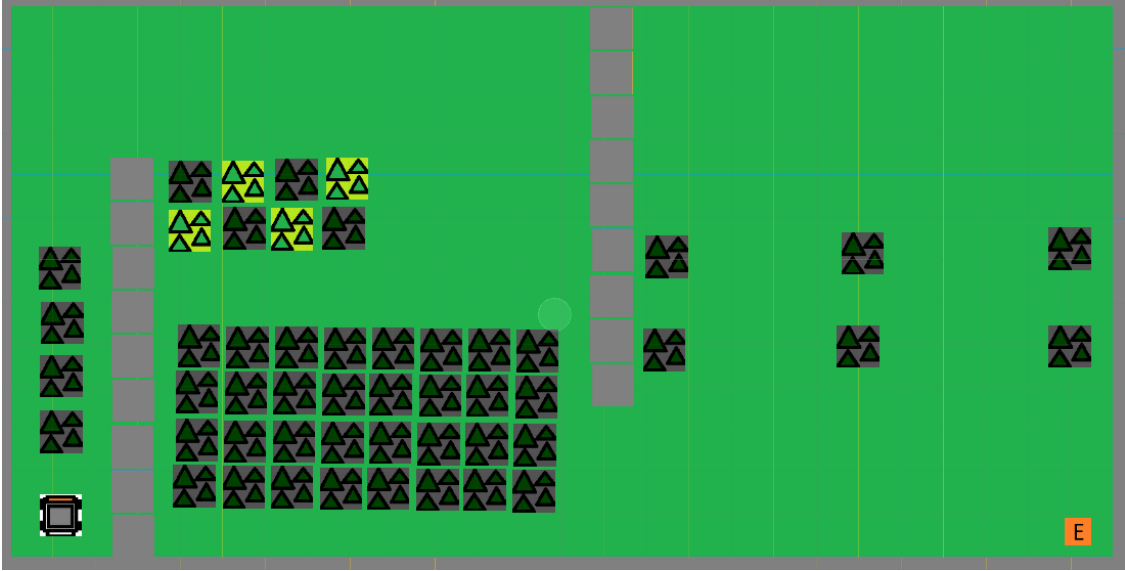
Figure C.2: LMS Scenario 2. Simulation entity introduced: Dark Grass.

The memory-based behavior required in this scenario is remembering which dark grass patches the human has already visited. A human with perfect memory will mow them in the same fashion as they would for regular grass patches. A human with a less confident memory may employ more random mowing behaviors in search of unmowed dark grass if they don't remember which exact patches were mowed. The main difficulty of this scenario is that the human does not have sufficient time to mow over all dark grass patches twice to make sure they were mowed — the human will have to remember at least some of the dark patches that were mowed in order to complete the scenario in time.

424

This scenario involves the following memory-based behaviors:

1. Remember whether a region of dark grass patches has been visited.

2. If mowing in a zig zag, circular, or random pattern, remember which part of the pattern (if any) was just executed. Execute the next part of the movement pattern.

Figure C.3 shows the third scenario. Its main purpose is to demonstrate that rocks should be avoided because they damage the lawn mower. This scenario also gives the human some practice with precise joystick handling.

Rocks, like grass, are entities that do not explicitly require memory on part of the human. Rocks are the hazard-equivalent of grass, except they can damage the lawn mower multiple times, since they are indestructible. However, memory may be required from a learning agent observing the human in order to remember which direction the human is going. This is because the scenario environment is laid out such that the human is almost never going directly toward the exit in order to avoid the rocks. Each pathway between the rocks has two exits and a learning agent must use memory to remember which way the lawn mower was going. Specifically when the human is making a turn around the rocks, the learning agent must remember which pathway the human came from in order to know that it should go for the other pathway.
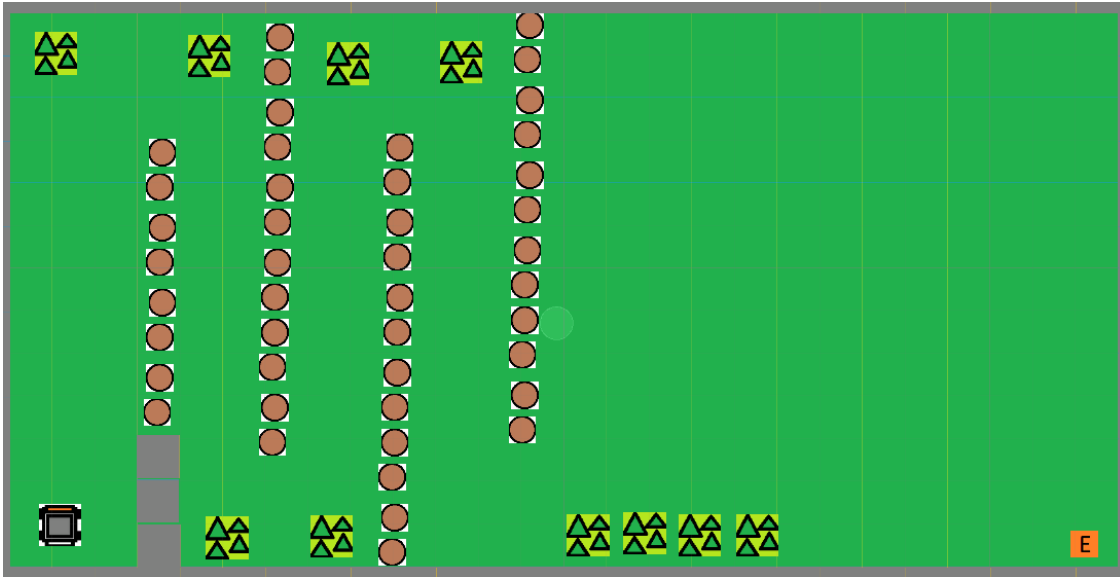
Figure C.3: LMS Scenario 3. Simulation entity introduced: Rocks.

This scenario involves the following memory-based behaviors:

1. Remember which direction the human was moving. Continue moving in that direction.

2. When rounding a corner around a column of rocks, remember which direction the human came from and don't go back in that direction.

Figure C.4 shows the fourth scenario. It combines grass, dark grass, and rocks into one scenario as a sort of "test scenario" to assess the human's abilities with all these entities all at once.

Figure C.4: LMS Scenario 4. Features grass, dark grass, and rocks.

The intended strategy is for the human to first mow the region of dark grass in the top left, and then proceed in a counter-clockwise fashion through the environment, though the human could use an alternate route. Dark grass patches are the main entity requiring memory on part of the human, especially those in the top left region. The main difficulty on part of the human would be handling the joystick precisely enough to successfully navigate through two rocks. The four regions of regular grass in the middle and the navigation from the eight regular grass patch region to the dark grass patch region to its right would require this behavior when using the intended navigation strategy. Because this scenario has four regions of dark grass, the human must also remember (on a higher level) which dark grass regions have been visited, in addition to which individual dark grass patches have been mowed within a given region.

This scenario involves the following memory-based behaviors:

1. Remember whether a region of dark grass patches has been visited.

2. Within a given dark grass patch region, remember which individual dark grass patches have been mowed.

Figure C.5 shows the fifth scenario, which introduces ground hogs. The screen shot shows the locations of all ground hog locations, with the ground hogs that are hidden grass enclosed in a red circle in the figure. However, one should note that ground hogs do not become visible to the human until the human gets close enough to them — the figure simply shows all of them as visible for the benefit of the reader.
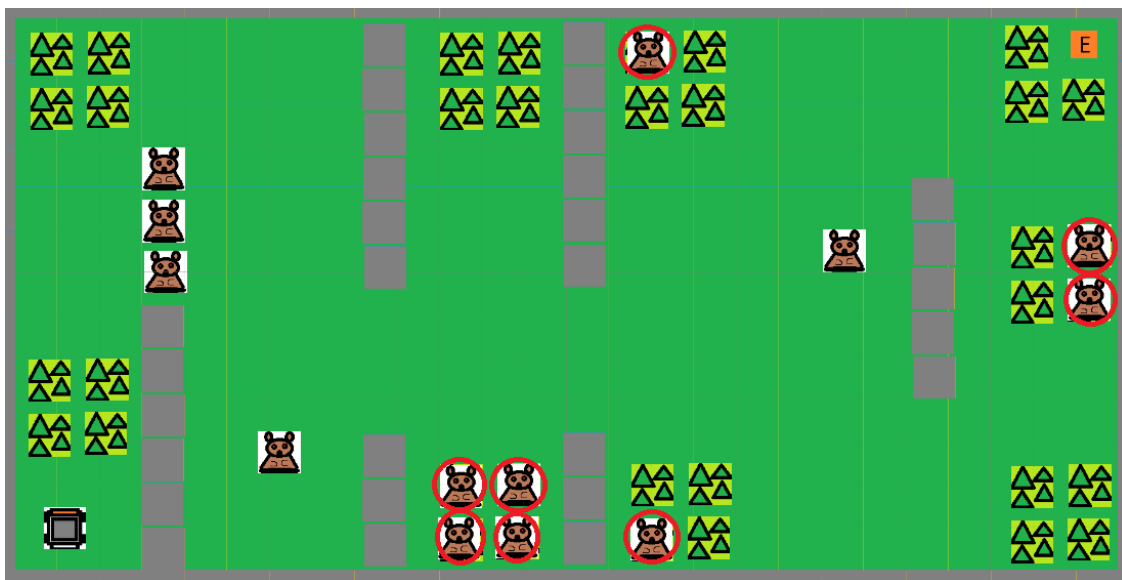
Figure C.5: LMS Scenario 5. Simulation entity introduced: Ground hogs.

Ground hogs are the first instance of static memory elements the human encounters — ground hogs are temporarily visible, then unmoving and invisible until a collision with the lawn mower. The grass patches not concealing a ground hog do not have to be mowed to enable the scenario's exit.

Ground hogs produce an interesting behavior in the human in that a human's intended trajectory

may be interrupted by a ground hog that reveals itself along that trajectory, thus requiring an immediate path readjustment by the human. The human does not need to remember the locations of all ground hogs that revealed themselves, but only those that are among grass patches that are yet to be mowed.

This scenario involves the following memory-based behaviors:

1. Remember whether a ground hog was seen a particular patch of grass. Avoid that grass patch if a ground hog peeked above ground in that grass patch.

2. Remember whether the human has come within close proximity of a grass patch. A ground hog may yet be revealed in that grass patch and an approach at slower speed may be advised for such a situation.
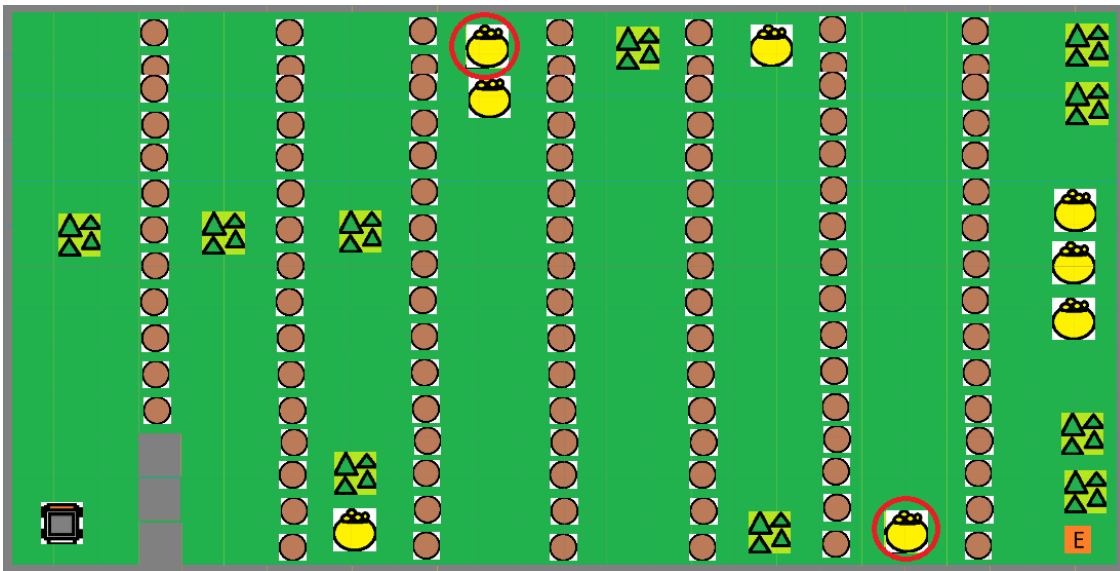
Figure C.6: LMS Scenario 6. Simulation entity introduced: Gold pots.

Figure C.6 shows the sixth scenario, which introduces gold pots. All gold pots are initially invisible

429

until the lawn mower gets close enough to them. The gold pots surrounded by a red circle in the figure will be obscured by a grass patch after they are revealed.

In this scenario, it is necessary for the lawn mower to take damage by passing over successive columns of rocks while replenishing health points with gold pots. Gold pots are similar to ground hogs, except they replenish health once before vanishing. This does not require as much memory from the human as ground hogs because 1) gold pots are only hidden by grass, which the human knows to collide with anyway, and 2) gold pots are only usable once. In contrast, ground hogs must be perpetually remembered and avoided for the duration of the scenario. Only two gold pots are obscured by grass in this scenario, but it was a curiosity as to how memory-based behaviors would be affected by a positive version of a ground hog.

This scenario involves the following memory-based behaviors:

1. Remember whether a gold pot was seen in a particular patch of grass. Go to that patch of grass.

2. Remember whether a particular end section of the environment between rock columns has been visited. A gold pot may reveal itself if that section is unvisited.

Figure C.7 shows the seventh scenario, which introduces snakes. It was created because the scenario which comes after it (which was created first) was a little difficult and we wanted a simpler version of that scenario to ensure the human subjects could exhibit correct behavior in reaction to snakes.
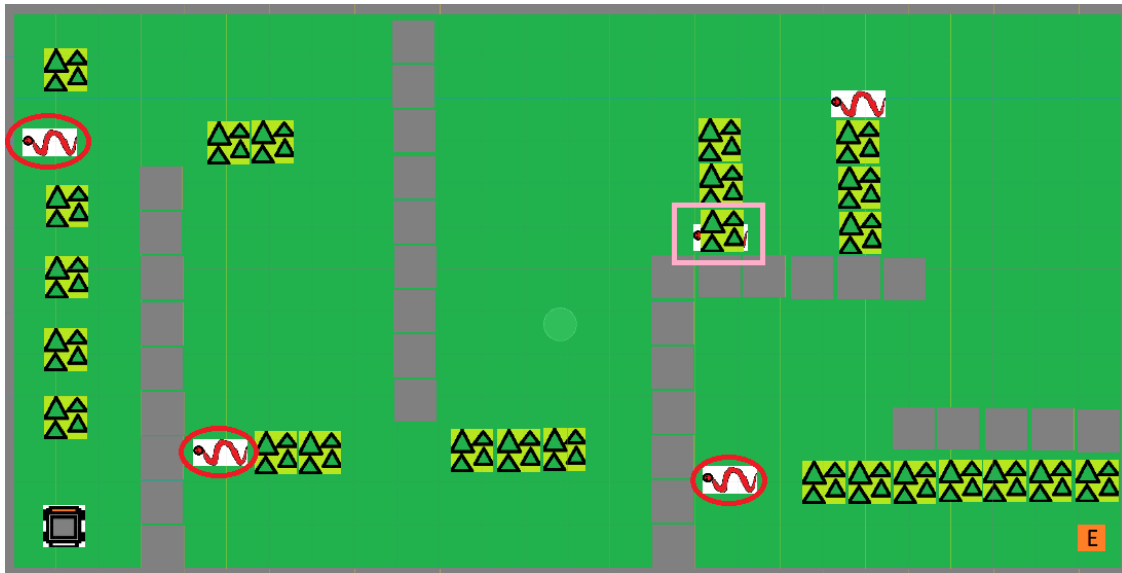
Figure C.7: LMS Scenario 7. Simulation entity introduced: Snakes.

The snakes that move horizontally are surrounded by a red circle in the figure; the rest move vertically (including the one with a pink square outline around it, which is partially occluded by grass in the figure). Snakes are invisible when moving through grass, which requires the human to remember where the snake first became invisible and then project where the snake's likely location is.

There are two memory-based behaviors the human may exhibit when dealing with snakes. First, if the human wants to take a path perpendicular to the snake's path, then the human can simply remember whether or not the snake is located in the position where the human wishes to cross. Most snake paths in this scenario only go through two or three consecutive grass patches at a time. However, the snake in the bottom right of the environment goes through seven grass patches, which requires significant memory from the human. Additionally, the human cannot simply mow across the path because of the barrier above the grass patches. However, if the human mows those grass patches, the snake will no longer be invisible on that part of the path, thus reducing the amount of

memory required by the human to address that snake.

This scenario involves the following memory-based behaviors:

1. Remember where a snake first enters a patch of grass and becomes invisible. Remember which direction the snake was moving.

2. Remember how long a snake has remained invisible. Based on this duration of invisibility, project whether the snake is located in the grass of patch that will be mowed next.

3. In the case of the snake in the bottom right of the scenario, project whether the snake has changed directions.

Figure C.8 shows the eighth scenario. It is a challenging scenario with snakes. The snakes that move horizontally are circled in red in the figure.
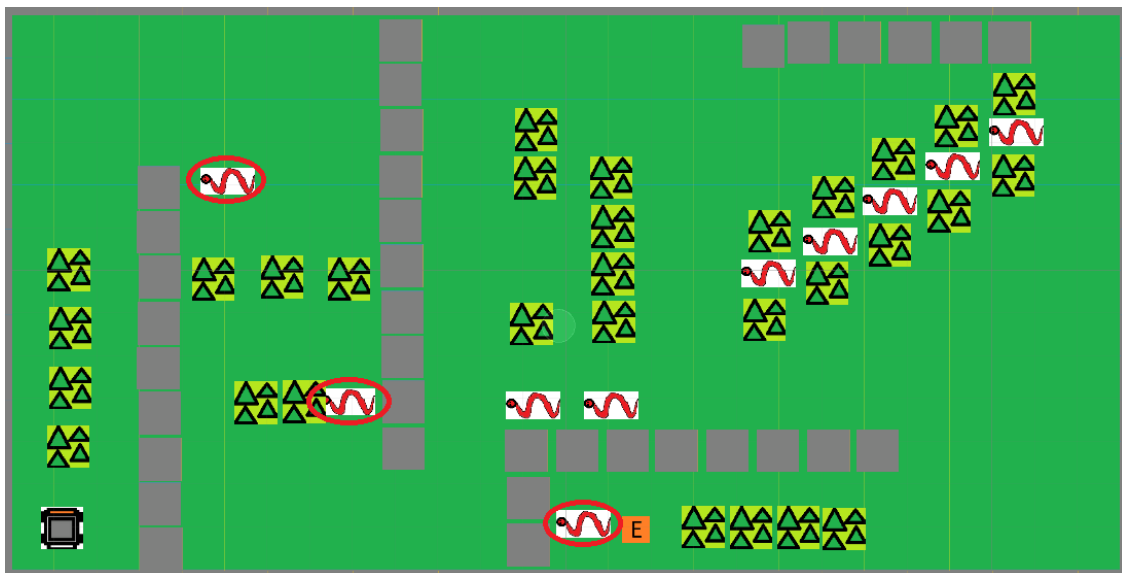
Figure C.8: LMS Scenario 8. Features snakes.

The first two snakes encountered have a very short path, because they switch directions upon hitting a barrier. Thus, the human must move quickly through these snakes. The second snake, whose path is covered by two grass patches, requires precise memory as a result. The next two snakes encountered move vertically at the same rate. Thus, if one of those snakes is hidden and the other is visible, the human can infer that the hidden snake is right next to the other snake within the grass, which reduces the memory requirement for those snakes. The five snakes arranged in a diagonal are difficult to navigate, but it is possible to avoid the grass patches in that region if more grass patches are mowed form other regions of the environment. In contrast, the snake at the bottom of the play area has a path that goes over the exit. The only way to hit the exit is to literally follow the snake through the line of grass patches leading to the exit, which requires careful memory of where the snake is. As an extra challenge, the lawn mower is slightly faster than the snake, so moving at full speed may result in a collision with the snake.

This scenario involves the following memory-based behaviors:

1. Remember where a snake first enters a patch of grass and becomes invisible. Remember which direction the snake was moving.

2. Remember how long a snake has remained invisible. Based on this duration of invisibility, project whether the snake is located in the grass of patch that will be mowed next.

Figure C.9 shows the ninth scenario, which introduces Leprechauns. Because Leprechauns replenish health, the scenario is designed similarly to the gold pots scenario so that it is necessary for the lawn mower to take damage in order to make it to the exit.
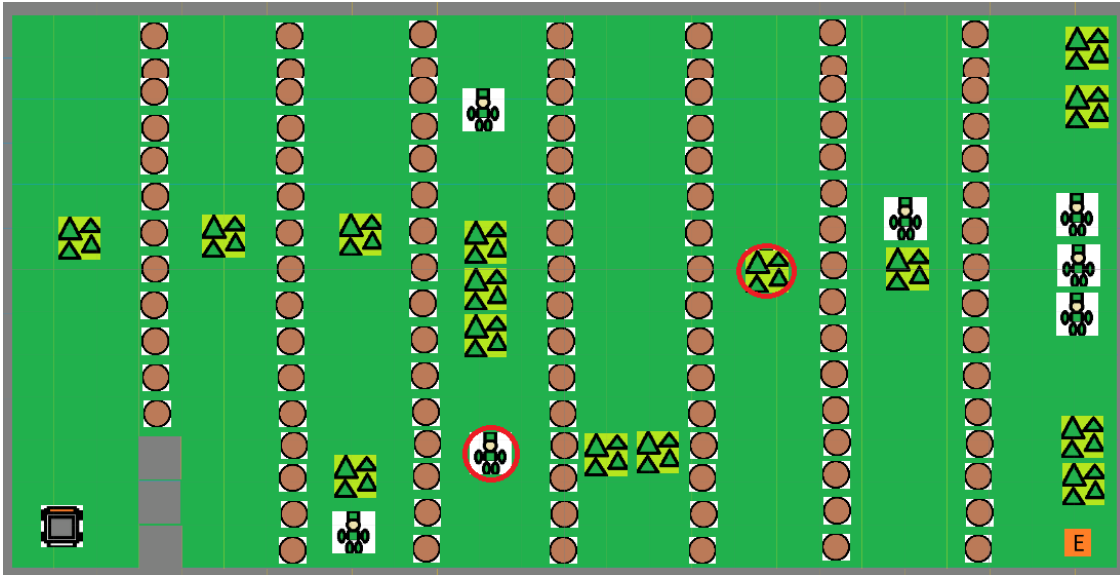
Figure C.9: LMS Scenario 9. Simulation entity introduced: Leprechauns.

The two Leprechauns that move horizontally (instead of vertically) are surrounded by red circles in the figure. (One of the Leprechauns is being rendered under the grass patch in the figure, just right of the fifth column of rocks from the left.) The intended strategy for this scenario is to remember which direction a Leprechaun is going move toward if it is moving vertically or else wait in its path if it is moving horizontally (because Leprechauns can move through rocks unencumbered).

This scenario involves the following memory-based behaviors:

1. Remember which direction a Leprechaun is moving (away or towards the lawn mower; horizontally or vertically).

2. Remember where a Leprechaun first entered a patch of grass and became invisible. Project its future location. Move toward that location if lawn mower is within close proximity to its projected path.

434

Figure C.10 shows the tenth scenario, which introduces Sprinklers. In this scenario, the grass patches represent "safe" locations beyond the reach of the sprinklers' spray radii. In the figure, all sprinklers are visible in the "spray" phase, but sprinklers do not begin their phased behavior until the lawn mower gets within a certain proximity of them and thus are all initially invisible when the human begins the scenario. The human must remember which "phase" a sprinkler is in (even if it is invisible) and how long the sprinkler has been in that phase.
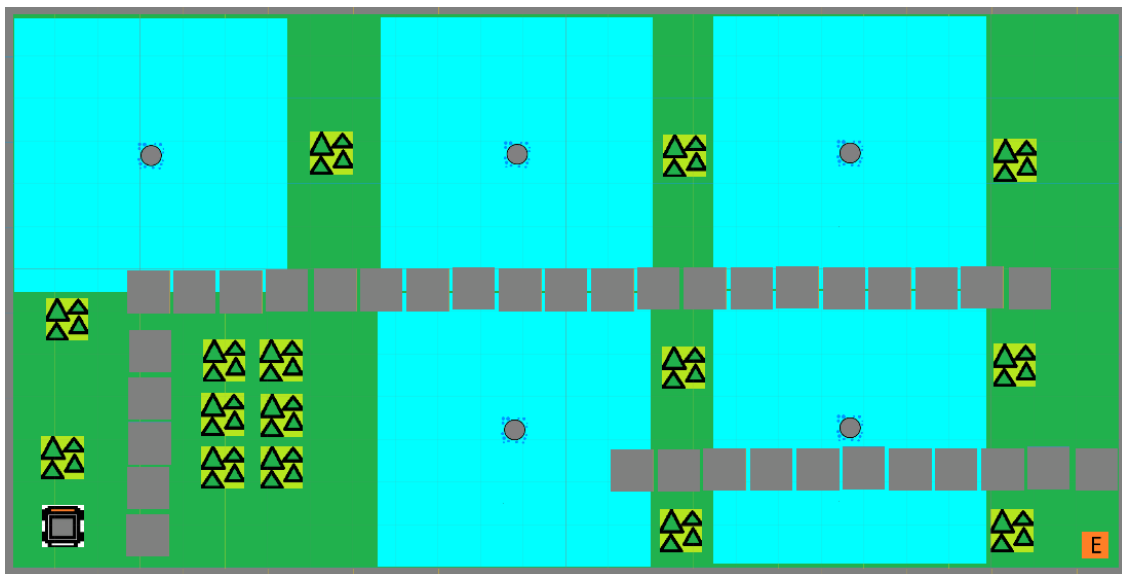
Figure C.10: LMS Scenario 10. Simulation entity introduced: Sprinklers.

This scenario involves following memory-based behaviors:

1. Remember which direction the human was previously moving in. This is critical because the human spends most of their time remaining still, waiting for a sprinkler to cease spraying.

2. If a sprinkler just revealed itself and entered the invisible "standby" phase, remember that there was a sprinkler there and get a safe distance away.

3. If the sprinkler has finished spraying and is in the "cooldown" phase, remember how long it has been since the sprinkler entered cooldown because it is safe to traverse the sprinkler's area of effect while it is in cooldown.

Figure C.11 shows the eleventh scenario, which features sprinklers.
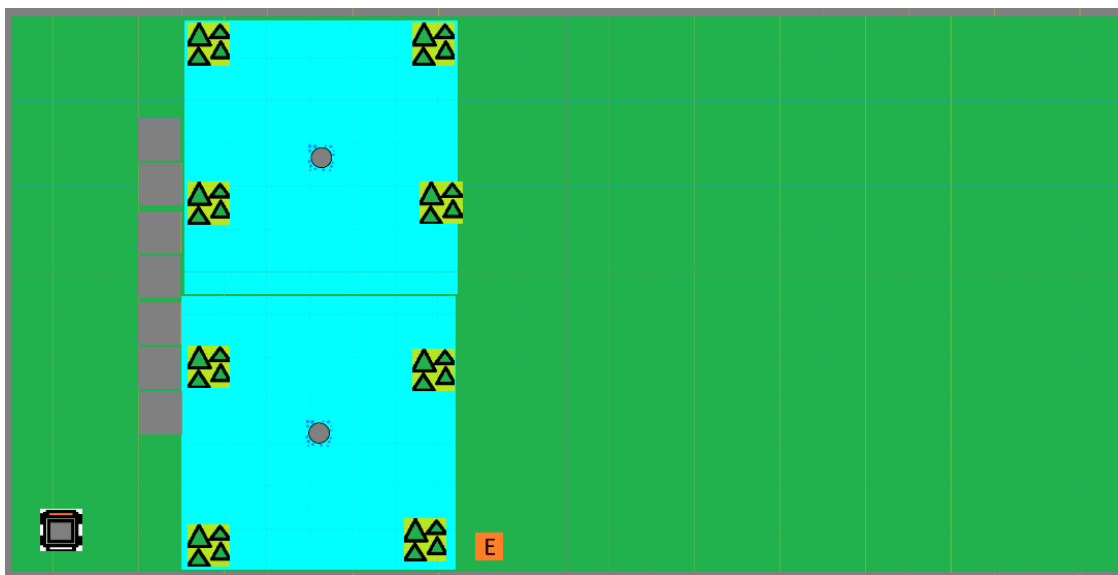
Figure C.11: LMS Scenario 11. Features sprinklers.

In contrast to the previous sprinkler scenario, the grass patches are located within the sprinklers' spray areas and the human must time their approach to coincide with the sprinkler cooldown phases. This scenario was the original sprinkler scenario, but since it only featured two sprinklers, the previous sprinkler scenario was created to facilitate more human behavior around sprinklers.

The human exhibits the same memory-based behaviors in this scenario as they do in the previous scenario, with some exceptions:

1. The two sprinklers are right next to each other, so there is no "safe" zone between them.

The human must remember the phases for both sprinklers simultaneously instead of just the "next" one on their path.

2. The grass patches are located within the sprinklers' areas of effect. Therefore, it is even more crucial for the human to remember how long a sprinkler has been in cooldown so that the human has enough time to leave the area of effect if the sprinkler pokes its head above ground to begin its cycle again.

Figure C.12 shows the twelfth scenario, which introduces fairies. Like the sprinklers in the previous scenario, the fairies are shown in the "spray" phase in this figure, but are initially invisible to the human until the human gets close enough.
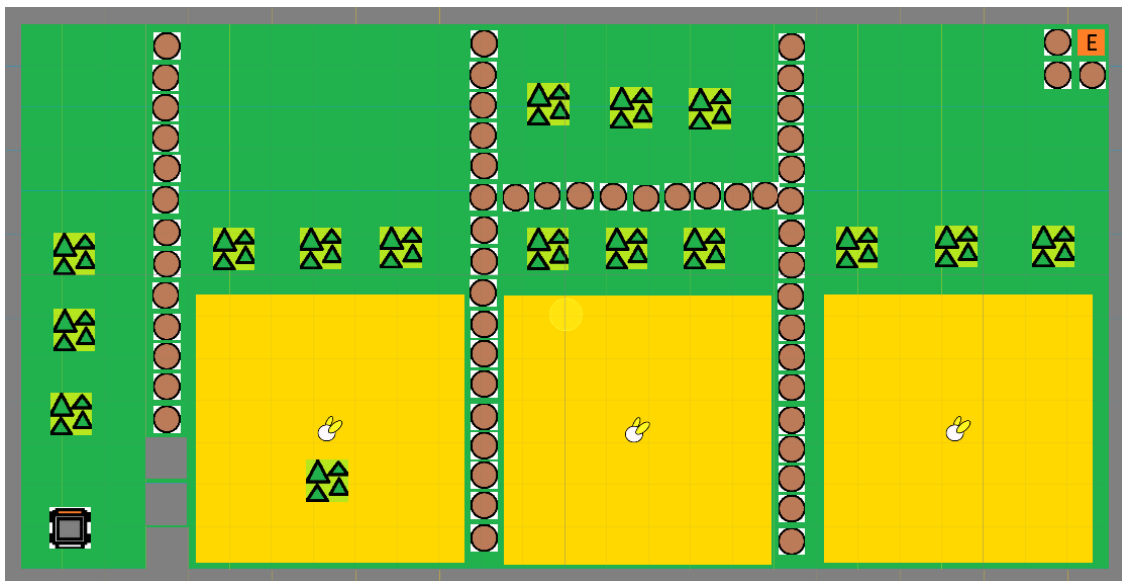
Figure C.12: LMS Scenario 12. Simulation entity introduced: Fairies.

There are a few possible routes through the scenario, but the lawn mower must take at least five instances of damage throughout the scenario to make it to the exit, ensuring that the human will need to use at least two fairies in order to fulfill the scenario requirements. Because fairies restore

all health, this scenario and the next fairy scenario are more forgiving of mistakes by the human than the sprinkler scenarios are.

This scenario involves following memory-based behaviors:

1. Remember how long a fairy has been in their spray phase. Go to the fairy if there is sufficient time to get within its area of effect before it exits spray phase.

2. Remember where a fairy previously spawned and how long it has been in cooldown. If health is sufficiently low, revisit a previous fairy location if a sufficient amount of time has passed since the fairy entered cooldown.
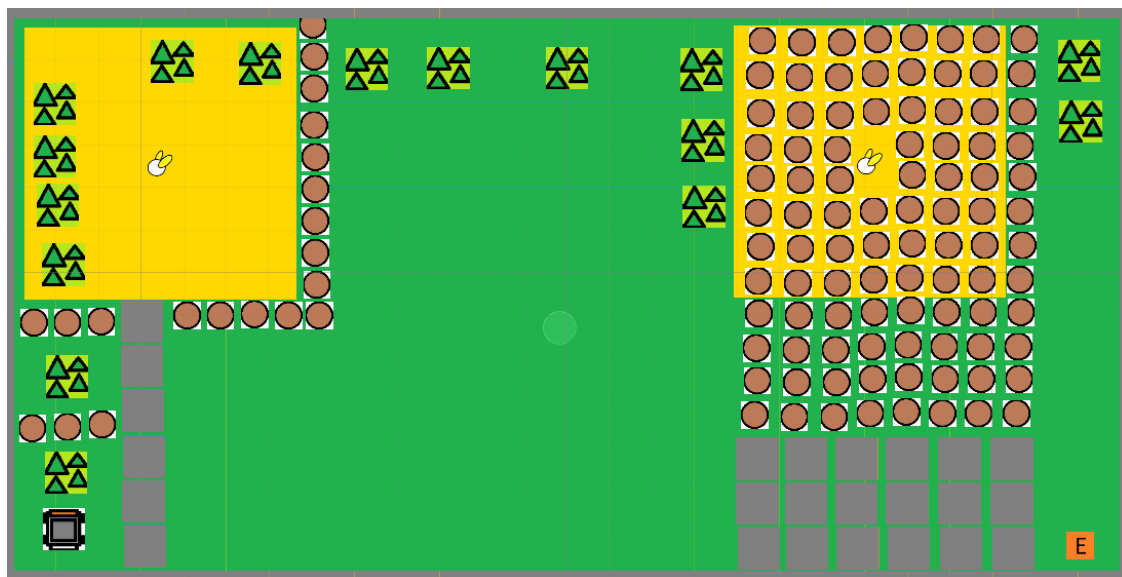
Figure C.13: LMS Scenario 13. Features fairies.

Figure C.13 shows the thirteenth and final scenario, which features fairies. This was the original scenario meant to introduce the fairy entity, but since there were only two fairies, an additional scenario (the previous one) was created to facilitate additional human behavior around fairies.

438

This scenario facilitates the same memory-based behaviors as the previous fairy scenario, except the second fairy (the one on the right of the play area) is especially critical. The deluge of rocks on the right of the environment can only be traversed while the fairy is in the spray phase. The lawn mower cannot just touch the fairy's area of effect once — the lawn mower must travel through it. If the human does not time their approach through the fairy's spray area just as the spray phase begins, then the human risks losing all their health to damage by the rocks. A more risky (but faster) approach would be to begin traversal through the rocks immediately after the fairy peeks (enters the standby phase) and time the approach such that the fairy's spray phase begins right before all health is lost.

APPENDIX D: LAWN MOWER STUDY SURVEY RESULTS

This appendix provides aggregate survey results from the research study that was conducted for the lawn mower domain. After each of our eleven human subjects experienced each scenario in the lawn mower simulation, they filled out a survey. The purpose of the survey was to solicit qualitative responses from the participants about memory-based strategies they used to address the entities in the simulation and to quantitatively gauge the degree to which each person reported using memory for each entity. The survey responses did not have a direct effect on the success of the research, but they provided interesting insights that we share here.

The survey solicited demographic information (e.g. gender, age, major) and information about the person's experience with joysticks and mowing lawns.

The following questions required a response via a five-point Lickert scale, where "1" was strongly disagree and "5" was strongly agree. Some of the questions were repeated for each simulation entity.

1. I found the lawn mower controls to be natural, as if operating a real lawn mower.

2. I found the simulation to be realistic.

3. I found that it was easy to understand the mechanics behind (simulation entity).

4. I found that it was easy to know/remember/infer the locations of (simulation entity).

5. The simulation contained a manageable number of (simulation entity).

6. It required a significant amount of memory on my part to address (simulation entity).

7. The lawn mower simulation required me to use memory in my mowing behavior.

The following questions were open-ended:

1. What strategy did you use to address (simulation entity) in the simulation?

2. If you could play the game again, what would you do differently?

3. How would you improve the simulation?

4. Please provide any final thoughts or comments about your experience.
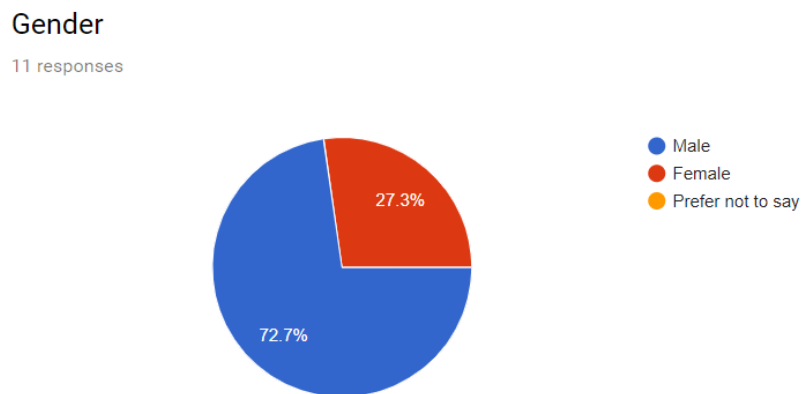
Figure D.1: Lawn Mower Survey – Gender

Figure D.1 shows the gender distribution of the human subjects. We had 3 females and 8 males participate in the study.
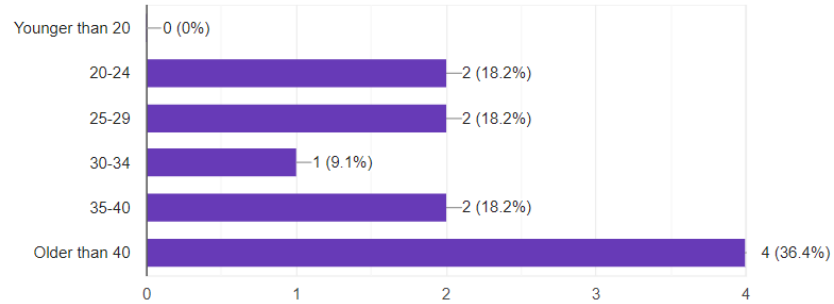
Figure D.2: Lawn Mower Survey – Age

Figure D.2 shows the age distribution of the human subjects. All human subjects were above the age of 18.
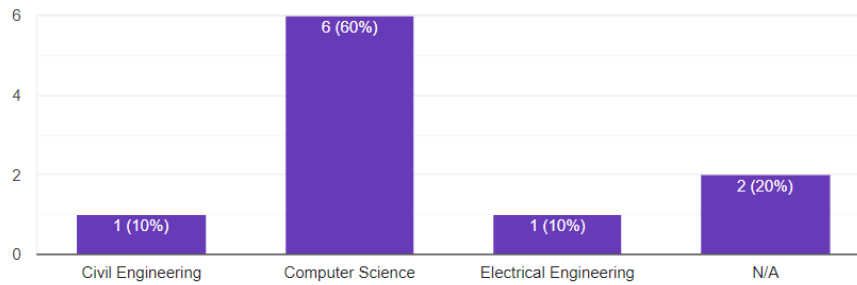
Figure D.3: Lawn Mower Survey – Major

Figure D.3 shows the educational backgrounds of our participants. The majority of our participants were Computer Science majors.

Experience with Joysticks

11 responses
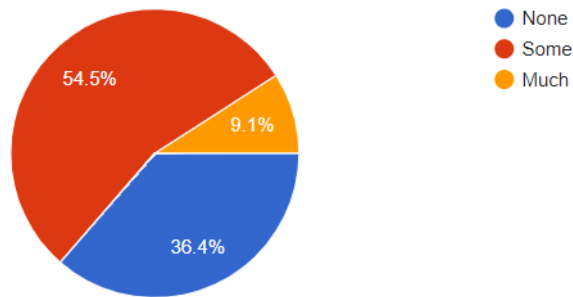
Figure D.4: Lawn Mower Survey – Joystick Experience

Figure D.4 shows how experienced our participants were with joysticks, which may have affected their performance in the simulation. The majority of our participants had at least some experience with joysticks.

Experience mowing lawns

11 responses
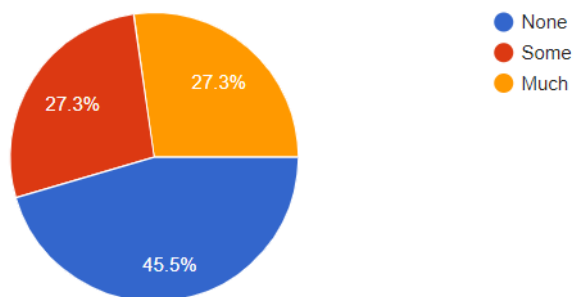
Figure D.5: Lawn Mower Survey – Mowing Experience

Figure D.5 shows the mowing experience that our participants had, which may have impacted their decisions in the lawn mowing simulation. About half of our participants had no experience while

half had at least some if not much experience with mowing lawns.

Lawn mower controls were natural
11 responses
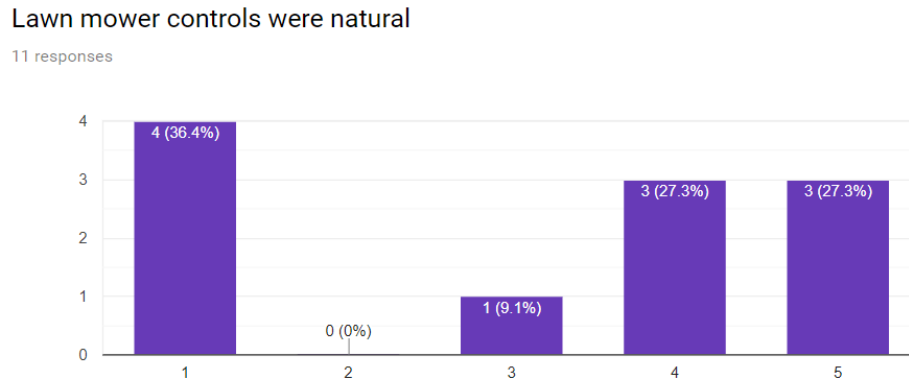
Figure D.6: Lawn Mower Survey – Naturalness of Controls

Figure D.6 shows the results for how participants perceived the naturalness of controls, when compared to mowing a lawn. Granted, about half of our participants had no lawn mower experience and had to guess. The responses were split roughly three ways between a strong belief that the controls were natural, a very strong belief that the controls were natural, and a strong disbelief in the assertion that the mower controls were natural. Of course, real lawn mowers are not operated by joysticks, but perhaps some of the participants judged that the movement of the mower in the simulation and its interaction with the simulation elements was believable and akin to those expected from a real mower.

Figure D.7 shows the degree to which our participants believed our simulation was realistic. Again, the responses were mixed, with the participants generally believing that the simulation was semi-realistic. Perhaps the presence of simulation elements modeled after mythical creatures (e.g. Leprechauns, Fairies) and our decision to not allow creatures to die when colliding with the mower detracted from the realistic quality of the simulation. However, the purpose of the simulation was not to recreate a real lawn mowing experience, but to facilitate the exhibition of memory-based
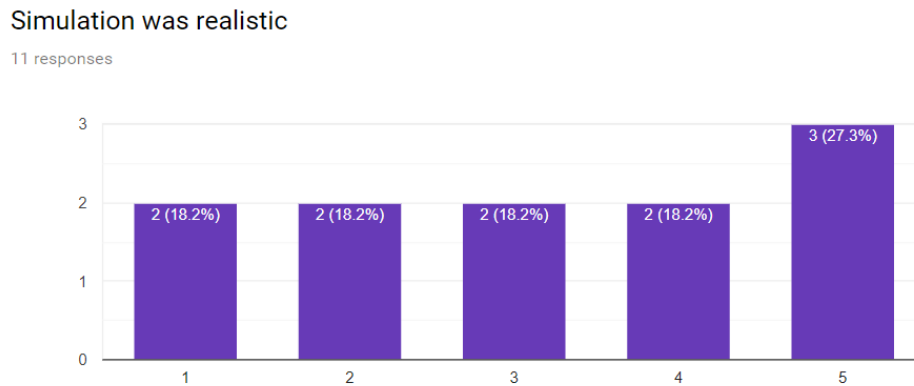
445

behaviors in humans.

Figure D.7: Lawn Mower Survey – Simulation Realistic Perception

Figure D.8 shows how easily the participants understood the mechanics behind the simulation elements. Any response other than "Agree" or "Strongly Agree" is a cause for concern. The simulation was designed to be a soft tutorial for addressing each entity, allowing the participant to learn from mistakes within a scenario (e.g. losing health) and correct them on subsequent attempts on the scenario if necessary. However, this design may be improved for certain simulation entities. Rocks and Grass were intuitive to all participants, while all other entities were at least partially misunderstood by certain participants.

Figure D.9 shows the ease with which each simulation entity's location was inferred by the participants. Generally, most participants found that knowing the locations of the entities was not difficult. Some participants found certain elements difficult to remember (e.g. Ground Hogs, Leprechauns, Fairies). Rocks, which are always visible, were labeled as being "challenging" to infer the location for by one participant, the reason for which is unknown (perhaps due to a misunderstanding of the question).

446
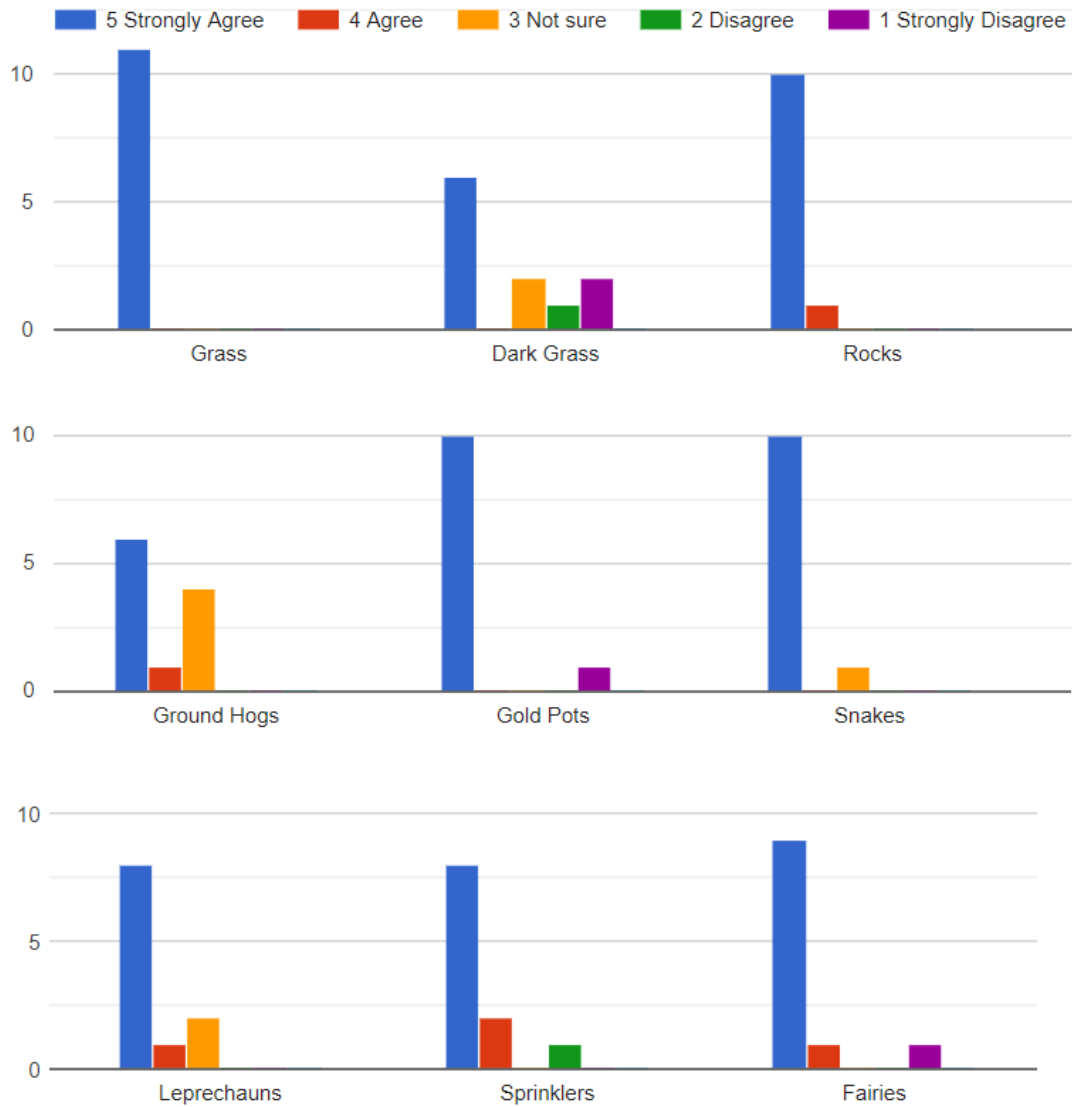
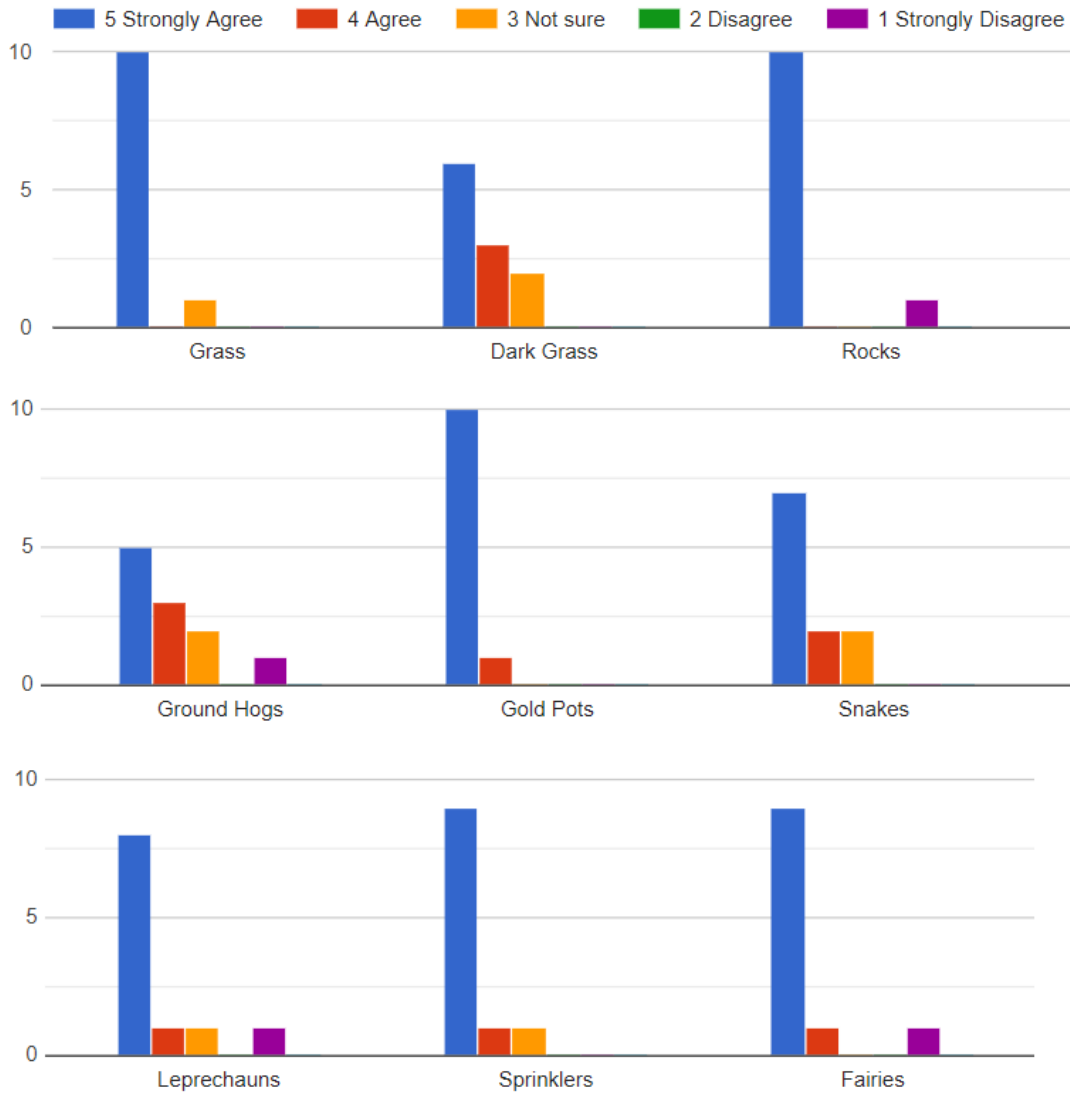Figure D.8: Lawn Mower Survey – Ease of Understanding Mechanics

Figure D.9: Lawn Mower Survey – Ease of Remembering Locations

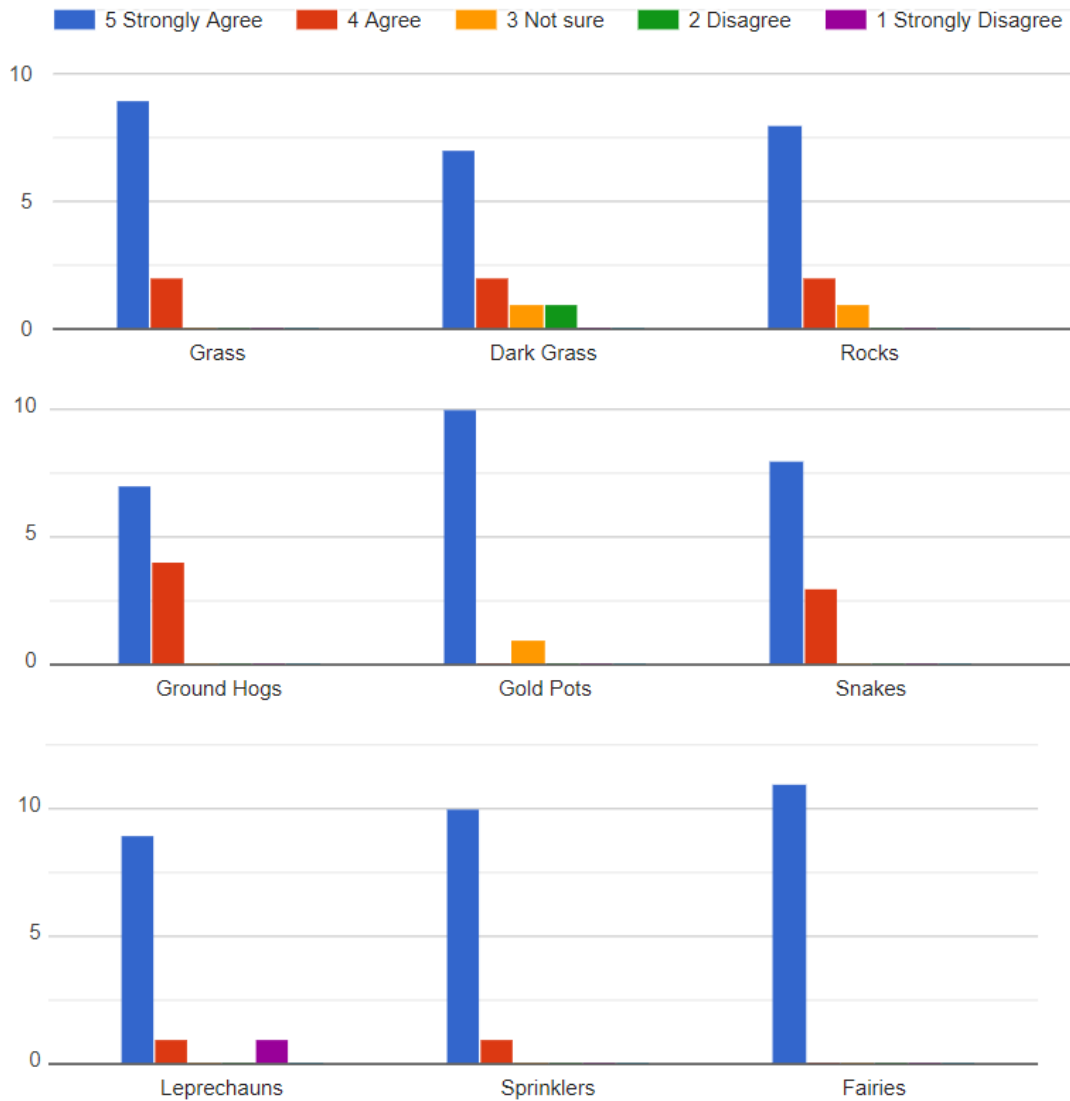The simulation contained a manageable number of...

Figure D.10: Lawn Mower Survey – Manageability of Entity Quantities

Figure D.10 shows how "manageable" the quantities of each simulation entity were perceived to be by the participants. In general, all simulation entities were presented in manageable quantities

449

according to the participant responses. Some entities that may have caused some trouble to partic-
ipants due to their quantity are Dark Grass, Rocks, Gold Pots, and Leprechauns (which received
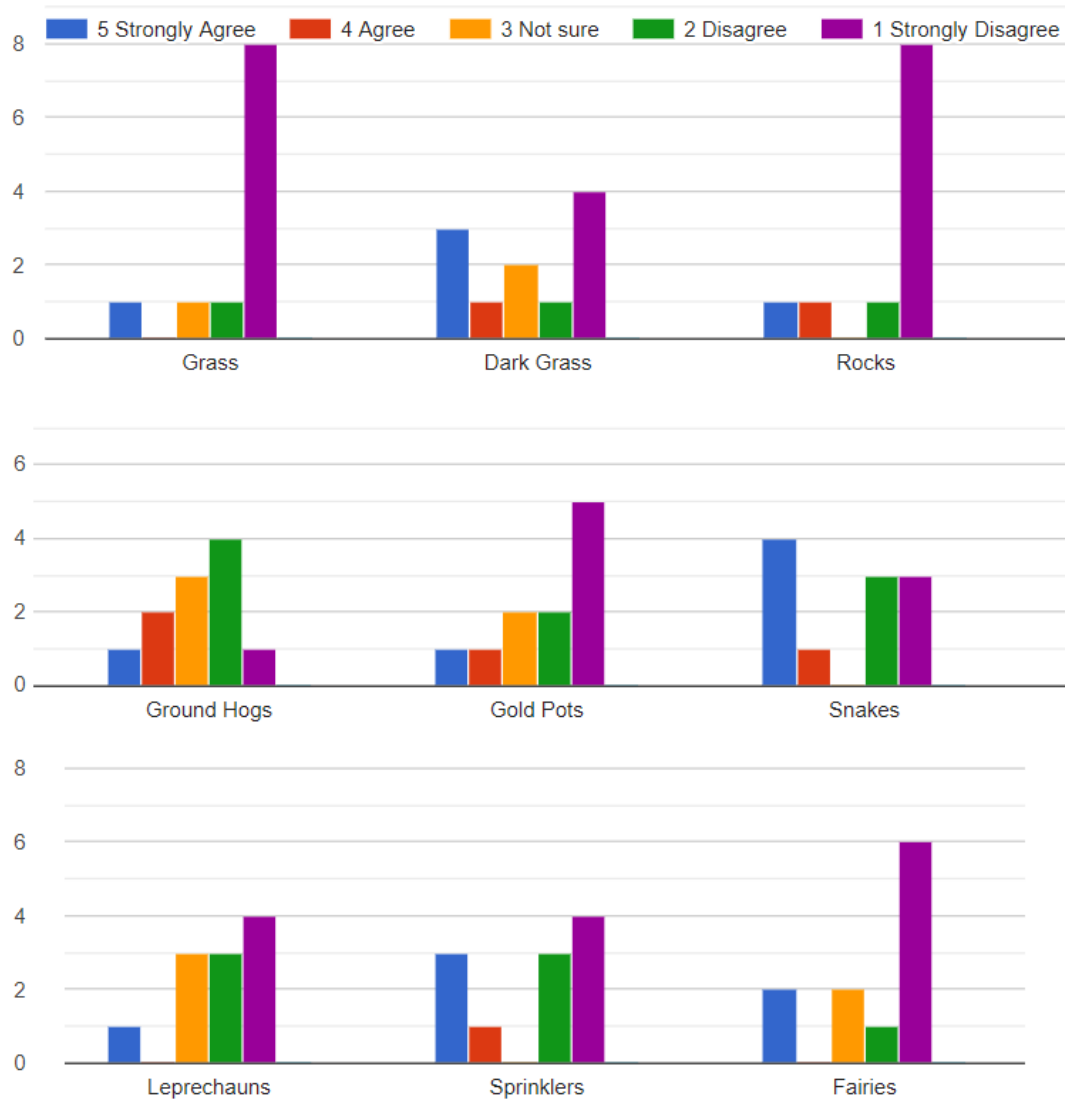one "Strongly Disagree" response).

Figure D.11: Lawn Mower Survey – Required Memory Per Entity

Figure D.11 shows how much memory the participants thought were required to address each simulation entity. The results are varied and we break them down here. Generally, participants thought Grass did not require memory. This is expected, given that grass is always visible. The same trend in responses is seen for Rocks, which too are always visible.

For Dark Grass, participants' responses generally were varied. Most responses labeled Dark Grass as not requiring memory. Some responses strongly thought memory was required. The remaining responses were middle of the road. On one hand, Dark Grass does require the human to remember which patches were mowed, so that patches do not get mowed a second time (which would give no points). However, it is possible to simply follow a path through dark grass regions without explicitly remembering which patches were mowed (i.e. trusting the path to mow all dark grass patches). Even though path following does require one to memorize which part of the path has already been executed, this may not have been perceived by the participants as requiring memory (which was not defined for the participants during the study).

For Ground Hogs and Gold Pots, the static memory elements of the simulation, participants generally though that these elements did not require a significant amount of memory. Perhaps they thought that it did require memory, but that it was very easy to remember. For Gold Pots, perhaps participants collided with the Gold Pots before they could disappear, thus removing the need to memorize their locations once the Gold Pots are consumed.

Snakes and Leprechauns are memory dynamic simulation elements. For Snakes, there was a strong belief by some participants that significant memory was required, but a majority of the participants still labeled Snakes as not requiring significant memory. For Leprechauns, there was less belief that memory was required. The Leprechaun scenario did have fewer grass patches for Leprechauns to hide under. Furthermore, one did not have to avoid the Leprechauns, but simply had to wait for them to come out of hiding.

For Sprinklers and Fairies, the phased memory simulation elements, there were some participants who found that these elements required significant memory. However, there was a strong disbelief of this assertion for Fairies. Because of the large areas of effect for Sprinklers and Fairies, there were fewer of them in a given scenario, so it was easier to track their locations. Furthermore, the memory required for Fairies does not need to be as precise as that for Sprinklers. To effectively address Sprinklers, one must remember the precise boundaries of the spray area so as to avoid damage. However, one simply needs to be within the general vicinity of a Fairy in order to move within the area of effect once the Fairy's spray phase begins (and the Fairy is visible during its spray phase).
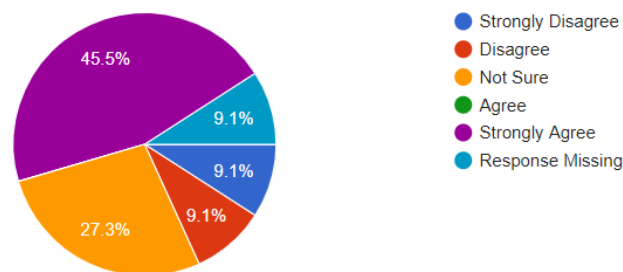

Figure D.12: Lawn Mower Survey – Required Memory Overall

Figure D.12 shows participant sentiment to the statement, "The lawn mower simulation required me to use memory in my mowing behavior." Five participants strongly agreed with the statement. Four were not sure or did not have a response. Two did not agree with the statement. It is thus important for future iterations of this survey to precisely define "memory" for participants, as some participants may have different definitions.

The following open-ended question "What strategy did you use to address (simulation entity) in the simulation?" was asked for each simulation entity. We paraphrase common responses for each simulation entity below.

Strategies for Grass:

- Mow over two patches of Grass simultaneously by mowing between them.

- Mow small areas of grass over multiple rows, rather than column by column.

- Use a zig zag pattern combined with visual confirmation of cut grass.

- Minimize overall economy of motion to reduce overall path length.

Strategies for Dark Grass:

- Memorize which patches were mowed on a patch-specific basis.

- Rely on audio feedback to know if a patch was cut.

- Skip dark grass patches whenever possible.

- Treat dark grass the same as grass and just move on if one forgets which patches were cut.

- Mow less chaotically to remember which patches were cut.

- Trial and error.

Strategies for Rocks:

- Avoid entirely when possible.

- Treat them like a wall obstacle.

- Mow slower near rocks.

- Count how many collisions with rocks occurred.

Strategies for Ground Hogs:

- Remember explicit locations from ground hogs (even from previous attempts on the scenario).

- Mow slower near known ground hog locations.

- Drive quickly over where they appeared once they have disappeared. It should be noted that this last strategy was used by participants who falsely assumed that ground hogs posed no danger once they were hidden from view.

Strategies for Gold Pots:

- Mow over gold pots as soon as they appear.

- Treat gold pots like grass patches.

- Count the number of times gold pot collisions occur.

Strategies for Snakes:

- Move to the furthest grass patch from snake's location.

- Observe movement pattern first, then mow.

- Mow slower near snakes.

- Mow in a path that follows the snake's trajectory.

- Use the passage of time to predict snake locations.

- Memorize the snake's direction of movement and avoid their path if they are approaching.

- Only mow when the snake is invisible.

- Memorize snakes' pattern of movement.

Strategies for Leprechauns:

- Immediately collide with a Leprechaun after colliding with a rock.

- No strategy.

- Count Leprechaun collisions (same as gold pots).

- Wait in a spot on the Leprechaun's trajectory until the Leprechaun collides with the mower.

- Memorize Leprechaun movement patterns.

Strategies for Sprinklers:

- Move once the sprinkler's spraying stops.

- Try to outrun sprinkler spray.

- Avoid. Memorize sprinkler locations from past attempts.

- Find "safe zone" away from sprinkler spray. Use passage of time to predict sprinkler behavior.

Strategies for Fairies:

- Seek out a Fairy immediately after colliding with a Rock.

- Mow over anything (e.g. grass, rocks) to get to a Fairy.

- Use health level to gauge when a Fairy is needed.

- Avoid if possible. (This strategy was probably listed by a person who did not understand the Fairy mechanic.)

The second open-ended question "If you could play the game again, what would you do differently?" received the following responses:

- Play the same way.

- Move faster.

- Read the directions.

- Count the number of rock collisions better.

- Pay more attention to the user-interface bar.

- Move slower.

- Practice using joysticks.

- Relax.

The third open-ended question "How would you improve the simulation?" received the following responses:

- Use better graphics (e.g. sprites with transparency, 3D assets).

- Have a joystick that requires more force to move.

- Have functionality to enable sharp stops in the mower.

- Have functionality to enable faster mower movement.

- Make a tutorial video.

Regarding the comments about joystick stiffness, several participants found that the mower would "drift" in the simulation, even when the human was not providing any joystick inputs. This was because the joystick's "haptic" mode was disabled during the study and the force of gravity allowed the joystick to tilt slightly when the human wasn't moving it. We originally decided to disable haptic mode because we found that the joystick was difficult to move when haptic mode was enabled. However, future iterations of the lawn mower simulation could enable haptic mode, but have the mower move faster in the simulation so that less force is required to achieve maximum speed.

We received the following responses for the final open-ended question, "Please provide any final thoughts or comments about your experience."

- The game was fun/cool/insightful/enjoyable.

- Prior video game experience made this simulation easy to handle.

- The research team should account for how the human uses memory of prior attempts on a scenario when going through the scenario again.

- The mower could have an optional self-propelling mechanic.

- The simulation could be used for improving memory.

457

APPENDIX E: PERMISSIONS FOR REPRINTING FIGURES
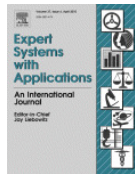
RightsLink®

A Dynamic-Bayesian Network framework for modeling and evaluating learning from observation

Author: Santiago Ontañón,José L. Montaña,Avelino J. Gonzalez

Publication: Expert Systems with Applications

Publisher: Elsevier

Date: 1 September 2014

Review Order

Please review the order details and the associated terms and conditions.

No royalties will be charged for this reuse request although you are required to obtain a license and comply with the license terms and conditions. To obtain the license, click the Accept button below.

☑ Licensed Content

Licensed Content Publisher	Elsevier
Licensed Content Publication	Expert Systems with Applications
Licensed Content Title	A Dynamic-Bayesian Network framework for modeling and evaluating learning from observation
Licensed Content Author	Santiago Ontañón,José L. Montaña,Avelino J. Gonzalez
Licensed Content Date	1 September 2014
Licensed Content Volume	41
Licensed Content Issue	11
Licensed Content Pages	15
Journal Type	S&T

Order Details

Type of Use	reuse in a thesis/dissertation
Portion	figures/tables/illustrations
Number of figures/tables/illustrations	2
Format	electronic
Are you the author of this Elsevier article?	No
Will you be translating?	No

About Your Work

Title	Learning Internal State Memory Representations from Observation
Institution name	University of Central Florida
Expected presentation date	Jan 2020

Additional Data

Portions	Figures 1 and 7

459

ELSEVIER LICENSE
TERMS AND CONDITIONS

Nov 18, 2019

This Agreement between Josiah Wong ("You") and Elsevier ("Elsevier") consists of your license details and the terms and conditions provided by Elsevier and Copyright Clearance Center.

License Number   4712240968903

License date   Nov 18, 2019

Licensed Content Publisher Elsevier

Licensed Content
Publication   Expert Systems with Applications

Licensed Content Title   A Dynamic-Bayesian Network framework for modeling and
                          evaluating learning from observation

Licensed Content Author   Santiago Ontañón,José L. Montaña,Avelino J. Gonzalez

Licensed Content Date   Sep 1, 2014

Licensed Content Volume   41

Licensed Content Issue   11

Licensed Content Pages   15

Start Page   5212

End Page   5226

A survey of robot learning from demonstration

Author: Brenna D. Argall,Sonia Chernova,Manuela Veloso,Brett Browning

Publication: Robotics and Autonomous Systems

Publisher: Elsevier

Date: 31 May 2009

Order Completed

Thank you for your order.

This Agreement between Josiah Wong ("You") and Elsevier ("Elsevier") consists of your license details and the terms and conditions provided by Elsevier and Copyright Clearance Center.

Your confirmation email will contain your order number for future reference.

License Number	4713410762087	🖨 Printable Details
License date	Nov 20, 2019	

✔ Licensed Content

Licensed Content Publisher	Elsevier
Licensed Content Publication	Robotics and Autonomous Systems
Licensed Content Title	A survey of robot learning from demonstration
Licensed Content Author	Brenna D. Argall,Sonia Chernova,Manuela Veloso,Brett Browning
Licensed Content Date	May 31, 2009
Licensed Content Volume	57
Licensed Content Issue	5
Licensed Content Pages	15
Journal Type	S&T

🗎 Order Details

Type of Use	reuse in a thesis/dissertation
Portion	figures/tables/illustrations
Number of figures/tables/illustrations	1
Format	electronic
Are you the author of this Elsevier article?	No
Will you be translating?	No

🗎 About Your Work

Title	Learning Internal State Memory Representations from Observation
Institution name	University of Central Florida
Expected presentation date	Jan 2020

📁 Additional Data

Portions	Figure 5

461

Copyright
Clearance
Center

RightsLink®

🏠
Home

❓
Help

📞
Live Chat

👤
Josiah Wong ⌄

Neural Networks

SPRINGER NATURE

Author: Alex Graves
Publication: Springer eBook
Publisher: Springer Nature
Date: Jan 1, 2012

Copyright © 2012, Springer-Verlag GmbH Berlin Heidelberg
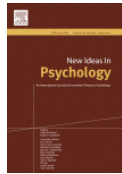
Order Completed

Thank you for your order.

This Agreement between Josiah Wong ("You") and Springer Nature ("Springer Nature") consists of your license details and the terms and conditions provided by Springer Nature and Copyright Clearance Center.

Your confirmation email will contain your order number for future reference.

License Number	4713440842466	🖨 Printable Details
License date	Nov 21, 2019	

☑ Licensed Content

Licensed Content Publisher	Springer Nature
Licensed Content Publication	Springer eBook
Licensed Content Title	Neural Networks
Licensed Content Author	Alex Graves
Licensed Content Date	Jan 1, 2012

📄 Order Details

Type of Use	Thesis/Dissertation
Requestor type	academic/university or research institute
Format	electronic
Portion	figures/tables/illustrations
Number of figures/tables/illustrations	5
Will you be translating?	no
Circulation/distribution	200 - 499
Author of this Springer Nature content	no

📄 About Your Work

Title	Learning Internal State Memory Representations from Observation
Institution name	University of Central Florida
Expected presentation date	Jan 2020

📁 Additional Data

Portions	Figures 3.3, 3.4, 4.1, 4.2, and 4.4

Memory systems within a cognitive architecture

Author: Ron Sun

Publication: New Ideas in Psychology

Publisher: Elsevier

Date: August 2012

Order Completed

Thank you for your order.

This Agreement between Josiah Wong ("You") and Elsevier ("Elsevier") consists of your license details and the terms and conditions provided by Elsevier and Copyright Clearance Center.

Your confirmation email will contain your order number for future reference.

License Number	4713430579917
License date	Nov 21, 2019

🖨 Printable Details

☑ Licensed Content

Licensed Content Publisher	Elsevier
Licensed Content Publication	New Ideas in Psychology
Licensed Content Title	Memory systems within a cognitive architecture
Licensed Content Author	Ron Sun
Licensed Content Date	Aug 1, 2012
Licensed Content Volume	30
Licensed Content Issue	2
Licensed Content Pages	14
Journal Type	S&T

📋 Order Details

Type of Use	reuse in a thesis/dissertation
Portion	figures/tables/illustrations
Number of figures/tables/illustrations	1
Format	electronic
Are you the author of this Elsevier article?	No
Will you be translating?	No

📄 About Your Work

Title	Learning Internal State Memory Representations from Observation
Institution name	University of Central Florida
Expected presentation date	Jan 2020

📂 Additional Data

Portions	Figure 2

Copyright Clearance Center

RightsLink®

Home

Help

Live Chat

Josiah Wong ∨

What Learning Systems do Intelligent Agents Need? Complementary Learning Systems Theory Updated

Author: Dharshan Kumaran,Demis Hassabis,James L. McClelland

Publication: Trends in Cognitive Sciences

Publisher: Elsevier

Date: July 2016

Order Completed

Thank you for your order.

This Agreement between Josiah Wong ("You") and Elsevier ("Elsevier") consists of your license details and the terms and conditions provided by Elsevier and Copyright Clearance Center.

Your confirmation email will contain your order number for future reference.

🖶 Printable Details

License Number	4713430783539
License date	Nov 21, 2019

✅ Licensed Content

Licensed Content Publisher	Elsevier
Licensed Content Publication	Trends in Cognitive Sciences
Licensed Content Title	What Learning Systems do Intelligent Agents Need? Complementary Learning Systems Theory Updated
Licensed Content Author	Dharshan Kumaran,Demis Hassabis,James L. McClelland
Licensed Content Date	Jul 1, 2016
Licensed Content Volume	20
Licensed Content Issue	7
Licensed Content Pages	23
Journal Type	S&T

📋 Order Details

Type of Use	reuse in a thesis/dissertation
Portion	figures/tables/illustrations
Number of figures/tables/illustrations	1
Format	electronic
Are you the author of this Elsevier article?	No
Will you be translating?	No

📄 About Your Work

Title	Learning Internal State Memory Representations from Observation
Institution name	University of Central Florida
Expected presentation date	Jan 2020

📁 Additional Data

Portions	Figure 1

Copyright Clearance Center

RightsLink®

Home

Help

Live Chat

Josiah Wong ⌄

Formalizing context-based reasoning: A modeling paradigm for representing tactical human behavior

Author: Gilbert Barrett, Brian S. Stensrud, Avelino J. Gonzalez

Publication: International Journal of Intelligent Systems

Publisher: John Wiley and Sons

Date: May 28, 2008

Order Completed

Thank you for your order.

This Agreement between Josiah Wong ("You") and John Wiley and Sons ("John Wiley and Sons") consists of your license details and the terms and conditions provided by John Wiley and Sons and Copyright Clearance Center.

Your confirmation email will contain your order number for future reference.

🖨 Printable Details

License Number	4713431041748
License date	Nov 21, 2019

☑ Licensed Content

Licensed Content Publisher	John Wiley and Sons
Licensed Content Publication	International Journal of Intelligent Systems
Licensed Content Title	Formalizing context-based reasoning: A modeling paradigm for representing tactical human behavior
Licensed Content Author	Gilbert Barrett, Brian S. Stensrud, Avelino J. Gonzalez
Licensed Content Date	May 28, 2008
Licensed Content Volume	23
Licensed Content Issue	7
Licensed Content Pages	26

▤ Order Details

Type of use	Dissertation/Thesis
Requestor type	University/Academic
Format	Electronic
Portion	Figure/table
Number of figures/tables	3
Will you be translating?	No

▤ About Your Work

Title of your thesis / dissertation	Learning Internal State Memory Representations from Observation
Expected completion date	Jan 2020
Expected size (number of pages)	1

🗁 Additional Data

Original Wiley figure/table number(s)	Figures 1, 2, and 3

465

Context Dynamic and Explanation in Contextual Graphs

SPRINGER NATURE

Author: Patrick Brézillon

Publication: Springer eBook

Publisher: Springer Nature

Date: Jan 1, 2003

Copyright © 2003, Springer-Verlag Berlin Heidelberg

Order Completed

Thank you for your order.

This Agreement between Josiah Wong ("You") and Springer Nature ("Springer Nature") consists of your license details and the terms and conditions provided by Springer Nature and Copyright Clearance Center.

Your confirmation email will contain your order number for future reference.

License Number	4713240007531	🖨 Printable Details
License date	Nov 20, 2019	

☑ Licensed Content

Licensed Content Publisher	Springer Nature
Licensed Content Publication	Springer eBook
Licensed Content Title	Context Dynamic and Explanation in Contextual Graphs
Licensed Content Author	Patrick Brézillon
Licensed Content Date	Jan 1, 2003

📋 Order Details

Type of Use	Thesis/Dissertation
Requestor type	academic/university or research institute
Format	electronic
Portion	figures/tables/illustrations
Number of figures/tables/illustrations	1
Will you be translating?	no
Circulation/distribution	200 - 499
Author of this Springer Nature content	no

📄 About Your Work

Title	Learning Internal State Memory Representations from Observation
Institution name	University of Central Florida
Expected presentation date	Jan 2020

📁 Additional Data

Portions	Figure 1

466

Re: Request to reprint figure

JR H
Wed 11/20/2019 5:52 PM
To: Josiah Wong <josiah.w.ucf@knights.ucf.edu>

Josiah,

You have my permission to use figure 21 from my dissertation.

James Hollister, PhD

From: Josiah Wong <josiah.w.ucf@knights.ucf.edu>
Sent: Wednesday, November 20, 2019 2:26 PM
To: JR H
Subject: Request to reprint figure

Dear Dr. James Hollister,

Greetings! I am a Ph.D. student at the University of Central Florida (UCF). I would like include in my dissertation Figure 21 from Chapter 4 (page 114) of your dissertation in my discussion of your Cooperating Context Method. My dissertation, titled "Learning Internal State Memory Representations from Observation", includes a discussion of various contextual reasoning frameworks, including the one you proposed in your dissertation. My dissertation will be published electronically by UCF.

Would you be willing to grant me permission to reprint this figure in my dissertation? Thank you and have a great day.

Sincerely,
Josiah Wong

Discovering Contexts from Observed Human Performance

Author: Viet C. Trinh

Publication: Human-Machine Systems, IEEE Transactions on

Publisher: IEEE

Date: July 2013

Copyright © 2013, IEEE

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

BACK | CLOSE

Copyright
Clearance
Center

RightsLink®

Home

? Help

Live Chat

Sign in

Create Account

Convolutional networks and applications in vision

Conference Proceedings:
Proceedings of 2010 IEEE International Symposium on Circuits and Systems

Author: Yann LeCun

Publisher: IEEE

Date: May 2010

Copyright © 2010, IEEE

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

BACK   CLOSE

LIST OF REFERENCES

[1] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. *Proceedings of the Twenty-First International Conference on Machine Learning*, pages 1–8. ACM, 2004.

[2] D. Aihe and A. Gonzalez. Context-driven reinforcement learning. *Proceedings of the Second Swedish-American Workshop on Modeling and Simulation*, 2004.

[3] J. R. Anderson. Act: A simple theory of complex cognition. *American psychologist*, 51(4):355, 1996.

[4] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

[5] M. A. Bauer. *Programming by Examples*, pages 317–327. Elsevier, 1986.

[6] D. C. Bentivegna and C. G. Atkeson. Learning from observation using primitives. *International Conference on Robotics and Automation (ICRA)*, volume 2, pages 1988–1993. IEEE.

[7] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[8] J. Bongard and H. Lipson. Automated reverse engineering of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 104(24):9943–9948, 2007.

[9] P. Brézillon. Context in problem solving: A survey. *The Knowledge Engineering Review*, 14(1):47–80, 1999.

[10] P. Brézillon. Context dynamic and explanation in contextual graphs. *International and Interdisciplinary Conference on Modeling and Using Context*, pages 94–106. Springer, 2003.

[11] C. Brom and J. Lukavsky. Towards virtual characters with a full episodic memory ii: The episodic memory strikes back. *Proc. empathic agents, AAMAS workshop*, pages 1–9, 2009.

[12] J. Campos and A. Paiva. May: My memories are yours. *International Conference on Intelligent Virtual Agents*, pages 406–412. Springer, 2010.

[13] T. G. Dietterich. Machine learning for sequential data: A review. *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 15–30. Springer, 2002.

[14] R. Dillmann, O. Rogalla, M. Ehrenmann, R. Zliner, and M. Bordegoni. *Learning robot behaviour and skills based on human demonstration and advice: the machine learning paradigm*, pages 229–238. Springer, 2000.

[15] M. Elvir, A. J. Gonzalez, C. Walls, and B. Wilder. Remembering a conversationa conversational memory architecture for embodied conversational agents. *Journal of Intelligent Systems*, 26(1):1–21, 2017.

[16] Z. Faltersack, B. Burns, A. Nuxoll, and T. L. Crenshaw. Ziggurat: Steps toward a general episodic memory. *AAAI fall symposium: Advances in cognitive systems*, pages 106–111, 2011.

[17] H. Fernlund, A. J. Gonzalez, J. Ekblad, and A. Rodriguez. Trainee evaluation through after-action review by comparison. *The Journal of Defense Modeling and Simulation*, 6(3):135–150, 2009.

[18] H. K. Fernlund, A. J. Gonzalez, M. Georgiopoulos, and R. F. DeMara. Learning tactical human behavior through observation of human performance. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 36(1):128–140, 2006.

[19] H. K. G. Fernlund. Evolving models from observed human performance. 2004.

[20] M. Floyd. *A General-Purpose Framework for Learning by Observation*. Carleton University, 2013.

[21] A. Giusti, J. Guzzi, D. C. Cireşan, F.-L. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. Di Caro, et al. A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotics and Automation Letters*, 1(2):661–667, 2015.

[22] A. J. Gonzalez and D. D. Dankel. *The Engineering of Knowledge-Based Systems, Theory and Practice*. Prentice Hall, New Jersey, 1993.

[23] A. J. Gonzalez, J. Nguyen, S. Tsuruta, Y. Sakurai, K. Takada, and K. Uchida. Using contexts to supervise a collaborative process. *Systems, Man and Cybernetics, 2008. SMC 2008. IEEE International Conference on*, pages 2706–2711. IEEE, 2008.

[24] A. J. Gonzalez, B. S. Stensrud, and G. Barrett. Formalizing context-based reasoning: A modeling paradigm for representing tactical human behavior. *International Journal of Intelligent Systems*, 23(7):822–847, 2008.

[25] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Advances in neural information processing systems*, pages 2672–2680, 2014.

[26] A. Graves. Supervised sequence labelling. *Supervised sequence labelling with recurrent neural networks*, pages 5–13. Springer, 2012.

[27] A. S. E. Gunaratne, B. Esfandiari, and A. Fawaz. A case-based reasoning approach to learning state-based behavior. *The Thirty-First International Florida Artificial Intelligence Research Society Conference (FLAIRS-31)*, pages 377–382. AAAI, 2018.

[28] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten. The weka data mining software: An update. 2009.

[29] A. Henninger, A. Gonzalez, M. Georgiopoulos, and R. DeMara. The limitations of static performance metrics for dynamic tasks learned through observation. 2001.

[30] J. Ho and S. Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, pages 4565–4573, 2016.

[31] W. C. Ho, K. Dautenhahn, M. Y. Lim, P. A. Vargas, R. Aylett, and S. Enz. An initial memory model for virtual and robot companions supporting migration and long-term interaction. *The 18th IEEE International Symposium on Robot and Human Interactive Communication*, pages 277–284. IEEE, 2009.

[32] W. C. Ho, K. Dautenhahn, and C. L. Nehaniv. Computational memory architectures for autobiographic agents interacting in a complex virtual environment: a working model. *Connection Science*, 20(1):21–65, 2008.

[33] J. Hollister. *A contextual approach to real time interactive narrative generation*. Thesis, 2016.

[34] A. Jackson and G. Sukthankar. Learning continuous state/action models for humanoid robots. *FLAIRS conference*, pages 392–397, 2016.

[35] C. L. Johnson and A. J. Gonzalez. Learning collaborative team behavior from observation. *Expert Systems with Applications*, 41(5):2316–2328, 2014.

[36] F. Kamrani, L. J. Luotsinen, and R. A. Lvlid. Learning objective agent behavior using a data-driven modeling approach. *Systems, Man, and Cybernetics (SMC), 2016 IEEE International Conference on*, pages 002175–002181. IEEE, 2016.

[37] B. Kim and J. Lee. Analysis of enumeration strategy use in the act-r cognitive architecture. *Natural Computation, 2007. ICNC 2007. Third International Conference on*, volume 5, pages 526–530. IEEE.

[38] D. Kumaran, D. Hassabis, and J. L. McClelland. What learning systems do intelligent agents need? complementary learning systems theory updated. *Trends in cognitive sciences*, 20(7):512–534, 2016.

[39] T. Knik and J. E. Laird. Learning goal hierarchies from structured observations and expert annotations. *Machine Learning*, 64(1-3):263–287, 2006.

[40] J. E. Laird, A. Newell, and P. S. Rosenbloom. Soar: An architecture for general intelligence. *Artificial intelligence*, 33(1):1–64, 1987.

[41] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[42] Y. LeCun, K. Kavukcuoglu, and C. Farabet. Convolutional networks and applications in vision. *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 253–256. IEEE, 2010.

[43] M. Y. Lim. *Memory models for intelligent social companions*, pages 241–262. Springer, 2012.

[44] M. Y. Lim, R. Aylett, W. C. Ho, S. Enz, and P. Vargas. A socially-aware memory for companion agents. *International Workshop on Intelligent Virtual Agents*, pages 20–26. Springer, 2009.

[45] Y. Liu, A. Gupta, P. Abbeel, and S. Levine. Imitation from observation: Learning to imitate behaviors from raw video via context translation. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1118–1125. IEEE, 2018.

[46] R. A. Løvlid, S. Bruvoll, K. Brathen, and A. Gonzalez. Modeling the behavior of a hierarchy of command agents with context-based reasoning. *The Journal of Defense Modeling and Simulation*, 2017.

[47] Merriam-Webster. Merriam-webster dictionary, 2017.

[48] R. S. Michalski and R. E. Stepp. *Learning from observation: Conceptual clustering*, pages 331–363. Springer, 1983.

[49] A. Mohammed Ali. *Machine learning from casual conversation*. Thesis, 2019.

[50] C. L. Moriarty and A. J. Gonzalez. Learning human behavior from observation for gaming applications. *FLAIRS Conference*, 2009.

[51] S. Nakaoka, A. Nakazawa, F. Kanehiro, K. Kaneko, M. Morisawa, H. Hirukawa, and K. Ikeuchi. Learning from observation paradigm: Leg task models for enabling a biped humanoid robot to imitate human dances. *The International Journal of Robotics Research*, 26(8):829–844, 2007.

[52] A. Y. Ng and S. J. Russell. Algorithms for inverse reinforcement learning. *ICML*, pages 663–670, 2000.

[53] M. Norouzitallab, A. Javari, M. Mohajer, S. Ismael, and S. Salehizadeh. Nemesis 2011 team description paper.

[54] S. Ontañón. Case acquisition strategies for case-based reasoning in real-time strategy games. *Twenty-Fifth International FLAIRS Conference*, 2012.

[55] S. Ontañón and M. Floyd. A comparison of case acquisition strategies for learning from observations of state-based experts. *The Twenty-Sixth International FLAIRS Conference*, 2013.

[56] S. Ontañón, Y.-C. Lee, S. Snodgrass, D. Bonfiglio, F. K. Winston, C. McDonald, and A. J. Gonzalez. Case-based prediction of teen driver behavior and skill. *International Conference on Case-Based Reasoning*, pages 375–389. Springer, 2014.

[57] S. Ontañón, J. Montaña, and A. Gonzalez. Towards a unified framework for learning from observation. *IJCAI Workshop on Agents Learning Interactively from Human Teachers*, 2011.

[58] S. Ontañón, J. L. Montaña, and A. J. Gonzalez. A dynamic-bayesian network framework for modeling and evaluating learning from observation. *Expert Systems with Applications*, 41(11):5212–5226, 2014.

[59] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[60] D. A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. *Advances in neural information processing systems*, pages 305–313, 1989.

[61] R. Raina, A. Battle, H. Lee, B. Packer, and A. Y. Ng. Self-taught learning: transfer learning from unlabeled data. *Proceedings of the 24th international conference on Machine learning*, pages 759–766. ACM.

[62] S. Ross and D. Bagnell. Efficient reductions for imitation learning. *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668, 2010.

[63] C. Sammut, S. Hurst, D. Kedzier, and D. Michie. *Learning to fly*, pages 385–393. Elsevier, 1992.

[64] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap. Meta-learning with memory-augmented neural networks. *International conference on machine learning*, pages 1842–1850, 2016.

[65] S. Schaal. Is imitation learning the route to humanoid robots? *Trends in cognitive sciences*, 3(6):233–242, 1999.

[66] T. A. Sidani and A. J. Gonzalez. A framework for learning implicit expert knowledge through observation. *Transactions of the Society for Computer Simulation*, 17(2):54–72, 2000.

[67] P. Sondhi. Feature construction methods: a survey. *sifaka. cs. uiuc. edu*, 69:70–71, 2009.

[68] K. Stanley, N. Kohl, R. Sherony, and R. Miikkulainen. Neuroevolution of an automobile crash warning system. *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, pages 1977–1984. ACM, 2005.

[69] G. Stein. Falconet: Force-feedback approach for learning from coaching and observation using natural and experiential training. 2009.

[70] G. Stein and A. J. Gonzalez. Building high-performing human-like tactical agents through observation and experience. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 41(3):792–804, 2011.

[71] G. Stein and A. J. Gonzalez. Learning in context: enhancing machine learning with context-based reasoning. *Applied intelligence*, 41(3):709–724, 2014.

[72] B. S. Stensrud and A. J. Gonzalez. Discovery of high-level behavior from observation of human performance in a strategic game. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 38(3):855–874, 2008.

[73] R. Sun. Memory systems within a cognitive architecture. *New Ideas in Psychology*, 30(2):227–240, 2012.

[74] W. Sun, A. Vemula, B. Boots, and J. A. Bagnell. Provably efficient imitation learning from observation alone. *arXiv preprint arXiv:1905.10948*, 2019.

[75] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 1998.

[76] C. Tîrnăucă, J. L. Montaña, S. Ontañón, A. J. Gonzalez, and L. M. Pardo. Behavioral modeling based on probabilistic finite automata: An empirical study. *Sensors*, 16(7):958, 2016.

[77] F. Torabi, S. Geiger, G. Warnell, and P. Stone. Sample-efficient adversarial imitation learning from observation. *arXiv preprint arXiv:1906.07374*, 2019.

[78] F. Torabi, G. Warnell, and P. Stone. Behavioral cloning from observation. *arXiv preprint arXiv:1805.01954*, 2018.

[79] F. Torabi, G. Warnell, and P. Stone. Recent advances in imitation learning from observation. *arXiv preprint arXiv:1905.13566*, 2019.

[80] J. G. Trafton, A. C. Schultz, M. Bugajska, and F. Mintz. Perspective-taking with robots: experiments and models. *ROMAN 2005. IEEE International Workshop on Robot and Human Interactive Communication, 2005.*, pages 580–584. IEEE, 2005.

[81] V. Trinh. *Contextualizing observational data for modeling human performance*. Thesis, 2009.

[82] V. C. Trinh and A. J. Gonzalez. Discovering contexts from observed human performance. *IEEE Transactions on Human-Machine Systems*, 43(4):359–370, 2013.

[83] R. M. Turner. Context-mediated behavior: An approach to explicitly representing contexts and contextual knowledge for ai applications. *Working Notes of the AAAI-99 Workshop on Modeling and Using Context in AI Applications, AAAI Technical Report*, 1999.

[84] M. van Lent and J. Laird. Learning by observation in complex domains. *Ann Arbor*, 1001:48109, 1998.

[85] X. Wang. *Learning by observation and practice: An incremental approach for planning operator acquisition*, pages 549–557. Elsevier, 1995.

[86] Wiktionary. Wiktionary, the free dictionary, 2017.

[87] J. Wong and A. J. Gonzalez. Learning behavioral memory representations from observation. *32nd International Florida Artificial Intelligence Research Society Conference. AAAI Press*, pages 86–91, 2019.

[88] J. Wong, L. Hastings, K. Negy, A. J. Gonzalez, S. Ontañón, and Y.-C. Lee. Machine learning from observation to detect abnormal driving behavior in humans. *31st International Florida Artificial Intelligence Research Society Conference. AAAI Press*, pages 152–157.

[89] M. Woolridge. Multi-agent systems: An introduction, 2001.