Famtile: An Algorithm For Learning High-level Tactical Behavior From Observation

Brian Stensrud
University of Central Florida

FAMTILE: An Algorithm for Learning High-Level Tactical Behavior from Observation

by

Brian S. Stensrud

B.S. Mathematics, University of Florida, 2001
B.S. Computer Engineering, University of Florida, 2001
B.S. Electrical Engineering, University of Florida, 2001
M.S. Computer Engineering, University of Central Florida, 2003

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the School of Electrical and Computer Engineering
in the College of Engineering
at the University of Central Florida
Orlando, Florida

Spring Term
2005

Major Professor: Avelino J. Gonzalez

Abstract

This research focuses on the learning of a class of behaviors defined as high-level behaviors. High-level behaviors are defined here as behaviors that can be executed using a sequence of identifiable behaviors. Represented by low-level contexts, these behaviors are known *a priori* to learning and can be modeled separately by a knowledge engineer. The learning task, which is achieved by observing an expert within simulation, then becomes the identification and representation of the low-level context sequence executed by the expert. To learn this sequence, this research proposes FAMTILE - the Fuzzy ARTMAP / Template-Based Interpretation Learning Engine. This algorithm attempts to achieve this learning task by constructing rules that govern the low-level context transitions made by the expert. By combining these rules with models for these low-level context behaviors, it is hypothesized that an intelligent model for the expert can be created that can adequately model his behavior.

To evaluate FAMTILE, four testing scenarios were developed that attempt to achieve three distinct evaluation goals: assessing the learning capabilities of Fuzzy ARTMAP, evaluating the ability of FAMTILE to correctly predict expert actions and context choices given an observation, and creating a model of the expert's behavior that can perform the high-level task at a comparable level of proficiency.

For my poo

Acknowledgments

Thanks to my advisor, Avelino, and to each member of my committee who put up with me coming in for update meetings. Thanks also to Amy Henninger, who found me a job when I ran out of dissertation funding and *then* helped me stick around until I was finished! Thanks to my maze and poker experts - Britt, Chad, Gil, Kacey, Kevin, and Mike - Howard Lederer and Jesus Ferguson just wouldn't return my phone calls, so you guys were the next-best thing. Thanks to Gil Barrett, who has been my partner in sorrow these past 3+ years - I appreciate your support, your friendship, and your, um, unique perspective. Thanks to my parents, Mike and Lynn Stensrud, and to my grandparents, Philip C. Stanley, Jr. and Frances. I've been in school now for nearly 22 (!) years, and my main reason for staying with it and working hard has been to make you proud. Finally, thanks to my mother-in-law, Kerry. You went out of your way to help get me started on this journey - oh, and you also allowed me to marry your daughter!

Table of Contents

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

This research explores the problem of developing a learning system that can learn portions of tactical human behavior merely by observing an expert perform the behavior within a simulation. The term *tactical behavior*, often reserved for behaviors involving military or war-related operations, is defined here to denote behaviors that:

- A well-defined goal or *mission*
- Are characterized by planning and/or maneuvering
- Are not well-defined as to their execution sequence, and thus their characteristics may vary greatly across individuals

For this research, we are interested in a subset of tactical behaviors which characterized by the execution of an undetermined sequence of sub-behaviors. These sub-behaviors possess the following characteristics:

- Can be identified and modeled by a knowledge engineer without the assistance of the expert under study
- Are atomic in that no two sub-behaviors co-exist at the same point in time

These type of situations are easily found when we consider tactical human behavior. The task of flying an airplane, for example, can be broken down into, in the most extreme case, trivial low-level actions - pushing buttons, guiding a throttle stick in a certain direction, etc. However, flying an airplane is certainly NOT a trivial task. The real knowledge is contained in the processes involved in deciding when to push a button, when to pull back on the throttle, etc., and in what sequence, depending on the situation at-hand. The knowledge is so complex, in fact, that there are hierarchies of sub-tasks that play a role in representing the behavior of flying a plane. Learning to fly is not achieved by learning button-pushing and throttle-maneuvering techniques per-say, but it is achieved by learning sub-tasks that involve those techniques - landing, taking-off, maintaining a heading, etc., and when to initiate these actions.

The argument posed by this example is that if we can identify and replicate the low-level functionality of the expert, learning his tactical expertise becomes an exercise in identifying a mapping between environmental and situational cues, which we will call *expert stimuli*, and the low-level function or behavior that the expert chooses in response to that cue.

The overall behaviors to be learned by the proposed system, therefore, are considered to be as *high-level* behaviors. The precise definition of a high-level behavior usually omitted in papers in spite of the fact that their implementation is a primary focus of the work. Jones and Laird refer to high-level behavior when describing the TacAir-Soar system ([JK99], [JL96]) but never define the term explicitly. Likewise, the work reported by Patterson et al ([PK03]) describes a method for learning high-level behavior by examining low-level sensors, and stops short of providing a definition of high-level behavior. A common thread found in all of the literature, however, is the presence of sub-behaviors that compose the high-level behavior described. In ([JL96]), the behavior of piloting a fixed-wing aircraft is described in terms of the composition of its lower-level functionality, such as communication, maneuvering the plane, etc.

For this research, high-level behaviors will be explicitly defined as as behaviors that can be represented by a sequence of simpler, identifiable sub-behaviors known as *low-level* behaviors. If it is assumed that each low-level behavior can be modeled and identified *a priori* by a knowledge engineer, the learning task becomes identifying the cues that determine the sequence in which those low-level behaviors are executed by the observed expert.

Proposed here is a learning system that gathers a sequence of observations from an expert performing one of these high-level behaviors (or high-level *tasks*). By examining the observations, the proposed system aims to correctly identify the low-level behaviors being executed (without feedback from the expert himself) and map them to stimuli within the observations that prompted their selection. With the help of a modeling paradigm, this proposed system can then be used to develop intelligent models of the learned high-level behavior. For this research, the *Context-Based Reasoning* (CxBR) paradigm [SG04] is used. Using CxBR, low-level behaviors are represented as individual *contexts*, while the high-level behavior to be modeled is represented using CxBR *missions*. CxBR is described in detail in section 1.2.

The potential utility of such a system is two-fold. On one hand, the time required to develop acceptable models of tactical behavior for such agents could be eliminated or significantly reduced. Instead of producing a complete high-level behavior model by hand, the most difficult portion (the cues that incite the expert to perform a new or different known low-level behavior during a task) of the logic could be automatically generated using this system. The knowledge engineer is therefore only responsible for specifying each low-level behavior - both how they are executed and how they are identified (i.e. what cues can allow an observer to recognize that the expert is executing a certain low-level behavior).

A second benefit involves the correctness of the knowledge extracted, and would denote a significant advance in the state-of-the-art of machine learning. Experts that perform their task with a high degree of proficiency often cannot elicit their knowledge to a third party [LL99]. A model constructed using an expert's explanation can therefore suffer from incompleteness (or even incorrectness) based on this shortcoming. In allowing a system to learn this behavior automatically by observing an expert in action, the intermediate step of asking the expert to recite his knowledge to a knowledge engineer (who would then be responsible for constructing a model for that expert) would be eliminated. Furthermore, learning such tactical behavior from observation allows for agents to study experts who are either unwilling or unable to elicit their knowledge to a third party. Automated reconnaissance vehicles, for example, can perhaps covertly observe an enemy terrorist training cell, extracting knowledge of certain tactical behaviors intended for use during (and perhaps before) an attack on friendly targets. That knowledge can, in turn, be used to create simulated terrorist agents for training counterterrorist forces. In a completely different sense, a learning system could perhaps observe the movement and behavior patterns of certain living organisms. If that behavior can be captured and replicated in simulation, it would be possible to gain a greater understanding of many mysteries in such fields as ecology or zoology.

The following sections provide background information on the abstract topics discussed above: high-level and low-level behaviors, learning from observation, intelligent agents, and Context-Based Reasoning.

1.1 Learning from Observation

The work reported here is dedicated to producing an algorithm that can model an observed high-level tactical behavior to the extent that its knowledge can be extracted, summarized, and reused. It is important, then, to define learning and how it is applicable within the arena of learning from observation.

As children, learning was a critical element of our day-to-day life. Much of it this learning is achieved by watching and emulating the activities of others. We learn to speak even at the early stages of our development by observing mouth movements of our parents and those around us, and replicate those movements to produce sounds of our own. We learn the various sounds made by animals, as well, by making a connection between sounds heard and images seen. When repeatedly presented a picture of a cow; for instance, along with hearing the sound 'moo', over time it is learned that a cow makes a sound of 'moo.'

The field of machine learning identifies several learning techniques. For example, *Learning by instruction* provides the required knowledge directly. On the other hand, Reinforcement learning allows the learning agent to learn by trial-and-error - in other words, allowing the agent to experience the successes and failures that correspond with his actions. *Inductive learning* uses classified historical examples to develop an induction tree from which rules can be derived. The examples can be discarded after the tree is built. *Explanation-based learning* is somewhat similar except that by including an explanation along with the examples, the number of examples necessary for adequate learning can be substantially reduced. *Connectionist learning* also uses classified historical examples to establish the values of weights in an artificial neural network. The examples can also be discarded after the weight values are set. Unlike the other techniques, neural nets generally suffer from opaqueness, as these weights are rather meaningless to someone casually inspecting the code. Case-based reasoning learns by adding historical examples progressively as it solves new and different problems. It does not discard the examples used, but rather, incorporates them into its own problem-solving approach. Additionally, unlike neural networks which must be trained all at once, case-based reasoning systems learn progressively as new cases are added to the case library. Evolutionary techniques can arguably also be said to learn by searching for an optimal way to accomplish some goal(s). All machine learning methods mentioned above have one thing in common - the need to somehow manipulate data from the real world, either provided a-priori, or collected as part of the process. This will permit us to put the term *learning from observation* in its proper context.

Within the research area of learning from observation, there also exists a distinction between *supervised* and *unsupervised* learning. Supervised learning is a technique by which the learning system is controlled, in terms of **what** it learns, by an outside party or system. An unsupervised learning system is free to learn on its own, without the aid or support of a teaching agent or expert, and is not guided in any fashion as to what to observe, what good and bad responses are, or what the overall goals or objectives of the observed behavior is.

The concept of learning from observation (LFO) is first mentioned in Michalski, Carbonel and Mitchell's book [MM86], where they associate learning from observation with unsupervised learning. In neural networks, the term "learning through observation" is often used to refer to the fact that the training data are "observations". It is true that much of the data in machine learning are based on actual observations. Nevertheless, they typically do not involve learning tactical behaviors. Even if observations are used to learn to recognize handwritten characters, an observed entity is not employed to teach handwriting skills to an observer. Fernlund [FG01] defines learning from observation as "the adoption of behavior through the use of data collected by means of observation." A more descriptive definition describes learning from observation as "inferring concepts by observation" [ZM00]. Here, observation is defined as the act of collecting "characteristics of the relevant environment" [ZM00]. What an observer infers from these observations, however, is a far different matter, and so there must be a clear distinction between what is observed and what is inferred about a given environment. One cannot assume that what is reported by an expert as 'observed' constitutes knowledge that hasn't already been asserted based on his *a priori* knowledge about his task or scenario. The goal is for the agent to develop, on its own, inferences about 'what it sees' based on how the expert reacts to his observations - not how he reports them. Therefore, observation must be considered as it pertains to the agent - we want to record what the agent sees through the expert's eyes. The observations must not include expressions of what the expert may annotate or report about his environment.

We define a single *observation* to be a point-acquisition of time-dependent inputs that can be used to infer assertions about an agent's environment. Included here is the concept of a time-dependent input, which is often an important factor in determining the relevance of a given observation or observation sequence (a group of observations each made during a given time interval). Because of this, we can use the time parameter to differentiate and make relationships between two otherwise independent observations.

$$O_t =< i_1, i_2, i_3, \ldots, i_n > \tag{1.1}$$

In the above equation, we define an observation O_t that occurred at time t. The vector O_t contains fields that represent each input that was introduced to the observer at time t. An observation sequence, therefore, can be considered to be the set of all observations occurring within an arbitrary period of time.

$$O_{t_0 - t_n} = \{O_{t_0}, O_{t_1}, \ldots, O_{t_n}\} \tag{1.2}$$

The assumption made here is that observations within a time interval occur in discrete points in time rather than continuously. This is potentially a dangerous assumption - considering that a gap in time-steps separating two adjacent observations can influence the overall completeness of the observation sequence. In other words, if observations are taken at two adjacent time-points t_i and t_{i+1}, anything that happens and ends between those two points will not be recorded. To ensure that this does not happen, each learning scenario will be such that occurrences within the simulation will be turn-based. Observations, then, will be made at a rate of one per turn. This eliminates the possibility of events occurring and disappearing between observation points.

As it pertains to our application, it is desired that a single observation include information about both the current environment and the current actions of the agent. This requirement is critical because we are attempting to draw a cause-effect relationship between the two. For this research, the learning system will develop tactical knowledge from an observation sequence by creating a certain mapping between an observation pattern and the observed response. However, it is necessary for this research to process these observations and, from them, learn the knowledge that produces these relationships between the environment and the reaction(s) of the observed expert. If we consider these observations as a set of training examples, learning then can be considered to be the process by which these examples are used to generate a knowledge-base about actions within the given scenario. Khardon [Kha98] infers a similar definition in his discussion on supervised learning, defining it to be an algorithm that takes examples and produces a strategy that attempts to mimic that of its teacher. In our case, however, the learning is to be unsupervised at the input. The expert being observed, in other words, does not directly interact with the agent, and learning is done merely by inferring how the expert has reacted to his observations. Nevertheless, we can define learning from observation in similar fashion:

learning from observation The use of data acquired, through observation, to assert knowledge from which an expert's behavior can be intimated

We can use the earlier definition of observation to formalize this definition. To do this, we consider the learning process for an expert E to be some function Λ of a given observation sequence O_E.

$$\Lambda\{O_E\} = A_E | A_E = \{A_1, A_2, \ldots, A_w\} \tag{1.3}$$

In the above equation, the learning algorithm designated by Λ operates on an arbitrary observation sequence O_E and outputs a set of assertions A_E that, in some fashion, describe the behavior that has been observed. As the abstraction of 'learning' does not imply a restriction in the format of what is learned, these assertions are likewise free to take on various types: equalities, thresholds, rules, etc.

In the following section, we introduce the modeling paradigm known as *Context-Based Reasoning.* This paradigm allows for the construction of models of intelligent behavior, as described above, that can represent the intelligence of an autonomous agent.

1.2 Context-Based Reasoning

Context-Based Reasoning, or CxBR, is a technique by which knowledge engineers can create *autonomous agents* able to demonstrate some tactical behavior. We define an *agent* to be any system operating within a real or simulated environment. An *autonomous agent*, then, is a system that operates unsupervised within that environment [TG04].

For this research, autonomous agents used to replicate observed expert behavior will do so using a model. A *model* is a construct that defines the behavior of a given entity within a specific scenario. The model is responsible, therefore, for all of the decisions made by the agent to which it is assigned - it is the 'brains' of the agent. We say that an agent is using a particular reasoning method if it is using a model constructed using that method. CxBR is a reasoning paradigm that allows for such models to be created for use in a variety of environments and scenarios where tactical expertise is necessary. CxBR is based on the idea that ([GA96], [SG04]):

- A situation calls for a set of actions and procedures that properly address the current situation
- As a mission evolves, a transition to another set of actions and procedures may be required to address a new situation
- What is likely to happen under the current situation is limited and influenced by the current situation itself

The motivation for CxBR is the idea that people tend to use only a fraction of their knowledge at any one given time [GA96]. For instance, let us consider an auto mechanic on his way to work. While he needs to keep in mind rules of the road, e.g., following speed and caution signs, avoiding pedestrians and other such obstacles, and being mindful of the other drivers in the area, his knowledge of how to rebuild a car's transmission is irrelevant to his current behavior of maneuvering his car on the road. In creating a model for this mechanic's behavior while driving to work, the representation of his expertise in fixing cars can be omitted. On the other hand, such knowledge would be required for a CxBR representation of the mechanic's day-to-day activities. While driving, however, our mechanic will not likely need to use his technical knowledge.

The technique of dividing the knowledge base into contexts is based on this concept. Given any behavior to model, contexts represent exclusive behavior substates which are relevant to that behavior. From them, the knowledge required to execute a specific behavior is contained within its associated context [SG04]. While this paradigm benefits from its intuitiveness, there are other advantages that make CxBR a viable solution, especially within the realm of tactical behavior. First, decomposing a model's behavior space - or behavioral capabilities - into contexts enables the model to carry a very broad understanding of its task.

While this understanding might, at times, be only on a general level, a context space representative of the entire domain in which the model is to operate, all but guarantees that it will operate on some level of intelligence at any point during its mission. There are many times where a certain skill may be helpful in more than one situation. Furthermore, a certain behavior might be needed in a variety of tactical tasks. CxBR models, in this sense, are modular. Contexts, which may have been constructed for one specific task, can be extracted from its model and inserted into a model for a new task in which that context is relevant. Because of this feature, CxBR models greatly benefit from an object-oriented software engineering approach.

Using CxBR, tasks assigned to the agent is encapsulated within a CxBR mission. This mission provides for the agent both a set of goals, which represent the criterion for completing the task, and a set of constraints specific to that task [SG04]. Also present within a mission is a list of *contexts* that serve to partition the agent's task-related knowledge by the situations in which they apply.

A context represents a situation, based on environmental conditions and agent stimuli, which induces a certain agent behavior specific to that context [SG04]. When an agent is executing a mission using a CxBR model, its behavior is controlled by the current *active context*. The determination of the current active context is made by the context-transition logic of the model. At each time step, context-transition logic examines the current stimuli on the agent and makes a determination of the active context for the subsequent time step. This logic is often in the form of *sentinel rules* that contain the conditions for a specific context-to-context transition; however the transition logic is not required to be rule-based.

1.2.1 Missions in CxBR

A *mission*, or mission context, is an abstraction defined within the model and assigned to a specific agent prior to run-time. Included within a mission is the goal, any imposed constraints, and the context topology that will dictate the high-level behavior of the agent. The goal provides the agent with the criterion for mission termination - end-game conditions for the agent's behavior. For example, consider the assignment of a mission X in which the criterion for completing X would be to satisfy conditions a, b, and c. Obviously, that goal can be represented formally using a Boolean function (e.g. $goal_x = f(a, b, c)$) and embedded within a CxBR model to indicate whether or not the agent has satisfied the requirements of X. The mission goal can be formally defined as a Boolean function g of a set of environmental and physical conditions E and P that exist at the time of query.

$$goal = g(E(t_0), P(t_0)) \tag{1.4}$$

In tactical missions, it is often the case where a 'goal' cannot be defined or is not applicable. It is not uncommon to assign an agent with the mission of performing a certain task or behavior for an indefinite amount of time. In this case, the goal can be construed as an end-game condition for the simulation or scenario. If, for example, an agent representing a scout plane is assigned the mission of performing general reconnaissance on a particular area, the 'goal condition' might be defined as the point where the agent has either been shot down or is ordered to discontinue the mission and return to base. The constraints on the mission provide the agent with a set of guidelines for operation. These constraints can be in the form of physical limitations placed on the sensing faculties of the agent, maximum and minimum counts for scenario-specific entities such as obstacles or enemies, or even map boundaries within which the agent is required to operate. We can consider the constraints on the mission M to be the union of the set of physical, environmental, and logistical constraints (denoted T_p, T_e, and T_l) placed on the agent as required by its mission. In this definition, a constraint c provides the agent with either a constant value or a range of valid values for a certain variable within the simulation.

$$constraints = \{T_p, T_e, T_l\} \tag{1.5}$$

While the notion of a context will be formally introduced in the following section, it is important to mention it here, as it is an essential part of the mission. It was mentioned earlier that to model a behavior with CxBR, that behavior must have the quality that it can be partitioned into sections representing all possible situations; the union of each of those partitions must represent that behavior in full. The reason for this requirement is that the behavior or task, as represented by any CxBR model, must be defined completely by the contexts that constitute it. It is because of this that the mission is also responsible for listing the contexts that are required to correctly execute the model's behavior in that mission. A default context is also listed within the mission, which is a behavior that the model can execute when it is unsure of a behavior to use for a certain situation. This context is also used as the initial context for the agent when it begins a scenario unless a more applicable context can be selected. The mission defines the high-level behavior of the agent by assigning it both a set of contexts and context-transition pairs, which indicate the specific context switches that will be allowed during the scenario. For example, consider the following two sets. The set C_x represents a set of five distinct major contexts present in a mission M_x, while set T_x includes all possible context-transition pairs applicable while executing M_x.

- $C_x = \{c_1, c_2, c_3, c_4, c_5\}$
- $T_x = \{< c_1, c_4 >, < c_2, c_3 >, < c_3, c_1 >, < c_4, c_2 >, < c_4, c_5 >, < c_5, c_1 >\}$

Since the context-transition pair $< c_1, c_4 >$ is a member of T_x, context c_4 is a possible transition from context c_1. In other words, if the agent is currently operating in context c_1, it is possible to switch contexts at a given time-step t_0 to context c_4, if certain conditions exist at t_0. The logic used to trigger these pairs is known as *context-transition logic*, and will be defined in the next section. A CxBR model's *context topology* CT_x consist of a set of contexts C_x, along with the set of context-transition pairs T_x, the Default Context (c_{DX}), and the scenario's universal transition criteria UTC_x. CT_x, along with the goal conditions and constraints, comprises mission M_x.

$$CT_x =< C_x, T_x, c_{DX}, UTC_x > \tag{1.6}$$

$$M_x =< goal_x, constraints_x, CT_x > \tag{1.7}$$

1.2.2 Contexts in CxBR

In [SH01], a context is defined as a set of environmental and physical conditions that may suggest a specific behavior or action. Within a CxBR model, however, a context is a functional state induced as a result of these conditions. Contexts are inserted within a mission to represent all possible conditions that can arise during the course of that mission. This ensures that a model can exhibit intelligent behavior no matter what occurs during mission execution.

CxBR models require that a single context be active at any one time-step during a scenario. It is said that a context within the model is 'active' if the conditions implying its validity exist and the agent is using its included knowledge to make decisions within a scenario. That context is then denoted the current *active context*. The knowledge engineer responsible for creating the model defines and creates each context. Contexts, therefore, are often constructed intuitive subsets of the behavior to be modeled. When encoding the knowledge for these contexts, the idea is to achieve a model that can take the same actions that an expert might take when in the same situation. Consider a mission M with context set $\mathbf{C} = \{c_1, c_2 \ldots c_n\}$. While the division of knowledge represented by these contexts is in the extreme case arbitrary, the knowledge engineer responsible for constructing the model will likely partition each context in a manner consistent with his understanding of the mission. Furthermore, the context-space might also be partitioned so that each context is intuitively coupled with a specific task or behavior that is necessary for the mission. This technique is often used for tactical models in which the sequence of activities and behavior is well known and bounded, and also where the mission itself entails the execution of a series of sub-tasks. It is important to note here, however, that the context-space must be partitioned in order to represent all possible situations that may exist for the agent during a scenario - not simply to divide all possible actions that the agent might take. For instance, consider the some high-level behavior where there are two distinct and unrelated situations under which a behavior b_i is activated. If contexts were partitioned by action, then the two unrelated situations would share the same context within the CxBR model. Partitioning the context-space by situation also ensures that the behavior space of the agent is completely spanned by the set of contexts - i.e. the CxBR model can address any situation and choose a viable active context for the agent to act in.

Within a CxBR model, individual contexts are nothing more than conduits between the current set of stimuli facing the agent and the behavior that will be executed in response. When a CxBR context is declared active, it references the appropriate behavior modules and fact-bases, which in turn determine the correct course of action. The command for that action is then passed from the context to the agent's interface for execution. The context will continue to repeat these steps every simulation cycle, until a different context is denoted as active or the mission terminates.

An active context controls the agent by referencing various knowledge and action modules. These modules are not restricted to a specific form - inference engines, neural networks, and expert systems are all valid modules. Using these modules, along with a local fact base in the agent interface, the active context derives an appropriate action. A *fact base* is a structure that stores parameters and inferences for a certain system, in this case the CxBR model.

The context logic for a context is composed of the control functions, knowledge and action rules that constitute the agent's 'behavior' within that context. We define F_{MC} as the set of functions that control the agent under a specific active context, such that

$$F_{MC} = \{cf_1, cf_2, cf_3, \ldots cf_n\} \tag{1.8}$$

Furthermore, we define the set of action rules for a specific context as AR_{MC}. Action rules are general purpose productions used to execute certain tasks necessary for behavior within a context. Action rules can use facts located in the agent's local fact base, or local variables in the functions that form part of F_{MC}. Some implementations of CxBR may additionally contain a global fact base upon which facts accessible to all models may reside. We can define AR_{MC} as:

$$AR_{MC} = \{ar_1, ar_2, ar_3, \ldots ar_k\} \tag{1.9}$$

Lastly, we define the knowledge contained by each context as a set of frames or classes whose attributes and methods/daemons are essential elements of the tactical knowledge required to successfully navigate the current situation. We refer to this knowledge, for lack of a better name, as knowledge frames or KF_{MC}.

Therefore, the context logic CL_{MC}, which controls the actions of the agent while under the control of a context MC, can be formally defined as:

$$CL_{MC} =< F_{MC}, AR_{MC}, KF_{MC} > \tag{1.10}$$

1.2.2.1 Sub-Contexts

CxBR supports the use of other context-like structures known as *sub-contexts*. Sub-contexts encompass a small functional section of a context not directly critical to the mission objectives. These structures share logical and physical similarities to contexts, but lack many of their attributes. A sub-context is called upon, like a function, to perform a subtask deemed necessary in the logic by a context. Unlike contexts, however, a sub-context does not need to be active at any given moment. Furthermore, when a sub-context has finished executing, it is immediately deactivated and control shifts back to the context that called it. In terms of its role, it is more convenient to think of sub-contexts as user-defined functions that are slightly more complex and specific to the model's mission. However, unlike user-defined functions - whose scope is typically the context that uses it - sub-contexts can be used by any context present within the model. This enhances re-usability of components in the model. Nevertheless, we can represent the sub-context by a vector function - whose input is an action rule of the calling context.

$$subContext_0 = f_0(AR_{MC_i}) \tag{1.11}$$

1.2.3 Context-Transition Logic in CxBR

The selection of an active context during a scenario is controlled by the context-transition logic. Knowing the active context and the recent stimuli on the agent, the context-transition logic selects the appropriate context transition amongst the pairs listed by the mission.

Context-transition logic is permitted to take any form within a CxBR model, so long as a context is chosen at each time step. The most popular representation of context-transition logic is with sentinel rules and *universal sentinel rules.*

With this implementation, the knowledge containing conditions under which a context transition is required are called sentinel rules, or *transition sentinel rules.* Sentinel rules indicate when the appropriate conditions for each applicable transition (each context-transition pair provided by the Mission) hold true. If, for instance, the mission provides a context-transition pair for context c_1 to c_3, a sentinel rule will be present within c_1 that monitors for the conditions warranting a transition from c_1 to c_3. If that condition arises, the transition sentinel rule corresponding to that pair will fire, and a transition will be instantiated. Sentinel rule antecedents may include the fact-base of the current context and the current status of the agent (e.g. inputs, physical state and location). While there are often universal conditions for a transition to a given context, sentinel rules defined to be unique to the context where they exist. This feature allows the agent to function in more complex tactical domains where transitions to a context might be a consequence of two entirely different motivations. When sentinel rules are implemented within a mission M_x, the CxBR model provides a set S_x of transition criteria that represent the conditions necessary for each transition listed in T_x (the set of legal context-transitions). Representing the rule defining the transition criteria from context c_i to c_j as s_{ij}, we can define the set of sentinel rules S_x as the combination of all s_{ij} where $< i, j >$ is a member of T_x (i.e. if $< i, j >$ is a valid transition within mission M_x).

$$S_x = \bigcup_{i,j=1}^{i,j \ni <i,j> \in M_x} S_{ij} \tag{1.12}$$

In many tactical scenarios, there exist conditions that require the agent to transition its context regardless of its current active context. To account for such conditions, universal sentinel rules are encoded within the mission. These rules hold precedence over all other transition criteria or sentinel rules.

$$US_j = \bigcup_j usr_{xj} \tag{1.13}$$

1.2.4 A Generic CxBR Model

Figure 1.1 below illustrates a block diagram of a generic CxBR model that can be generated using the current CxBR framework developed by Norlander [Nor99]. This framework serves as both an engine for CxBR models as well as a foundation on which they are constructed. The *agent interface* module stores any sensor data that is read-in by the agent, and includes any necessary low-level functionality needed to implement the actions indicated by a context. When a model is run, this module is instantiated and assigned a mission. The CxBR model controls the agent by determining proper actions and calling the appropriate functions defined within this interface.

As illustrated, CxBR missions define a context topology for the model as well as valid context-transition pairs (illustrated by the dashed lines); agent con-

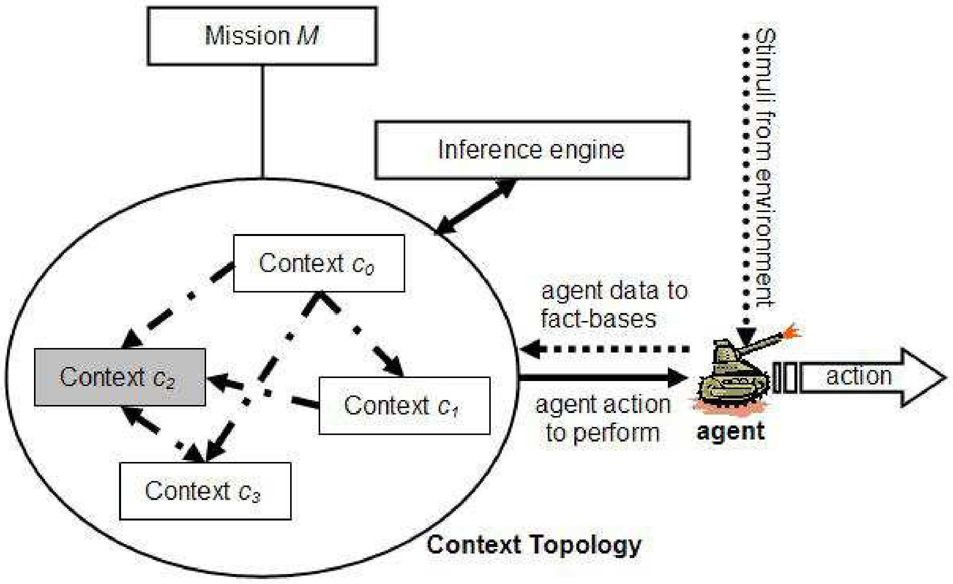


Figure 1.1: Block diagram of a CxBR model

straints, universal sentinel rules, and mission objectives (goals). They are also responsible for identifying the *default context*, which is the context that the agent will operate in at the start of the scenario. If no sentinel rules fire within the current context and it is also found that the current context is not valid, the model will revert to this default context.

As an example of a CxBR model, we present the iRobot Scenario developed in [TG04]. This scenario was an exercise in implementing a CxBR model on a physical platform. In this scenario, the mission is to maneuver an iRobot around an open area looking for a single enemy entity.

Upon detection, the agent first determines the hostility level of the enemy. If it approaches, consider it hostile and retreat. If the enemy retreats, follow it at a close distance. If the enemy is not responsive (i.e., stationary), execute an end of mission signal and retreat to the original starting position.

The context topology for this scenario is provided as figure 1.2. The agent interface connects the CxBR model to the iRobot and defines its low-level functions (move, turn, activate sonar).

`Context Names for iRobot Behavior`

- $goal =< findAndRespondToStationaryEnemy >$
- $C_x = \{c_1 = locateEnemy, c_2 = determineEnemyHostility,$
- $c_3 = approachEnemy, c_4 = retreatFromEnemy,$
- $c_5 = stationaryEnemySignal\}$

`Sentinel Rules for iRobot Behavior`

- $T_x = \{< c_1, c_2 >, < c_2, c_1 >, < c_2, c_3 >, < c_2, c_4 >, < c_2, c_5 >, < c_3, c_2 >, < c_4, c_2 >\}$
- $s(1, 2) = foundEnemyOnSonar$
- $s(2, 1) = lostEnemyOnSonar$
- $s(2, 3) = enemyRetreating$
- $s(2, 4) = enemyApproaching$
- $s(2, 5) = enemyStationary$
- $s(3, 2) = enemyStoppedRetreating$
- $s(4, 2) = enemyStoppedApproaching$

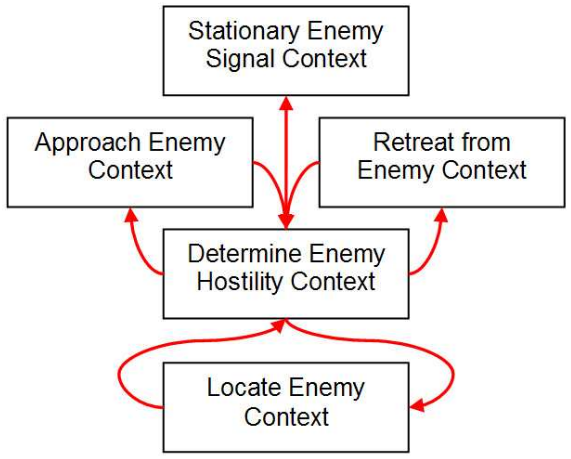

Stationary Enemy Signal Context
Approach Enemy Context
Retreat from Enemy Context
Determine Enemy Hostility Context
Locate Enemy Context

Figure 1.2: Context Topology for iRobot Scenario

`Context Behaviors for iRobot Behavior`

- $CL_1 = (lookForEnemy)$
- $CL_2 = (watchEnemyWaitForMovement)$
- $CL_3 = (pursueEnemyUntilHeStopsRetreating)$
- $CL_4 = (retreatFromEnemyUntilHeStopsPursuing)$
- $CL_5 = (waitWhileEnemySits, spinAroundIfSittingForFiveSeconds)$

1.2.5 Knowledge Representation in CxBR

As discussed in the previous sections, the CxBR paradigm itself provides a way of representing knowledge through the use of the agent interface, mission, context, and *context moderator* objects.

At some level, knowledge is contained in all CxBR components. Some of this contained knowledge is directly responsible for the action of the agent, such as the high-level behavioral knowledge represented within contexts. Other knowledge contained in these CxBR objects is concerned with the dynamics of the paradigm itself, such as the context topology contained in the mission object. Regardless of whether the knowledge is used for directly controlling the agent or the dynamics of the paradigm, CxBR does not constrain nor specify the use of any particular type of knowledge representation paradigm. A knowledge schema illustrating the potential facets of a CxBR model's knowledge-base is illustrated below as figure 1.3

The importance of not demanding a specific knowledge-representation paradigm is in the flexibility offered to the modeler. Any knowledge or associated reasoning mechanisms employed must be determined by the knowledge engineer responsible for model construction. For most systems, a rule-based structure may prove to be the most efficient. However, if learning is to be incorporated or the details of decision-making are not easily classified in terms of rules, structures such as a neural network may be employed. The CxBR paradigm does not limit the type or types of knowledge representation used; rather it is a decision to be made by the knowledge engineer, based on the requirements of the behavior to be modeled.

1.2.6 Intrinsic Low-Level Knowledge of Autonomous Agents

Low-level behaviors in CxBR models are considered to be behaviors that are closely related to dynamic physical and behavioral characteristics of the agent. Such behaviors may include motor skills, sensory data, what the agent perceives

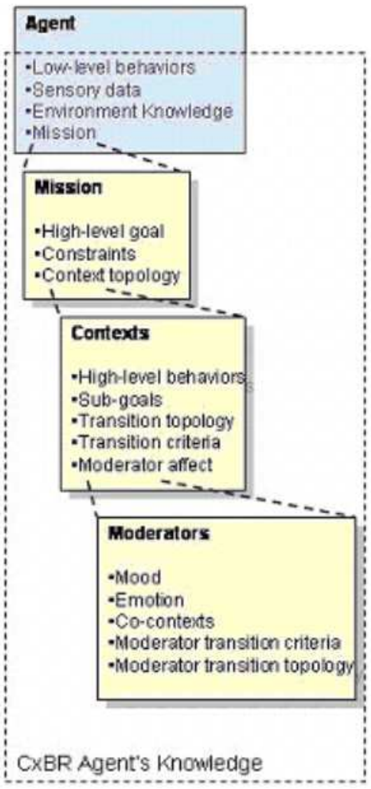


Figure 1.3: Knowledge Schema for a CxBR Agent (reprinted from [SG04]

about its world, environmental knowledge, or even what the agent remembers with regard to its historical perception of the world. These low-level behaviors are fundamental in defining the agent. This is true in that the agent is defined by the low-level behaviors of which it is capable and. It is also true in the sense that the constraints of the behaviors help to define the agent. Consider a behavior such as movement and a corresponding function *move*() to represent this behavior. Different agent types should be characterized in distinctly different ways by how *move*() defines them. For example, *move*() to a helicopter allows for three dimensional movement through space, but there are certain constraints that must be adhered to regarding maximum velocity, maximum altitude, attitude of the aircraft, etc. A fish would also have a low-level behavior defined by *move*(). However, the maximum velocity or maximum altitude of a fish will obviously differ from that of a helicopter.

In addition to low-level behaviors, in CxBR each agent has some perception of and knowledge about its surrounding world. What is of particular importance here, as in the other areas of knowledge representation employed by CxBR agents, is the flexibility the modeler is permitted in choosing knowledge representation paradigms. The method in which memory is implemented for a model is not constrained by the CxBR paradigm. A set of data structures stored in memory could be used to allow fast retrieval of information. Alternatively, a database could be interfaced with the model to allow storage and retrieval of large quantities of data.

1.3 Summary and Discussion of Introduction Topics

In this chapter, a brief overview of the problem space was defined that introduced the topic of learning from observation and the research challenges that it poses. The following chapter describes relevant work that has been done in these areas. Chapter 3 specifically defines the problem addressed for this research. Chapter 4 describes the methodology developed to address the problem, and chapter 5 introduces a prototype implementation of this methodology. The final two chapters are devoted to reporting relevant data, results, and conclusions from the testing of the prototype learning system.

CHAPTER 2

LITERATURE REVIEW

In this chapter, other research both directly and indirectly related to the topics of this work is presented and summarized. The research discussed here is organized by how it is related to this work. Section 2.1 introduces some cognitive architectures and behavior-modeling techniques related to the Context-Based Reasoning paradigm. Section 2.2 summarizes an assortment of techniques used for learning expert behavior that use neural networks in their approach, while section 2.3 outlines other techniques for learning.

2.1 Related Cognitive Architectures and Behavior Modeling Techniques

Recognition-primed Decision Making, or RPDM, is a behavior-representation paradigm developed by Klein [CK02]. The major focus of RPDM is to define how experts make decisions during situations highlighted by time constraints, uncertainty, and 'high-stakes' [SH01]. RPDM has been used to model decision-making processes in such arenas as route planning, computer security, and even nursing and weather forecasting.

The model for RPDM, illustrated in figure 2.1, is based on two variations that represent the level of recognition made by the expert about the current situation [SH01]. In the first variation, the expert easily recognizes the situation he is in. Here, the expert likely makes a decision on a course of action in a direct and methodical manner, and there is little doubt that the course of action is correct for that instance.

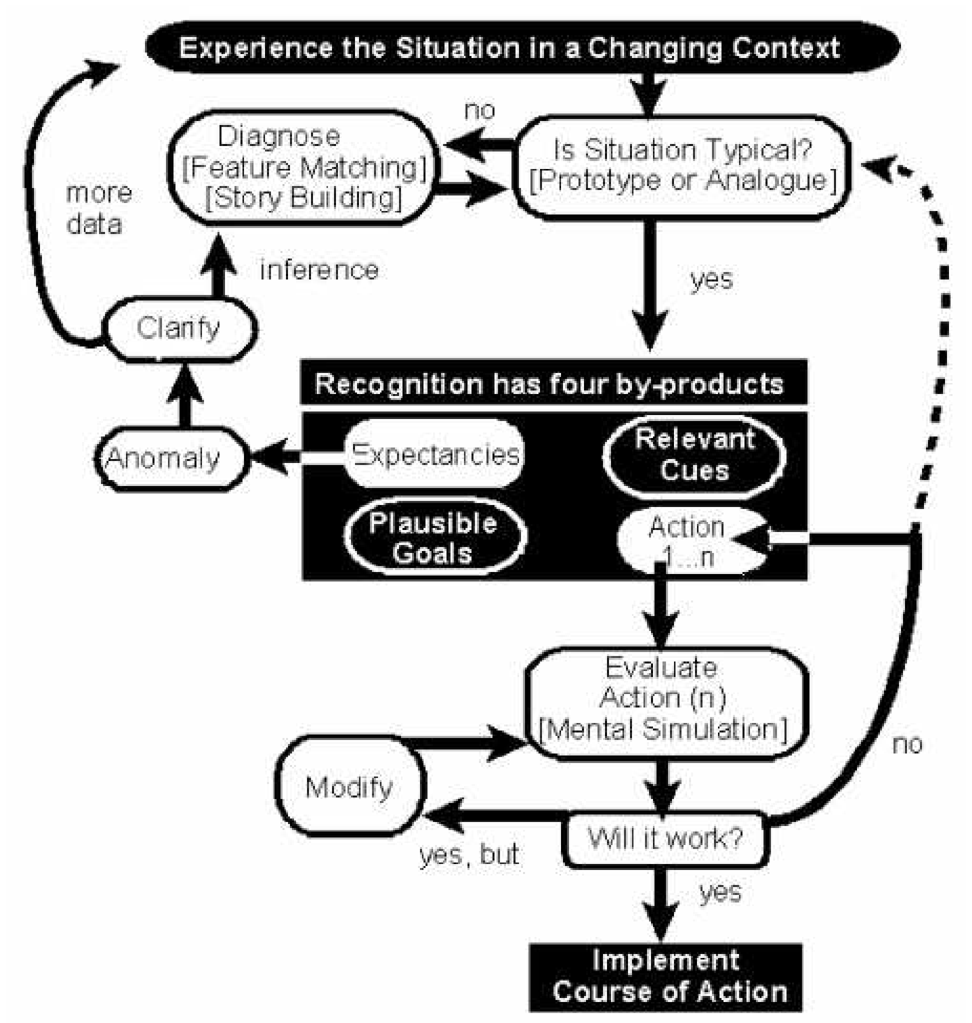

Experience the Situation in a Changing Context
no
Diagnose [Feature Matching] [Story Building]
Is Situation Typical? [Prototype or Analogue]
more data
inference
yes
Clarify
Recognition has four by-products
Anomaly
Expectancies
Relevant Cues
Plausible Goals
Action 1...n
Evaluate Action (n) [Mental Simulation]
no
Modify
yes, but
Will it work?
yes
Implement Course of Action

Figure 2.1: Block Diagram of RPDM Model (reprinted with permission from [SH01])

In the second variation, the situation is either not immediately recognized, or is initially recognized incorrectly. When the situation is not immediately identified, the expert will enter a diagnostic model to determine appropriate course of action. After a course of action is taken, the expert may realize at some point that the consequences of his actions do not coincide with the situation he has chosen. Here, the expert would revert back to re-diagnose the situation as if his initial choice had not been made.

The concept of RPDM shares many similarities to the Context-Based Reasoning (CxBR) paradigm, specifically in its relationship to sentinel rules. Like CxBR, RPDM models put a premium on recognizing and acting upon the introduction of a new situation that requires a new and immediate change in tactical behavior. In CxBR, sentinel rules provide specific and clear-cut conditions that indicate a change in behavior due to a new situation. This behavior is, in fact, tied directly to that new situation through the abstraction of a context. Similarly, RPDM looks for changes in the environment that indicate a new situation, through the diagnosis phase, and then chooses a new mode of action to compensate. Furthermore, both techniques employ a system that identifies and corrects incorrect assessments of the situation. In the case of CxBR, the process of selecting a next-context allows for multiple iterations in order to insure that the proper context is entered. More specifically, a context can be chosen by a sentinel rule and then discarded if it is found that its premises are invalid. These mechanics are not unlike the steps taken in RPDM for variation 2 - if the expectancies assumed by the selected situation do not occur, that situation is discarded and a new one chosen.

There has been significant work done, over the past two years, in developing computational models for RPDM in the tactical behavior arena [WM]. These models have been successful in distinguishing among courses of action even in complex tactical areas such as air-traffic control and on-ground enemy detection. In these cases, the diagnostic logic processes executed are often difficult to express as a set of if-then rules.

Context-mediated Behavior (CMB) is a technique for developing intelligent, autonomous agents much like CxBR. It was originally developed by Turner [Tur98] at the University of Maine. Turner defines a context as "...any identifiable configuration of environmental, mission-related, and agent-related features that has predictive power for behavior". Through this definition, he is better able to justify the close relationship between the identification and selection of context and the knowledge used when acting within that context. He argues the relationship between context and decision-making processes, for instance citing the "gambler's fallacy" of a person skewing the probability of a given outcome because of his recent observations of previous outcomes [TK74].

Much like CxBR, Context-mediated Behavior partitions the knowledge space of an expert into 'contexts' which outline implicitly when that knowledge is appropriate for use. In CMB, the vehicle used to represent this knowledge is the contextual schema, or *c-schema*. A c-schema is a frame-like abstraction that contains several pieces that define the parameters for entering the context and the behavioral knowledge to employ when the context is active. The context description of the c-schema contains three fields - 'actors,' 'objects,' and 'description.' These three fields are used to define the situation under which the context is valid by providing values for 'how much' each feature is expected within that context. By doing this, CMB models are able to use these context descriptions to assess their context at each time step. Context descriptions can also be used by the behavioral knowledge within their respective c-schema to further enhance its models situational awareness about his environment. The standing orders of a c-schema define the appropriate actions to take whenever its context is entered or exited. Similarly, the events field within a c-schema defines unanticipated events within a certain context that must be responded to and provides the knowledge to make an appropriate response. The goals field gives direction to the overall behavior of the agent when it is applying a certain c-schema. This field provides the agent with a general direction to follow when executing actions. Finally, the actions field provides the agent with a list of various moves that it can employ to reach the goals outlined in his current c-schema.

CxBR models are in fact quite similar to those modeled using CMB. In representation, CMB uses several structures to represent knowledge within each c-schema, whereas in CxBR knowledge representation is not as well-defined. In CxBR, the models are more rigidly structured at the mission level. Absent in CMB models, a mission context provides structure in an effort to control the context-flow rather than the execution of individual contexts. Furthermore, CxBR model design places a premium on separating the lower-level contextual knowledge from the contexts themselves, emphasizing the notion that contexts are cues for behavior, and do not also embody the actual behaviors themselves. This line of thinking follows closely with the ideas of Brezillon [BS97], who in his work separates context from action through the use of *proceduralized contexts*.

Soar is a cognitive architecture developed at Carnegie Mellon University in by Laird, Rosenbloom and Newell [LR87]. It was at first an attempt to develop a Unified Theory of Cognition [LR95], and has since been used to develop robust, high-fidelity behavior models in systems from rotary and fixed-wing aircrafts [JK99] to computer-game AI *bots* [WJ02] and models that perform natural-language processing.

The driving force behind all Soar models is the production rule. These rules are responsible for allowing models to reason about its environment, make changes to both its short and long-term memory, and to select appropriate actions based on relevant stimuli. The Soar architecture has a unique rule-firing mechanism, based on the Rete Algorithm, that identifies when each rules' conditions are met. This mechanism allows for rules in a Soar model to activate 'in parallel', meaning they are executed in the same production cycle. Soar production rules are categorized by their specific function within the model. *Elaboration rules*, or elaborations, are responsible for updating the Soar agent's situational awareness by editing working memory with new information. The other two rule categories are both related to the abstract operator structure. *Operators* are the structures within Soar that are responsible for allowing the agent to react and make actions either directly or indirectly in response to his environment. Two types of rules are associated with operators: *operator proposal rules* are Soar rules that allow the agent to select from (or set preference values to) a list of possible operators. *Operator application rules* are then responsible for executing the operator that has been selected as a result of the firing of the operator proposal rules.

While operators are often responsible for making a direct response to the outside world, operators are also involved with the selection of agent *substates*. Agent substates serve to decompose the action/behavior space into goal-defined structures from which the agent can execute more specific operators relevant to that goal. Substates exist underneath the agent's main state (representing its presence within whatever global task it is participating in), and can also exist within other substates creating a hierarchy structure.

An analog to this substate-hierarchy is also present in the ACT-R theory of cognition [And96]. Within ACT-R, these structures are identified as subgoals.

When a goal in ACT-R is identified, the requirements to meet that goal are themselves partitioned into subgoals which, if achieved, will complete the top-level goal.

It is this hierarchical organization of goal-oriented templates that draws the biggest comparison of the Soar (and ACT-R to some extent) cognitive architecture to the modeling paradigm of Context-based Reasoning. Within CxBR, as described in Chapter 1, contexts exist to partition the behavior space just as substates do within Soar. Furthermore, context-transition logic exists within CxBR to select an appropriate active context at each time-step. This is a close analogue to any operator proposal rules within a Soar model that maps to a substate-activating operator application rule. The nature of these operator proposal rules, in conjunction with the state-information and relevant task and agent constraints, help to indirectly form a 'substate-topology' which is similar to the context-topology that is defined by CxBR models' transition logic, mission/agent constraints and goals.

This section introduced four paradigms for behavior representation which are related to Context-Based Reasoning. In each, behaviors are partitioned in some fashion (*c*-schemas in CMB, substates in Soar, etc.) and selected when applicable to the situation. While the CxBR paradigm is instead used to model behaviors, it is this selection process that is central to the theme of this research.

2.2 Approaches to Learning from Observation Using Neural Networks

Henninger [Hen01] describes the design of a neural network to more accurately predict low-level behavior of vehicles in a distributed simulation (MODSAF). More specifically, her work involved predicting Abrams M1 tank positions while moving by using a neural network to extrapolate an updated location accurately. Her model alleviates network bandwidth requirements by allowing a system to accurately predicting tank positions, thereby reducing its need to frequently query the simulation.

In a distributed simulation with many 'nodes' (stations that control individual or several entities), network bandwidth is at a premium. Because of this, it is infeasible for a node to be constantly updated with state information. A dead-reckoning algorithm is typically used to allow each node in the simulation to predict the positions of each vehicle. However, since dead-reckoning is merely a linear approximation of a moving vehicle's position, its predictive accuracy can be quite poor in a simulation where vehicles are constantly changing speed and direction.

Henninger uses a feed-forward, backpropagation neural network to predict entities' location in the simulation. By doing so, she significantly improved upon the accuracy of the dead-reckoning model. The neural network achieves this accuracy by recording previous activity of the tanks and learning how the position of the tank is affected by its previous positions. After learning, the network uses a history of previous tank-position updates to predict its current position. This model proves to be a significant improvement over a straight dead-reckoning algorithm, which simply extrapolates the previous position and heading of the tank's last updated position to predict it's current position.

While relevant to this research because of the learning aspect, Henninger's work is strictly relevant to low-level behavior. All of the learning done for her research centered around predicting positional data, there is no hierarchy of behavior that is considered as in this research.

Gerber [Ger01] employs a Template-Based Interpretation (TBI) engine that predicts tank-position information by first selecting its inferred behavioral context. TBI is a method of inferring tactical intent that was developed in [GG00]. In this method, behaviors are represented by templates that contain the expectations of what an expert would do if executing that behavior. When identifying the presence of a specific behavior compete with each other. Attributes in templates are referenced and marked when observed. Since TBI will be used for this research, a more thorough description of the algorithm is included in chapter 4.

In Gerber's work, the behaviors to be identified were encapsulated as CxBR contexts. He used a neural network to learn and modify the attribute weights for each template, where each template represented a specific context for a tank's path . Through this series of modifications, Gerber's model is better able to correctly identify the current context of the tank than with pre-existing template definitions.

While still confined to tank-driving behaviors, Gerber's work is highly relevant to this research. He decomposes the behavior into a set of contexts which are represented using TBI templates, and attempts to optimize the identifying weights associated with the templates using a learning algorithm. By contrast, this research assumes an accurate definition of a set of context templates and attempts to learn the cues that result in a specific context selection.

Sidani [SG00] introduces a framework for learning implicit expert knowledge through observation within a simulation. This framework operates by combining symbolically represented explicit knowledge with implicit knowledge represented using a multi-layer neural network, into a structure he denotes as a SAM (Situational Awareness Module). The goal of the hybrid structure of SAM is to be able to handle learning complex recognition patterns of the expert. These patterns may even be temporal in nature, varying across a sequence of events in time.

The system works by partitioning the learning space into situations where certain behaviors are expected, such as the contexts described in chapter 1. In Sidani's work, these situations are identified by a global symbolic reasoner. This module is assigned the task of assessing the overall situation facing the expert, and selecting the knowledge unit most appropriate.

The knowledge unit selected, then, learned more primitive (or low-level as defined in this text) knowledge by employing a set of multi-layer neural networks. One trained network is used for each slice of knowledge. By doing this, each network is confined to learning lower-level knowledge confined to particular situations. Training patterns, therefore, can focus on implicit cues and skills that maximize the utility of the neural network.

Sidani's developed framework is quite relevant to this research because of its attention to partitioning knowledge bases by situation. However, the research presents almost the opposite approach. More specifically, here it is assumed that the low-level behavior (denoted as 'primitive' in [SG00]) is defined explicitly and it is the actual *situation identification* knowledge that is learned using a neural network. This concept will be introduced in further detail in Chapter 4.

Johnson [JD02] describes a Fuzzy ARTMAP-based system that allows computer-generated forces to gradually learn behavior on-line during a real-time simulation.

Fuzzy ARTMAP [CR92](or FAM) is reported to have several key benefits, including a relatively few number of parameters and the ability to extract and easily explain the results of the learning.

The neural network is initially trained off-line prior to simulation-time using data that can either be extracted from the simulation or created by-hand by the knowledge engineer responsible for the model. With this data, Johnson's algorithm determines the appropriate parameter values for FAM so that a maximum classification-accuracy is achieved in the testing set.

After off-line training, the FAM model is embedded within a simulation as a computer-generated object (CGO) that imitates the inferred behavior learned from the training data. The online-learning phase of this research then commences when the simulation starts. At this point, all decisions for the new CGO are made by the trained FAM model. Training now occurs incrementally after each decision-step. If the CGO reports that the FAM model has made a favorable decision, that decision's associated pattern is immediately presented as a training pattern thereby strengthening FAMs ability to correctly respond to those type of decision patterns. However, if a poor decision is made (i.e. the results of FAM's decision at that time-step are unfavorable), FAM evaluates what would have been the second-chosen decision for that input-type and uses that pattern for training.

Carpenter and Tan [CT95] describe a technique for extracting individual rules from a trained FAM neural network based on their confidence factor. The authors refer to a fuzzy rule within FAM as the mapping between a vector input cluster and its output pattern. The goal of the research was to extract these fuzzy rules through techniques they call pruning and quantization, thereby giving the learned network a set of usable, readable rules that can be extracted and used.

Rule pruning is the process by which unnecessary and misleading input clusters are removed from the trained neural network. To begin this process, each input cluster in the neural network is assigned a confidence factor that represents its usage frequency within the network along with its accuracy in output predictions. Fuzzy rules that are assigned the lowest confidence values are removed from the network based on the specifics of the pruning policy used.

After pruning is finished, the fuzzy values in each rule are quantized to allow for the analog description of the rule to be re-expressed as a logical statement. To do this, the range of values for each field is divided into quantization levels. These levels define a set of features associated with each field (i.e. 'high', 'medium', and 'low'). Each analog value within the fuzzy rule is then re-expressed in terms of the quantization level in which it falls.

2.3 Other Approaches for Learning from Observation

Van Lent and Laird [LL01] outline the development of KnoMic, a system that extracts knowledge from an expert through observation and then generalizes this knowledge in the form of rules that can be used by an agent to perform a similar task to that of the expert. Whereas Henninger's work focused on learning low-level behaviors, KnoMic is assigned to learn how to execute specific and detailed tasks, like flying an airplane to a certain destination and in a certain fashion. The authors refer to these types of tasks as performance tasks.

KnoMic was developed as a derivative of two previously-developed techniques. The first technique, known as behavioral cloning, also attacks the problem of learning performance tasks. The idea behind this technique is to exactly mimic an expert's actions in performing a very specific and well-defined task. Behavioral cloning revolves around building decision trees which classify appropriate control values based on the values of the sensor inputs to the system during specific stages of the performance task. For example, 36 decision trees would be created for an agent that is learning to control 4 four flight parameters during a 9-stage performance task.

While this technique has shown to be effective in its ability to duplicate expert behavior in a single task, it is easy to see that the same learned behavior would not be sufficient should the environment or task be modified in a similar domain. For example, an agent trained in a flight plan from Orlando to Denver would not be able to use his learned knowledge effectively in a trip from Orlando to Boulder, much less from San Francisco to Pittsburgh. Furthermore, the way that knowledge is represented in a behavioral cloning system's decision trees is extremely limited in scope and cannot be generalized in any form so that it might be used in other systems. KnoMic's other parent system, known as OBSERVER, represents knowledge in terms of a specific scripting language that is flexible to generalization (i.e. the system accepts wild-card values and ranges). Its limitation is its ability to learn complex behaviors such as the flight plan example handled well using a system employing a behavioral cloning learning scheme. KnoMic represents knowledge through scripts like OBSERVER, but the operators that use those scripts can be applied over multiple time-steps and in conjunction with other operators. This facet can lead to a hierarchy of operators operating in parallel, allowing for the system to be more reactive to changes in the environment and to better represent the knowledge base used by the expert. Because of this, much more complex tasks can be learned using KnoMic. Furthermore, the use of scripts also gives KnoMic a more flexible representation scheme, the best trait of the OBSERVER system.

As follow-up research to van Lent's KnoMic system, Tolga Konik's work [KL04] involves the learning of goal hierarchies using a technique known as Inductive Logic Programming (ILP). In the observation mode of this algorithm, the expert is again asked to execute a task while annotating goals that he/she has completed during the task. The learning algorithm is then responsible for learning the selection and termination conditions of each goal (when the behavior to execute each goal should be turned on/off) based on the situations the expert is presented and the annotations made in response to those situations.

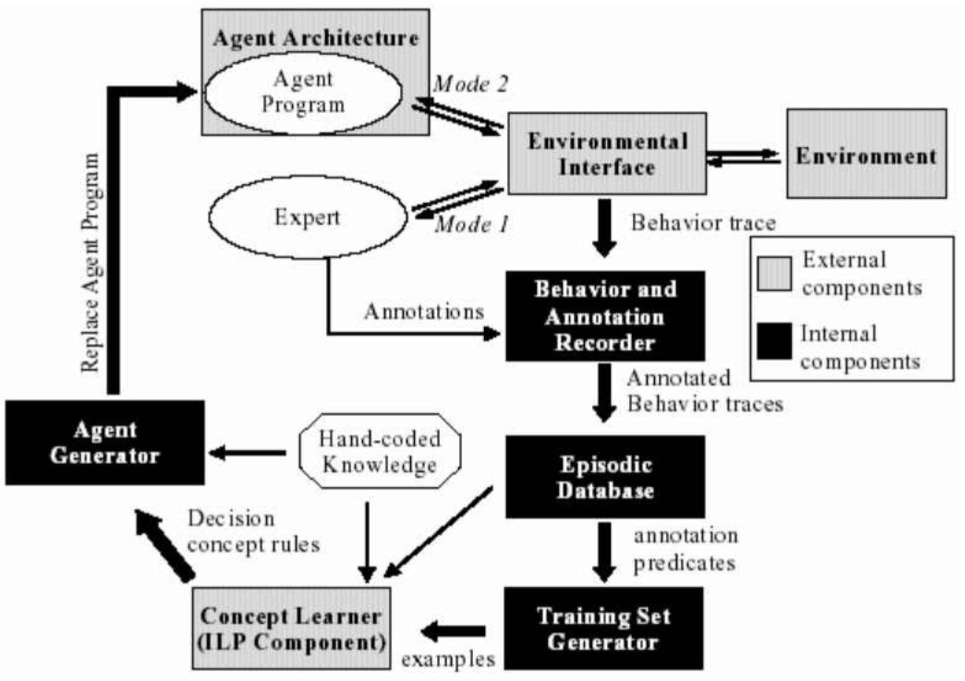


Figure 2.2: Konik's Learning from Observation Framework (reprinted with permission from[KL04])

The ILP algorithm used for this research is called inverse entailment, a technique defined and detailed in [Mug95]. One significant advantage of using ILP for learning is its tolerance for both noise within the training data and the ability to cope with training data where the expert has made an action that is inconsistent with actions he has made at a previous time-step.

The target architecture for Konik's work is Soar, which has been reviewed earlier in this chapter. The goal conditions referred to in [KL04] draws analogies to the concept of context-transition logic, just as the operator abstraction in the Soar Architecture is similar in function to a CxBR context.

Using an array of Bayesian Networks, Laskey [LW04] describes a system by which human-behavior models can be employed to detect security threats imposed by authorized users of computer information systems - situations where users attempt to access portions of the system for which they do not have an appropriate level of clearance. The networks employed are defined as Multi-Entity Bayesian Networks (MEBNs) - extensions to generic Bayesian Networks. This extension allows the network to add arguments to the network known as *MFrags* that have a significant impact at building a hypothetical, probabilistic case for a particular action sequence of a user model.

In Laskey's implementation, seven MFrags are used that represent each potential query and document retrieval command that can be entered by an authorized user. Each of these MFrags is arranged to construct a model representing a single secure query by one individual user. The model is then able to provide probabilities of the users likelihood of malicious intent based on these factors. Learning in this system occurs every time an individual makes a retrieval or query. When this happens, the probabilities associated with the MEBR are updated to reflect the current action.

The work by Rabiner [Rab89] contains a description of both the Markov Chain and Hidden Markov Model (HMM). Although the applications Rabiner selected for his paper do not directly apply to learning from observation, it is possible that a properly trained HMM can output state/context transition data, based on agent observations. This data, in the form of HMM event probability matrices, can be analyzed in parallel with sensor data to extract transitional rules.

Before introducing the HMM, it is important to first understand the concept of a Markov Chain. A Markov Chain is essentially a collection of abstract states and events that stimulate a change in states that is represented as a directed graph. An example of a Markov Chain is provided in figure 2.3 below.

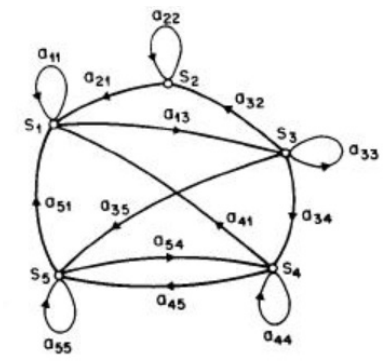


Figure 2.3: A Markov Chain

The nodes on this chain are known as discrete-time states, and each directed edge is known as an event, or a transition between states in the chain. Each transition is the means by which the state q_1 at time n becomes a new state q_2 at time $n+1$. Note that transitions can occur to and from the same state, and states do not necessarily have to have a corresponding event that leads them directly to every other state in the chain. For example, in state s_3 there is no event that will stimulate a direct transition to state s_1.

A Markov Model is simply a probabilistic way by which to represent a given Markov Chain. Note that in all of the states in Figure 2.3, multiple events are possible that will cause a transition to different states in the chain. The act of choosing, or perhaps predicting, the event that will occur prior to a subsequent time-step, is the job of the state-transition probability matrix. This matrix contains heuristic values that can be conceived as probabilities that each specific event will (or should) occur. Consider a Markov Chain with three states and the following state-transition probability matrix:

$$A = \begin{matrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{matrix} \tag{2.1}$$

Here, e_{11} represents the probability that, in the current time-step, the state should/will change from state 1 to state 1. e_{23} is the probability of a change from state 2 to state 3, and so on. If there is no possible transition to a state a to a state b, the value e_{ab} will be zero.

In a Hidden Markov Model, the observation that is output from the system is itself another probabilistic function of the current state. The urn-and-ball example can more clearly illustrate this concept. Consider a set of n urns that each contains a different distribution of colored baseballs, where at each time step one ball is selected from one urn and then placed back into the urn. If that output of the system is to predict what color ball will be selected from the urn, the prediction will be different depending on which urn the ball is removed from. This concept is extrapolated to a general HMM by considering the output of the system to be the color of the ball selected, and each urn to be a state in the Markov Chain.

Wu et al [WP99] introduce the Baum-Welch Re-estimation algorithm for Hidden Markov Models. Though it is in the context of speech-recognition, the algorithm uses a module that trains an HMM to predict a specific state-sequence and observation response to the sensor inputs it receives.

The Baum-Welch Algorithm attempts to generate a model that maximizes the probability of a certain observation sequence given a time-valued input. Two values are generated to initiate this process, $\alpha(t)$ and $\beta(t)$. $\alpha(t)$ is the joint probability that the model will arrive at a specific state i at time t and that it will have generated the correct partial observation sequence from time 0 to time t. This probability is known as the forward probability. The backward probability, $\beta(t)$, is the probability that the correct partial observation sequence is generated in reverse for inputs starting at the end-time T and ending at the current time t.

After $\alpha(t)$ and $\beta(t)$ have been computed for the current time-step, the HMM parameters and transition matrix can be updated based on those values. The number of epochs that the input sequence must be presented to the model to be properly trained is unknown, and is a topic for future research and/or experimentation.

Pineau and Thrun [PT02] introduce the concept of a POMDP and Hierarchical POMDP in the context of controlling the high-level behavior of an autonomous robot designed to guide and assist the elderly.

Partially Observable Markov Decision Processes (POMDP's) are designed to extend the traditional Markov Decision Processes (MDPs, used in Markov Models discussed earlier in this review). They are capable of making informed decisions in domains where state uncertainty is probable. While containing a set of states, actions, and observations, a POMDP model is also capable of high-level problem-solving, and so also includes a heuristic for evaluating action consequences and rewards. Unlike MDP models, POMDP models choose action based on belief of the current state rather than the observed state, because the observations themselves are vulnerable to noise.

The concept of Hierarchical POMDPs is also introduced in this paper. Because of the fact that the current state will not always be completely observable, hierarchical POMDPs involve the partitioning of the action space instead of the state space to provide a more robust problem solution set to the agent.

Here, the actions that are available to the agent are divided into categories that serve as smaller POMDP tasks, which themselves are governed by what the authors call a local policy. The execution of a high-level task now becomes an exercise in extracting all of these local policies and forming a global policy - one that will govern the actions of the agent when it performs the assigned task.

Kocabas et al [KK95] describes a system by which agents that can explain their own learned actions. The authors indicate that most of the previous research on agent explanation has been based on static knowledge instead of machine learning, which was part of the motivation for their project.

Explanation-based Learning (EBL) is a machine learning technique that has previously been used in smaller domains for learning concepts, control rules, and planning. In [KK95], the technique is used to create autonomous agents to serve as opponents and targets in a real-time air simulation. For such domains, they explain, acquiring knowledge for such a task without machine learning is nearly impossible.

RSIM is the name of their integrated system designed in [KK95] to learn the tactical behavior needed for these targets. Within RSIM is a learning and explanation subsystem which is also described in the text.

The learning and explanation subsystem is a rule-based system where action rules are learned in real-time by querying the trainer during the simulation. When a situation arises when action is required, the subsystem searches its rule database for an appropriate response. If no rules match the current situation, the trainer is asked for an appropriate action and the reasoning/explanation behind the choice. If a similar situation occurs in the future, even in the same run of the simulation (the rule is stored in what the authors call dynamic memory), RSIM applies the action described by the rule and provides the included explanation.

Yairi et al [YN00] consider a completely different direction to learning by observation. The concept of a state-space, in earlier work, has always been provided to the agent prior to simulation-time. Learning, therefore, is an exercise in optimizing state-to-state actions and transitions. Manually defining the state-space, however, is often very difficult to do when it is complex, thereby making the task quite inefficient and tedious. Unfortunately, the opposite is also true. Allowing the agent to construct the state-space from scratch, the authors contest, has also shown to be very inefficient.

The concept of reconstructing a provided state-space during simulation-time is introduced in this report. Here, a state-space definition is constructed a priori and provided to the agent. A state-action mapping is learned using a technique known as *Q-learning*. After the mapping is achieved, the state-space is reconstructed that groups similar behavior experiences (a behavior experience is defined by the authors to be an ordered triple of the sensor input, action taken and behavior outcome) and generalizes them to a state. After the state-space has been reconstructed, the Q-values (found during Q-learning - though the algorithm applies when Q-learning is combined or replaced with another method) for the initial state-space are converted for the reconstructed state-space.

Q-learning is a reinforcement learning method, intended for domains that can be modeled using a Markov model (such as the one illustrated in figure 1). The algorithm for Q-learning involves the updating of an SxA matrix of Q-values, where S is the number of states in the Markov chain, and A is the number of actions possible at each state. At each state s', the agent chooses an appropriate action a' (which is based on applying a function involving Q-values from the previous time-step). That action typically will transition the agent to a new state, and the environment will typically provide some type of feedback to the user - feedback which can be construed to the form of a reward value for taking the action. That reward value is then used to update the Q-value that corresponds to taking action a' at state s'.

When state-spaces are reconstructed, the authors claim that three different criteria can be used as preference for considering behaviors 'similar'. These are (1) goal achievement, (2) reward obtained, and (3) sensor-input change. While each of these three criteria come with it different advantages and disadvantages (optimality vs. efficiency, etc.), the authors attempt to combine each of the criteria by considering what they call entropy minimization of multiple behavior outcomes. This is executed by calculating the 'information entropy' for each behavior in a classified state, applying a weight to each, and summing to obtain a Weighted Sum of Entropies (WSE) for the state being evaluated. Minimizing this sum, the authors claim, is equivalent to finding states whose corresponding behavior outcomes are most similar (thereby creating the most accurate state-space).

The act of minimizing that sum, of course, is a much more difficult task than just taking a derivative. Because of this, the authors impose an algorithm that gradually decreases the WSE for each state. This process that decreases the exploration space dramatically and provides an acceptable state-space for the agent, however it will not in non-trivial cases be able to produce the optimal solution.

Takeda et al [TA00] define a new Q-learning algorithm, *Continuous Valued Q-learning* (CVQ) in an attempt to handle some of the shortcomings and inaccuracies encountered when applying standard Q-learning to a Markovian domain.

The authors describe the major problem with standard Q-learning to be its reliance on well-defined action and state-spaces. In a complex domain, it is argued, it becomes very difficult to provide the agent with an appropriate state-space a priori. Attempting to represent Q-values by quantizing the state and action spaces into a finite number of 'cells' that contain identical state and action values, the authors argue, is a generalization technique that is acceptable but runs into the problem of poor efficiency with a small cell-size. Other methods - such as neural networks and statistical techniques - perform a similar task, require a large set of training data often unavailable or time-consuming to prepare.

Introduced in [TA00] is Continuous Valued Q-learning. In this technique, the same general algorithm of Q-learning is applied. However, a critical difference is in the way states and actions are represented. Consider a system with n sensor inputs and m actions available to the agent at each time-step. The CVQ method uses an n-dimensional state-space that can be visualized as a grid of n-dimensional hypercubes representing a continuous state-space. In this continuous state-space, each vertex corresponds to one of n representative states. A perceived continuous state x, then, can be considered as a weighted linear combination of each of the representative states. The weights correspond to how a representative state x_i influences the continuous state x.

New Q-value functions and action policies are also redefined in the CVQ method which takes into account the weights found using the method described above. This method is shown to be both more successful and 'smoother' (smoother in terms of robot movement) than standard Q-learning in the experiments performed.

Laurent and Piat [LP01] discuss a parallel Q-learning procedure for automating the control of a micro-manipulator system with two degrees of freedom. The agent in question is assigned the task of maneuvering several small blocks across a glass platform. The immediate problem that the authors encountered was the potential size of the state-space if each of the objects on the table were considered. According to the text, representing n objects on a table using a Markov diagram would require a $2n$-dimensional state space, which would be too large for effective learning to occur.

Their solution was to create what they call a parallel Q-learning algorithm. In this algorithm, all objects on the table are assumed identical in terms of their size and movement. Because of this, the Q-values generated assume identical rewards for two objects that are pushed in the same manner.

The algorithm takes this into account, and generates a set of n Q-values based on the current state for each of the n objects on the table. The maximum Q-value generated, then, is chosen as the global strategy (action taken) for the agent at that time step. A theoretical analysis was performed on this algorithm and the result is that this method will converge to a local solution that is often acceptable in complex domains such as this.

Fernlund [FG01] incorporates the concepts of genetic programming and Context-Based Reasoning to generate his technique of learning from observation. More specifically, Fernlund evolves CxBR models - models of agents exhibiting tactical behavior - using genetic programming (GP). In this algorithm, the observation takes place on a human performing the very task that the program is attempting to duplicate in a CxBR model.

The observational phase of the learning, for this approach, takes place prior to run-time and results in a recorded set of data that is used to generate fitness values. An evolved model is evaluated by simulating its behavior and comparing its outputs to those generated by the human.

The learning that occurs in this algorithm takes place on three levels: context, transitional, and sub-context. At the context level, models learn what environments constitute a context and what actions should be associated with it. In a driving scenario, for instance, a program should be able to distinguish between interstate driving and city-street driving. Coupled with this idea is learning transitions between contexts. Here, the learning mechanism should develop recognition of when the observed expert has changed his major context. This recognition would then be translated to sentinel rules in the model. Learning at the sub-context level involves learning how to execute the various actions associated with each context.

One problem with a scheme of this type is its scope. By expecting a GP to evolve programs that have no knowledge of either the context domain or the environment, one opens up a search-space many dimensions larger than necessary, especially when the task to be learned is at least somewhat familiar to the developer. By providing the model with at least an initial set of contexts on which to evolve, we can shift the focus of the learning to that of transitions and sub-contexts, knowledge that often separates a novice from an expert.

One of the advantages of a CxBR model is its intuitiveness, in terms of its logic structure. Contexts are defined as logical pieces of a tactical mission, and transitions are rules that an expert would most likely provide as reasoning to proceed from one context to another. Genetic programming techniques, however, do not hold these qualities in the same esteem. Individuals in a population are evaluated on how well they score on a fitness function - a function tied to performance, not clarity of design. In a system such as the one presented by Fernlund, it is conceivable that the GP technique produces models that do not resemble any CxBR models that a typical systems engineer might produce. For example, contexts and transitional rules might be expressed in terms of variables that do not directly translate to one physical object or idea. They might, instead, represent some random combination that only makes sense within the chromosome itself. Though these models might indeed represent the most-fit program based on the fitness specifications, they abandon the spirit of CxBR - readability and modularity.

As in Fernlund's work, the approach in Gustafson [gH01] uses GP to evolve models using a technique known as layered learning. In layered learning, problems are presented to a population individually, and a new problem is presented only after the population has successfully evolved to solve the previous one. This technique is in contrast to a standard GP system where a population is asked to solve a set of problems concurrently, complicating the fitness function and increasing the search-space of the algorithm.

Gustafson develops autonomous, intelligent soccer players that compete on a team with other agents. To develop this behavior, it was deemed necessary to evolve knowledge of the tactics involved from the bottom-up - much like a coach would teach young children the very basics of soccer before moving on to strategy and the various tactics of the game. In this paper, these basics are referred to as ADFs (automatically defined functions).

ADFs are code segments that evolve at the early stages of the layered learning process, and are later used as building blocks for models learning higher-level behavior. For example, consider the action of heading a soccer ball. This fundamental skill is learned independently of an actual soccer match, but is used throughout a match nonetheless. The same concept holds true for this GP scheme: the ADF of heading is learned in the early stages, and stored as a code-segment and reused during latter learning stages - stages where match-specific behavior is being observed.

Pentland and Liu [PL99] propose that human behaviors can be modeled using a Markov chain of dynamic activities, represented by Kalman filters, and then be used to predict future actions based on observations of a human's preparatory activities.

According to the authors, simple human behaviors can be represented using a simple dynamic model known as a Kalman filter. A Kalman filter is, basically, a next-state estimator whose inputs are the current state, sensor measurements, and a Kalman gain matrix that constitute a vector of scalar constants that apply weights to the inputs. However, the authors note that human behavior cannot be accurately modeled by using one of these filters. A multiple model approach, therefore, is proposed where several filters are connected using a Markov chain. A human is then observed to be behaving in one of the states in that chain, and the appropriate filter is applied to predict future behavior.

In this type of setup, the need to predict when the human will change states within the Markov chain. In dealing with this, the authors present a Markov Dynamic Model (MDM). A MDM is similar to a Hidden Markov Model (described earlier in this review), but instead of generating state-transition probability matrices based on observations, they are based on the prediction errors generated by the Kalman filter output of the current state.

Using this approach, the authors were able to achieve 95% accuracy in predicting human behavior in an automobile driving scenario. However, they do not discuss how the Kalman filters or transition probability matrices were generated except that they were designed based on experimental data. In other words, no automated learning was performed in this work.

Seymore and McCallum [SR99] outline a project to extract information from the headings of research papers using a Hidden Markov Model. Many topics were addressed in this paper, including the comparison of using multiple HMM states per information class, and how best to utilize data of different types (labeled, unlabeled, 'distantly labeled'). However, most relevant to our project was the sections of the paper devoted to the automatic generation of an HMM structure.

To generate a model automatically from labeled training data, the authors begin by assigning each word in a labeled paper and assigning it a state. A new state is then created for the next word, along with a transition to it from the previous state. This process will continue until the training data has been exhausted, at which time state merging can begin. The authors identify two types of state merging, named *neighbor merging* and *V-merging.*

In neighbor merging, states that are connected by a transition and share a common label are merged into one state with a self-transition loop (transition from the state to itself). V-merging, on the other hand, merges states that share a transition either to or from the same state, and also share the same label. For example, if state A and state B both contain the label $'X'$ and point to the state C, they would merge using this technique. After the merging is complete, the model structure used to predict the actual data is learned using a Bayesian model-merging strategy that attempts to maximize a balance between fit of the model to the data and the size of the model.

Oliver and Pentland [OP00] describe a system capable of learning driver maneuvers from observation. Known as SmartCar, this system collects inputs - in real-time and from real vehicles - of both the vehicles movement parameters (acceleration, brake, steering angle, etc.) and the surrounding environment (road, other drivers, etc). With these inputs, the SmartCar system is used to predict seven different simple maneuvers that drivers typically make while driving. To accomplish the learning, the authors construct HMMs for each of the seven maneuvers that the SmartCar system is required to recognize. More specifically, a CHMM - or Coupled Hidden Markov Model - was developed and used for this project. A CHMM allows for multiple chains to be used and for interactions to exist between them. In the case of SmartCar, separate HMMs were required to represent an interaction between the other drivers on the road and the subject driver. The authors found that SmartCar was able to recognize and predict a maneuver nearly one second prior to when signals designating that maneuver occur.

Khardon [Kha98] provides a formalization of supervised learning procedures and how they can be applied to rule-based paradigms in both a flat and hierarchical setting (i.e. the layered learning technique introduced in [gH01]).

Khardon first provides a model on which to base his formalizations. This model is a push-button game that embodies some unknown state machine, where some unknown 'goal' state must be reached (note that a context-based model can easily be extrapolated to mirror these requirements). He then defines what the student agent (the thing doing the learning) can do in terms of collecting information and interacting with the environment and provides definitions of terms based on this syntax. Using this setting, Khardon produces a proof that learning a rule-based strategy (a gambit by which to proceed from a certain state to a goal state) can be achieved in polynomial time.

All of the literature introduced in this chapter provides meaningful insight into the state-of-the-art in behavior modeling and machine learning. However, there is a significant gap in this literature in that little work has been done to address the learning of high-level behaviors. While the work by Konik [KL04] is the most relevant, his learning algorithm required the expert to identify when a sub-task was completed and a new one begins. In this research, no direct contact is made between the expert and observer, nor is any indication given when a context transition is made.

Gerber's work involves the learning of context-template weights, however, it was confined to the low-level task of driving a tank. Contexts were implicitly distinguished by only the features of its path and not a result of high-level planning on the part of the expert. In this work, a shift in context is representative of a clear tactical decision made on the part of the expert which is identified unobtrusively by the system.

By contrast, this research is distinguished by the current state-of-the-art in the following areas:

- Low-level behavior/goal transitions are identified automatically by the system
- Learned behavior is is not limited to lower-level behaviors such as maneuvering a tank or navigating a room
- Learned behavior can ultimately be expressed as explicit sentinel rules that define the conditions for each low-level action switch

CHAPTER 3

PROBLEM DEFINITION

The purpose of this chapter is to define the purpose, scope and contributions of this research. In section 3.1, we review the general topic of learning from observation and the research opportunities that relate to this work. Section 3.2 details the specific problem addressed by this research, and postulates the potential contributions that the results of this work can provide. The final section contains the specific hypothesis to be tested. The system design to test this hypothesis is described in detail in Chapter 4, and the prototype implementation of this design is provided in Chapter 5.

3.1 Research Challenges

This research can be best described as an exercise in learning from observation. While Chapter 1 defines the term, the concept of implementing a system that 'learns from observation' is ambiguous at best. What is it that we are attempting to learn, and how are we going to go about observing it? How much interaction does the learning system have with the expert? What aspects of the expert's stimuli and response patterns do we intend to observe?

As shown in the previous chapter, work in learning from observation addresses the task of learning a variety of different patterns and tasks. While work by Henninger [Hen01] addressed the task of learning low-level, task-specific movement models of tanks, Hovland [HM97] was interested in learning the task of inserting pegs into holes. Other research involved LFO challenges such as navigating an airplane [LL99], driving a car [FG01], and even the seemingly simple task of exiting a room [KL04]. The learning challenges associated with these tasks, therefore, often imply very unique qualities that are representative of how difficult the ability to 'learn from observation' truly is.

As discussed in Chapter 1, the overall behaviors to be learned by the proposed system are defined as *high-level* behaviors. High-level behaviors are considered to be those that can be represented by a sequence of simpler, identifiable sub-behaviors known as *low-level* behaviors. If it is assumed that each low-level behavior can be modeled and identified *a priori* by a knowledge engineer, the learning task becomes identifying the cues that determine the transitions from one low-level behavior to another, as executed by the expert.

This process is then potentially capable of expanding about itself. For example, consider behaviors X, Y, and Z that are composed by a set of known low-level behaviors a, b, and c. This research attempts to learn how an expert executes tasks X, Y, and Z by creating a mapping between the expert's observations and the action sequence consisting of sub-actions. Assuming that this task is done successfully, a higher-level behavior A can then be learned in the same manner provided that its execution is composed of behaviors X, Y, and Z. A diagram illustrating this point is provided in figure 3.1 below.

This example identifies the nature of a high-level behavior versus a low-level behavior. In terms of the behaviors X, Y, and Z, the behaviors a, b, and c

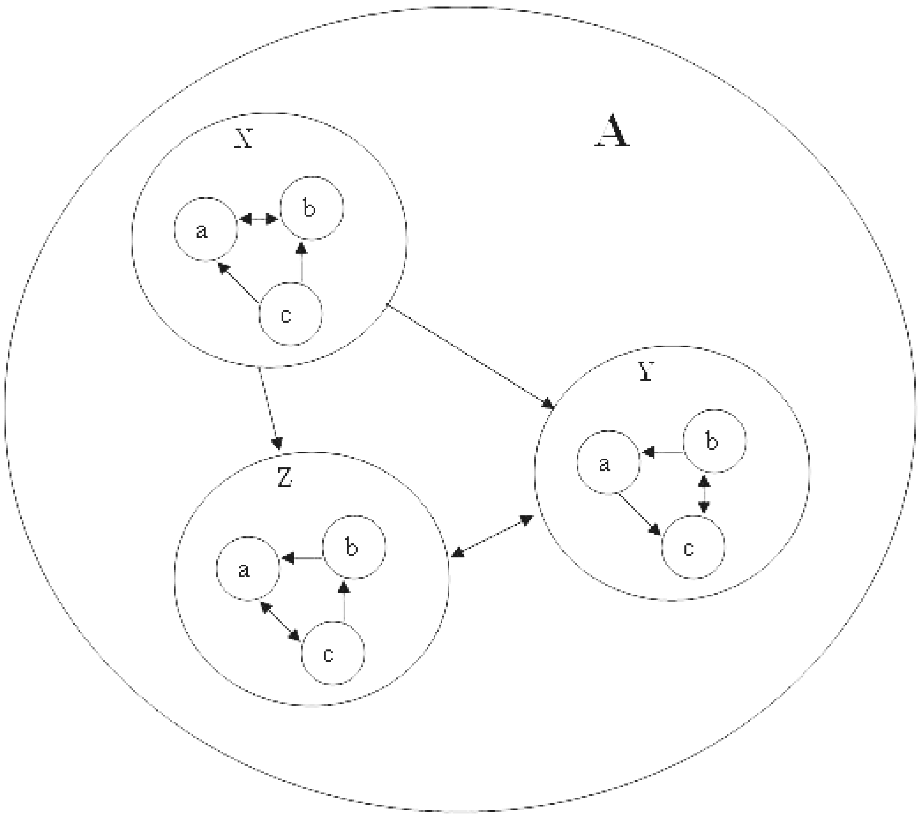


Figure 3.1: Learning Behaviors By Mapping Relationships Between Known Sub- -Behaviors

are considered to be low-level behaviors. However, since X, Y, and Z in-part compose behavior A, they are considered as low-level behaviors with respect to A.

This research, therefore is interested in a class of behaviors and tasks which are composed of lower-level tasks that (a) can be identified during observation, (b) do not need to be learned individually, and (c) are known to be characteristic of the task/behavior we do wish to learn. A behavior B_i, therefore, is learned by determining the situations under which our expert decides to use the sub-behaviors $b_0, b_1 \ldots b_k$ that compose B_i. This behavior B_i will be considered the high-level behavior. The pre-defined contexts that compose that behavior, therefore, will model the low-level behaviors $b_0, b_1 \ldots b_k$ that compose B_i.

3.2 Problem Statement

For this research, we consider a generic task that can be modeled by hand using the Context-Based Reasoning paradigm (introduced in Chapter 1). This paradigm calls for a specific action or action-sequence to be executed in response to identifying the active context that best applies to the model's current situation within his assigned task/mission.

Given such a task, the research challenge was to design an algorithm that creates a mapping between a presentation of a certain stimuli and the expert's observed choice of active context in response to that stimuli. Since this observation is chosen to take place without the expert's assistance, the idea of observing the active context requires two key assumptions: (1) that the expert operates within these contexts and (2) that the system is able to correctly identify the context in which the expert is acting. The first assumption can be justified by choosing the context-set so that it is both transparent to the expert and representative of the choices the expert makes. For example, we can define a *highwayDriving* context for an expert participating in a *driveCarToWork* mission, and infer from observation when he is acting within the *highwayDriving* context. This way, it is possible to identify how the expert has chosen to respond to a situation without disrupting the process by asking him.

3.3 Overview of Approach

To address this problem, this research creates an algorithm that will observe a human expert within a simulation and develop a set of rules that define his high-level behavior. To accomplish this, the algorithm employs two key components. The first component is a Template-Based Interpretation (TBI) engine. This engine is responsible for observing the situation, along with the expert's observed action, and outputting the current context under which the expert is acting. The behaviors and circumstances that define each context are known *a priori* by a knowledge engineer, who is responsible for encoding the properties specific to each relevant context.

The second component is a Fuzzy ARTMAP neural network, which creates a mapping between a set of stimuli and the context chosen by the expert in response to that stimuli. This mapping is created after presenting the neural network with a set of training patterns - each training pattern is represented by an input-output pair. The input is the observed stimuli on the expert, and the output is the context identified by the TBI engine as the response chosen by the expert. The Fuzzy ARTMAP neural network operates by creating a set of *clusters* for the input patterns presented that group patterns with similar features that map to the same output. These clusters are represented within the neural network in such that they can be extracted and represented as a set of input-output rules.

This technique is summarized below for clarity:

1. Expert is placed within a simulation and performs a pre-defined, high-level tactical mission or behavior

2. At discrete time-points, the simulation records the current stimuli (referred to as an *observation*) and the response made by the expert

3. The observation and resultant expert action will be examined by a TBI engine to interpret the context of the expert's action

4. The interpreted low-level context will be paired with the observation, and transformed into an input/output training pattern

5. The learning module within the algorithm will learn the patterns within a given observation that result in each certain context to be activated by the expert

6. After training, the algorithm will contain pattern clusters which can be transformed into transition sentinel rules for a model of the expert's high-level behavior

A more detailed description of the algorithm is provided in chapter 4. That chapter also includes a thorough description of both Template-Based Interpretation and Fuzzy ARTMAP neural networks.

3.4 Contributions of this Research

Below are the contributions of the research described in this dissertation, a few of which have been discussed in earlier sections of this chapter.

- A definition of high-level behavior as a sequence of low-level behaviors combined with the knowledge about when to transition from one behavior to another

- An algorithm by which a high-level behavior - one that can be decomposed into a pattern or sequence of several lower-level, identifiable actions - can be learned through observations of an expert performing the behavior within a simulation.

- A learning algorithm that is both supervised and unsupervised. The algorithm is supervised in the sense that the output mapping that the system makes is limited to a set of possible output contexts. However, it is unsupervised in that the output context does not need to be provided to the system by the expert at run-time.

- An algorithm which can automatically generate a set of sentinel rules for a Context-Based Reasoning Model by observing a human expert perform the behavior within a simulation

- A system that can replicate an expert's high-level behavior by facilitating the construction of a CxBR model of that behavior

- A system that can identify and replicate the different strategies and heuristics used by two different experts in a tactical situation such as a Poker game

- A Java-based simulation environment that can be easily used for LFO tasks.

Chapter 4 documents the design of the learning algorithm and provides thorough descriptions of the key pieces used to realize it. Chapter 5 describes the implementation of this methodology, as well as the simulation testbed and testing scenarios on which it was evaluated. Finally, chapters 6 and 7 detail the results of the testing procedures, discuss conclusions of the work, and suggest future research that can be done to expand or improve the system.

CHAPTER 4

METHODOLOGY

As discussed in the previous chapter, the main focus of this research is the production of an algorithm that can learn high-level behavior by observing the sequence of transitions for the lower-level actions that constitute that behavior. This chapter proposes an algorithm that identifies low-level behaviors when executed by the expert and creates a mapping between them and the scenario-specific observations that precede them. The name given to this algorithm is FAMTILE, which stands for *Fuzzy ARTMAP / Template-based Interpretation Learning Engine.*

Based on our definition of low-level behavior, we assert that all low-level behavior can be specified and identified during observation. After learning the set of conditions that call for each possible low-level behavior transition, a CxBR model can then be constructed that represents the high-level behavior of the expert observed during the simulation. This model contains both the low-level contextual knowledge developed *a priori* and the knowledge learned by this system that identifies when each low-level context is applicable.

The first two sections of this chapter introduce the two main components that are used within FAMTILE. Section 4.1 outlines Template-Based Interpretation, the technique defined in [GG00] and used by FAMTILE to identify expert actions by inferring the low-level context that supports them for each observed decision-point. The subsequent section describes Fuzzy ARTMAP, the neural network architecture used to learn the relationship between the recorded observations during the scenario and the low-level contexts representing the actions taken by the expert in response to those observations. In each section, the description of the systems is followed by a detailed explanation of how FAMTILE uses them to support the learning task.

The last section defines the FAMTILE algorithm, and how it is used to learn high-level behavior. Concluding this chapter is a description of how a CxBR model is developed using the knowledge gained by using the FAMTILE system.

4.1 Template-Based Interpretation

We assume no contact between the expert and the observing system. Because of this unobtrusive observation, however, there is a level of uncertainty that exists when recording an expert's actions for use as training data. A system that can effectively learn expert behavior, therefore, cannot do so by creating a relationship between stimulus and expert response without the ability to recognize that observed response.

For this research, we assume that high-level behaviors are composed of lower-level behaviors which are executed in some sequence. This sequence is dependent on the goals, the environments, and the individual experts themselves. It is also assumed that each of these low-level behaviors can be modeled as CxBR contexts, and are known and specified prior to learning-time by a knowledge engineer.

On the other hand, it cannot be assumed that an expert operates under the same set of contexts identified for the task. Though the contexts may involve the same set of sub-goals that the expert assumes, there is no communication between the expert and the system during learning. Thus, the low-level contexts that the expert **actually** uses cannot be assumed. Therefore, at no point can the expert's low-level context be unequivocally identified. This limitation creates an interesting challenge for this research. If no contact with the expert is to be made (through interview, system feedback, etc.), how can his observed behavior be represented as a sequence of low-level context transitions?

Nevertheless, since the low-level behaviors that embody these contexts are known, we employ a technique known as Template-Based Interpretation (TBI). This algorithm allows for an inference to be made of the low-level context sequence for the expert. TBI is a method of inferring tactical intent [GG00].

4.1.1 Context Templates

In TBI, contexts are represented by *context templates*, or *templates*, that list the expectations of what an expert would do if acting within that context. Within each template is a set of attributes that indicate actions and conditions; each attribute within a template is considered relevant to the context represented by that template. TBI operates by considering each template attribute for all possible contexts for a specific observation or observation sequence. By identifying which attributes are true, TBI will compute a score for each template. The template earning the highest score will then be flagged by the engine, and the context that template represents is considered to be the context under which the expert is operating at the time of observation.

Consider the tactical behavior of driving a car. As a high-level behavior, driving includes several lower-level behaviors that are executed in support of the high-level task: stopping at a red-light, passing slower traffic, avoiding and being aware of pedestrians, etc. Often times there are attributes and cues, from either the driver or from the surrounding environment, that could indicate to an observer what low-level behavior is being executed by the driver. For instance, a passenger does not need to ask the driver to indicate when he's attempting to pass a slower car, he can simply look out the window - the driver has changed lanes and increased his speed, the passed car is likely driving too slow, etc.

Using TBI, we consider these cues to be *attributes* of a context, and group them together within a *context template.* These attributes are then assigned a weight indicating their importance in identifying the context. When observing an expert in action, then, these templates are then referenced to see which has the most (and most important) attributes in effect at that time.

Since the behavior expected within each context is known *a priori*, creating templates with useful attributes is a reasonable task for a knowledge engineer. During the observation phase, then, the intent of the observed performer is inferred, using TBI, amongst the templates created prior to learning-time.

For convenience, we will consider an arbitrary set of contexts $\mathbf{C} = C_1, C_2, \ldots C_n$ and corresponding set of context templates $\mathbf{T} = T_1, T_2, \ldots T_n$. Using this representation, we say that a template T_j includes all attributes and weights common to the context C_j. In a given scenario, all contexts C_i are represented within TBI by a specific template T_i that defines the attributes of C_i.

Each attribute a_i in template T_j is a representation of a condition that is prevalent in the context C_j. The weight w_i represents the importance of a_i in determining context C_j. A low weight value for w_k indicates that attribute a_k is not an imperative nor important characteristic of context C_j. Conversely, a high weight value for w_m indicates that the attribute a_m is a highly relevant condition for context C_j, perhaps even essential. This representation was used in both the work by Drewes ([GG00])and by Gerber [Ger01].

$$T_j = < a_0, w_0 >, < a_1, w_1 >, \ldots < a_n, w_n > \tag{4.1}$$

Here, we reintroduce the high-level behavior of driving a car. If we were to break that behavior down into lower-level contexts of behavior, one context we might create is that of driving in a school zone. For this context, how would we represent its attributes within a context template?

First, we must consider the factors that would indicate that the driver is driving within a school zone:

- a_0: A school is within sight of the driver
- a_1: Children are present and walking the sidewalks
- a_2: Crossing guards are seen controlling nearby traffic
- a_3: Driver is not exceeding 25 mph
- a_4: School Zone signs are visible and blinking

Each of these conditions favor the presence of this context, though none of them are necessarily required. For instance, the driver may know that he is in a school zone despite the fact that he has not seen a sign for the school zone. Likewise, these conditions may exist despite the fact that the driver is not operating within that context. This would certainly be the case at night, where a school may be in sight of the driver though school-zone driving laws are not enforced after school hours or on weekends.

To these ends, we apply a set of weights to each of these attributes that indicate their importance in identifying the school-zone driving context:

- $w_0 = \mathbf{3}$ - A school is within sight of the driver
- $w_1 = \mathbf{4}$ - Children are present and walking the sidewalks
- $w_2 = \mathbf{5}$ - Crossing guards are seen controlling nearby traffic
- $w_3 = \mathbf{6}$ - Driver is not exceeding 25mph
- $w_4 = \mathbf{9}$ - School Zone signs are visible and blinking

The existence of blinking school-zone lights is more indicative of a school-zone driving context than, for instance, the fact that the school is in sight of the driver. Because of this, that attribute is assigned a higher value.

Attributes are not limited to boolean conditions. Perhaps as an alternative to attribute a_3, we want to consider the attribute $\acute{a}_3$ that represents *how close* the driver's speed is to the school-zone recommended speed of 25 mph. By doing this, we eliminate the situation where the driver's speed exceeds 25mph but not by much. Using attribute a_3, the boolean condition considers a speed of 27mph equivalently to one of 72 mph.

4.1.2 Template Selection in TBI

The TBI engine infers a context for each decision-step by first evaluating the condition or value of each attribute present in its set of predefined templates. After each attribute is assigned a value, a weighted sum is computed for each template T_j and used as its *template score*. This template score, s_j, is computed as follows:

$$s_j = \sum_{i=0}^{n} a_{ij} * w_{ij} \tag{4.2}$$

The value assigned to each attribute a_i in template T_j is dependent on the nature of the attribute. We consider four types of template attributes. Below is a short description of each along with an example:

boolean Boolean attributes are either true or false. True attributes are assigned the value 1, false attributes are assigned 0.

$a_{boolean}$=`"does the driver have his windshield wipers on"`

amount less than These attributes are assigned an absolute value scaled between 0 and 1, representing the amount less than a certain value an attribute is

$a_{amount*less*than}$=`"how much slower the driver's speed is to 25 mph"`

amount greater than These attributes are assigned an absolute value scaled between 0 and 1, representing the amount greater than a certain value an attribute is

$a_{amount*greater*than}$=`"how much faster the driver's speed is above 25 mph"`

how close to These attributes are assigned an absolute value scaled between 0 and 1, representing how close a certain value is to an attribute

$a_{how*close*to}$=`"how close is the driver to driving 25 mph?"`

To assign values to each attribute, the attribute's type is considered. For boolean attributes, the value 1 is assigned if the attribute is true, and 0 if the attribute is false. *Amount less than* and *amount greater than* attributes are computed using the expression

$$(value_{att} - value_{target})/max * range(att) \tag{4.3}$$

$$(value_{target} - value_{att})/max * range(att) \tag{4.4}$$

How close to attributes are computed using the expression

$$value_{att} = 1 - (|value_{att} - value_{target}|/max * range_{att}) \tag{4.5}$$

For these expressions, the $max * range$ value represents the maximum possible difference between the *attribute condition* (the test variable for the attribute, e.g. the driver's speed for the above descriptions) and the *target condition*, e.g. 25 mph. Using these expressions, the attribute "how close is the driver's speed is to 25mph" would be evaluated as $1 - (|25 - 12|/25) = 0.48$ if the driver was traveling at 12mph. Note that this value is maximized when the driver is traveling at exactly 25mph.

After each attribute and weight within a template is assigned a value, the template score is calculated using equation 4.2. A context C_x is then chosen by TBI as the inferred context if *and only if* its corresponding template T_x achieves a template score s_x greater than or equal the scores of all other templates. In FAMTILE, the TBI engine will only select one context per observation.

Figure 4.1 represents a TBI engine that considers a set of m context templates and n attributes per template. On the left side of the figure, we see the composition of a generic context template score. Note that the score is generated using a simple weighted sum of each attribute score (computed using the equations above) multiplied by its associated weight. The right side of the figure illustrates the comparative portion of the engine - each score is reviewed and the maximum score selected. The context associated with s_{max} is chosen as the inferred context for that observation.

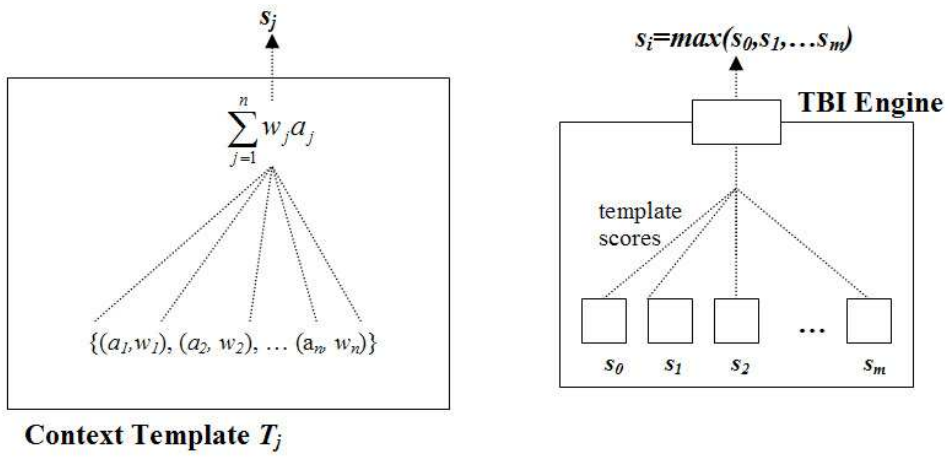


Figure 4.1: A Generic Context Template and TBI Engine

4.2 Fuzzy ARTMAP

Fuzzy ARTMAP is a neural-network clustering technique developed in the early 1990's. The network was introduced by Carpenter et al in [CR92]. It is also described in detail in [GC01]. The goal behind this technique is to produce a neural network proficient at dealing with 'misbehaved' batches of test patterns - patterns where a minority of the testing patterns share little in common with the majority used to train the neural network, but are equally (if not more so) relevant.

A block diagram of the Fuzzy ARTMAP architecture is shown in figure 4.2. The ART_a and ART_b modules within Fuzzy ARTMAP are responsible for generating pattern *clusters* that correspond to a certain pattern form. Each cluster created within the ART_a module represents an input-pattern type that corresponds to a specific output template created by the ART_b module. The Inner-ART module is then responsible for creating a many-to-one mapping between the templates within ART_a and those within ART_b.

For example, consider a situation where a neural network is trained to recognize alphabetic letters when seen and, in response, produces a specific sequence of numbers based on the letter input: $A = 1010, B = 1011, C = 1100, D = 1011$.

When training a Fuzzy ARTMAP module, the ART_a module is responsible for learning to recognize each input letter (A-D), while the ART_b module is responsible for learning to recognize each output sequence (1010-1101). The Inner-ART module creates the mapping between specific letters and their corresponding output sequence.

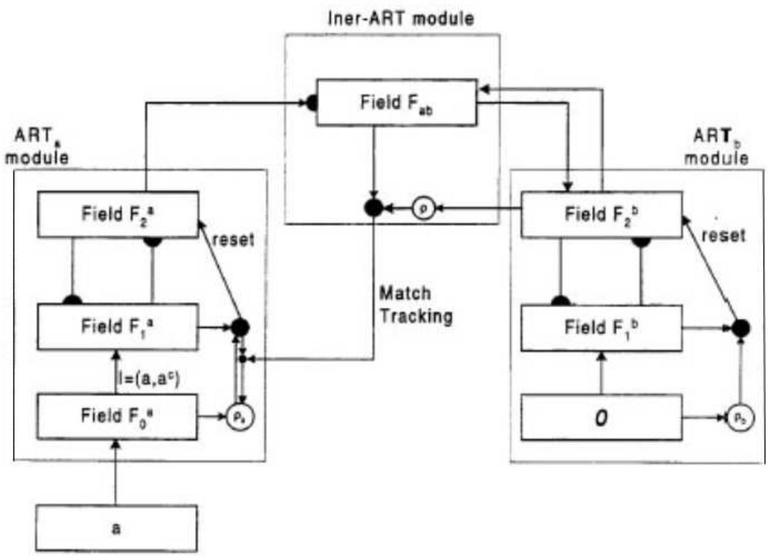


Figure 4.2: Block Diagram of a Fuzzy ARTMAP Architecture[GC01]

4.2.1 Cluster Construction and Modification in Fuzzy ARTMAP

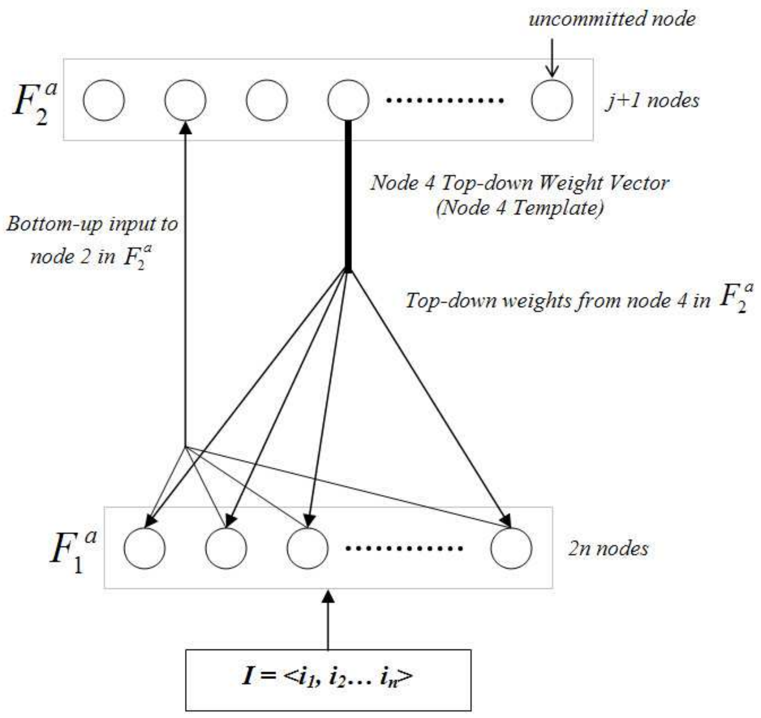
Fuzzy ARTMAP learns in part by developing *clusters* that represent similarities amongst the inputs present within the training sequence. These clusters are developed within both the ART_a and ART_b modules. During the training phase, a cluster is either modified or created (in both modules) to accommodate each input-output pair presented. A mapping between the two clusters is then created within the Inner-ART module. This mapping represents the idea that for a pattern $< a, b >$, the output b is the desired response to the input a.

As stated above, the training patterns used to teach Fuzzy ARTMAP are presented in the form of an input/output pair. This pair is then presented to the ART_a and ART_b modules, respectively. Before this presentation, however, both the input and output patterns must be preprocessed so that they are in a proper form and can be useful to Fuzzy ARTMAP.

First, the pattern must be converted into a vector of real numbers ranging between $[0, 1]$. This step is taken to ensure no learning bias is placed on the magnitudes of each individual field [GC01]. For example, consider an input pattern containing two fields ranging between $[0, 100]$ and $[0, 1]$. The operations done within a neural network often contain factors that represent magnitudes. Because of this, the first field will have a much greater impact on the learning capabilities of the network simply because it can represent numbers of greater magnitudes. By scaling each input to ensure an identical range across each field, this potential problem is eliminated.

Before presentation to Fuzzy ARTMAP, the input pattern undergoes a process known as complement coding within the F_0^a field of the ART_a module. In complement coding, the fuzzy complement of each field within the pattern is taken and then appended to the end of the vector. The fuzzy complement of a number $x \epsilon [0, 1]$ is denoted x^c, where $x^c = 1 - x$. For example, consider vector $v =< 0.2, 0.7, 0.35 >$. Complement coding v results in the vector $v^c =< 0.2, 0.7, 0.35, 0.8, 0.3, 0.65 >$. After complement coding the input patterns, both patterns are presented to Fuzzy ARTMAP at fields and F_1^a and F_1^b. Before discussing further the training procedure, however, it is necessary to define the relevant components and parameters within Fuzzy ARTMAP.

Fields F_1^a and F_1^b represent the position where the input and output patterns are presented to the ART modules. When a pattern enters one of these fields, each component of the pattern vector is represented by a node. Within F_1^a, then, there exist $2n$ nodes, where n represents the number of fields present within each input pattern. Fields F_2^a and F_2^b contain nodes that are connected to the and nodes via top-down weights and bottom-up inputs. One node exists for each cluster created in the ART_a module, plus an additional node representing the uncommitted node. All other nodes within F_2^a and F_2^b are known as committed nodes. These interconnections are illustrated below as figure 4.3.



uncommitted node
F_2^a
j+1 nodes
Node 4 Top-down Weight Vector (Node 4 Template)
Bottom-up input to node 2 in F_2^a
Top-down weights from node 4 in F_2^a
F_1^a
2n nodes
$I = <i_1, i_2 \ldots i_n>$

Figure 4.3: Interconnections Between F_1^a and F_2^a

When input pattern I is presented at F_1^a, a bottom-up input is calculated for each of the existing nodes within F_2^a. This bottom-up input to node n in F_2^a is a function of the input pattern, the choice parameter (a Fuzzy ARTMAP constant ranging from $(0, \infty)$), and the top-down weight vector for n. This vector is known as a cluster. The node n_{max} in that induces the maximum bottom-up input from is chosen as the representative for I, so long as the node satisfies the vigilance criterion for ART_a. The vigilance criterion for node n_{max} is met when a specific scalar function of the input I and the top-down weight vector w_n is found to be greater than the vigilance parameter a. The vigilance parameter is a measure of how selective a node is to admitting patterns, and is a value initialized to the baseline vigilance parameter $\bar{\rho}_a(\bar{\rho}_a \epsilon [0, 1])$ before each pattern presentation during the training phase. If the vigilance criterion is not satisfied, n_{max} is disqualified and the node inducing the next largest bottom-up input from F_1^a is selected, so long as it satisfies the vigilance criterion. If no committed node satisfies the vigilance criterion, the uncommitted node (which will always satisfy the criterion) is selected as the representative for I. When this occurs, a new uncommitted node is generated within before the presentation of the next training pattern. A similar interaction occurs between fields and when an output pattern O, though the vigilance parameter ρ_b for ART_b can not change during the training phase.

If the node selected for I is the uncommitted node, that node must be mapped to the node in F_2^b that was selected for the output pattern O. This task is performed by the Inner-ART module, which creates an internal connection weight between the nodes selected by ART_a and the ART_b. If the node selected for I is a committed node, however, a connection has already been made to a node in ART_b. If that node is the same node selected by ART_b for O, a correct mapping has been achieved. In either of these cases, the top-down weight vectors for the nodes are updated to reflect the addition of new patterns to the cluster. However, it is possible for the ART_a module to select a node for I that maps to a node in ART_b that does not correspond to the node chosen for the output pattern O. In this case, the mapping is incorrect and must be removed from memory. Furthermore, the vigilance parameter in ART_a must be increased as a precaution against future incorrect mappings. As previously stated, large values for the vigilance parameter in ART_a lead to tighter clustering of input patterns. After this parameter is increased, the bottom-up inputs from F_1^a are reexamined and a new node is selected. This process will continue until either the uncommitted node is selected for I, or a committed node is selected that maps to the node selected for O in ART_b.

A more thorough description of cluster construction and Inner-ART mappings can be found in [GC01].

4.2.2 Learning Parameters for Fuzzy ARTMAP

There are several Fuzzy ARTMAP parameters that affect the performance of the learning algorithm in a variety of ways. These parameters are listed along with their description (reprinted from [GC01]).

β_a**:** This parameter is called the ART_a choice parameter, and takes values in the interval $(0, \inf)$. Its value affects the bottom-up inputs that are produced at the F_2^a nodes due to a pattern presentation at F_1^a.

$\bar{\rho_a}$**:** This parameter, called the baseline vigilance parameter, determines the initial value of the vigilance parameter ρ_a in ART_a. The range of $\bar{\rho_a}$ is the interval $[0, 1]$. Small values of $\bar{\rho_a}$ result in coarse clustering of the input patterns presented in ART_a, while large values of $\bar{\rho_a}$ result in fine clustering of the input patterns presented in ART_a.

ρ_a**:** This parameter, called the vigilance parameter, is related to the baseline vigilance parameter. Prior to an input/output pair presentation in fuzzy ARTMAP, ρ_a is set equal to $\bar{\rho_a}$. During training with an input/output pair, the value of ρ_a is allowed to increase above the $\bar{\rho_a}$ value; it is reset back to $\bar{\rho_a}$ when a new input/output training pair is presented to Fuzzy ARTMAP. This parameter affects the granularity of the clusters created in ART_a

N_a**:** This parameter corresponds to the number of committed nodes + 1 in F_2^a during the Fuzzy ARTMAP training phase. During the training phase, Fuzzy ARTMAP operates with all the committed nodes in F_2^a and one committed node. A committed node in F_2^a is a node that has coded at least one input pattern. An uncommitted node is a node that is not committed.

ϵ**:** This parameter is used to evaluate the value of the vigilance parameter, when the vigilance parameter is required to increase during Fuzzy ARTMAP training to a level above the baseline vigilance parameter level. It is usually taken to be a very small positive constant.

β_b**:** This parameter is called the ART_b choice parameter and takes values in the interval $(0, \inf)$. Its value affects the bottom-up inputs that are produced at the F_2^b nodes due to a pattern presentation at F_1^b.

ρ_b**:** This parameter is called the vigilance parameter in ART_b. Small values of ρ_b result in coarse clustering of the output patterns presented in ART_b, while large values of ρ_b result in fine clustering of the output patterns presented in ART_b.

N_b**:** This parameter corresponds to the number of committed nodes +1 in F_2^b during the Fuzzy ARTMAP training phase. During the training phase, fuzzy ARTMAP operates with all the committed nodes in F_2^b and one uncommitted node. A committed node in F_2^b is a node that has coded at least one output pattern. An uncommitted node is a node that is not committed.

4.2.3 Extracting Rules from Fuzzy ARTMAP Clusters

Chapter two includes a synopsis of the work done by Tan and Carpenter [CT95]. In this paper, the authors describe a rule-extraction algorithm for Fuzzy ARTMAP where clusters in a trained Fuzzy ARTMAP neural network are converted into a set of rules that describe the mapping learned by the network. In the case of FAMTILE, each cluster developed represents a certain observation pattern that implies a specific inferred output context reaction by the expert. For example, consider the input pattern attributes a, b, c, output patterns d, e, and f, and a Fuzzy ARTMAP cluster generated by a series of pattern presentations:

$< a_0^{min}, b_0^{min}, c_0^{min}, a_0^{max}, b_0^{max}, c_0^{max}, d_0, e_0, f_0 >=< 0.2, -0.1, -0.7, 0.4, 0.5, -0.3, 0, 0, 1 >$

Using the technique from [CT95], the cluster above can be extracted to form the following rule:

$\mathit{if}(a_0 > 0.2, a_0 < 0.4, b_0 > -0.1, b_0 < 0.5, c_0 > -0.7, c_0 < -0.3)$
$\mathit{then}(d_0 = 0, e_0 = 0, f_0 = 1)$

4.2.4 Motivation For Using Fuzzy ARTMAP

The ability of a neural network to handle 'misbehaved' training sets is of particular relevance to learning from observation. Consider the knowledge required to drive an automobile, an example of a tactical skill. The ability to handle a tire blowout while driving, especially when at high speeds, is certainly important. However, this skill is rarely required simply because tires rarely blow out. If one were to observe an automobile driver in order to train a neural network how to drive; then, it can be assumed that the training pattern corresponding to a blown-out tire on the freeway would represent a very small minority of the training set.

In a CxBR model for a tactical simulation, it is possible that important events requiring a specific context change occurs infrequently. Because of this, training patterns representing these types of context-change cues will most likely be under-represented within a training set. In such situations, traditional neural networks will have a difficult time learning these patterns as a result to the strong emphasis of the other patterns. In these cases, the neural network tends to 'over-learn' the more frequent patterns, discarding the others as noise within the training set. In the case of this research, this noise may represent an interesting and important observation, making the expert's response to it very important to record. Fuzzy ARTMAP neural networks are adept at recognizing the infrequent patterns without reversing the knowledge of any well-learned patterns [GC01].

The idea of creating clusters representing recognition-pattern clusters is also compatible with the RPDM model discussed in chapter one - a model that shares many common features with CxBR sentinel rules. In the RPDM model, situations are diagnosed at critical points during a scenario when the context of that scenario changes. If the situation is 'typical', i.e. if the model recognizes the situation as familiar and has the knowledge to generate an appropriate response, that response is determined and implemented. If the situation is not recognized, the model must generate a new response. This new response will be based on what is known about the situation, any new sub-goals that will be required, and the expectations of outcome based on various responses to the situation.

Within CxBR models, decision making at the highest level is controlled by sentinel rules. At each decision step, sentinel rules choose an appropriate next-context for the model to activate based on the current context and stimuli. If the sentinel rules do not call for a transition to a specific next-context based on the situation presented, and the current active context is not valid, the default context is then chosen.

Through the creation of clusters, Fuzzy ARTMAP also has the ability to handle the large sample of training patterns necessary for a complete observation of an expert's behavior. This clustering process has the effect of reducing the complexity of a decision-space significantly based on the size of the clusters created. The advantage here can be visualized by again considering the task of learning driver behavior. Since recording a decision-making cue (to change lanes, to brake, to turn) often requires a fine granularity across observations, several hundred observations may be recorded of the driver/expert throughout a few-minute driving task. Furthermore; values for the driver's speed, heading, distance to other vehicles, and other potentially significant factors will certainly fluctuate at least nominally along a several-second stretch where no significant behavioral change is executed. This is not because the driver consciously decides to make these changes (decisions that should be recorded and learned), but simply because of the dynamics of the environment and the driver's inherent inability to hold an identical speed and course. A Fuzzy ARTMAP system, however, allows for nearly identical input patterns such as these (that map to the same output) to be represented by a single cluster. By creating a less complex decision space, we not only reduce the order of the learning task but also create a set of clusters from which it may be easier to extract the decision cues they represent and express their knowledge outside of the neural network. The knowledge learned by FAMTILE can be extracted into rules that determine context transitions within a CxBR model.

4.3 FAMTILE: Fuzzy ARTMAP / Template-Based Interpretation Learning Engine

In this section, the components and mechanics of FAMTILE will be explained. As stated earlier, the knowledge extracted with this algorithm is high-level in that it identifies situations where the expert has chosen to chose a specific lower-level course of action or behavior. The situations and goals requiring these lower-level behaviors represented by contexts. Assuming that these lower-level behaviors are known and accurately represented by existing contexts, the algorithm proposed provides the transitional knowledge necessary to construct a CxBR model of the behavior. The knowledge within each context, combined with the overall goals of the expert's behavior (his mission) and the logic that defines when transitions are necessary (the context-transition logic), constitutes a CxBR model for the expert's behavior.

For this research, a mission is defined as a specific behavior partitioned into a set of contexts. These contexts are designed to represent all possible situations that can arise within the mission. Because we assume no direct interaction with the expert, the creation and partitioning of the mission's context-space is done independently. This is a fundamental design requirement of this algorithm, as in Chapter 1 it is asserted that this research intends to learn from experts that may be either unable or unwilling to cooperate with the learning task. Furthermore, the expert is not expected or assumed to know anything about contexts, nor is it expected or assumed that the expert reasons in a manner consistent with Context-Based Reasoning. Part of the design task for the knowledge engineer, therefore, is to construct a set of contexts that are generic enough to represent most experts but detailed enough to be identifiable using TBI and to have meaning as a context.

Before the learning process can begin, the expert must understand clearly the mission he is to perform. He must also be in an environment (either live or simulated) in which he can operate normally. Furthermore, the observational system must be situated so it has the most direct access to the stimuli on the expert without impeding him. To test the learning algorithm, all learning takes place within a simulation so that the observing system can have identical perspective of the scenario as the expert. This simulation is described in Chapter 5.

The following sections describe in sequence the steps used to learn a high-level behavior using FAMTILE, beginning with the generation of an observation sequence.

4.3.1 Generating an Observation Sequence

When an expert executes a high-level mission within the simulation, FAMTILE records all relevant and visible stimuli on the expert along with the actions taken by the expert at the time those stimuli are presented. A recording is made at each decision point i reached during the execution of the behavior to be learned. In the simulated world, these decision points can either be continuous points or segments in time, or they can be planned decision points where time is not relevant, as in a turn-based mission (such as playing chess, for instance, or cards). To account for the reactive nature of the expert's actions at any decision point i, we will refer to the time at which the stimuli are presented as time i^-, and the time at which the expert switches his active context and chooses a course of action as time i^+. At the point when the expert completes the scenario, the learning system will have compiled a set of recordings that should encompass all relevant stimuli and expert actions taken. This set is known as the observation sequence for the executed scenario. Individual members of this sequence are distinguished by the simulation-time at which they were recorded, and are referred to as observations. These observations, labeled as ω_i denote the decision-point i, along with the set of visible stimuli Φ that existed at i^- and the set of actions Γ taken by the expert at i^+.

$$\omega_i =< \Phi_{i^-}, \Gamma_{i^+} > \tag{4.6}$$

$$\Phi_{i^-} = \overbrace{o_0, o_1, \ldots o_n}^{traits-of-observation-i} \tag{4.7}$$

$$\Gamma_{i^+} = \overbrace{a_0, a_1, \ldots a_n}^{actions-taken-by-expert-inresponse-to-observation-i} \tag{4.8}$$

We define the complete observation sequence, Ω_n, to be the set of observations ω_i taken of the expert throughout an entire scenario n.

$$\Omega = \bigcup^{i} \omega_i \tag{4.9}$$

The algorithm for generating this observation sequence is enumerated below:

1. `Determine all variables to be recorded in each observation`
2. `For each variable` k`, assign a representation for that variable and denote it` ϕ_k
3. `Denote the set of all variables` ϕ `in an observation` x `to be` Φ_{x^-}`. If there are` n `variables in an observation,` Φ_{x^-} `can be defined by` $\Phi_{x^-} = \bigcup_{k=0}^{n} \phi_k$

4. Determine all expert actions to be recorded

5. For each action j, assign a representation for that action and denote it γ_j

6. Denote the set of all variables γ in an action set y to be Γ_{y^+}. If there are m variables in an observation, Γ_{y^+} can be defined by $\Gamma_{y^+} = \bigcup_{j=0}^{m} \gamma_j$

7. Set $i = 0$; $\Delta i = 1$; define the number of decision points to observe as N

8. Record current state of variables ϕ present at decision point i

9. Save the set of variables as Φ_{i^-}

10. Record all actions γ of the expert at decision point i

11. Save the set of actions as Γ_{i^+}

12. Save the observation sequence ω_i as $\omega_i =< \Phi_{i^-}, \Gamma_{i^+} >$

13. If $i = N$, go to step 15

14. Set $i = i + \Delta i$, go to step 8

15. For N decision points, save the observation sequence Ω as $\Omega = \bigcup_{n=0}^{N} \omega_n$

Algorithm for Generating an Observation Sequence for FAMTILE

4.3.2 Inferring Low-Level Contexts from Expert Actions and Observations using TBI

After the observations of the expert is complete, the entire observation sequence Ω is presented to FAMTILE. At this point, the actions of the expert must be interpreted by a TBI engine, which will convert Ω_n into a new observation sequence $\acute{\Omega_n}$ where the set of actions taken (represented by ω_i in Ω_n) are replaced with the interpreted low-level context. This context, inferred by TBI for decision point i, is represented by Ψ_{t^+} in equation 4.10. Also represented within $\acute{\omega_i}$ is the inferred active context of the expert prior to decision point i. This context is denoted Ψ_{i^-}, but is identical to the context inferred from the previous decision step ($\Psi_{(i-1)^+}$).

$$\acute{\omega_i} =< \Phi_{i^-}, \Psi_{i^-}, \Psi_{i^+} > \tag{4.10}$$

$$\acute{\Omega} = \bigcup^{i} \acute{\omega_i} \tag{4.11}$$

FAMTILE's TBI engine achieves this transformation by making an interpretation of each low-level action, as described in detail in section 4.1. Prior to observation-time, a knowledge engineer (KE) creates a specification for each low-level behavior necessary for the execution of some high-level behavior (the behavior the system will observe). From these specifications, the KE also creates a set of context templates. Each of the templates' attributes are derived from fields within the observation ω_i.

The algorithm for this step is provided below:

```
1. Determine all parameters $P$ from each observation needed by TBI
   engine to infer a context
2. Set $i = 0$; $\Delta i = 1$; define the number of decision points as $N$
3. If $i = 0$, set $\Psi_{i^-} = null$, else set $\Psi_{i^-} = \Psi_{(i-1)^+}$
4. Extract necessary parameters $P_i$ from $\omega_i$ and input them to TBI
5. Record the inferred context by TBI as $\Psi_{i^+}$, the expert's inferred
   context after decision point $i$
6. Extract the input variable set $\Phi_{i^-}$ from $\omega_i$
7. Record the new observation sequence $\acute{\omega_i} =< \Phi_{i^-}, \Psi_{i^-}, \Psi_{i^+} >$
8. If $i = N$, go to step 10
9. Set $i = i + \Delta i$, go to step 3
10. For $N$ decision points, save the observation sequence $\acute{\Omega}$ as $\acute{\Omega} =$
    $\bigcup_{i=0}^{N} \acute{\omega_i}$
```

Algorithm for Generating Revised Observation Sequence with Context Inferences

4.3.3 Converting Observation Sequence to Training Patterns for Fuzzy ARTMAP

The set $\acute{\Omega}$ is at this point transformed into a form usable by Fuzzy ARTMAP. This operation is done by converting each $\acute{\omega_i}$ into a single training pattern. For a training pattern to be readable by the neural network, each field must be a *fuzzy value* (some real number between $[-1, 1]$). Within FAMTILE, the input portion of the training pattern is derived from Φ_{i^-} and Ψ_{i^-}, while the output pattern is derived from Ψ_{i^+}.

4.3.3.1 Input Pattern Generation

The subset Φ_{i^-} of the observation sequence $\acute{\Omega}_n$ consists of fields representing the expert's complete observation at time i^-. The expert's active context at i^- is denoted by Ψ_{i^-}.

The specific technique for converting the observation arguments must be determined by the knowledge engineer, and depend on the nature of the observations required for the learning. Specific examples of converting an observation into a partial input pattern is included as part of chapter 5, which introduces the prototype FAMTILE system. Converting the observation for Ψ_{i^-}, the observed active context at i^-, involves the same procedure regardless of the scenario. To convert the identified active context into a field within the input pattern, one field is set aside for every possible context in the scenario. If a context j is identified as the active context, the j^{th} field is assigned a value of 1, and the other 'context fields' within the input pattern are assigned a 0. This is done to persuade input patterns with different active contexts to bind to different templates in ART_a. Equation 4.12 represents an arbitrary input pattern converted from Φ_{i^-} that can be presented to Fuzzy ARTMAP. We refer to this converted pattern as $\dot{\Phi}_{i^-}$

$$\dot{\Phi}_{i^-} = \underbrace{o_1, o_2, o_3, \ldots, o_{k-1}}_{observationfields}, \overbrace{c_1, c_2, c_3, \ldots, c_{n-1}}^{activecontext_{n-1}} \tag{4.12}$$

The algorithm for generating the input patterns for FAMTILE is provided below.

1. `Determine a conversion technique for each parameter` ϕ `in the input observation` Φ
2. `If there are` C `total contexts in the scenario observed, assign each context a number` b `between 0 and` $C-1$`.  Assign the representation for the contexts represented by` Ψ_{i^-} `as a` C`-bit binary number.  In a representation for a certain` Ψ `value where` Ψ `is assigned context` b_0`, each bit gets set to 0 except for the` b_0^{th} `bit, which is assigned a 1.`
3. `Set` $i=0$`;` $\Delta i=1$`; define the number of decision points as` N
4. `Perform the conversion for each parameter` ϕ_k `in` Φ_i`, denote each converted field as` o_k
5. `Convert the context` Φ_{i^-} `into a` C`-bit binary integer using the technique from step 2`
6. `Construct the input pattern` $\dot{\Phi}_{i^-}$ `by combining each` ϕ_k `value with the` C`-bit binary integer, as shown in equation 4.12`
7. `If` $i<N$`,set` $i=i+\Delta i$`, go to step 4`

Algorithm for Converting Observation Sequence into a Set of Input Patterns for Fuzzy ARTMAP

4.3.3.2 Output Pattern Generation

The output pattern Ψ_{i^+} is simply a representation of the inferred active context at i^+. Because of this, Ψ_{i^+} can be represented as a j-bit binary number to identify one of j distinct low-level contexts as active, just as is done for the inferred context at i^-. Within Ψ_{i^+}, all bits are set to 0 except for one. If that one set bit is the i^{th} bit (i.e. oc_i in equation 4.13), that means that context i has been identified as the active context for i^+. This representation scheme will make for a trivial clustering task for ART_b, because exactly one output cluster will be generated per context. Representing a context name in this manner allows for the output of ART_b to be both readable and unambiguous for either a knowledge engineer or a separate module created to read its output. Equation 4.13 represents an arbitrary input pattern converted from Ψ_{i^+} that can be presented to Fuzzy ARTMAP. We refer to this converted pattern as $\dot{\Psi}_{i^+}$

$$\dot{\Psi}_{i^+} = \underbrace{oc_1, oc_2, oc_3, \ldots, o_{k-1}}_{activecontext_j} \tag{4.13}$$

The algorithm for generating the output patterns for FAMTILE is provided below.

1. `If there are` C `total contexts in the scenario observed, assign each context a number` b `between 0 and` $C-1$`. Assign the representation for the contexts represented by` Ψ_{i^+} `as a` C`-bit binary number. In a representation for a certain` Ψ `value where` Ψ `is assigned context` b_0`, each bit gets set to 0 except for the` b_0^{th} `bit, which is assigned a 1.`

2. `Set` $i = 0$`;` $\Delta i = 1$`; define the number of decision points as` N

3. Convert the context Φ_{i+} into a C-bit binary integer using the technique from step 1
4. The C-bit binary integer is assigned to $\dot{\Phi}_{i+}$, as shown in equation 4.13
5. If $i < N$,set $i = i + \Delta i$, go to step 3

Algorithm for Converting Observation Sequence into a Set of Output Patterns for Fuzzy ARTMAP

4.3.4 Applying Fuzzy ARTMAP to the Learning Algorithm

The input and output patterns $\dot{\Phi}_{i-}$ $\dot{\Psi}_{i+}$ presented to Fuzzy ARTMAP reflect observations recorded at specific times during the scenario along with the active contexts at those times as identified by the TBI engine. Input patterns are represented by quantitative values for each stimulus on the expert - enemy movements, environmental conditions, current physical conditions, etc. The output patterns represent the action taken by the expert in response to the input pattern presented, where each action reflects a transition from the provided context at the input to a new active context which is inferred using TBI. The implication here is that every action (and thus every output pattern) will represent a transition to a new context, which is of course not the case. Rather, actions representing no context transition are also represented by patterns that require a transition to the current context - the equivalent to no context change.

A training pattern is generated and presented to Fuzzy ARTMAP for each observation made of the expert during the execution of a scenario. Learning occurs through the creation of clusters in the ART_a and ART_b modules and of a many-to-one mapping between those templates. ART_a templates represent clusters of input patterns, similar in their representation, to which the expert has responded by making a specific context transition. That transition is stored in a template in the ART_b module, and a mapping between the two templates is created. When the network subsequently encounters an input that matches the input pattern cluster represented by that template in ART_a, it will know that the appropriate response is stored in its mapped template in ART_b.

Figure 4.4 depicts FAMTILE in learning mode. An observation recorded includes both the stimuli on the expert and his resultant decision. A decision is considered to be the action made by the expert in response to a set of stimuli presented at i, and is expressed as the context that the agent enters (makes active). This stimuli, along with the active context in which the expert is operating at immediately prior to i (i^-), constitutes the input pattern that is presented to ART_a. The actions that the agent executes in response to these inputs (at i^+) are analyzed by a TBI module, which then outputs the most likely candidate for the context that corresponds to those actions. That context name is then presented to ART_b as the output pattern for i, and is also stored for the next decision-point $i+1$, where it will be presented as part of the input pattern as the active context prior to the stimuli presented and actions taken at $i+1$.

The task for Fuzzy ARTMAP, then, is to learn the correct context transition given the current active context and the input stimuli on the agent. To do this, the network will create templates in ART_a that effectively cluster similar input patterns that induce a specific context transition by the expert. The template

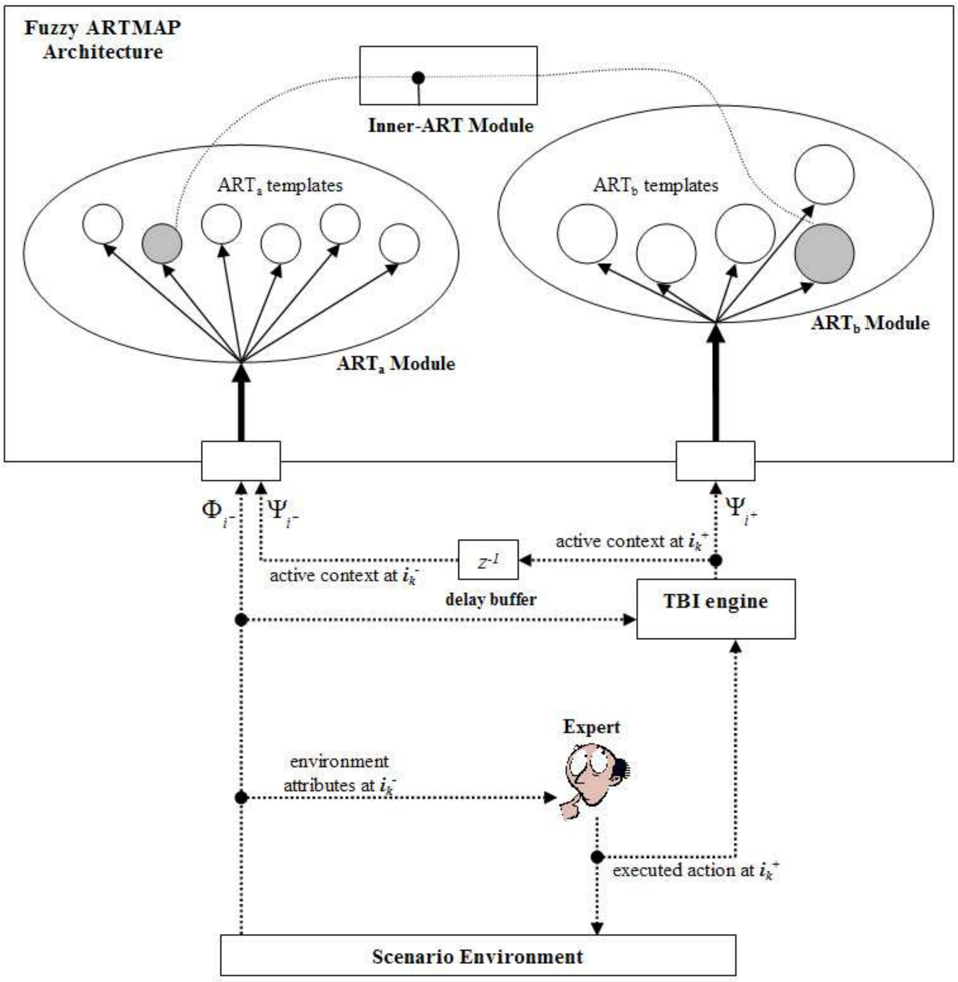


Figure 4.4: Training Context Transitions with Fuzzy ARTMAP

corresponding to the actual transition made will be stored in ART_b, and the Inner-ART module will create a link representing a mapping between the two templates. After the training phase is complete, there will exist a many-to-one mapping between the input-pattern templates in ART_a and the context-transition templates in ART_b.

4.3.5 Converting Fuzzy ARTMAP Clusters into CxBR Sentinel Rules

In section 4.2.3 a technique for converting Fuzzy ARTMAP clusters, as introduced in [CT95], is described. For FAMTILE, rule extraction is used to convert the clusters into sentinel rules for a CxBR model of the expert's high-level behavior. To illustrate the use of this technique, we again consider consider the input-pattern attributes a, b, g, and output patterns d, e, and f. In addition, we add the input-pattern attributes c_0, c_1, and c_2 that collectively represent the expert's active context at point i^- - $< 0, 0, 1 >$ would correspond to context 1, $0, 1, 0$ to context 2, and $< 1, 0, 0 >$ to context 3.

- $ic_x =< a_{min}, b_{min}, g_{min}, c_{0_{min}}, c_{1_{min}}, c_{2_{min}}, a_{max}, b_{max}, g_{max}, c_{0_{max}}, c_{1_{max}}, c_{2_{max}} >$
- $oc_x =< d, e, f >$
- $ic_0 =< 0.2, -0.1, -0.7, 1, 0, 0, 0.4, 0.5, -0.3, 1, 0, 0 >$
- $oc_0 < 0, 0, 1 >$

Consider the input and output clusters ic_0 and oc_0 above. This cluster is identical to the example given in 4.2.3. However the input pattern includes fields for the active context at i^-. This cluster corresponds to the following rule:

```
if (c0==1 and c1==0 and c2==0)
and (a>0.2 and a<0.4)
and (b>-0.1 and b<0.5)
and (g>-0.7 and g<-0.3)
then (d=0, e=0, f=1)
```

If we consider the fields d, e, and f to represent the expert's active context at point i^+, this rule represents a CxBR sentinel rule that indicates the conditions necessary for a transition from context 1 ($c = 100$) to context 3 ($c = 001$). The rule can be re-written as:

```
if (a>0.2 and a<0.4)
and (b>-0.1 and b<0.5)
and (g>-0.7 and g<-0.3)
then (transition to context 3)
```

If the behaviors for these low-level contexts have been coded *a priori* by a knowledge engineer, they can be integrated with the set of rules generated from each Fuzzy ARTMAP cluster, along with any necessary mission requirements, to develop a CxBR model for the expert's observed behavior. This process is illustrated in figure 4.5.

The overall measure of how well FAMTILE is able to deduce expert decision-making cues must ultimately be determined by allowing an autonomous agent to imitate the expert's knowledge in similar scenarios. To do this, the CxBR model - constructed partially by extracted sentinel rules from FAMTILE and partially *a priori* by a knowledge engineer - is assigned to an agent that attempts to execute the expert's behavior in a separate physical or simulated environment. If it is assumed that the behaviors represented by the individual contexts closely match that of the expert, the success of the agent will rely most heavily on the ability of FAMTILE to generate the correct active context sequence based on the stimuli presented to the agent during the scenario. It is the degree of this success that this investigation must ultimately use to evaluate the effectiveness of the learning algorithm.

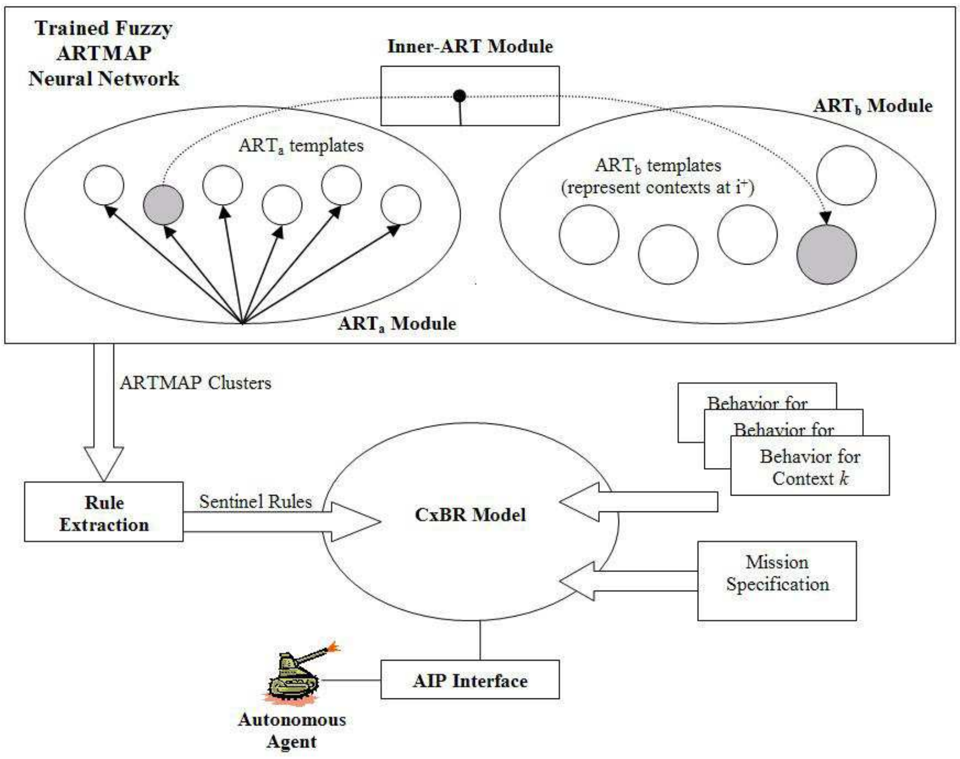


Figure 4.5: Creating a CxBR Model of Observed Expert Behavior using FAMTILE

4.4 Summary of FAMTILE Algorithm Sequence

In the previous sections, the individual components of FAMTILE were introduced and described, as well as many of the processes that take place within the system. To summarize the algorithm, the sequence of events for this algorithm are repeated below.

1. SME runs simulation and begins a scenario involving some high-level behavior

2. FAMTILE collects an observation sequence which spans the length of the expert's activity (see section 4.3.1)

3. Expert actions are deciphered by TBI Engine to determine sequence of output contexts (see section 4.3.2)

4. Input observation sequence converted into a set of input patterns for Fuzzy ARTMAP (see section 4.3.3.1)

5. Sequence of output contexts converted into a set of output patterns for Fuzzy ARTMAP (see section 4.3.3.2)

6. Input/Output patterns are re-paired and presented sequentially to Fuzzy ARTMAP

7. Fuzzy ARTMAP training is complete after each pattern has been presented. Input pattern clusters represent 'interesting' observation patterns that result in a certain context being activated

8. CxBR Sentinel Rules are extracted from Fuzzy ARTMAP clusters (see section 4.3.5)

9. Extracted sentinel rules are combined with pre-coded context behaviors and mission specification to construct CxBR model of expert's high-level behavior (see figure 4.5)

In Chapter 5, a prototype of FAMTILE is described. Also documented the chapter is the simulation environment developed for this research, as well as the four training scenarios that were generated for testing purposes.

CHAPTER 5

A PROTOTYPE IMPLEMENTATION OF FAMTILE

In this Chapter, the prototype system used to evaluate FAMTILE is introduced and described. Section 5.1 describes the simulation developed to serve as the training environment for FAMTILE. The following section details each module of FAMTILE and how they relate to the functionality of the system. Section 5.2 introduces the two high-level behaviors used to evaluate the prototype FAMTILE implementation. The final section, section 5.3 describes the prototype FAMTILE system and each of its components.

5.1 Simulation Environment for FAMTILE Training and Testing

In order to test the FAMTILE prototype, it was first necessary to construct a testbed with which training vignettes can be developed. This environment was developed in Java, and was designed to interface FAMTILE with the testing vignettes and to provide a graphical user interface for the expert.

A block diagram of the simulation environment is provided as figure 5.1, as well as the systems in which it interacts with (including the human expert).

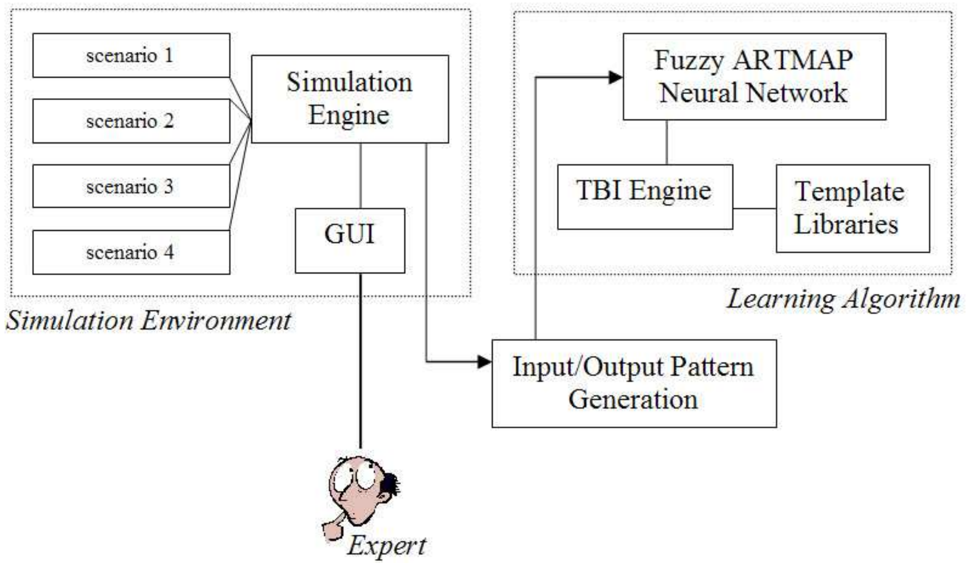


Figure 5.1: Block Diagram of Testing Environment

5.1.1 Interface between Simulation and FAMTILE

Each module involved in this system interacts with other necessary pieces through an interface class named *ArtmapInterface*. This class is responsible for reading the raw data representing observation sequences of the expert. This class interprets the incoming data, transforms it into a set of input/output patterns, and finally presents that set to FAMTILE's neural network for training and testing. The raw data is saved in a text file generated by the simulation engine at run-time.

The interface contains pointers to classes constituting both the simulation and the learning algorithm so that it can interpret the actions of the simulation and report them to FAMTILE. This interface operates as follows:

1. Expert executes a vignette. The interface generates a text file containing his responses to all presented stimuli
2. Interface reads text file
3. Interface passes observation sequence to TBI engine
4. Interface creates a second text file containing the input/output patterns which are readable by a generic FAM neural network.
5. Interface randomizes the list of patterns, and then divides them into training and testing data
6. Interface instantiates FAM and presents it with training data
7. FAM neural network is trained with data
8. Interface presents FAM with testing data
9. Results are compiled and presented
10. If training results are satisfactory, FAMTILE replaces the context-transition logic for a CxBR model

5.1.2 The Simulation Engine Module

The simulation engine provides both the logic of the expert vignettes and their graphical user interface, developed using the Java programming language. This interface was developed in an attempt to both attract experts to participate and to provide them with as realistic a vignettes as possible. Each vignette is introduced and described later in this Chapter.

The simulation engine implements the logic and execution engine for each of the four vignettes. When the expert selects one of them, the simulation instantiates it and presents the expert with his first decision-point. Each vignette is such that the expert actions are *turn-based*, and observations for a certain decision-step represent a set of stimuli and resultant action for one turn. In a turn-based simulation, decision-steps are triggered on expert actions, and not actual clock-time. This property ensures for FAMTILE that the expert is making decisions in response to a known set of observations, and that there is a correct pairing between those observations and that action. Otherwise, the system could not guarantee that the expert was making decisions based on the observation that were recorded for that corresponding time-step.

Listed below are the actions that take place within the simulation during training mode.

1. Simulation prompts expert to enter his name
2. After name is entered, expert selects a training vignette
3. When a vignette is selected, the simulation engine calls the initial commands that begin that vignette. That vignette then displays the situation for the expert, and then pauses until the expert has made his response. That response triggers an event in the simulation that brings up the next situation, and writes the stimuli/response pair to a text file which is read by the interface class after the training session is complete (*offline learning*).

Table 5.1: Vignette Features

	Single Decision-Point	Complete Task/Mission
Simple Behavior	Vignette A	Vignette B
Complex Behavior	Vignette C	Vignette D

5.2 Four Training Vignettes for FAMTILE

In order to make a thorough evaluation of the learning algorithm, four different test vignettes are developed. These vignettes are based on two types of behaviors - moving within a maze environment and playing a game of poker.

The first two vignettes (A and B) consider a relatively simple behavior. Set in a maze environment, only the expert's choice of direction is required at each decision-step. The second pair of vignettes (C and D) involve the game of poker. For these vignettes, we instead assume that the actions made by the expert represent contexts that must be inferred using FAMTILE's TBI engine.

Two vignettes are used for each vignette-type to allow for decisions to be made both repeatedly at the same decision point (vignettes A and C) and throughout a series of consecutive decision-points (B and D).

5.2.1 Vignettes A and B: Navigating a Maze

The first two training vignettes involve the navigation of a 2-dimensional maze. For each vignette, the expert is asked to navigate from his position within a virtual maze to a specified goal position. At each point during the vignette, the player is provided a compass-like directional icon that indicates the distances - in

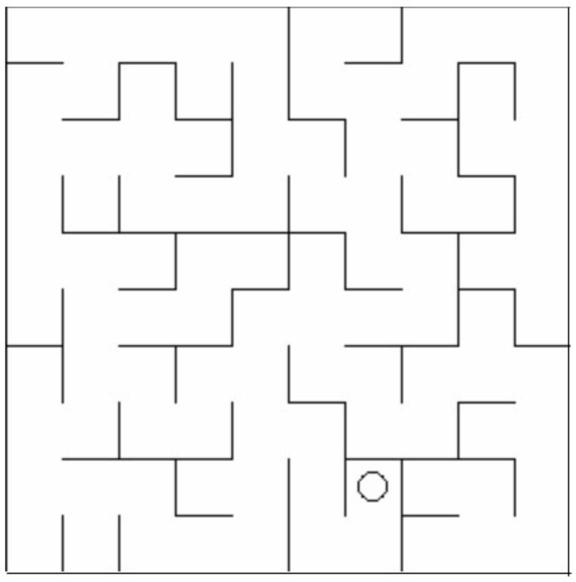


Figure 5.2: Maze Used for Vignettes A and B

both the x and y directions - to the goal position. If the goal position is located within the player's field of view, its position is marked on the map.

In Figure 5.3 above, the circular shape occupying the center position in the maze indicates the position of the expert's avatar. Also illustrated are the walls in all directions that are one space in all directions from the avatar's position. From the observations of this figure, the expert makes a decision on which direction to move. In the first vignette, the avatar and goal positions are re-initialized after each expert action.

In the second vignette, the expert is asked to navigate the avatar towards a goal-position and is given a larger frame of view as illustrated in Figure 5.4. In addition, the simulation records the spaces that have been visited by the avatar along his path to the goal position, and marks these spaces with a square shape on the maze view.

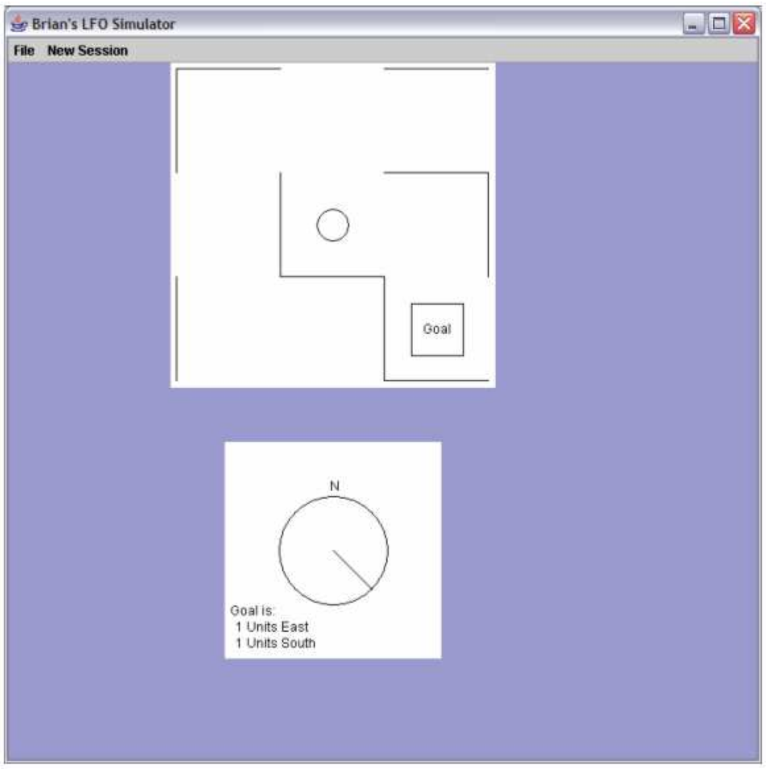


Figure 5.3: Screen Shot of Vignette A

In these two vignettes, there is no conversion from the player's action (left, right, up, down) to a context. This is because of the relative simplicity of the maze-navigating task. With no context inference, the FAMTILE algorithm reduces to a simple Fuzzy ARTMAP neural network. Vignettes A and B, therefore, will serve as a baseline for evaluating FAMTILE in more complicated vignettes where context inference is required. In addition, these vignettes will also be used to evaluate the effectiveness of using Fuzzy ARTMAP individually to learn expert behaviors. The first vignette is intended to produce a basic learning task for Fuzzy ARTMAP. The second vignette is designed to pose a slightly more difficult task for the neural network. The viewing area for the expert is more than twice that as the area in Vignette A. In addition, the network is provided the last move

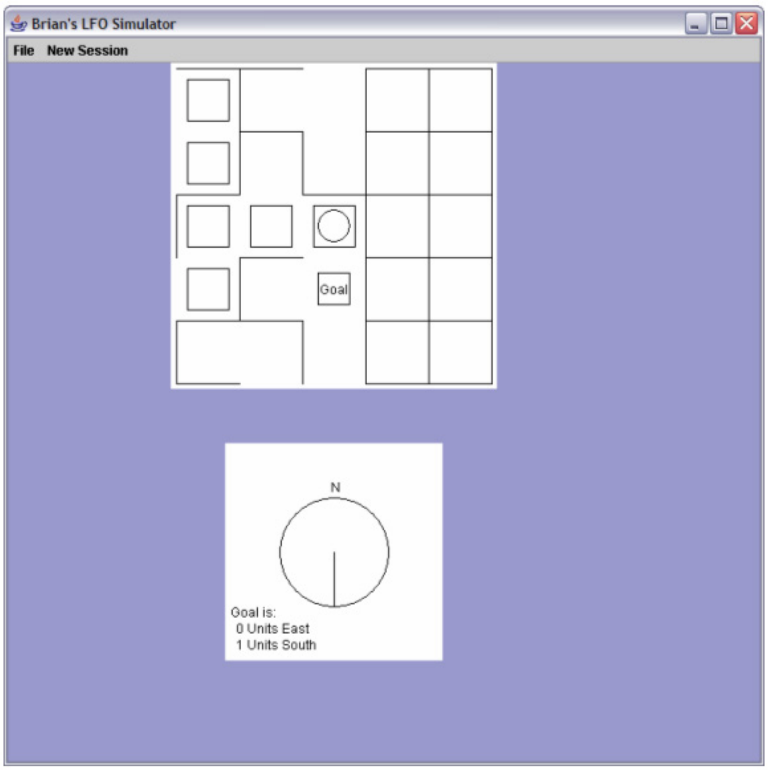


Figure 5.4: Screen Shot of Vignette *B*

of the expert as well as the spaces on the board visited during the vignette. The extra information that may or may not affect the expert's decision is included in an attempt to make the clustering process for Fuzzy ARTMAP more difficult than in Vignette *A*.

5.2.1.1 Expert Behavior within a Maze Environment

Navigation of a maze is, for the most part, an exercise in trial-and-error. A particular path is chosen, and when it is found to be the wrong path a new direction is taken. However, there are certain strategies that can be employed by an expert that take advantage of the observations provided by our simulation. In Vignette A, the expert is provided only a view of one space in either direction along with a vector indicating the distance and direction to the goal position. Here, the strategy for choosing a direction to move is simple: Move in a direction towards the goal that is not blocked by a wall. Unless the expert is able to predict the walls in positions he has not visited (which he cannot do, as his and the goal's position are reset after each move), there is no reason to navigate the maze in a different manner, unless the goal's position is not in the field-of-view of the avatar.

If the goal's position is visible to the player, his movement strategy will likely be altered. For instance, if the goal position is located one unit away from the avatar but in a direction in which he is blocked, the expert will likely choose the best direction that navigates the avatar around the wall.

In Vignette B, there are two additional enhancements that provide more tactical options to the expert. First, the expert's field-of-view is widened by one space, giving him an $5x5$ view of the maze environment. With this view, the expert can 'look ahead' to what his options will be if he moves in a certain direction. With this extra information, the expert can make a move in anticipation of making another move to get to a desired position. This planning step can be made at each decision point, since the field-of-view changes after each move to keep the player in the center of the $5x5$ view.

The second interesting enhancement is the presence of markers on each position in the maze that has been visited by the player (refer to Figure 5.4. With these markers, the expert can now recognize paths within the maze that he has already traversed. If the paths lead to a dead-end, the expert can make a different choice if he is in the same position a second time.

5.2.2 Vignettes 3 and 4: Texas Hold'em Poker

The final two training vignettes involve the game of Texas Hold'em Poker. The following sections assume a basic understanding of the concepts of poker and of Hold'em Strategy. A description of the rules and strategies behind these games can be found in Appendix C. For advanced Texas Hold'em or general Poker strategy, please refer to [SM03] or [Bru79].

5.2.2.1 Description of Poker Vignettes

For this research, two training vignettes were developed using Limit Hold'em game. In the first vignette (Vignette C), only one betting round occurring prior to the *flop* is considered. The expert is placed at a random position at a poker table and seated with seven computerized opponents. The dealer button is placed at a random position, and each player is dealt two hole-cards. Starting with the player to the left of the big-blind bet, each opponent will make an action (either to fold, to call, or to raise) until it is the expert's turn to act. At this point, the expert will know his two hole-cards, his position at the table, and the actions of each opponent who has acted before him. The simulation will then prompt the expert to make an action - either to fold, call, or raise. The expert's action will be recorded along with all applicable observations at that point, and then a new hand is dealt and the player is re-seated. This process continues until the simulation has collected a requisite number of expert observations. A screen shot of the simulation for this vignette is provided below as Figure 5.5.

In this figure, the player at the top of the table is on the button and will act last. Action proceeds in a clockwise direction starting with the player on the

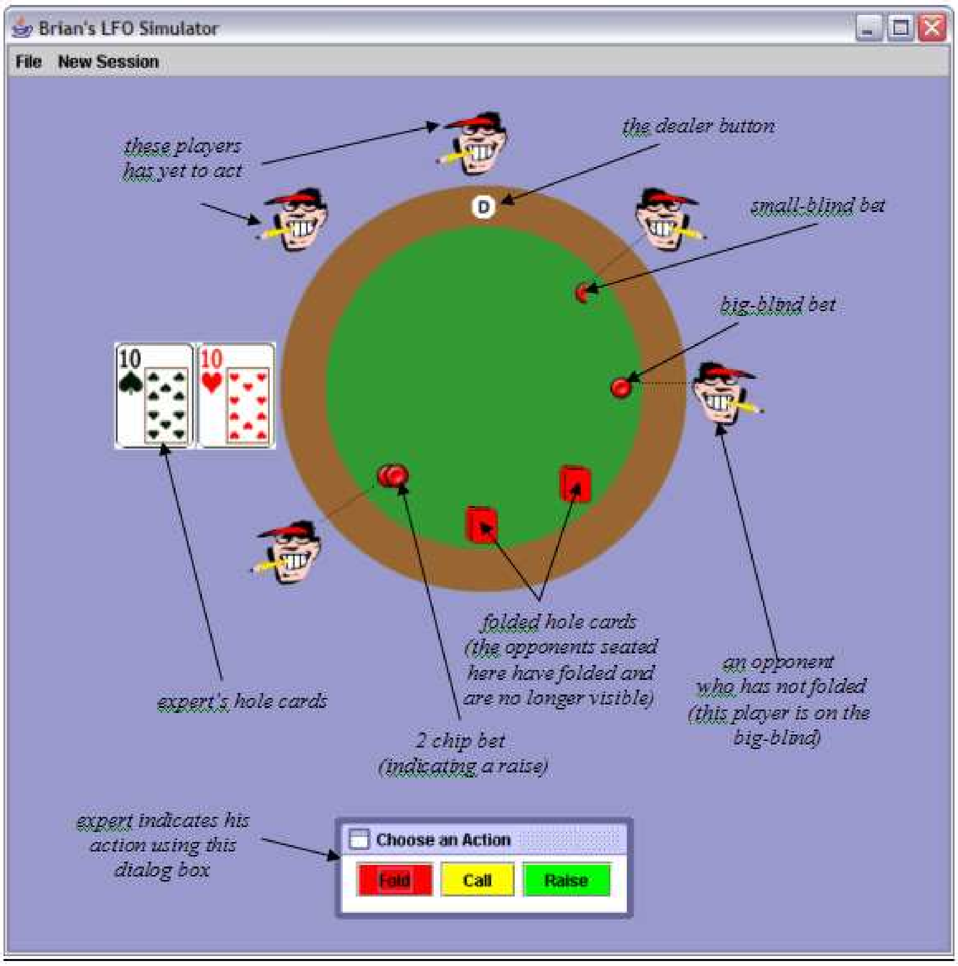

Brian's LFO Simulator
File New Session
these players has yet to act
the dealer button
D
small-blind bet
big-blind bet
10
10
folded hole cards (the opponents seated here have folded and are no longer visible)
an opponent who has not folded (this player is on the big-blind)
expert's hole cards
2 chip bet (indicating a raise)
expert indicates his action using this dialog box
Choose an Action
Fold
Call
Raise

Figure 5.5: Screen Shot of Vignette C

small blind. The two players to the left of the big blind have folded. The player to their left has raised the big blind, so the bet the remaining players must call to remain in the hand is two.

To the raiser's left is the expert's seat. He has been dealt 10♠10♡, a pair of tens. His options are to raise the bet (to three), to call the raise, or to fold. These options are shown to the player using the window at the bottom of the figure.

For the second poker vignette (Vignette D), the expert is asked to make decisions throughout entire hands and accumulate chips throughout the vignette. This vignette begins just as the first poker vignette - the expert is placed at the table with seven opponents, and the button is placed at a random position at the table. A hand is dealt, and each opponent makes an action on their cards until it is the expert's turn to act. When the expert acts, however, the betting round continues as well as the hand, and proceeds just as a standard round of Limit Hold'em. After each round, the dealer button rotates one chair to the left and a new hand is dealt. A *chip-count* is stored for the expert, which will reflect the amount of money won/lost during the sequence of hands played.

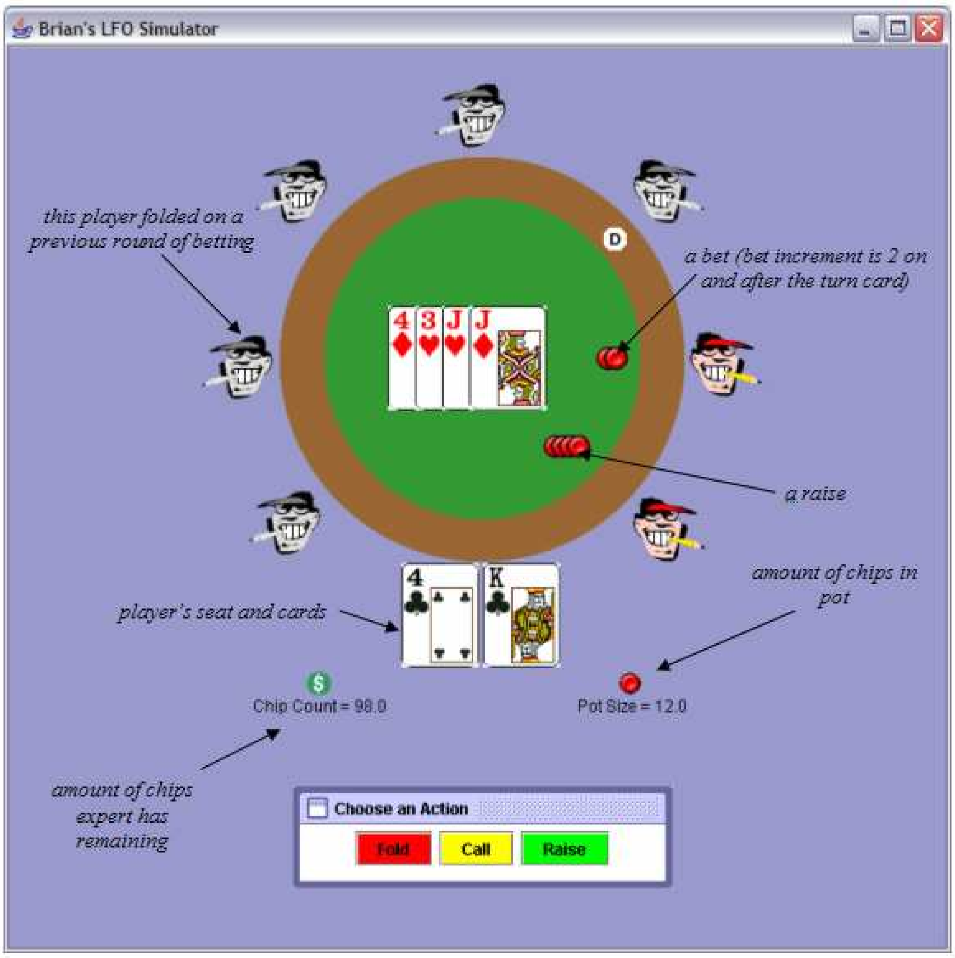

Brian's LFO Simulator
this player folded on a previous round of betting
a bet (bet increment is 2 on and after the turn card)
a raise
amount of chips in pot
player's seat and cards
Chip Count = 98.0
Pot Size = 12.0
amount of chips expert has remaining
Choose an Action
Fold
Call
Raise

Figure 5.6: Screen Shot of Vignette D

In this vignette, the situations encountered by the expert are far more robust and are designed to challenge his playing ability. Because the vignette involves entire rounds, the opponents at the table react to the expert's decisions and use many of the strategies outlined in [SM03] to try and win hands. The intelligence coded for the opponents is described in 5.2.2.3.

Figure 5.6 illustrates a hand being played out in Vignette D. Here, the turn card has just been dealt. The betting round begins to the left of the dealer, and that player bets. Note that in the final two betting rounds (after the turn card and river card), the bet amount is doubled. The player to his left raises, and so the amount is 4 units to call for any remaining player. The expert is next to act. After the turn card, the player's best hand is $J\heartsuit, J\diamondsuit, 4\diamondsuit, 4\clubsuit, K\clubsuit$. He has the option to raise (making it 6 units), to call the 4 unit bet, or to fold. Since a bet has already been made, the player is not allowed to check.

In Figure 5.7, another hand is taking place and the flop has just been dealt. The subsequent betting round begins to the left of the dealer, and that player has checked along with the player to his left. The expert has the $5\heartsuit, 10\diamondsuit$ and can either check or bet. Folding is not an option, since both players who acted before him checked.

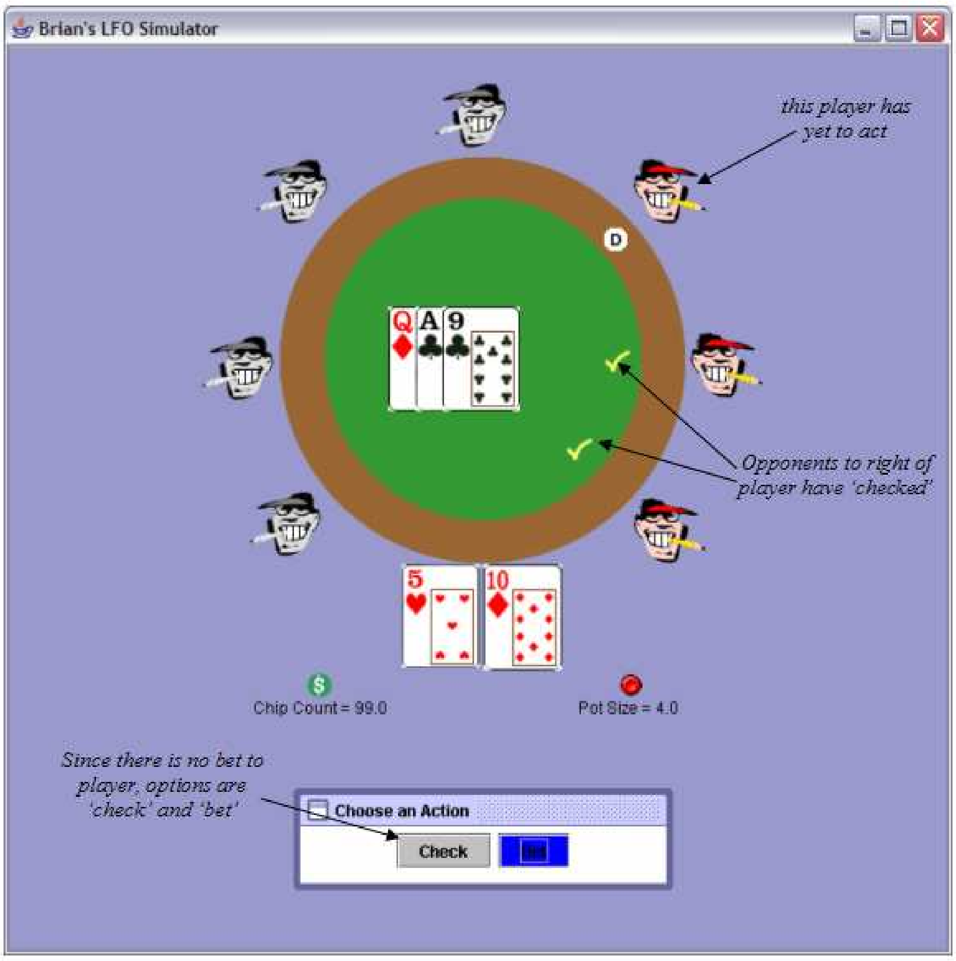

Brian's LFO Simulator
this player has yet to act
D
Opponents to right of player have 'checked'
Chip Count = 99.0
Pot Size = 4.0
Since there is no bet to player, options are 'check' and 'bet'
Choose an Action
Check
Bet

Figure 5.7: Alternate Screen Shot of Vignette D

5.2.2.2 Expert Strategy and Techniques in Texas Hold'em

Unlike the basic strategies that can be employed when navigating a maze, advanced Texas Hold'em strategy is very complex and requires a variety of skills. For instance, advanced players are generally well-versed in the mathematics of the game. and often use the random nature of the cards to their advantage. These players are also adept at *reading* other players - assessing the strength of their hands by observing betting patterns, mannerisms, and behaviors at the table - while frequently changing the style and patterns of their own play so that they are less likely to be read.

The strategy in which the *hole cards* (the two cards dealt face-down to the player) are played prior to the flop is considered to be the most important skill in the game of Limit Hold'em. While these hole cards will inevitably only make up at most 2/5 of the player's eventual 5-card hand, the players who begin with the best hole cards are those who have the best odds to win the hand [SM03]. Likewise, players who begin with poor hole cards - for instance, two cards of low and unpaired rank - have a relatively low chance of their hand turning into the eventual winner, and are best served to fold their hand before the flop is even presented.

Another important concept in Texas Hold'em is that of position. Players who act near the right dealer button are among the last few to act, and have a considerable advantage for that round of play. This is because they are able to observe how the players acting before them play their hand. Because of this important facet of the game, identical starting hands are often played in different ways based on the position of the player holding the hand.

There are many factors that affect the overall action of an expert player besides besides hand strength and position. For example, players will often feign weakness in the opening rounds of betting with a superior hand, in an attempt to trap their opponents in later rounds when the bet amount has doubled. This gambit is known as *slowplaying*.

Expert players will often choose to make a raise when their hand is *on the come* or *on a draw* - meaning that hand is one card away from a *made hand* (when a player has successfully drawn to a hand that will likely win). Consider a player who holds $7\heartsuit J\heartsuit$ with the *board* (the set of community cards currently displayed) showing $8\heartsuit 7\diamondsuit 10\diamondsuit 6\heartsuit$. At this point, the player only has a pair of 7's as his best hand (a relatively weak hand given the cards on the board), however there are many cards that can come on the 'river' that will make a significantly stronger hand. Any 9 will give the player a Jack-High Straight, and any $\heartsuit$ will make the player a flush. However, the player may choose to raise here employing what is known as a *semi-bluff*. Not only does this bluff induce more money to be thrown in the pot (which will result in a larger payoff if his hand is made), but with only a few players in the pot the semi-bluff may induce them all to fold. In this case, the player would win the pot without even making his hand.

The various tactics associated with raising brings to light important reasons why Texas Hold'em Poker works well in an intelligent model where contextual actions can be identified accurately using templates. In these two Hold'em-based vignettes, there is a many-to-one mapping between the context the player employs when making an action and the action itself. For example, consider potential contexts *RaiseToBluff*, *RaiseToSemiBluff*, and *RaiseWithSuperiorHand*. While each context in this set induces the same action (raise), the stimuli for activating them are likely completely different depending on the style and preference of the expert who is playing. However, with knowledge of the player's hole cards, his position at the table, other players at the table, the board, and the previous activity in earlier betting rounds, the context in which the raise was made can likely be inferred. To make this inference, we call to use the Template-Based Interpretation engine described in section 5.3.1. The context topology and associated context templates for both poker vignettes in Chapter 6.

Playing the opening betting round well in Texas Hold'em requires a good sense of hole-card strength. While it is reasonable to assume that two Kings are strong hole cards, there are many hole-card pairings whose strength is not as obvious.

To assist in determining the relative strength of a player's hole cards, we refer again to Sklansky's "Hold'Em Poker for Advanced Players" [SM03]. In this text, the author breaks down the best starting hands and places them into groups according to their relative strength. The strongest hands are in group 1 and so on, as provided below. In this list, the suffix 's' at the end of a card-pairing means the two cards share the same suit. The suffix 'x' refers to any card lower in rank than 10.

Group #	Hole Cards
1	AA, KK, QQ, JJ, AKs
2	TT, AQs, AJs, KQs, AK
3	99, JTs, QJs, KJs, ATs, AQ
4	T9s, KQ, 88, QTs, 98s, J9s, AJ, KTs
5	77, 87s, Q9s, T8s, KJ, QJ, JT, 76s, 97s, Axs, 65s
6	66, AT, 55, 86s, KT, QT, 54s, K9s, J8s, 75s
7	44, J9, 64s, T9, 53s, 33, 98, 43s, 22, Kxs, T7s, Q8s
8	87, A9, Q9, 76, 42s, 32s, 96s, 85s, J8, J7s, 65, 54, 74s, K9, T8

Table 5.2: Pre-Flop Hand Groupings ([SM03])

5.2.2.3 Intelligent Opponents for Poker Training Vignettes

Since this vignette involves the observation of experts playing against opponents, it was important to create opponents who are able to pose at least a minimal challenge. Opponents for the vignettes are programmed with:

- Basic understanding of the strength of its hole cards before the flop
- Basic understanding of hand-strength relative to cards on the board
- Basic understanding of hand-potential relative to cards on the board
- Ability to bluff
- Ability to trap, or slowplay

- Ability to change play based on position and amount of action in betting round

Since Vignette *C* only involves a single decision-point that occurs towards the beginning of a Hold'em round, it was not necessary to require each opponent to reason about their cards. However, having each opponent make a random decision of call, fold, raise, check, or bet would not make for a realistic situation for the expert (and thus would result in unusable expert data). Because of this, the likelihood of each opponent decision option was weighted based on a realistic estimate of the number of players who that will be in the hand after the first round of betting. To do this, the opponent's option weights were skewed towards folding (the most likely opponent move) but favored players either calling (or better-still raising) in good position (within 0 to 2 seats away from the dealer) or on the blinds (a player who has already committed money to a pot is less-likely to fold than someone who can get out for free). A random number then ultimately decides the action taken by the opponent, creating a close approximation of tight, aggressive pre-flop opponents. This is the playing style that expert players tend to employ. A tight, aggressive style calls for the player to fold all but the best of hands, but when they play a hand to be aggressive and choose to raise rather than calling.

For Vignette *D*, however, it became necessary to construct intelligent opponents that could make informed Hold'em decisions throughout entire hands. The first step in doing this was providing the opponents with a knowledge of the strength of their hole cards. To do this, the hand grouping technique from Sklansky ([SM03]) is encoded within each opponent's knowledge-base. During the pre-flop betting round, each opponent will make a decision on the strength of his cards and make an action accordingly. For example, if the opponent is dealt $A\spadesuit K\spadesuit$, he will first examine the table and determine that this set of cards denotes a group 1 hand. Being the strongest group, the opponent will likely make a raise. The actual play that the opponent makes in this situation will be a probabilistic function biased heavily towards raising.

Hand Strength. The most crucial piece of the opponent's intelligence to encode is the ability to read and understand relative hand-strengths based on the cards present on the board at the time of betting. Consider the following vignette. A player is dealt $A\diamondsuit A\clubsuit$ and recognizes that he has a group 1 pre-flop hand. However, after the turn card is dealt, the cards on the board are $5\spadesuit 6\spadesuit 8\spadesuit 9\spadesuit$. At this point any player who has a 7 has a straight, and any player with a spade has a flush. Since both of these hands beat a pair of aces, the opponent should recognize that his hand is weak with respect to the board and fold.

In 1997, Billings et al [BS98] developed an intelligent model of expert Hold'em behavior called Loki. Within Loki, hand strength is calculated at each point in the hand for a player by considering

a=**numHandsPlayerBeats** the number of opponent hand combinations that the player can beat

b=**numHandsPLayerTies** the number of opponent hand combinations that the player can tie

c=**numPossibleOpponentHands** the number of possible opponent hand combinations

From these, a fuzzy value representing the relative strength of the player's hand can be calculated by

$$relativeHandStrength = (a + (b/2))/c \tag{5.1}$$

This equation calculates the probability that the player's hand can beat a random hand. Each opponent makes this calculation to gain an awareness of their relative hand strength at each stage of the betting. After the model computes this value, it examines the value to make an assignment to the enumerated variable handStrength, a qualitative description of the player's hand. The code to make this assignment is duplicated below:

```
if (rhs < .3) handStrength = awful;
    else if (rhs < 0.6) handStrength = mediocre;
    else if (rhs < 0.75) handStrength = playable;
    else if (rhs < 0.9) handStrength = good;
    else if (rhs < 0.98) handStrength = great;
    else handStrength = monster;
```

For our model, the drawing strength of a hand is considered quantitatively using the *pPot* calculation developed in [BS98]. This value represents the potential of the player's current hand to draw to a winning hand. For example, consider the vignette where the player holds $T\spadesuit, J\spadesuit$ and the board is the $A\spadesuit K\spadesuit, 3\diamondsuit$. At this point, the player only holds ace-high $(A\spadesuit, K\spadesuit, J\spadesuit, 10\spadesuit, 3\diamondsuit)$, which is likely not the best hand. However, with two cards to come, the player has many draws at a good and likely winning hand. Any spade makes the player a flush, while any Queen makes the player a straight. The $Q\spadesuit$ would make the player an unbeatable royal flush!

In Loki ([BS98]), *pPot* is determined by considering the player's current hand against all possible hands held by an opponent. All remaining two-card combinations are then dealt as the turn and river, and the two hands are compared. For each possible opponent hand and turn/river combination, the following cases are counted, and a value for pPot is computed using equation 5.2

a=**numBehindToAhead** number of times player had an inferior hand to the opponent, but drew to a better hand

b=**numBehindToTied** number of times player had an inferior hand to the opponent, but drew to the same hand

c=**numTiedToAhead** number of times player had an equal strength hand to the opponent, but drew to a better hand

d=**numBehind** number of times player had an inferior hand to the opponent

e=**numTied** number of times player had an equal strength hand to the opponent

$$pPot = (a + b/2 + c/2)/(d + e) \tag{5.2}$$

Similarly, a player's hand can be the best hand at some point but be vulnerable to getting *drawn out on* by an opponent. This likelihood is represented by the value *nPot* ([BS98]). Consider a vignette where the player holds $A\spadesuit, J\heartsuit$ while the board cards are $J\clubsuit, A\clubsuit, 10\clubsuit$. At this point during the round, the player very likely holds the best hand with two pairs ($A\spadesuit, J\heartsuit, A\clubsuit, J\clubsuit, 10\clubsuit$). However, two cards are left to come, and any opponent with a $\clubsuit$ can make a flush if another $\clubsuit$ is dealt on the turn or river. Other opponents can also make a straight with one card if they hold a Jack or King - both a straight and a flush would beat the player's two pair.

Like the *pPot* value, *nPot* is determined by considering the player's current hand against all possible hands held by an opponent. All remaining two-card combinations are then dealt as the turn and river, and the two hands are compared. For each possible opponent hand and turn/river combination, the following cases are counted:

f=**numAheadToBehind** number of times player had a superior hand to the opponent, but the opponent drew to a better hand

g=**numAheadToTied** number of times player had an inferior hand to the opponent, but the opponent drew to a hand of equal strength

h=**numTiedToBehind** number of times player had an equal strength hand to the opponent, but the opponent drew to a better hand

j=**numAhead** number of times player had a superior hand to the opponent

k=**numTied** number of times player had an equal strength hand to the opponent

pPot is then computed by:

$$nPot = (f + g/2 + h/2)/(j + k) \tag{5.3}$$

The concept of a bluff is a very important concept in poker, though severely overused by amateur players. A 'bluff' is when a player makes a strong play (either a bet or a raise) with a hand that cannot win in a showdown. The idea behind the bluff is to misrepresent the strength of your hand in order to induce your opponents to fold. Not only does this play allow for players to win pots with hands that would otherwise be thrown away, it also allows for players to 'mix up their play' in order to confuse their opponents and keep from being readable. If a player only raises with the most premium of hands, for instance, expert opponents will pick up on this and be able to quickly identify his hand strength by whether or not he is raising. On the other hand, the more often a player makes a bluff, the more often he will be unsuccessful in running other players out of pots. In order to keep one's opponents guessing about your style of play, it is necessary to make a bluff or two to let opponents know that you don't always put money in with the best possible hands.

The bluff is integrated within the model's playing logic within the probabilities of his actions. When an opponent determines his hand to be an 'awful' hand, depending on the state of the hand it introduces a distribution of action-probabilities that favor folding, such as the following code segment:

```
if (handStrength == awful) && (betAmount < 4)
{
    check-likelihood = 0;
    bet-likelihood = 0;
    fold-likelihood = 90; \\
    raise-likelihood = 8; \\
    call-likelihood = 2; \\
}
```

Notice, however, that the raise likelihood for this situation is assigned the value 8. This means that the player will raise $8/(0+0+90+8+2) = 8\%$ of the time. This percentage represents the likelihood that the model will bluff. Note also that the call likelihood is assigned the lowest probability. This is because calling is the worst move that can be made in this situation - an awful hand will only win if the other player decides to fold, which cannot happen in response to a call.

Each opponent model also incorporates the idea of *slowplaying.* As explained in an earlier section, this tactic is employed by misrepresenting the strength of your hand by feigning weakness with your actions - checking and calling instead of raising, acting unimpressed with your hand, etc. While our model is incapable of acting, it does execute slowplaying by favoring checking and calling in the early stages of betting with a monster hand. The code for this tactic is shown below for making an action with a monster hand on the flop.

```
case monster:
{
   if (betAmount == 0) // slowplay most of the time
   {
       check-likelihood = 80;
       fold-likelihood = 0;
       call-likelihood = 0;
       bet-likelihood = 20;
       raise-likelihood = 0;
   }
   else if (betAmount - myBetOnTable < 2) // slowplay
   {
       check-likelihood = 0;
       fold-likelihood = 2;
       call-likelihood = 78;
       bet-likelihood = 0;
       raise-likelihood = 20;
   }
   else // less likely to slowplay with more than 2 bets
   {
       check-likelihood = 0;
       fold-likelihood = 2;
       call-likelihood = 48;
```

```
            bet-likelihood = 0;
            raise-likelihood = 50;
        }
    }
```

The final ability that was incorporated into the intelligent opponent models is the concept of changing play as a function of the player's position. As explained earlier, the *button* is the strongest position at the table. The player on the button, in all rounds after the first round (because of the blinds), acts last and is able to watch everyone else's play prior to playing. Because of this advantage, players on and near the button have the luxury of being more aggressive with weaker cards [SM03]. This advantage is maximized before the flop because of the number of possible hands is minimized, and because it is relatively straightforward to determine a correct course of action based on the betting and the player's position.

To incorporate the idea of position into our model, we simply modify the values for the decision probabilities based on a qualitative representation of the player's position. This representation is determined by the code segment below:

```
// determine player's distance from button
while (distFromButtonCount != position)
{
    distFromButtonCount--;
    if (distFromButtonCount == -1)
        distFromButtonCount = 7;
    distFromButton++;
}
// is player in good, middle, or bad position?
goodPosition = (distFromButton < 3);
```

```
    middlePosition = ( (distFromButton >= 3) && (distFromButton < 5));
    badPosition = ! (middlePosition || goodPosition);
```

This variable is updated after every hand, as the button rotates. It is used in code-segments such as the one below to derive appropriate action probabilities for the player's turn. This code segment is used for action before the flop with a group 6 or group 7 hand.

```
    else if ( (handGroup < 8) && goodPosition)
    {
        check-likelihood = 0;
        fold-likelihood = 45;
        bet-likelihood = 0; // cannot bet
        call-likelihood = 55;
        raise-likelihood = 10;
        if (betAmount-myBetOnTable >= 2)
        {
            fold-likelihood = 70;
            call-likelihood = 25;
            raise-likelihood = 5;
        }
    }
    else if ( (handGroup < 8) && middlePosition) )
    {
        check-likelihood = 0;
        fold-likelihood = 60;
        bet-likelihood = 0; // cannot bet
        call-likelihood = 45;
        raise-likelihood = 5;
        if (betAmount-myBetOnTable >= 2)
        {
            fold-likelihood = 75;
            call-likelihood = 12;
            raise-likelihood = 3;
        }
    }
    else if (handGroup < 8) // bad position
    {
        check-likelihood = 0;
        fold-likelihood = 65;
        bet-likelihood = 0; // cannot bet
        call-likelihood = 35;
        raise-likelihood = 5;
        if (betAmount-myBetOnTable >= 2)
        {
            fold-likelihood = 92;
            call-likelihood = 2;
            raise-likelihood = 8;
        }
    }
}
```

Here, the model is given a better chance to make a raise or call if he is in better position. Note also that there are other factors at work here, such as the amount of the bet at the time it is the model's turn to act.

Finally, since there will be seven models in Vignette D acting as opponents to the expert, a random value representing the model's aggression is computed once for each model. Each model's aggression value is between $[0.5, -0.5]$ and represents the player's capacity for making aggressive and loose plays during given rounds. Since this value is randomly generated for each opponent, no two opponents will act in an identical fashion. The code segment below represents how this value affects each model's action probabilities.

```
// adjust for aggression
if (handStrength <= good) // average-weak hand
{
fold-likelihood *= (1-aggression);
call-likelihood *= (1+aggression);
}
else // good-monster hand
{
    raise-likelihood *= (1+aggression);
    call-likelihood *= (1-aggression);
}
```

If a model is given an aggression value close to 0.5, this code tells us that he is more apt to call a bet with a weak hand and raise a bet with a good hand. On the other hand, a player with aggression value close to -0.5 is more apt to fold with an average-weak hand, and more apt to call (rather than raise) a bet with a good-monster hand.

It is noted that these components were not designed to construct robust models of opponent poker behavior. The only purpose of these models are so that the expert could play against realistic computer-generated opponents in terms of their betting patterns. If the opponent folded every time when raised, for instance, the expert would likely pick up on this and modify his normal playing style to compensate. This would be undesirable for collecting training data.

Since each opponent has been provided the intelligence to play the game autonomously, the poker vignettes can be run as a regular game without scripting the order or sequencing of the cards in play. Cards are dealt randomly to each player, and the opponents play according to what they are dealt. This forces the expert to make intelligent decisions in order to succeed in the game, which makes for a good training exercise for FAMTILE. Using the algorithm, the expert's decision is analyzed by the TBI engine which infers the context that supported it. FAMTILE then learns the criteria for choosing a certain context based on the player's cards, position, the board cards, the action around the table, and the player's previous context.

As discussed earlier, there is no doctrine or field manual for playing poker as there would be for negotiating an assault or ambush on the battlefield. It is as much of an art as a science, and the choices to make particular actions during the game are very much dependent on the individual who is making the decisions. Within our opponent models, these variations are handled probabilistically - in a particular spot with a hand of a certain strength, the opponent will be instructed to raise 80% of the time, but the other 20% of the time he may fold or call. This is the type of information we wish to extract using FAMTILE: when expert A gets a certain hand in a certain position with a certain opponent-betting pattern, what action will he choose in response? For example, consider the case where the player has the $10\diamondsuit, J\diamondsuit$ and the flop comes $Q\diamondsuit, K\diamondsuit, A\diamondsuit$ - the player has an unbeatable Royal Flush! If there are four opponents still in the hand and one opponent bets, what should the expert do? Should he raise with immediately, or string his opponents along by slowplaying and attempt to build the pot? What if the player is first to act - should he lead out and bet, or *check* hoping that another opponent will bet for him? These decisions are very much dependent on both the situation and the individual player making the decision. The goal of FAMTILE is to successfully learn the patterns of a variety of expert players and predict them when similar situations occur.

This task is simplified in Vignette C with no cards on the board and, therefore, only two cards and the opponents' actions to consider. Furthermore, the expert's past context is assumed null because it is the first betting round of a new hand. This vignette is an analog to Vignette A in that past actions are not considered, and the expert makes a set of unrelated decisions instead of a string of decisions throughout a running vignette. The main difference is that a context is identified for the expert based on the environment and his resultant action. Because of this, the entire FAMTILE algorithm is used to perform the learning rather than simply Fuzzy ARTMAP, which is used by itself to learn the experts' behaviors in Vignettes A and B.

Vignette D adds a new degree of complexity to the learning task with the addition of previous context considerations, multiple betting rounds, and community cards that are present during entire rounds of Hold'em poker. These added inputs are intended to break the FAMTILE algorithm so that its deficiencies can be identified, analyzed, and ultimately posed for future work.

5.3 A Prototype Implementation of FAMTILE

The major components of FAMTILE are the TBI Engine, the Fuzzy ARTMAP Neural Network, and the rule extraction engine. Descriptions for each of these subsystems is provided in the following sections.

5.3.1 The TBI Engine

For Vignettes C and D, each action taken by the expert must first be interpreted by the TBI engine before presenting a corresponding output pattern to FAM. This output pattern is the context of the action taken as interpreted by TBI. Individual actions performed by the expert are assumed to be a consequence of the expert acting in a particular context.

To make an interpretation of the context embodied by the expert's recorded action, the TBI engine matched each template against the appropriate conditions present in the observation. The engine then infers the context in which the expert is likely to be acting. This determination is then recorded by the interface module and transformed into a bitstream representing the output pattern for Fuzzy ARTMAP using the technique discussed in Chapter 4.

5.3.1.1 Context Templates

Context Templates are a heuristic description of 'what it means' to exist in a certain context. These structures are most analogous to the context description fields present in each c-schema of Turner's CMB models [Tur98]. Each structure includes a weighted 'checklist' of parameters that represent conditions that often or occasionally exist when in a certain context. When the TBI engine receives all appropriate variables relevant to the current expert action to be interpreted, it will attempt to match them to each context template present for the given vignette. The template that yields the highest score (based on the number of field matches and their appropriate weight) will be chosen as the context most likely to have reflected the expert's action.

For Vignettes A and B, no context templates are required because there are no contexts implied with expert's movement. However, context templates are used for both of the poker vignettes introduced in 5.2.2.1. Each context used for Vignettes C and D is described below. For Vignette C, the contexts calling for a bet or a check are not used. When the player is on the big blind and the pot is not raised, we consider it a call if a check is made.

foldWithWeakHandContext player folds because his hole cards are not strong

foldToStrongBettingContext player folds an otherwise playable hand due to raising and other aggressive table action

foldWithWeakHandContext player folds because his hole cards are not strong

foldToStrongBettingContext player folds an otherwise playable hand due to raising and other aggressive table action

foldInWeakPositionContext player folds an otherwise playable hand

foldWithStrongHandContext player incorrectly folds a strong hand

checkWithStrongHand player checks with a strong hand, possibly with the intention to raise when it is his action. This strategy is known as a 'check-raise', commonly referred to as the strongest play in hold'em

checkWithWeakHand player checks with a weak hand, likely with the intention to fold if there is a bet made

checkWithMarginalHand player checks with a marginal hand, likely to observe the action at the table and gain more perspective on the strength of his hand

callToTrapContext player calls with a strong hand either in or out of position attempting to induce action in later rounds

callWithMarginalHandContext player makes a 'loose call' with a hand that 'tighter' players would likely fold. A 'tight' player typically only plays with very strong hands and draws

callWithDrawingHandContext player calls with good *multiway* hole cards to see a flop, or if he is on a good draw (to a flush, straight, etc.)

callWithWeakHandContext player makes an extremely loose call with a weak hand

raiseWithMultiwayHandContext player makes a raise with a 'multi-way' hand before the flop cards in order to increase the size of the pot and thereby increase the implied odds on his bet. Implied odds are the pot-odds the player is getting on his bet based on the assumption that he will earn extra bets if and when his hand 'hits' (i.e. he makes a winning hand)

raiseWithDrawingHandContext player makes a raise with a strong drawing hand, in an attempt to induce either folds or 'free cards' in later rounds. A free card is when a player on the come acts strong and in a later round induces other players to 'check' around to him, allowing him to see a card without betting or calling.

raiseInPositionContext player makes a raise based mainly on his position at the table

raiseWithStrongHandContext player makes a raise with a strong opening hand

raiseToBluffContext player makes a raise with a weak hand in order to induce the table to fold out

foldInWeakPositionContext player folds an otherwise playable hand

foldWithStrongHandContext player folds a strong hand for no discernable reason

checkWithStrongHand player checks with a strong hand, possibly with the intention to raise when it is his action. This strategy is known as a 'check-raise', commonly referred to as the strongest play in hold'em

checkWithWeakHand player checks with a weak hand, likely with the intention to fold if there is a bet made

checkWithMarginalHand player checks with a marginal hand, likely to observe the action at the table and gain more perspective on the strength of his hand

callToTrapContext player calls with a strong hand either in or out of position attempting to induce action in later rounds

callWithMarginalHandContext player makes a 'loose call' with a hand that 'tighter' players would likely fold. A 'tight' player typically only plays with very strong hands and draws.

callWithDrawingHandContext player calls with good [multiway] hole cards to see a flop, or if he is on a good draw (to a flush, straight, etc.)

callWithWeakHandContext player makes an extremely loose call with a weak hand

raiseWithMultiwayHandContext player makes a raise with a 'multi-way' hand before the flop cards in order to increase the size of the pot and thereby increase the implied odds on his bet. Implied odds are the pot-odds the player is getting on his bet based on the assumption that he will earn extra bets if and when his hand 'hits' (i.e. he makes a winning hand)

raiseWithDrawingHandContext player makes a raise with a strong drawing hand, in an attempt to induce either folds or 'free cards' in later rounds. A free card is when a player on the come acts strong and in a later round induces other players to 'check' around to him, allowing him to see a card without betting or calling

raiseInPositionContext player makes a raise based mainly on his position at the table

raiseWithStrongHandContext player makes a raise with a strong opening hand

raiseToBluffContext player makes a raise with a weak hand in order to induce the table to fold out

In poker, a context is assumed to be a circumstance and/or rationale for making a particular play. The *raise* action, for instance, is divided into contexts that differentiate the inferred reason for the raise. As discussed in [SM03], there is a variety of purposes behind making a raise - to force weaker hands to fold, to get more money into a pot, to bluff thereby causing stronger hands to fold, etc. While the expert's intent cannot be recorded through strict observation, it can be inferred if each of these purposes are encoded by a context. Using expertise gathered from poker experience and from various texts [SM03], [Skl89], [Bru79]), a set of contexts were generated that result in each possible action (raise, call, bet, fold check) in both vignettes. When an observation is presented to FAMTILE's TBI engine, it is compared against the attributes of each context template and generates a score for that template. Consider the template below for the *RaiseInPosition* context. This context refers to a situation where the expert has made a raise based mostly on his strong position relative to the dealer button. As stated earlier, players *on the button* get to act last on each post-flop betting round, giving them a significant advantage of being able to react to each opponent's play.

```
playerAction == Raise;            weight = 6
distanceFromDealerButton = 0;     weight = 3
numPlayersInPot = 2;              weight = 0.5
numBetsToCall = 1;                weight = 0.5
```

Note the weights associated with each attribute. The most heavily weighted attribute is the player's action - if the player does not make a raise, this weight induces the TBI engine to calculate a low score for this template. The other weights are assigned based on their relevance to the context. Aside from the player's action, all attribute scores are assigned based on the following formula introduced in Chapter 4 for 'how close to' attribute-types:

$$score_{att} = (1 - |att_{observed} - att_{template}|/range_{att}) * weight \tag{5.4}$$

Consider the *distanceFromDealerButton* attribute. If that observed value for an observation is 0, the player is on the button and is more likely to raise because of it. To reflect this, the *distanceFromDealerButton* attribute score is maximized when the expert is on the button:

$score_{distanceFromDealerButton} = (1 - |0 - 0|/7) * 3 = 3$

Likewise, as the button moves further away from the expert, this attribute score decreases, making it less likely for the TBI engine to select this template as the inferred context.

For this prototype, context templates are instantiated as Java objects in the following form. The TBI engine is itself a class that stores an array of Template objects. When passed an observation from the FAMTILE interface, it compares the attributes of that observation with those of each template in its array. A total score is computed for each template using the sum of each attribute score as calculated above.

```
public class Template
    final int numAttributes;
    String templateName;
    double [array] attributes;
    double[array] weights;
    double[array] templateScore;
    int[array] range;
```

Using equation 5.4, a score is generated for each context template and each expert observation o_j recorded. The template for which a maximum score is computed is chosen the most likely active context of the expert at time j. This selection will be used for the output pattern of time j, while the observation o_j will be used to construct the input pattern at that point. Presented in an appropriate form, this pairing of the observation o_j and the resultant active context ctx_j inferred by TBI will be presented to Fuzzy ARTMAP for training.

Consider the case where the expert is on the big blind with $8\heartsuit, 9\heartsuit$ and 4 opponents call the forced blind. Given the option to check or raise, the expert chooses to raise with this hand. TBI uses the information given to it and examines the top 5 candidates for the expert's inferred action:

- raiseWithWeakHandContext (template score = 8.2)
- raiseWithDrawingHandContext (template score = 8.7)
- raiseWithMediocreHandContext (template score = 8.0)
- raiseWithStrongButVulnerableHandContext (template score = 6.7)
- raiseWithMonsterHandContext (template score = 6.1)

With the highest template score, raiseWithDrawingHandContext is chosen as the inferred context of the expert at that decision point. The task of FAMTILE is therefore to cluster similar situations that result in that context selection and use them to predict the *raiseWithDrawingHandContext* for future decision-points not seen by the expert. This clustering process begins by presenting the observation to FAMTILE, which is discussed in the following sections.

5.3.2 Generation and Representation of Training Data

Since the training patterns for the neural network come directly from the observations of the expert under study, the amount of diversity among those training patterns is completely dependent on the robustness of the vignette in which that expert operates. Knowledge used for training can only be extracted from observations, and thus any relevant expert knowledge not executed within an observed simulation will not be learned by the neural network. Because of this, there will be gaps in the tactical knowledge representing situations not encountered by the expert during the observation phase. If these gaps are ignored by the learning system, the resultant autonomous agent will have no intelligent response if presented with that unlearned situation. The only defense against these gaps in knowledge is to train the network with as many examples as possible in hopes that they sample as much of the expert's knowledge as possible - in other words, provide vignettes in which the expert must use all or most of his tactical knowledge.

5.3.2.1 Generation and Representation of Training Data for Maze Vignettes

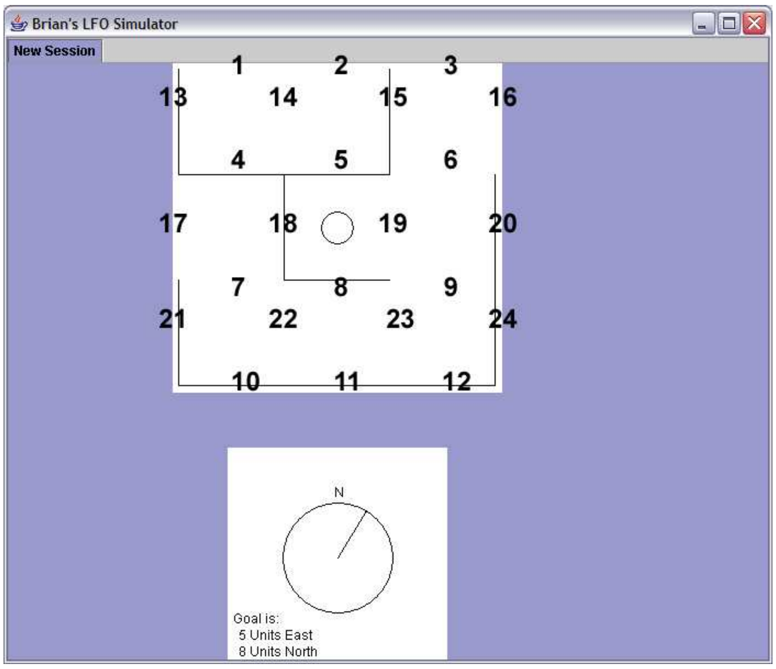
Generating training points for the maze vignettes is a matter of placing the player and goal at random locations within a fixed maze. Each time the player makes a move, the next training point input pattern becomes either a new random position for both him and the goal (as in Vignette A) or the updated maze state based on the direction of the player's previous movement (as in Vignette B. The output pattern for that training point is then the action taken by the expert for the corresponding maze state represented by the input pattern. Each of these patterns, however, must first be translated into a readable form so that they can serve as useful training patterns for FAMTILE.

Because the Fuzzy ARTMAP Neural Network is the component of FAMTILE responsible for doing the low-level learning of the expert's knowledge, it is necessary to pre-process the stimulus data (the maze and goal-state) along with the interpreted context of the expert's action. In this case, the expert's contexts are considered to be simply the actions of moving left, right, up, or down. The observation of the maze state is recorded, and the composition of each processed piece is presented to the network as a single input-pattern. Each input pattern generated represents the position of the player at a given time; the visual of the maze according to the player, his distance to the goal, and the move he takes in response.

As illustrated in figure 5.4, the visible portion of the maze for Vignette B is a 5x5 window in which the player is at the center. To represent this portion of the maze in terms of a bit-stream, each edge on the maze is represented by a 1-bit binary number. This number represents whether or not a wall is present at that edge. In addition to this information, the distance to the goal in both the vertical and horizontal direction is also parsed into the input pattern. To do this, both distances are scaled to a value between 0 and 1 by dividing by the length and width of the maze. A Boolean value is inserted into the data stream representing whether or not the goal position is actually visible on the screen. The reason for adding this value is that the player is given a much better idea as to the correct move to make when the goal is not visible to the player (exists outside the $5x5$ window).

Vignette B requires an additional piece of information - the player's movement history. This is because the board is marked with spaces previously visited by the player during the vignette. These data must be represented in the input pattern because it is potentially an input that the expert uses to make his next decision. To represent this, an additional bit is added to the 4-bit wall-state vector for each space on the visible board that indicates whether or not that space has been visited.

The output pattern is simply the context that the expert has chosen as a response to the stimuli represented by the input pattern. In both maze vignettes, there are four possible transitions that exist. Just as with the wall-state of a space on the maze, a 4-bit binary integer is used. However, here the '1' bit represents the action taken and the other bits hold value '0', meaning that only 4 possible output-patterns exist. These 4-bit representations of the players' move will also be inserted into the input patterns for the next move made by the expert (for Vignette *B*).



Goal is:
5 Units East
8 Units North

Figure 5.8: Wall-State and Other Map Inputs for Vignette A

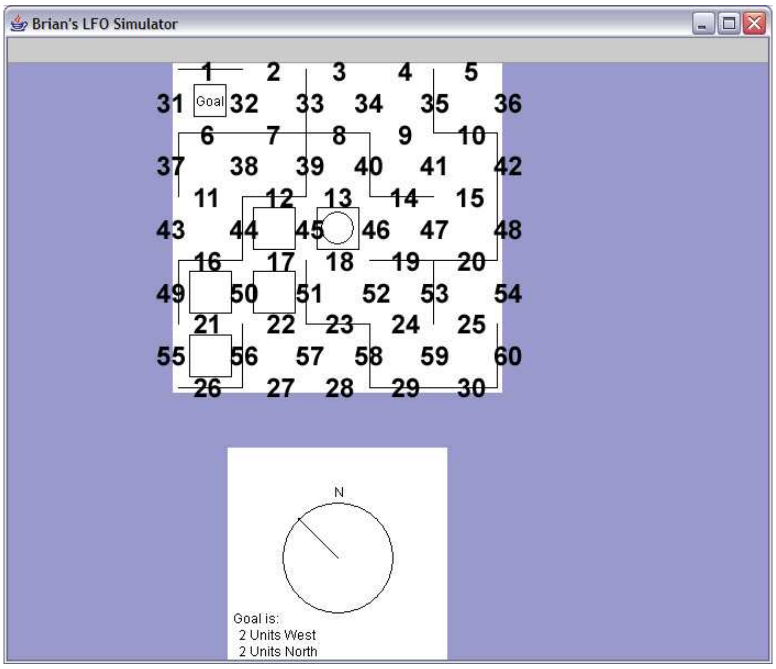
The representations for the input and output vectors used for Vignettes *A* and *B* are summarized below.

$$\overbrace{0.5, 0.8}^{distance} \quad \underbrace{0}_{goal*visible} \quad \overbrace{000110010111\ldots\ldots}^{wall*states*for*3x3*window} \tag{5.5}$$

Input Vector for Vignette A

$$\underbrace{0001}_{action*4*player*goes*down} \tag{5.6}$$

Output Vector for Vignette A



Goal is:
2 Units West
2 Units North

Figure 5.9: Wall-State and Other Map Inputs for Vignette B

$$\overbrace{-0.2, 0.2}^{2*west*2*north} \quad \underbrace{1}_{goal*is*visible} \quad \overbrace{0100}^{action*taken*last*move*right} \quad \underbrace{001011\ldots}_{visited*spaces} \quad \overbrace{1000011101010......}^{wall*states*for*visible*5x5*window} \tag{5.7}$$

Input Vector for Vignette B

$$\underbrace{0100}_{action*2*player*goes*right} \tag{5.8}$$

Output Vector for Vignette B

$$\underbrace{0}_{space*0*not*visited} \quad \overbrace{1}^{space*1*visited} \quad \underbrace{1}_{space*2*visited} \quad \ldots \tag{5.9}$$

Representing Visited Spaces in Vignette B

5.3.3 Generation and Representation of Training Data for Poker Vignettes

To accurately represent the situation presented to the expert player, the simulation must generate and record the following pieces of information for each observation:

- Player's hole cards
- Board cards (Vignette D)
- Player's position

- Position of the Button
- Action of all players acting before the player
- Amount of money in pot (Vignette D)
- Player's action

To generate this information, the simulation deals a random hand to the expert and seven automated opponents. Each opponent makes an action until it is the player's turn. At this point, the state of the hand is recorded along with the action made by the player for his turn. For Vignette C, each of these points occur during the betting round prior to the flop. Vignette D includes points throughout an entire hand of Hold'em.

5.3.3.1 Representation of Training Data for Vignette C

The simulation records the hand-state information in character form, and that character sequence is then converted by the interface into an appropriate input/output vector to present to Fuzzy ARTMAP. This process is similar to that used to record and convert the stimuli/action data for vignettes 1 and 2. However, observation for Vignette C also involves the conversion of the player's action to an interpretation of the expert's context.

After the expert observation sequence has been generated, representations for each situation are in the format seen in equation 5.10. In these equations, hole cards are identified by rank and suit (e.g. *8s* is the 8 of spades, *tc* is the 10 of clubs, etc.). Opponent actions are either f (fold), c (call), or r (raise) in vignette C. In vignette D, players also have the option to bet (b) and check (k). Opponents who have yet to act are identified by a '-'. Consider the situation illustrated by figure 5.10.

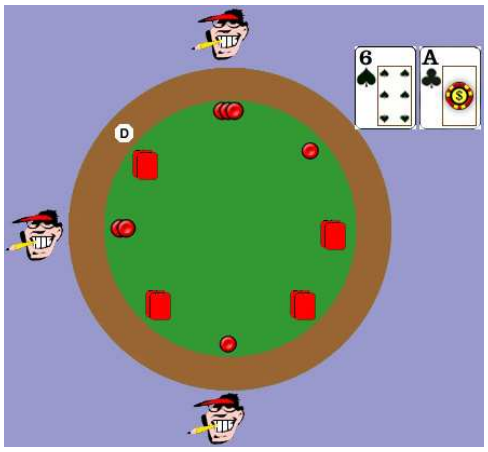


Figure 5.10: An Example Decision Point in Vignette C

To represent this situation, we record the player's hole cards, his position at the table, and the actions of his opponents. This situation is represented by the expression below. Note that the opponent actions begin with the player to the left of the big blind.

$$< \overbrace{6s, Ac}^{holecards} \quad \underbrace{6}_{distancefrombutton} \quad \overbrace{f, f, c, f, r, f, r, -}^{opponentactions} > \tag{5.10}$$

The player's hole cards hc_1 and hc_2 are converted into an input pattern first by rank, then by suit. The rank of each card is scaled into a fuzzy value between 0 and 1 (a *fuzzy bit*) by simply dividing the card's rank by 14 - the rank of the highest card (Ace). One fuzzy bit is then used to represent whether or not the cards are *suited*(share the same suit). If they are, that bit is set to 1, otherwise it is set to zero The player's distance to the button, db, is also represented as a fuzzy bit. This value is determined by gauging the player's position at the table relative to the button. The strongest position is to be on the button, so this position is assigned the value 1. The remainder of the seats are assigned according to figure 5.11

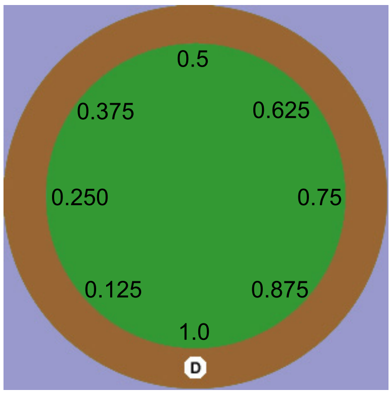


Figure 5.11: Table Position Value Assignments for Vignette C Input Patterns

Each action in the action sequence can be represented by a 3-bit binary integer that designates the action taken. For the action sequence, a 'c' represents a call, an 'f' represents a fold, an 'r' represents a raise, and an 'x' represents a player who has yet to act. We can represent these actions by representing the amount of money each player has put into the pot.

The minimum amount of chips that a player can have for his bet is 0, which occurs if he has folded or has yet to act. Since there is a maximum of three raises, the maximum number of chips that a player can have as his bet is 4. Therefore, we can consider each chip bet by a player to hold a value of 0.25. If a player is on the small blind and has yet to act, his 1/2 chip can be represented by $0.25/2 = 0.125$. Going around the table, the amount of chips each player has in the pot is converted to a fuzzy bit.

Combining each of the bits and fuzzy bits described above, we can now express an input pattern for Vignette C that represents the action at the table directly prior to the expert's decision point.

$$< \overbrace{0.4286}^{hc_1}\overbrace{1.000}^{hc_2}\underbrace{\;0\;}_{suited}\overbrace{0.250}^{db}\underbrace{0, 0, 0.25, 0, 0.5, 0, 0.75, 1}_{actions} > \tag{5.11}$$

5.3.3.2 Representation of Training Data for Vignette D

For Vignette D, this observation is expanded to include interpreted information about the player's hand and position relative to the rest of the table. To do this, the following parameters are used:

hole cards rank and of player's two hole cards. Both are scaled to values < 1

suited boolean value indicating whether cards have the same suit, e.g. $5\heartsuit, J\heartsuit$

hand strength fuzzy value of player's hand, as determined in equation 5.1 ([BS98])

pPot fuzzy value representing the potential of the player's hand drawing to a winning hand, as determined in equation 5.2

nPot fuzzy value representing the potential of the player's hand decreasing in strength due to future board cards, as determined in equation5.3

distance from dealer button

betting round 4-bit binary value, representing the current betting round. Either preFlop (0 0 0 1), flop (0 0 1 0), turn (0 1 0 0), or river betting round

last action 4-bit binary value representing what the player did on his last turn to act. Either nothing (0 0 0 0), fold (0 0 0 1), check (0 0 1 0), call (0 1 0 0), or raise (1 0 0 0)

pot size number of chips currently in the pot, scaled to a fuzzy value < 1

opponent bets in pot Scaled to a fuzzy value < 1 by the size of the largest bet

Here is an example observation for vignette D:

$$\overbrace{tdks}^{hole*cards}\ \underbrace{tc9h8h--}_{board*cards}\ \overbrace{6}^{position*of*button}\ \overbrace{r}^{last*action}\ \underbrace{f}_{betting*round}\ \overbrace{2.5}^{pot*size}\ \underbrace{ffrfc--}_{opponent*actions}$$

Based on the conversions listed in the above descriptions, here is the corresponding input pattern for this observation that would be presented to Fuzzy ARTMAP:

$$\overbrace{0.714, 0.929}^{hole*cards}\ \underbrace{0}_{suited}\ \overbrace{0.78}^{hand*strength}\ \underbrace{.3}_{pPot}\ \overbrace{.2}^{nPot}\ \underbrace{5}_{distance*from*button}\ \overbrace{1000}^{last*action}$$

$$\underbrace{0010}_{betting*round}\ \overbrace{0.025}_{pot*size}\ \underbrace{0.75}^{num*players*in*pot}\ \overbrace{0, 0, 1, 0, 1, 0, 0}^{opponent*bets*in*pot}$$

5.3.3.3 Generation of Output Patterns for Poker Vignettes

When the simulation records the expert's action during observation, the result is simply a character value representing either a raise, fold, or call. For both poker vignettes, however, Fuzzy ARTMAP is used to create a mapping between the observed situation and the expert's choice of context, not simply his action. To make this transformation, the interface extracts necessary variables from the input pattern to present to the TBI engine, which makes a prediction of the most likely context that the expert has chosen. For Vignette C, there are 12 contexts from which the expert can select.

foldWithWeakHandContext Player folds because his hole cards are not strong.

foldToStrongBettingContext - Player folds an otherwise playable hand due to the bet amount and other aggressive table action

foldInWeakPositionContext - Player folds an otherwise playable hand

foldWithStrongHandContext - Player folds a strong hand for no discernable reason

callWithDrawingHandContext - Player calls a good *multiway* hand in order to see a flop.

callToTrapContext - Player calls with a strong hand either in or out of position attempting to induce action in later rounds

callWithMarginalHandContext - Player makes a *loose call* with a hand that *tighter* players would likely fold. A loose call indicates a call when the player holds a relatively mediocre or poor hand. A tight player typically only plays with very strong hands and draws.

callWithWeakHandContext - Player makes an extremely loose call with a very weak hand

raiseWithDrawingHandContext - Player makes a raise with a strong drawing hand, in an attempt to induce either folds or *free cards* in later rounds. A free card is when a player on the come acts strong and in a later round induces other players to 'check' around to him on the next round, allowing him to see a card (and possibly catch his draw) without betting or calling.

raiseInPositionContext - Player makes a raise based mainly on his position at the table

raiseWithStrongHandContext - Player makes a raise with a strong opening hand

raiseToBluffContext - Player makes a raise with a weak hand in order to induce the table to fold out A 12-bit binary number, therefore, is used to represent the context identified for the expert's action and constitutes the entire output pattern that is presented to Fuzzy ARTMAP at ART_b.

An output pattern for Vignette C would therefore be a 12-bit binary number with all but one number set to zero. That number, in the j^{th} position, represents that the TBI engine has identified context j as the active context for the observation represented by the input pattern.

In Senario D, there are 23 contexts:

foldWithWeakHandContext Player folds because his hole cards are not strong.

foldWithMediocreHandContext Player folds an average hand

foldWithDrawingHandContext Player folds a hand that could draw to a winner

foldWithStrongHandContext Player unknowingly folds a strong hand

checkWithWeakHandContext Player checks with a weak hand, likely with the intention to fold if there is a bet made

checkWithDrawingHandContext Player checks a hand that is on the come to a possible winning hand, and would like to see another card for little to no money

checkWithMediocreHandContext Player checks with a marginal hand, likely to observe the action at the table and gain more perspective on the strength of his hand

checkWithMonsterHandContext Player checks with a monster hand, to fake weakness and induce action from his opponents

checkWithStrongButVulnerableHandContext Player checks with a strong hand that may get drawn out on

callWithWeakHandContext Player makes an extremely loose call with a weak hand

callWithMediocreHandContext Player makes a 'loose call' with a hand that 'tighter' players would likely fold. A 'tight' player typically only plays with very strong hands and draws.

callWithDrawingHandContext Player calls with good *multiway* hole cards to see a flop, or if he is on a good draw (to a flush, straight, etc.)

callWithMonsterHand Player calls with a monster hand, attempting to slow-play his hand

callWithStrongButVulnerableHandContext Player calls with a strong hand vulnerable to drawing hands

betWithWeakHandContext Player bets with a weak hand to bluff

betWithMediocreHandContext Player bets with a marginal hand, either to bluff or to induce an even weaker hand to fold

betWithDrawingHandContext Player bets a drawing hand on a semi-bluff.

betWithStrongButVulnerableHandContext Player bets with a strong and likely winning hand

betWithMonsterHandContext Player bets with a hand that cannot be beaten

raiseWithWeakHandContext Player makes a raise with a weak hand in order to induce the table to fold out (a bluff)

raiseWithMediocreHandContext Player makes a raise with a mediocre hand, either to bluff or to induce a weaker drawing hand to fold

raiseWithDrawingHandContext Player makes a raise with a strong drawing hand, in an attempt to induce either folds or 'free cards' in later rounds. A free card is when a player on the come acts strong and in a later round induces other players to 'check' around to him, allowing him to see a card without betting or calling.

raiseWithStrongButVulnerableHandContext Player makes a raise with a strong hand that could get drawn out on

raiseWithMonsterHandContext Player has a nearly unbeatable hand, and is raising to extract the most amount of chips out of his opponents

An output pattern for Vignette D would, likewise, be a 23-bit binary number with all but one number set to zero. That number, in the j^{th} position, represents that the TBI engine has identified context j as the active context for the observation represented by the input pattern.

$$\overbrace{00000100000000000000000}^{context-6-chosen} \tag{5.12}$$

To evaluate the utility of representing expert actions through the context in which they were executed, a second set of output patterns are generated that represent the expert's actual action. The expert has three available actions in Vignette C - fold, call, and raise, therefore his selected action can be represented by a 3-bit binary integer (there are four options in vignette four because the player can 'check'). Presenting the input patterns paired with output patterns of this form provides a learning task similar to those of vignettes A and B. It also provides baseline results that can be measured against the set generated using context selections as the output patterns.

5.3.4 The Fuzzy ARTMAP Neural Network

After the expert has executed a vignette, and the results have been interpreted by the TBI engine and the simulation interface, all of the expert data now exists as simply a sequence of binary strings. These strings represent a set of input and output patterns that are then presented to the neural network for learning. Before this presentation, a neural network class is instantiated and provided with values for its learning parameters. Next, the training and testing set are read in by the neural network, complement-encoded, and stored into two arrays of sizes m and n, where m is the number of training examples and n is the number of testing examples (see Chapter 4). The neural network then trains by presenting itself each training pattern individually and in sequential order. Since Fuzzy ARTMAP only requires one pattern presentation *epoch* for learning; the training ends once all pattern have been presented once. One trait of the Fuzzy ARTMAP neural network is its sensitivity to the order in which the patterns are presented during training. Two identical Fuzzy ARTMAP neural networks trained by the same set of training data will produce different results if the order of patterns presented during training differs. Therefore, it became necessary that, for each vignette and expert tested, the neural network be trained multiple times with a different order of pattern presentation given for each learning session. By doing this, and computing an aggregate score for the network's testing results, the possibility for skewing the results because of a specific pattern-presentation order is eliminated.

After FAM has learned each training pattern, a set of clusters will exist at the input and output. Each cluster in the ART_a module of FAM will map to an action cluster in ART_b for the maze vignettes, and to an output-context cluster for the poker vignettes. The ART_a clusters represent a group of closely matched input patterns that each map to the same cluster in ART_b. In the maze vignettes, patterns in the same cluster may represent similar positions within the maze where the expert chose the same direction. In the poker vignettes, such patterns may share features such as hole-card strength, player position, or even opponent betting patterns. The only guaranteed similarity amongst patterns in the same cluster are their input patterns. However, the relative *granularity* of each cluster (which represents the similarity that must exist for patterns to share the same cluster) can be adjusted for FAM using the $\bar{\rho}_a$ parameter discussed in Chapter 4.

Since the output patterns contain only one piece of information (the index of the output context inferred for the input), only one output cluster exists per output context.

For this prototype, FAM clusters are stored as 1-dimensional arrays - one for each cluster in the ART_a and ART_b modules. Each entry in these arrays represents a field value of that cluster. To store the mappings, a separate array is created that represents the InnerART module of FAM. This array contains one field for each cluster created in ART_a. The value stored in each field is the index of its mapped cluster in ART_b. For instance, if ART_a cluster i is mapped to cluster j in ART_b, the InnerART array would look like

$$[ia_1, ia_2, \ldots, ia_c = j, \ldots] \tag{5.13}$$

Here the field containing the value j is stored in the i^{th} slot.

5.3.5 Rule Extraction and CxBR Model Composition

When implementing CxBR models in the framework developed by Norlander ([Nor99]), all context-transition logic must be represented in the form of sentinel rules. For the simulation used in this prototype, however, this framework is not required to develop CxBR models of the expert's observed behavior. The reason for this is that each context used for the training vignettes map to a single, atomic action. For instance, the *raiseInPosition* context implies that a raise is made - no further intelligence is needed to define that context. This condition exists for each context used. Because of this, no rule extraction prototype was developed for this implementation.

The rule extraction algorithm [CT95] discussed in Chapter 4, however, can be implemented within FAMTILE to make the transformation. To do this, first each cluster array from ART_a along with the InnerART array is retrieved from the network. To convert each cluster to a rule, the following steps are executed:

- Count the number of fields n within the ART_a clusters
- Field j represents the minimum value for that cluster
- Field $n/2 + j$ represents the maximum value for that cluster
- Calculate the range for each cluster field and interpret those ranges in terms of the observation sequence
- Each range constitutes a condition for entering the output context represented by the corresponding InnerART value
- The conjunction of each range stored in a cluster represents the transition rule for that cluster
- Compose each rule using the format given in [Nor99]

CHAPTER 6

EVALUATION OF FAMTILE PROTOTYPE

In this Chapter, we outline the testing procedures used to evaluate FAMTILE along with the summarized results of those tests. The testing procedure uses eight testing Scenarios, with each using one of the four Vignettes described in Chapter 5. To populate each Vignette with data for training and testing, three human test subjects were observed over a fixed number of decision points. The observations collected from these test subjects were used as the source of training and testing data for both Fuzzy ARTMAP and FAMTILE.

As discussed in Chapter three, the primary interest of this research is the class of behaviors and tasks which are composed of lower-level behaviors that (a) can be identified during observation, (b) do not need to be learned individually, and (c) are known to be characteristic of the task/behavior we do wish to learn.

6.1 Overview of Testing Scenarios

Seven testing Scenarios are used to evaluate Fuzzy ARTMAP and FAMTILE. In this section, each Scenario will be briefly introduced. Table 6.1 contains an overview of the algorithms and Vignettes used for each testing Scenario described here.

Table 6.1: Summary of Test Scenario Parameters

Scenario	Vignette Used	Expert Contexts	Learning Algorithm Evaluated
1	A	no	FAM
2	B	no	FAM
3	C	no	FAM
4	D	no	FAM
5	C	yes	FAMTILE
6	D	yes	FAMTILE
7	D	n/a	both

6.1.1 Testing Scenarios #1 and #2

With Scenarios A and B, we consider a basic instantiation of this class of behaviors. Here, low-level actions are represented by direction-choices - either left, right, up, or down. These directions are also representative of the entire action-space of the behavior, as no other actions are permitted within the maze. Scenarios #1 and #2 are an evaluation of Fuzzy ARTMAP's ability to learn subject behavior from Vignettes A and B, respectively.

When learning subject behaviors for Vignettes A and B, all possible contexts that may provide motivation for each action is ignored during training. For instance, the motivation of going left because the goal state is in that direction is considered identical to the motivation of going left simply because that is the best alternative. Because of this, contexts for making particular moves are not considered in these two testing Scenarios.

6.1.2 Testing Scenarios #3 and #4

In Scenarios #3 and #4, the subjects perform the more complex activities related to Vignettes C and D - participating in hands of Texas Hold'em. As discussed in Chapter 5, these Scenarios involve reasoning about several observations, where each may have a significant impact on the subject's eventual decision. Furthermore, each action taken by the subject may be the result of one or more motivations. A raise or bet in one situation, for instance, may be used for a completely different purpose than it would in another situation.

Scenarios #3 and #4, however, intentionally ignore this fact. When a player makes an action, it is presented to Fuzzy ARTMAP as that action regardless of any supposed context behind it, as illustrated in Figure 6.1. Because of this, these tests will mirror those of Scenarios #1 and #2, but with a more complex behavior.

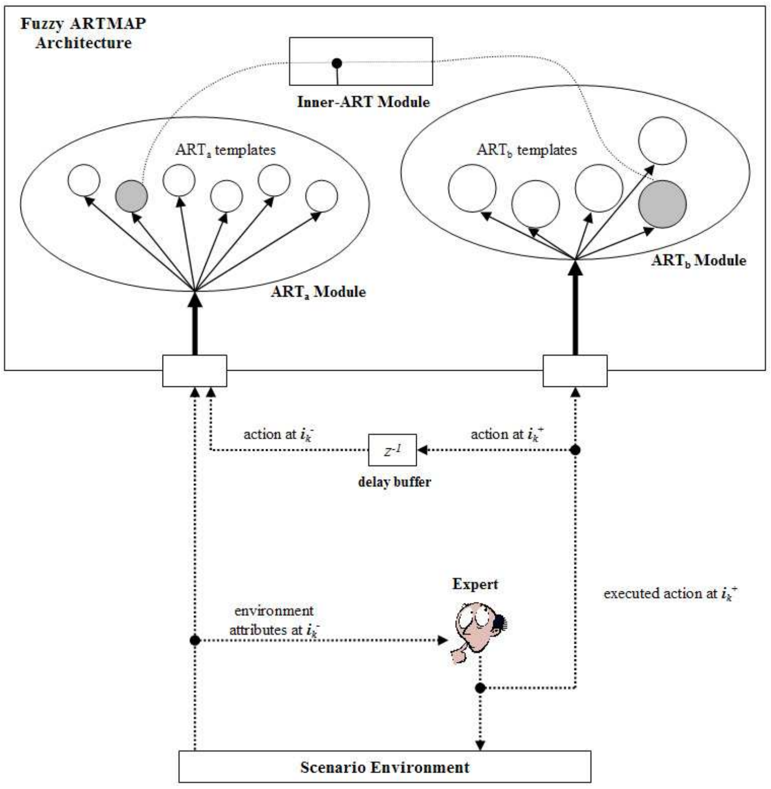


Figure 6.1: Training FAM With Subject Actions

Figure 6.1 represents the process of training FAM with the actions of a specific test subject. the subject is a presented with some stimuli that triggers a decision point at i_{k^-}. In response to that stimuli, the subject executes an action (considered to occur at i_{k^+}). That action is presented to the ART_b module of Fuzzy ARTMAP. At the input module ART_a, both a representation of the stimuli and the action performed at the previous time step $i_{(k-1)^+}$ is presented.

6.1.3 Testing Scenarios #5 and #6

By contrast, Scenarios #5 and #6 consider the context of each subject action prior to creating a training pattern for the neural network. Before running these test Scenarios, a set of contexts were developed for both Vignettes C and D in an attempt to capture all possible motivations for each action. During training, therefore, the subject's action at each decision point is first examined by a TBI engine to infer a context for that point. This training technique is illustrated in Figure 6.2.

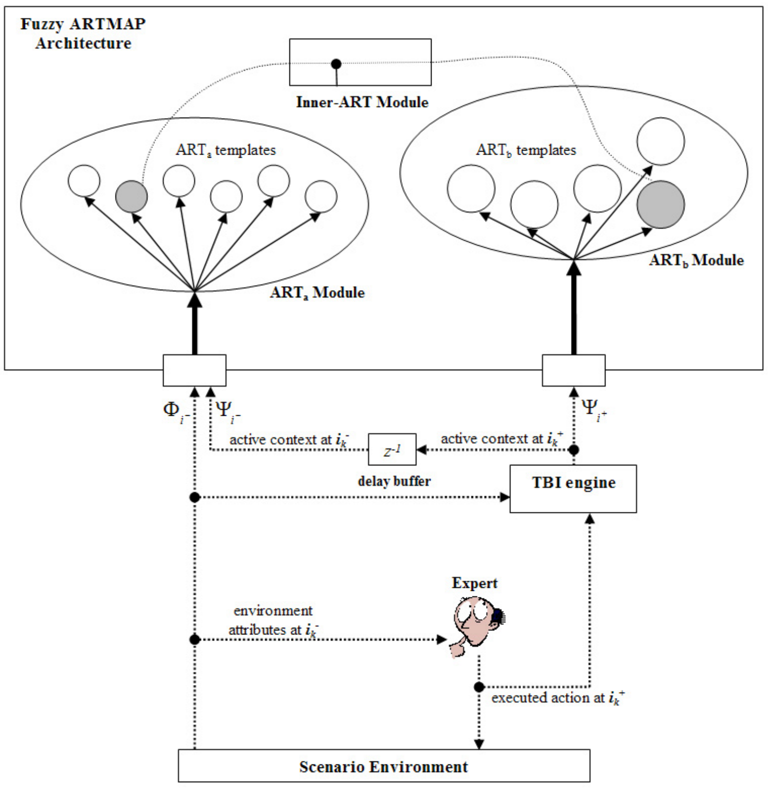


Figure 6.2: Training FAMTILE With Inferred Subject Contexts

Here, ART_a is instead presented with the subject's *inferred active context* determined for i_{k^+}. At ART_b, similarly, the context inferred for $i_{(k-1)^+}$ is introduced along with the a representation of the stimuli present at i_{k^-}

6.1.4 Testing Scenario #7

In Scenario #7, both Fuzzy ARTMAP and FAMTILE attempt to execute Vignette D, making a sequence of decisions just as the subject did during observation. To perform these tests, the entire observation sequence collected from the subject's activity was used for training. This sequence is then used to train both Fuzzy ARTMAP and FAMTILE.

For Fuzzy ARTMAP, just as in Scenarios #3 and #4, the actions of the subject are presented output patterns regardless of the motivation behind the action.

1. The entire observation sequence gathered from subject i is used to generate a set of training patterns - no validation set is generated
2. FAM is trained with the complete set of patterns and generates a mapping between observation and action
3. FAM takes the place of the test subject within the simulation and executes the training Scenario
4. For each decision cue presented by the simulation, FAM predicts an action
5. That action is then executed in the simulation and the Vignette continues

6. The overall performance of both subject i and FAM are compared based on metrics collected throughout the execution of the Scenario

When testing FAM separately, the network is trained with the subject's action being presented at its output. For FAMTILE, the actions of the subject are first translated to an inferred context for each decision point, and a representation of that context is instead presented to the FAM network within FAMTILE.

After training of each system was complete, the simulation was run again. This time, each decision cue was presented to the algorithm. Based on its knowledge, then, FAMTILE predicts a low-level context and the actions associated with that context were executed. The standalone FAM outputs only a predicted action.

1. The entire observation sequence gathered from subject i is used to generate a set of training patterns - no validation set is generated
2. FAMTILE is trained with the complete set of patterns, and generates a mapping between observation and low-level context
3. FAMTILE takes the place of the subject within the simulation and executes the Vignette
4. For each decision cue presented by the simulation, FAMTILE predicts a low-level context
5. The knowledge for that low-level context implies an action that is then executed, and the Vignette continues
6. The overall behaviors of both subject i and FAMTILE are compared based on metrics collected throughout the execution of the Vignette

6.2 Evaluation Procedures for FAM and FAMTILE

In this section, we introduce the two methods used in the testing Scenarios to quantify the accuracies of FAM and FAMTILE in predicting subject actions in Vignettes A through D.

6.2.1 Computing a Predictive Accuracy

In Scenarios 1 through 6, both FAM and FAMTILE are evaluated using the following technique:

1. Subject is observed while executing a specific behavior, such as navigating a maze or playing poker
2. An observation sequence is recorded for that subject and converted into a set of training patterns
3. These patterns are randomly divided into a training set and a validation set
4. The neural network is trained using each pattern in the training set
5. The network is then presented with the input patterns from the validation set
6. A percentage is taken of how many times the network correctly predicts the output pattern out of the total number of validation-set presentations

This percentage is referred to as the *predictive accuracy* of the system for a particular validation set. For example, a network would achieve a predictive accuracy of 86% if it correctly predicts 86 out of 100 inferred output patterns from a validation set. As stated earlier, this validation set is comprised of patterns from the same set of observations as those making up the training set. However, the patterns generated for validation are separate than the ones generated for training.

For these testing Scenarios, predictive accuracies will be taken in three types of instances. For Scenarios 1 through 4, this accuracy will represent how well Fuzzy ARTMAP correctly predicts the action of the test subject.

When examining FAMTILE in Scenarios 5 through 7, we first consider the case where the predictive accuracy represents the ability to predict the subject's *inferred active context* (not his action!). This process is summarized below.

1. Subject is observed while executing a specific behavior, such as navigating a maze or playing poker
2. An observation sequence is recorded and converted to a set of input patterns in the manner described in section 4.12. This set of patterns is randomly separated into training and validation sets
3. During training, the network is provided the input observation at ART_a and the predicted active context of the subject at ART_b
4. During validation, the network is only provided the input observation at ART_a
5. the network identifies a cluster in ART_a to match with the input pattern, and from the mappings created during training, follows the mapping to the corresponding cluster in ART_b. This cluster represents a specific output context

6. The output context predicted by FAMTILE is compared with the output context inferred by TBI for each test pattern and the actual action performed by the subject for that decision point

7. If the inferred context matches the predicted context, then FAMTILE has made a correct prediction of that pattern

In a separate set of runs, a new predictive accuracy for FAMTILE is calculated that represents its ability to predict the subject's **action** (equivalent to the predictive accuracies for FAM):

1. Subject is observed while executing a specific behavior, such as navigating a maze or playing poker

2. An observation sequence is recorded and converted to a set of input patterns in the manner described in section 4.12. This set of patterns is randomly separated into training and validation sets

3. During training, the network is provided the input observation at ART_a and the inferred active context of the subject at ART_b

4. During validation, the network is only provided the input observation at ART_a

5. the network identifies a cluster in ART_a to match with the input pattern, and from the mappings created during training, follows the mapping to the corresponding cluster in ART_b. This cluster represents a specific output context

6. The output context predicted by FAM is then translated to an action. That action is then compared with the actual action performed by the subject for that decision point

7. If the actual action matches the predicted action, then FAMTILE has made a correct prediction of that validation pattern

The reason for this second type of predictive accuracy calculations for FAMTILE is so that its results can be more easily compared to those of FAMTILE. Since there is a many-to-one relationship between the contexts and their resultant actions, comparing the ability of FAM to predict actions against the ability of FAMTILE to predict contexts is irrelevant.

Furthermore, if FAMTILE achieves a statistically significant increase in predictive accuracy versus FAM, it indicates that there is indeed utility in considering context for our test Scenarios.

Though our test Scenarios for FAMTILE only consider contexts that imply a single action, it is easy to consider cases where they could instead imply more complex behaviors. For example, consider the tactical behavior of maneuvering a squad for an assault on a enemy's location. At the lowest level of behavior, 'actions' would involve mechanical movements such as crawling, walking, raising a gun or shooting. However, these movements are the consequence of higher-level behaviors (such as seeking cover, gaining an angle on the enemy, etc.) which themselves are low-level behaviors comprising the assault.

6.3 Learning Parameters for FAM

For each Scenario, the following parameters (defined in section 4.2.2) within Fuzzy ARTMAP were held constant:

- $\epsilon = 0.00001$

- $\beta_a = 1$
- $\beta_b = 1$
- $\rho_b = 1$

The only parameter that was modified during the testing phase was the baseline vigilance $\bar{\rho_a}$. As stated in Chapter 4, this parameter has a direct effect on the granularity of the clusters generated in the ART_a module. These clusters represent groups of input patterns presented to ART_a where each pattern maps to the same output pattern (either an action as in Scenarios #1 and #2, or a context as in Scenarios #3 and #4) and are closely matched with respect to their individual fields. $\bar{\rho_a}$ affects this granularity by raising the vigilance parameter, which is responsible for rejecting the addition of new input patterns to a certain cluster if it fails to match a certain criteria. This change ultimately increases the number of input pattern clusters created in ART_a by decreasing their overall size (and inclusiveness). This effect is illustrated quantitatively in the following sections.

6.4 Scenario #1 Testing

This section defines the purpose, motivation and results for the first of four testing Scenarios. This Scenario is a baseline evaluation of Fuzzy ARTMAP.

As described in the previous section, Vignette A involves a maze environment. The test subject is placed at the center of a $3x3$ subsection of a $10x10$ maze, and provided a vector indicating the distance and direction towards the goal position (see Figure 5.3). He will then choose the direction that he would begin moving towards that goal state. After this direction is chosen, he will be presented a brand new position and goal location, and will again choose a direction to proceed. The subject does not continuously navigate towards the goal in this Scenario; that behavior is reserved for Vignette *B*.

The observation recorded by the simulation represents the $3x3$ subsection of the maze, along with the vector that provides the distance and direction from the character (maneuvered by the subject) and the goal state. The subject's action in response to that observation is recorded by the simulation - either left, right, up, or down.

In Scenario #1, we examine the predictive accuracy of FAM for subject action for Vignette *A*.

6.4.1 Scenario #1 Motivation

Essentially, the task for FAM in this Scenario is to create a mapping between the maze topology and a predicted direction for the test subject facing that situation - either left, right, up, down. In this Vignette, there is no implied notion of 'context' used within the learning environment for this Scenario, and thus there is no interpretation made by the TBI engine.

The purpose for this testing Scenario, therefore, is to confirm the effectiveness of the FAM within FAMTILE. The results of the Scenario #1 tests illustrate the effectiveness (or ineffectiveness) behavior of Fuzzy ARTMAP for this Vignette. From here, we then proceed to gradually more complex Scenarios involving more complicated observation landscapes and, eventually, subject actions that must be interpreted. Vignette *A*, as discussed above, is the simplest of the four Scenarios.

This Vignette examines learning the high-level behavior of navigating a maze by examining the lower-level actions of moving up, down, left and right.

6.4.2 Scenario #1 Results

Three subjects were used for Scenario #1. Within the Scenario, each subject executed an action in 1000 different maze/goal position combinations, generating 1000 training patterns. Those patterns were used to train and evaluate the neural network. For the first set of tests, 90 separate *runs* are executed. A *run* consists of the following steps:

1. Randomize the order of the 1000 training patterns
2. Choose 900 patterns at random to train the neural network, use the final 100 patterns for testing
3. Choose values for $\bar{\rho_a}$ and $\bar{\rho}_{atest}$
4. Train the neural network using the 900 chosen training patterns
5. Test the neural network using the remaining 100 points
6. Record the number of correct predictions made by the neural network out of the 100 testing patterns. This percentage is considered the *predictive accuracy* of the network for that run.

The purpose of a run is to calculate a predictive accuracy for FAM given for a specific pattern presentation and vigilance parameter $\bar{\rho_a}$.

6.4.2.1 Selecting Values for $\bar{\rho_a}$ and $\bar{\rho}_{a_{test}}$

Several runs were executed in order to observe the behavior of the network against the baseline vigilance parameter $\bar{\rho_a}$. To do this, ten of the 90 runs were executed with $\bar{\rho_a}$ set at 0.1, ten at 0.2, and so on up to the final ten runs with $\bar{\rho_a}$ at 0.9. Within each set of five runs for each $\bar{\rho_a}$ value, five were run where $\bar{\rho_a}$ was re-initialized to 0 prior to testing. This reinitialization of $\bar{\rho_a}$ for testing forces the neural network to make a prediction no matter the input pattern. As discussed earlier, $\bar{\rho_a}$ determines how close patterns must match with a particular cluster to become a member. Because of this, if a pattern from the validation set matches a particular cluster but does not satisfy this vigilance parameter, the neural network will not return a prediction. The distribution of runs are illustrated in table 6.2.

6.4.2.2 Predictive Accuracies of FAM for Scenario #1

The predictive accuracy achieved by these runs for each test subject were then averaged across each run-type ($\bar{\rho_a}$ and $\bar{\rho}_{a_{test}}$ pairing). These averages are tabulated below as Table 6.3. The tabulated results for each individual run are provided in the appendix. Here, predictive accuracy results are listed as a number correct out of the 100 testing patterns used. In Table 6.4, the predictive accuracy of Fuzzy ARTMAP across all 90 runs are averaged for each subject.

Table 6.2: Distribution of 100 Testing Runs for each Subject, Scenario #1

$\bar{\rho_a}$	$\bar{\rho_a}(test)$	numRuns	numTrainingPoints	numTestingPoints
0.1	0.1	5	900	100
0.1	0	5	900	100
0.2	0.2	5	900	100
0.2	0	5	900	100
0.3	0.3	5	900	100
0.3	0	5	900	100
0.4	0.4	5	900	100
0.4	0	5	900	100
0.5	0.5	5	900	100
0.5	0	5	900	100
0.6	0.6	5	900	100
0.6	0	5	900	100
0.7	0.7	5	900	100
0.7	0	5	900	100
0.8	0.8	5	900	100
0.8	0	5	900	100
0.9	0.9	5	900	100
0.9	0	5	900	100

Table 6.3: Results for Scenario #1: Average Number Correct of 100 Testing Patterns

$\bar{\rho_a}$	$\bar{\rho_a}(test)$	Subject 1	Subject 2	Subject 3
0.1	0	93.2	84.2	77.4
0.1	0.1	95.2	81.6	75.8
0.2	0	96.6	85.8	78.4
0.2	0.2	93.8	83.2	79
0.3	0	93.6	86.2	77.8
0.3	0.3	95.6	81.8	75.8
0.4	0	94.4	82.8	78.6
0.4	0.4	94.8	83.8	78.4
0.5	0	94.2	86.2	76.4
0.5	0.5	93.2	84.4	78.4
0.6	0	**95.8**	84	77.2
0.6	0.6	94.4	86.4	76.4
0.7	0	**95.8**	87.2	78.6
0.7	0.7	92.4	86.8	79.8
0.8	0	94.8	90	**82.4**
0.8	0.8	92.2	86.8	81.2
0.9	0	94.8	**88.2**	80.4
0.9	0.9	92	85.8	81.8

Table 6.4: Summarized Results for Scenario #1: Average Number Correct out of 100 Testing Patterns over 90 Total Runs

	Number of Runs	Mean (μ)	Standard Deviation (σ)
Subject 1	100	94.2	2.33
Subject 2	100	85.3	4.02
Subject 3	100	78.5	4.87

6.4.2.3 Obtaining the Best-Case Predictive Accuracy of FAM for Scenario #1

For each subject, the $\bar{\rho_a}$ and $\bar{\rho}_{atest}$ values that yielded the best average result over their 5-run trials were used for a second set of 1000 runs. These values are highlighted in table 6.3

Across each of these 1000 runs, $\bar{\rho_a}$ and $\bar{\rho}_{atest}$ were fixed. Runs for this set proceeded in the following sequence (repeated from description above):

1. Randomize the order of the 1000 training points (note: these are the same training points used in the previous section's tests)
2. Choose 900 points at random to train the neural network, use the final 100 points as testing patterns
3. Train the neural network using the 900 chosen training points
4. Test the neural network using the remaining 100 points
5. Record the number of correct predictions made by the neural network out of 100 testing patterns

Table 6.6 displays the results of the 1000-run sets for each subject, including the sample mean predictive accuracy $\bar{\mu}$ and associated standard deviation $\bar{\sigma}$. A 2-tailed t-test was used on each set of data to validate that the computed sample mean $\bar{\mu}$ for each subject approaches the actual mean μ. Using an α value of 0.01, the test computed a 99% *confidence interval* for the actual mean.

Table 6.5: Summarized Results for Scenario #1: 1000 Runs for each subject, Using $\bar{\rho_a}$ and $\bar{\rho}_{atest}$ Values that Yielded Best Accuracy (see Table 6.3)

	Number of Runs	$\bar{\rho_a}$	$\bar{\rho}_{atest}$	$\bar{\mu}$	$\bar{\sigma}$	99%CI	p-value
Subject 1	1000	0.6	0	94.7	2.38	(94.5524,94.9416)	1.00
Subject 2	1000	0.8	0	87.3	3.27	(87.055,87.589)	1.00
Subject 3	1000	0.8	0	80.6	3.76	(80.336,80.950)	1.00

Figures 6.3, 6.4 and 6.5 illustrate the distribution of predictive accuracies reached across each subject's 1000-run set. In Figure 6.3, for instance, the bar graph shows a frequency of about 75 for the predictive accuracy 0.92 (92 correct out of 100 total). This means that the neural network correctly predicted 92 of 100 testing patterns in 75 of the 100 runs executed.

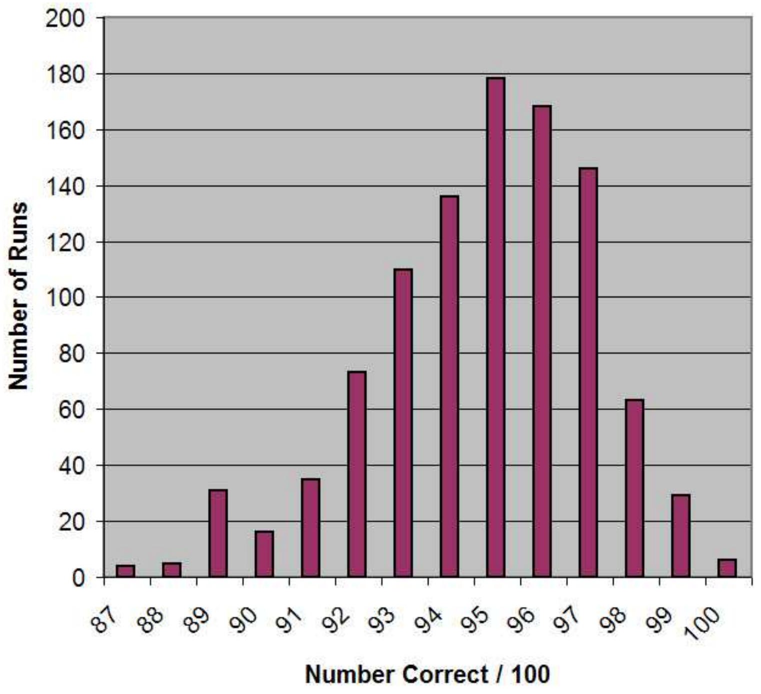


Figure 6.3: Scenario 1 Results: Frequency of Correct Predictions over 1000 Runs for Subject 1

The only parameter that differed across these runs was the order and set of pattern presentations during testing and training, and this order was based on a random variable. Because of this, the Central Limit Theorem [sta90] implies that these results for both the frequency of correct predictions generated will follow a normal distribution centered about the sample mean. The mound-shaped pattern of each set of data support this implication.

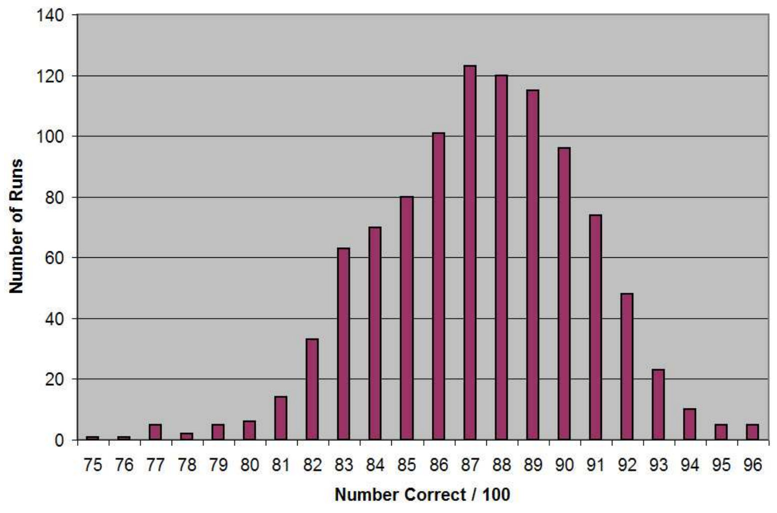


Figure 6.4: Scenario 1 Results: Frequency of Correct Predictions over 1000 Runs for Subject 2

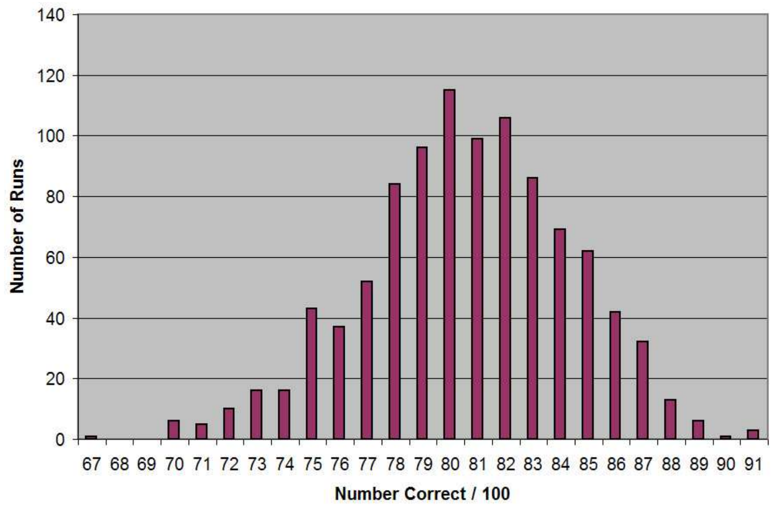


Figure 6.5: Scenario 1 Results: Frequency of Correct Predictions over 1000 Runs for Subject 3

6.4.3 Analysis of Scenario #1 Results

As expected, Fuzzy ARTMAP is able to successfully learn the movement patterns for each of the three subjects. Success, here, is defined as better than random. A random guess at the subject's action, for Vignette A would yield on average a 25% predictive accuracy (because there are four possible actions). As a qualitative comparison, consider the accuracies achieved by each subject. For subject #1, the network was able to predict, on average, almost 95 of the 100 testing patterns. Even for the worst-cased subject (#3), Fuzzy ARTMAP was able to predict nearly 81% of the testing patterns.

In order to show that Fuzzy ARTMAP achieves statistically different prediction accuracies for each of the subjects, a set of 2-sampled t-test is performed for each combination of subject-pairs. The results are tabulated below.

Table 6.6: Statistical Comparison of Predictive Accuracies for Scenario #1; $H_0 \rightarrow \mu_i - \mu_j = 0$, $H_1 \rightarrow \mu_i - \mu_j \neq 0$

	Sample Means	99%CI	Test Statistic	p-value
Subject 1 vs. Subject 2	94.7,87.3	$(7.095, 7.755)$	57.99	0
Subject 1 vs. Subject 3	94.7,80.6	$(13.741, 14.467)$	100.14	0
Subject 2 vs. Subject 3	87.3,80.6	$(6.272, 7.086)$	42.36	0

Here, the 2-tailed t test confirms that the predictive accuracy of Fuzzy ARTMAP is statistically different across each subject. Qualitatively, this indicates that the neural network is learning a different set of rules for choosing a direction from each subject, and is not just learning a simple heuristic for how to navigate a maze. Even the simple tactical task of navigating a maze, in fact is dependent on the subject and his distinct decision-making processes. Based on the results of the testing, it is clear that Fuzzy ARTMAP learns a *different* set of rules based on the subject observed, and is able to achieve high predictive accuracies for each.

By creating such a simple Vignette such as this one, the intent is to create an environment where the actions of the subject are closely tied to the primary goals of the behavior. In this Vignette, the subject makes only a single move in response to being told where and how far away the goal position is. Each low-level move, therefore, is made in direct accordance with the objective of reaching the goal. In the next few Vignettes, the behavior required will become increasingly complex, and the relationship between the low-level action required by the subject will become less dependent on the overall objective and more dependent on the context in which the subject is operating.

The intent is for these results to serve as one baseline for evaluating the performance of FAMTILE. With this baseline, we can in turn evaluate FAM (and ultimately FAMTILE) and examine how this notion of context affects their predictive accuracy.

6.5 Testing Scenario #2

This section defines the purpose, motivation and results for the second of four testing Scenarios used to evaluate the FAMTILE system.

In this Scenario, the subject is shown a $5x5$ subsection of the same $10x10$ maze used for Scenario #1, along with a vector indicating his distance and direction towards the goal (see Figure 5.4). In this Scenario, however, the subject is able to make a sequence of moves to advance towards the goal position. When the subject makes a move, his position within the maze and goal direction vector is updated to reflect that move, and his viewable $5x5$ area of the maze is updated. Since the character's icon remains at the center of this area, each move gives the subject new viewable sections of the maze. This Scenario also displays spaces in the maze that have already been visited by the subject during the current run. The subject will move in a sequence of directions that navigate it towards the goal, with one input/output observation being collected for each move. When the subject reaches the goal, his position and the goal's position are reset, and the subject again navigates the maze. This cycle continues until the simulation collects the requisite number of observations.

6.5.1 Scenario #2 Motivation

Like Vignette B, Vignette A is set in a maze environment where the only four actions can be made - move up, move down, move left, or move right. However, there are differences in this Vignette that make it significantly more complex than the first. The most important difference is that the subject is not re-placed at a random position on the maze after every action. Instead, he/she makes a sequence of moves towards the goal while the simulation records and displays the positions the subject has already visited while on that path. In addition, the viewable area of the maze is increased by nearly 200% (from 9 squares to 25).

The intent of these changes is to obscure the relationship between the action of the subject and the objective of reaching the goal. While the objective has not changed, the reasons behind making one move over another can now include more intermediate information. For instance, a player could be making a move based on the wall states further away from him, or because he wants to avoid revisiting a space that he has already been to. In this case, there are motivations beneath the objective of reaching the goal - motivations that could possibly be considered to be contexts.

Nevertheless, the potential for context inference within Vignette B is intentionally omitted for this test Scenario. Instead, the testing is conducted just as it was for Scenario #1. The objective, then, is to observe how this increased complexity affects the predictive accuracy of FAM.

6.5.2 Scenario #2 Results

As discussed above, the tests for Scenario #2 were executed in the same manner as Scenario #1, and the same three subjects were used. Within the Scenario, each subject makes consecutive moves within a 10x10 maze, with the board and goal positions resetting each time the subject reaches the goal. The Scenario ends when the subject has generated 1000 training points - each training point represents a specific maze state and the action the subject makes in response to that state. Those points were used to train and evaluate the neural network. For the first set of tests, 90 separate *runs* are executed. As in Scenario #1, a *run* consists of the following steps:

1. Randomize the order of the 1000 training points
2. Choose 900 points at random to train the neural network, use the final 100 points as testing patterns
3. Choose values for $\bar{\rho_a}$ and $\bar{\rho}_{atest}$
4. Train the neural network using the 900 chosen training points
5. Test the neural network using the remaining 100 points
6. Record the number of correct predictions made by the neural network out of 100 testing patterns. This percentage is considered the *predictive accuracy* of the network for the run.

6.5.2.1 Predictive Accuracies of FAM for Scenario #2

The runs were again partitioned to observe the behavior of the network against the baseline vigilance parameter $\bar{\rho_a}$. To do this, ten of the 90 runs were executed with $\bar{\rho_a}$ set at 0.1, ten at 0.2, and so on up to the final ten runs with $\bar{\rho_a}$ at 0.9. Within each set of 10 runs for each $\bar{\rho_a}$ value, 5 were run where $\bar{\rho_a}$ was re-initialized to 0 prior to testing.

The predictive accuracy reached by these runs for each subject were then averaged across each run-type ($\bar{\rho_a}$ and $\bar{\rho}_{atest}$ pairing). These averages are tabulated below as Table 6.7. The tabulated results for each individual run are provided in the appendix. Once again, the predictive accuracy results are listed as a number correct out of the 100 testing patterns used.

Table 6.7: Results for Scenario #2: Average Number Correct of 90 testing patterns

$\bar{\rho_a}$	$\bar{\rho_a}(test)$	Subject 1	Subject 2	Subject 3
0.1	0	90.8	85.2	82.2
0.1	0.1	85.6	84.4	82.2
0.2	0	91.0	81.6	81.0
0.2	0.2	88.0	81.8	84.6
0.3	0	89.2	82.0	82.0
0.3	0.3	90.8	82.0	82.0
0.4	0	90.6	80.4	84.8
0.4	0.4	91.2	82.4	86.4
0.5	0	**93.6**	84.8	83.4
0.5	0.5	89.2	84.6	86.6
0.6	0	91.0	85.2	84.4
0.6	0.6	91.4	79.4	84.6
0.7	0	93.2	82.4	86.6
0.7	0.7	91.6	81.8	**88.4**
0.8	0	**93.6**	**85.8**	85.6
0.8	0.8	91.0	84.6	85.2
0.9	0	93.4	82.2	87.8
0.9	0.9	91.4	81.8	80.2

Table 6.8: Summarized Results for Scenario #2: Average Number Correct out of 100 testing patterns, over 100 Total Runs

	Number of Runs	Mean (μ)	Standard Deviation (σ)
Subject 1	100	90.9	3.59
Subject 2	100	82.9	3.84
Subject 3	100	84.3	4.42

In Table 6.8, the predictive accuracy of Fuzzy ARTMAP across all 100 runs are averaged for each subject.

6.5.2.2 Obtaining the Best-Case Predictive Accuracy of FAM for Scenario #2

For each subject, the $\bar{\rho_a}$ and $\bar{\rho}_{atest}$ values that yielded the best average result over their 5-run trials were used for a second set of 1000 runs, as in Scenario #1. Across each of these 1000 runs, $\bar{\rho_a}$ and $\bar{\rho}_{atest}$ were fixed.

1. Randomize the order of the 1000 training points
2. Choose 900 points at random to train the neural network, use the final 100 points as testing patterns
3. Train the neural network using the 900 chosen training points
4. Test the neural network using the remaining 100 points
5. Record the number of correct predictions made by the neural network out of 100 testing patterns

Table 6.9: Summarized Results for Scenario #2: 1000 Runs for each subject, Using $\bar{\rho_a}$ Values that Yielded Best Accuracy (see Table 6.7)

	Number of Runs	$\bar{\rho_a}$	$\bar{\rho}_{atest}$	$\bar{\mu}$	$\bar{\sigma}$	99%CI	p-value
Subject 1	1000	0.8	0	92.5	2.63	(92.3074,92.7366)	1.00
Subject 2	1000	0.8	0	84.5	3.42	(84.181,84.739)	1.00
Subject 3	1000	0.7	0	85.6	3.31	(85.308,85.848)	1.00

Table 6.10 displays the results of the 1000-run sets for each subject, including the sample mean predictive accuracy $\bar{\mu}$ and associated standard deviation $\bar{\sigma}$. A 2-tailed t-test was used on each set of data to validate that the computed sample mean $\bar{\mu}$ for each subject approaches the actual mean μ. Using an α value of 0.01, the test computed a 99% *confidence interval* for the actual mean.

The following Figures 6.6, 6.7 and 6.8 illustrate the distribution of predictive accuracies reached across each 1000-run set.

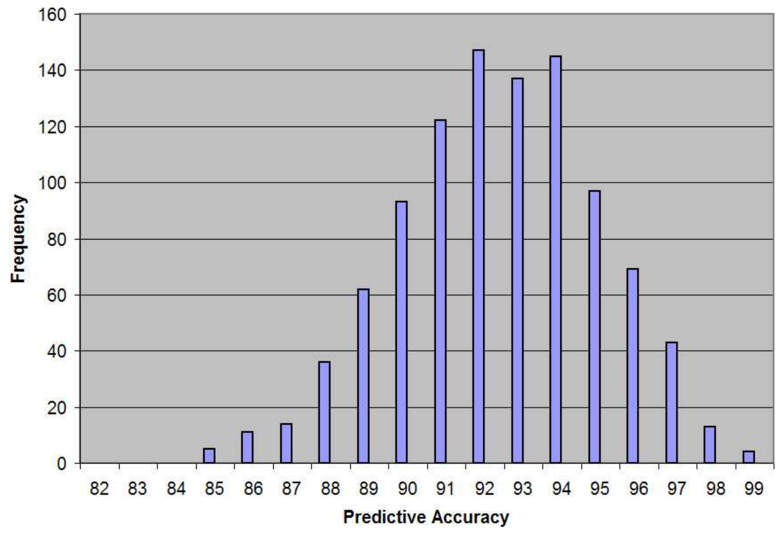


Figure 6.6: Scenario 2 Results: Frequency of Correct Predictions over 1000 Runs for Subject 1

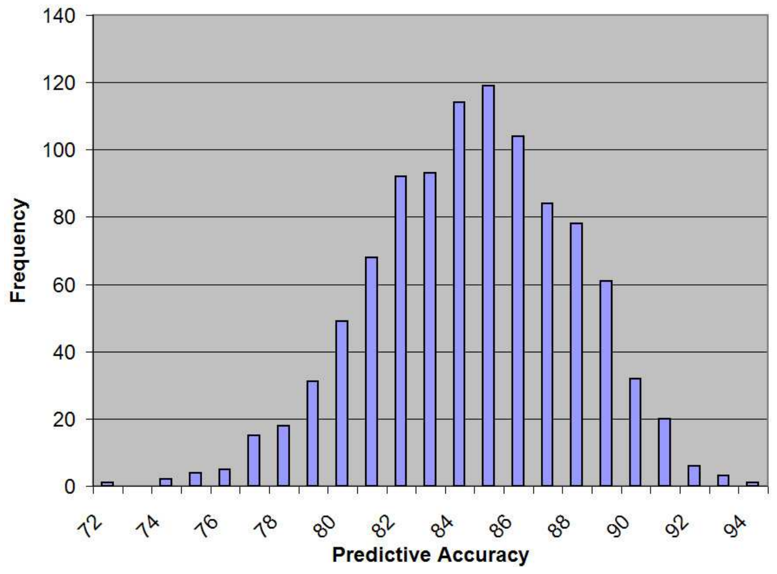


Figure 6.7: Scenario 2 Results: Frequency of Correct Predictions over 1000 Runs for Subject 2

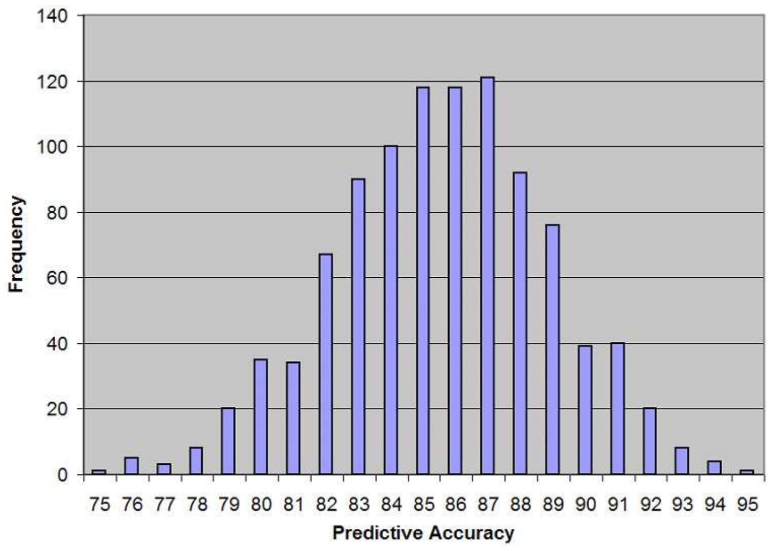


Figure 6.8: Scenario 2 Results: Frequency of Correct Predictions over 1000 Runs for Subject 3

6.5.3 Analysis of Scenario #2 Results

In this Scenario, Fuzzy ARTMAP is able to adequately learn the movement patterns for each of the three subjects. Furthermore, the predictive accuracy varied significantly across subjects, just as it had in Scenario #1. FAM achieved a predictive accuracy of nearly 93 of 100 for subject #1 versus 84.5 and 85.6 for the other two. We again show that these sample accuracy means are statistically significant using 2-sampled t-tests. The results of these tests are are tabulated below. The 2-tailed t test again confirms that the predictive accuracy of Fuzzy ARTMAP is statistically different across each subject, just as in Scenario #1. This means that the network is again learning different rules for navigating the maze from subject to subject, and is not just learning generic rules.

Table 6.10: Statistical Comparison of Predictive Accuracies for Scenario #2; $H_0 \rightarrow \mu_i - \mu_j = 0$, $H_1 \rightarrow \mu_i - \mu_j \neq 0$

	Sample Means	99%CI	Test Statistic	p-value
Subject 1 vs. Subject 2	92.5,84.5	$(7.710, 8.414)$	59.05	0.00
Subject 1 vs. Subject 3	92.5,85.6	$(6.599, 7.289)$	51.96	0.00
Subject 3 vs. Subject 2	85.6,84.5	$(0.730, 1.516)$	7.43	0.00

Because the three subjects for Vignettes A and B were the same, a similar analysis was performed on the best-case predictive accuracies achieved for both test Scenarios. In these computations, a 2-sampled t-test was performed the best-case predictive accuracy values for each subject, as computed in sections 6.4.2.3 and 6.9.2.2. The results of these tests are summarized in Table 6.11.

Results of this comparison were mixed. For subjects #1 and #2, FAM was more accurate in predicting their actions within Vignette A (94.7 to 92.5 for sub-

Table 6.11: Statistical Comparison of Predictive Accuracies for Scenario #1 versus Scenario #2; $H_0 \rightarrow \mu_i - \mu_j = 0$, $H_1 \rightarrow \mu_i - \mu_j \neq 0$

	Sample Means	99%CI	Test Statistic	p-value
Subject 1	94.75,92.52	$(2.005, 2.445)$	19.82	0.00
Subject 2	87.32,84.46	$(2.568, 3.156)$	19.11	0.00
Subject 3	80.64,85.58	$(-5.197, -4.670)$	-36.75	0.00

ject #1, 87.3 to 85.6 for subject #2). For subject #3, however, the performance of FAM actually *improved* by nearly 4 percentage points (80.6 to 84.5). This is in spite of the fact that input patterns for Vignette B were over three times the size of those used for Vignette A - primarily due to the increased viewable area. For Scenario #1, 27 bits are required to represent each input pattern. This is compared to **96** bits required to represent an input pattern for Scenario #2.

From these results, there are several possible conclusions that can be drawn from the similarity in predictive accuracy of FAM in Scenarios #1 and #2. Perhaps the subjects decisions were not significantly affected by the increase in Scenario complexity or the extra information might have made the subject's decision more clear than when he/she was able to see only one square in each direction. It is also possible that the subjects' decision making for Vignette B was somehow affected after first executing Vignette A. This is quite likely, as each of the three subjects were used for both Vignettes A and B.

Regardless of which item was the primary cause of these curious results, the logical next step was to evaluate the system in an environment involving significantly more complex decisions, as in Vignettes C and D. Again returning to the main goals of the research, the goal is to analyze and learn subject behaviors as a sequence of lower-level *behaviors*, not necessarily lower-level actions. If a Vignette is put into play that significantly affects the predictive accuracy of FAM, that in turn provides a good backdrop for the introduction of a system that instead learns context transitions rather than just action sequences. We can then use the high predictive accuracies generated by FAM in Scenarios #1 and #2 to provide a ceiling of performance for both systems in learning behavior in the more complex Vignettes.

As described exhaustively in the previous Chapter and in Appendix C, the game of Poker involves decision-making at several different levels, and the choice of action is heavily dependent on the style and skill of the player. Because of this, it is hypothesized that FAM will perform significantly worse in predicting actions in these Vignettes than it did for this Scenario.

6.6 Testing Scenario #3

This section defines the purpose, motivation and results for the third of four testing Scenarios.

In Vignette C, each test subject is placed at a simulated Texas Hold'em game with seven computer-generated opponents. Each of these opponents were encoded with a basic knowledge of how to play the game. The details of how these opponents were modeled is provided in section 5.2.2.3. For each observation, a player is placed at a random position at the table and dealt his two hole cards. In turn, each opponent who acts before the player makes an action. Each observation recorded by the simulation records the player's hole cards along with his position at the table and the actions of the computer-generated opponents who act before him. The subject is asked to look at his hole cards, examine the actions of the players before him, and either raise, call, or fold. This decision is recorded as the subject's action for that decision point.

In this section we evaluate the ability of FAM to predict subject's action decisions within Vignette C.

6.6.1 Scenario #3 Motivation

The strategies and tactics in a Poker game are far more complex than the maze navigation used for Vignettes A and B. Whereas the goal when traversing a maze is simply 'to reach the goal', the goal in a Poker game is far less concrete; as players are not trying to win every hand but rather to accumulate chips over a period of time. To do this effectively, the appropriate strategy is to make the right moves at the right time. Many times, the correct play in Texas Hold'em is simply to fold your cards and commit no chips to the pot. While this action will never wins chips for the player, it is still correct within the context of the overall goal - it keeps the player from *losing* chips for reasons he/she could have avoided.

The same is true in the opposite case. Consider a situation where the player is in the big blind with $9\heartsuit, 10\heartsuit$ and five players (including the small blind) call the blinds. The flop now comes $7\heartsuit, 8\heartsuit, 9\heartsuit$, and the player now has a straight flush that is guaranteed to be the best hand (known as the *stone cold nuts*). The small blind, however, immediately bets at this hand. The first reaction of the player may be to raise - after all, he does have the best hand - however this action would not play to the overall goal of winning chips. By making a raise, the three players yet to act will now have to call two bets to stay in the pot. However, if the player simply calls, the bet will only be one to call. By making the bet more enticing to call, the player keeps more people in the hand - and the more people who are in the hand, the more chips will be put into the pot. Since the player's hand cannot be beat, there is no utility in raising when it will likely drive opponents out.

The actions made during a poker game involve thousands of situations such as these where decisions are made based on concepts such as these and others discussed in Appendix C. Furthermore, these decisions are heavily influenced by the style and personality of the subject making the decisions, making this a far more conceptually complex behavior than that of a maze. There is no one specific manual that outlines how to play subject poker as there are military field manuals that describe how to engage in an assault, fly a fixed-wing aircraft [JK99] or negotiate a turn during a road march [Hen01] [Ger01]. Because of this, it is possible to observe several subject poker players playing and get several different notions of the tactics used to win. From the perspective of a learning system, then, it is possible to construct a set of Poker models that differ greatly in their knowledge because of the differing styles of the subjects used to generate the knowledge.

The purpose of this testing Scenario is to observe how well FAM performs when learning behavior in this more complex Vignette. As reported in the previous two Scenarios, the neural network was able to achieve predictive accuracy ranges from 80 to 95% for Vignettes A and B depending on the test subject. Vignette B was an attempt to create a slightly more complicated learning task for FAM, however the results did not seem to be affected.

Vignette C, however, represents a far more significant increase in complexity affecting the mapping between situation and action. Because of this, we expect that this would result in a significant performance degradation in predictive accuracy versus those of the first two Scenarios.

6.6.2 Scenario #3 Results

As in all of the test Scenarios, the learning system is tested with the data from three separate subjects. FAM learns a training set for each subject and then uses the trained network to correctly predict each subject action recorded in a separate validation set.

6.6.2.1 Predictive Accuracies of FAM Across Values of $\bar{\rho_a}$

A series of runs were executed to determine the value for $\bar{\rho_a}$ where FAM achieves the best predictive accuracy for each subject, as done in Scenarios #1 and #2. These runs are performed for each each subject, in turn, using 300, 600, and 900 training patterns. This is done to gain perspective as to how the performance of improves with more training patterns.

For each run, 100 patterns were used for testing. The results in tables 6.12 represents averages over 100 runs for both algorithms and for each particular subject, number of training points and $\bar{\rho_a}$ value. Among these averages, the $\bar{\rho_a}$ value that results in the best predictive accuracies was used for the direct comparison texts between FAMTILE and FAM. Unlike Scenarios #1 and #2, $\bar{\rho}_{atest}$ was set to 0 for each of the runs reported below. We observed in the first two Scenarios that setting the value for $\bar{\rho}_{atest}$ for a testing run did not result in any positive influence on FAM's predictive accuracy. Because of this, it was deemed unnecessary to keep $\bar{\rho}_{atest}$ as an independent variable for this or any subsequent tests.

Table 6.12: Average Predictive Accuracies of Subject *Actions* for 100-run sets of FAM, 300 Training Points

$\bar{\rho}_a$	Subject 1	Subject 2	Subject 3
0.05	66.98	59.91	66.67
0.10	66.08	60.68	65.78
0.15	66.62	60.64	67.39
0.20	66.16	60.06	66.74
0.25	65.97	60.25	66.47
0.30	66.48	60.90	66.25
0.35	67.22	60.64	66.25
0.40	67.97	59.40	65.69
0.45	66.89	60.48	66.16
0.50	66.58	60.76	66.74
0.55	65.95	59.47	66.22
0.60	67.50	60.14	65.86
0.65	67.22	60.20	66.91
0.70	67.67	60.25	68.24
0.75	68.58	61.38	69.12
0.80	70.78	61.77	69.75
0.85	71.46	64.13	71.36
0.90	**74.00**	66.28	**72.40**
0.95	73.47	**66.64**	72.11

Table 6.13: Average Predictive Accuracies of Subject *Actions* for 100-run sets of FAM, 600 Training Points

$\bar{\rho_a}$	Subject 1	Subject 2	Subject 3
0.05	67.93	61.34	67.51
0.10	69.03	61.08	67.65
0.15	67.79	60.03	67.47
0.20	67.80	61.90	67.95
0.25	68.16	62.06	66.92
0.30	67.69	61.37	67.69
0.35	68.38	61.30	66.40
0.40	68.38	61.89	66.87
0.45	67.92	61.96	68.07
0.50	68.21	60.78	67.20
0.55	69.11	61.92	67.04
0.60	68.39	60.71	67.81
0.65	68.39	62.36	67.28
0.70	69.10	61.10	68.28
0.75	70.63	61.57	69.53
0.80	70.69	62.88	70.35
0.85	72.05	64.80	72.58
0.90	74.94	66.29	74.24
0.95	**75.62**	**67.18**	**74.30**

Table 6.14: Average Predictive Accuracies of Subject *Actions* for 100-run sets of FAM, 900 Training Points

$\bar{\rho_a}$	Subject 1	Subject 2	Subject 3
0.05	69.59	61.65	67.52
0.10	69.67	63.12	68.68
0.15	68.79	62.51	68.01
0.20	70.35	62.06	67.71
0.25	69.12	62.69	68.19
0.30	69.32	63.09	69.45
0.35	69.99	62.47	68.01
0.40	69.54	62.45	68.69
0.45	69.26	62.49	68.00
0.50	70.45	61.80	68.22
0.55	69.30	62.68	68.57
0.60	69.02	61.87	67.71
0.65	69.58	62.26	67.71
0.70	70.37	62.72	68.96
0.75	71.20	62.37	69.99
0.80	72.10	63.72	72.07
0.85	72.56	65.44	72.51
0.90	74.53	67.52	74.79
0.95	**75.23**	**68.74**	**74.95**

6.6.2.2 Obtaining the Best-Case Predictive Accuracy of FAM for Scenario #3

The results are first tabulated across the baseline vigilance parameter, $\bar{\rho}_a$, in order to ascertain an acceptable value for the comparison test. For each subject, a value for $\bar{\rho}_a$ is chosen that maximizes the average predictive accuracy of FAM over a set of random validation sets. That value is then set as a constant parameter, and the system is then re-tested in three batches of 1000-run tests: one using 300 training points, one using 600, and the last using 900 of the 1000 points generated. The best values for $\bar{\rho}_a$ are extracted from the above tables and summarized in table 6.15.

Table 6.15: $\bar{\rho}_a$ Values Yielding Best Average Predictive Accuracies for Scenario #3

Subject	300	600	900
1	0.9	0.95	0.95
2	0.95	0.95	0.95
3	0.9	0.95	0.95

The purpose of these batches of runs is to obtain best-case predictive accuracies of FAM. The results are tabulated in full in Appendix F, while the mean values of each 1000-run test are tabulated in Tables 6.37 - 6.39.

Table 6.16: Average Predictive Accuracy for 1000-run Sets for Scenario #3 Using Optimal Values for $\bar{\rho_a}$

Subject	300	600	900
1	72.99	74.94	75.04
2	66.01	67.55	68.54
3	71.94	73.95	75.56

6.6.3 Analysis of Scenario #3 Results

As expected, the predictive accuracy of FAM degraded significantly when tested using Vignette C. By the numbers, the network achieved best case predictive accuracies of 75.0, 68.5, and 75.6 for each player respectively, versus 92.5, 84.5 and 85.6 for Scenario #2.

Subject #1 from Scenario #2 did not participate as an subject for Vignette C. Subjects #2 and #3 did, however, and are also represented as subject #2 and #3 for this testing Scenario. Comparing the predictive accuracies of FAM on these two subjects for Scenarios #2 and #3, there is an 18.9% decrease for subject #2 and an 11.7% decrease for subject #3. This is a sharp contrast to the statistically insignificant performance difference between Scenarios #2 and #1, where the network's predictive accuracy changed 1.95% and 4.84% for subjects #2 and #3.

These results confirm that the poker environment of Vignette C is much harder for FAM to learn human tactical behavior versus the simpler Vignettes in a maze. What this means in terms of the network itself is that FAM had a more difficult time effectively creating clusters with similar data points that mapped to the output patterns representing correct predictions of the subject's action. In Scenario #5, Vignette C is re-used and FAMTILE attempts to learn subject actions just as FAM attempted in Scenario #3. It is hypothesized that the reintroduction of the subjects' actions as inferred contexts will help the network more effectively make finer clusters representing more closely related patterns, thereby increasing the predictive accuracy of the system.

An interesting result of this test was the sharp contrast in the predictive accuracy of FAM for subject #2 versus the other two subjects. As reported above, FAM was only able to predict 68.54% of subject #2's actions versus 75.04 and 75.56% for the other two subjects. Since the results on subject #2 were not this deviant in the other two Scenarios, it seems as though the problem is related to the subject's behavior within this particular Vignette C.

To gain more perspective on this issue, a questionnaire was sent out to several amateur poker players including each of the three who participated as subjects for Scenario #3. This questionnaire is duplicated below.

1. `How long have you been playing Poker?`
2. `How long have you been playing Poker seriously/professionally?`
3. `How often do you play poker?`
4. `Do you consider yourself to be a tight or loose player? Explain.`
5. `Do you consider yourself to be a passive or aggressive player? Explain.`
6. `Do you read literature on the game of Poker? If so, how does it affect the strategies you employ during a game?`

7. List all conditions where you would consider a *check* to be a legitimate action. Give card-by-card examples for each.

8. List all conditions where you would consider a *bet* to be a legitimate action. Give card-by-card examples for each.

9. List all conditions where you would consider a *call* to be a legitimate action. Give card-by-card examples for each.

10. List all conditions where you would consider a *raise* to be a legitimate action. Give card-by-card examples for each.

11. List all conditions where you would consider a *fold* to be a legitimate action. Give card-by-card examples for each.

12. In the cases where multiple actions are appropriate, what other factors do you consider?

The purpose of this questionnaire was to identify the skill level of each subject tested. As it turns out, the questionnaire found that subject #2 had significantly less experience playing Limit Hold'em than subjects #1 and #3, and also had not read any literature on theory or strategy in the game. This lack of experience likely affected the continuity and predictability of her play, whereas the other two subjects tended to play a consistent style based on clear-cut observations such as hand-strength, position, etc.

In Texas Hold'em, proper play before the flop is both the easiest piece of strategy to learn and the most crucial ([SM03]). Strategy after this round becomes much more complex because of the explosion of information present with community cards on the board. Because of this, Limit Hold'em play before the flop round of betting tends to be somewhat mechanical among experienced players. This is supported by the data on subjects #1 and #3, who shared similar experiences and have read much of the same literature. Subject #2, on the other hand, has much less experience, and therefore her play is likely to be more erratic and therefore less predictable.

This finding is somewhat of a confirmation of a conclusion drawn from Scenarios #1 and #2, that FAM is not merely predicting basic rules for behavior. Instead, the neural network learns specific tendencies of the player, and its performance in doing so is affected by how consistently he/she plays.

Another interesting piece of information gathered from Scenario #3 was the relative invariability in predictive accuracy given increased numbers of training data. This trait was present in each of the three subjects, where FAM averaged only a 2.64% increase in predictive accuracy using 900 patterns for training versus 300 patterns. This modest increase is an indication that each subject provided a sufficient amount of training data to train FAM. It is also possible, though unlikely, that the system could benefit from increased numbers of training points for this Vignette, and that this invariability between 300 and 900 training points represents a temporary plateau in the learning capacity of FAM. Investigating this possibility is left for future research.

In the following test Scenario, a Vignette D is examined. This Vignette represents the most complex behavior with which our learning systems are evaluated. It is therefore expected to produce another significant decrease in predictive accuracy for FAM similar to the one produced by the jump from Vignette C to Vignette D.

6.7 Testing Scenario #4

In Scenario #4, predictive accuracies for FAM are collected an analyzed for the last and most complex Vignette, Vignette *D*. Just as Vignette *C*, this Vignette is set at the poker table with 7 computer-generated agents playing against the subject in games of Texas Hold'em. Here, however, the subject's decision points are not limited to the first round of action. Instead, a series of entire hands are carried out to their completion - if an subject folds, a new hand is dealt; if an subject raises, the opponents react to that raise accordingly; a flop, turn, and river are dealt and betting rounds follow just as in an actual hand. The subject is also given a stack of 100 'chips' that is maintained throughout the Vignette.

6.7.1 Scenario #4 Motivation

Just as Vignette *C* represented a significant jump in complexity from the two maze Scenarios, the complexity in subject behavior throughout an entire hand of poker is significantly more complex than that observed in the initial betting round.

While Scenario #3 showed a decrease in FAM performance, the system still performed relatively well considering the added complexity of the behavior. For the best values of $\bar{\rho_a}$, FAM was able to predict over 75% of the actions of the two experienced subjects and close to 70% of the actions of the less experienced subject #2.

While Vignette *C* does represent a complex behavior, the number of situations that can exist for a player to reason about in the first round of betting is far fewer than those that are possible after community cards are dealt. There are less than 200 combinations of two ranked cards, and the only other significant variables are the player's position and the initial actions of the 7 (or fewer) players acting before him. Throughout a round of poker, as previously discussed, there are several other variables - including betting history in previous rounds, making actions to set up future actions, not to mention to fact that there are 3-5 more cards visible for the player to consider. Because of this explosion of possible tactics, literature about Hold'em strategy in post-flop action is far less robust and specific.

It is this Vignette that best represents the concept of high-level tactical behavior discussed in Chapters 1 and 3. Repeated from that discussion, a tactical behavior was defined as:

- A well-defined goal or *mission*
- Are characterized by planning and/or maneuvering
- Are not well-defined as to their execution sequence, and thus their characteristics may vary greatly across individuals

Poker behavior meets each of these requirements, albeit in a turn-based game. The goal of poker is to accumulate chips over a long sequence, however the execution path for achieving that goal is dependent on a large number of variables, the most of which being the style and skill of the subject playing. While it is a game of individual actions, planning and maneuvering are essential, specifically during individual rounds. Actions are made in anticipation of future actions and betting rounds, for example, and decisions are heavily influenced by the style and skill of one's opponents.

6.7.2 Scenario #4 Results

In this fourth and final evaluation of FAM, we continue to examine its ability to learn subject actions as a function of his cards, his position at the table, and the betting action at the table. In Scenario #7, we will perform a similar evaluation of FAMTILE. However, FAMTILE will instead learn the inferred *context* of the subject during training, and then translate predicted contexts to actions during validation. The results of this Scenario will lead us into an analysis of the utility of using FAMTILE versus FAM for learning tactical behaviors similar to Vignette D.

6.7.2.1 Predictive Accuracies of FAM Across Values of $\bar{\rho_a}$

A series of runs were executed to determine the value for $\bar{\rho_a}$ where FAM achieves the best predictive accuracy for each subject, as done in the previous three Scenarios. These runs are performed for each subject using all but 100 of the total number of patterns generated for training. The remaining 100 patterns are used for the validation set, from which a predictive accuracy can be generated.

For this Vignette, subjects were asked to play a fixed number of hands, and therefore the number of actual patterns generated by the subject depended on the actions they made during those hands. For instance, a single hand could generate only one training pattern if the subject immediately folds, or several if he/she stays in the hand for future rounds of betting. Subject #1 generated 2009 patterns, while subjects #2 and #3 generated 1731 and 1735 patterns, respectively.

Table 6.17: Average Predictive Accuracies of Subject *Actions* for FAM

$\bar{\rho_a}$	Subject 1	Subject 2	Subject 3
0.05	55.7	59.3	52.7
0.1	56.4	56.7	**54.9**
0.15	55.8	57.5	53.2
0.2	53.4	57.3	50.7
0.25	56.3	**60.9**	53.8
0.3	57.6	58.6	50.9
0.35	54.6	56.8	53.1
0.4	56.2	57.6	52.3
0.45	52.1	57.6	49.8
0.5	54.3	57.9	51
0.55	55.9	59.6	51.8
0.6	55.5	58.1	52.6
0.65	56.9	58.7	53.8
0.7	54	57.1	51.8
0.75	55.3	55.6	51.2
0.8	54.8	58.6	53.3
0.85	57.7	57.1	53.8
0.9	57.4	56.6	52
0.95	**58.3**	57.7	51.2

6.7.2.2 Obtaining the Best-Case Predictive Accuracy of FAM for Scenario #4

For each subject, the $\bar{\rho_a}$ the trials summarized in Table 6.17 were used for a second set of 1000 runs. Across each of these 1000 runs, $\bar{\rho_a}$ was fixed at the value that previously produced the greatest average predictive accuracy for that subject.

Table 6.18: Average Predictive Accuracy for 1000-run Sets

Subject	$\bar{\rho_a}$	Predictive Accuracy
1	0.95	58.22
2	0.25	60.18
3	0.10	55.32

6.7.3 Analysis of Scenario #4 Results

Once again, the increase in complexity of Vignette D compared to Vignette C resulted in further erosion in FAM's predictive accuracy. Best-case accuracies of 55.32, 58.95 and 58.12 are an average of over 20% worse than those of Scenario #3 - which is nearly twice the decrease observed between Vignette C and the maze Scenarios.

Subjects #1 and #2 were the same two people for Vignettes C and D. Keep in mind that in Scenario #3, the network performed significantly worse on subject #2 than on the other two subjects. Furthermore, it was noted that subject #2 had several years less experience than the other two, which likely affected the predictability and consistency of her actions.

The complexity of this Scenario, however, seems to have neutralized this effect. In fact, according to the summarized results from Table 6.18, FAM was slightly more effective in the best-case at predicting her actions than for the other two subjects. As it turns out, subject #3 (who did not participate in Vignette *C* or the maze Vignettes) had comparable experience to subject #1. His responses to the poker questionnaire are also included in the appendix.

6.8 Testing Scenario #5

The objective for testing Scenario #5 is to evaluate FAMTILE's ability to predict both the subject's inferred active context and his resultant action. Vignette *C* is used for this testing Scenario, the same one used to evaluate FAM in testing Scenario #3. Because of this, the results of Scenario #3 will serve as a baseline performance metric for the results achieved here. Unlike FAM, however, FAMTILE instead attempts to predict the subject's inferred active context. In order to make a comparison between Fuzzy ARTMAP and FAMTILE, therefore, the predicted contexts of FAMTILE must then be converted to a predicted action for the subject. For example, if FAMTILE chooses the *callToTrap* context for some decision point, it can be determined that the predicted action is a call. Since FAM does not make context predictions, this determination is necessary to compare the predictive accuracies of the two learning systems.

6.8.1 Context Templates Used For Vignette C

Based on the reviewed literature and personal experience, 12 contexts were generated for Vignette C that will be used by FAMTILE. These contexts are a potential justification for each possible action.

As discussed in Chapter 5, there are many motivations and situations where one particular action may be desirable. A raise, for instance, does not always imply that the player believes he has the best hand. It could also be because the player wishes to make a bluff or semi-bluff. This is also the case with a fold or a call - a player may fold an otherwise playable hand because of a raise by an opponent or because the player is in 'weak' position. Likewise, a player can call a mediocre or weak hand because he is on the small blind and must only call 1/2 a bet to stay in the hand, or for several other reasons.

The context templates developed for Vignette C serve to partition these reasons for each of the three possible actions (raise, call, fold). Four templates are defined for each action, and contain a subset of the following attributes:

1. playerAction (what the player did - fold, call, or raise)
2. holeCardRanking (the rank of the player's hole cards as provided in section 5.2.2.2)
3. distanceToDealerButton (number of seats the player is away from the dealer button, going counter-clockwise)
4. numBetsToCall (the amount it would take to call the current bet. Prior to the flop, this amount will be 1 unless there has been a raise.
5. numPlayersInPot (the number of players who have yet to fold their hand and are still eligible to win the pot)

Listed below are the 12 templates used in Vignette C.

foldWithWeakHandContext Player folds because his hole cards are not strong (e.g. $6\spadesuit, J\spadesuit$)

Attribute	**type**	**weight**
$playerAction = fold$	boolean	6
$holeCardRanking = 9$	how close to	4

Table 6.19: Context Template for *foldWithWeakHandContext*

foldToStrongBettingContext Player folds an otherwise playable hand (e.g. $6\heartsuit, 6\spadesuit$) because one or more of his opponents have raised

Attribute	**type**	**weight**
$playerAction = fold$	boolean	6
$distanceFromDealerButton = 3$	how close to	0.5
$holeCardRanking = 5$	how close to	0.5
$numBetsToCall > 0$	amount greater than	3

Table 6.20: Context Template for *foldToStrongBettingContext*

foldInWeakPositionContext Player folds an otherwise playable hand (e.g. $K\heartsuit, Q\diamondsuit$) because he is in weak position in relation to the dealer

Attribute	**type**	**weight**
$playerAction = fold$	boolean	6
$distanceFromDealerButton = 3$	how close to	2
$holeCardRanking = 3$	how close to	2

Table 6.21: Context Template for *foldToInWeakPositionContext*

foldWithStrongHandContext Player folds a strong hand (e.g. $J\clubsuit$, $J\diamondsuit$)for no discernable reason

Attribute	**type**	**weight**
$playerAction = fold$	boolean	6
$holeCardRanking = 1$	how close to	4

Table 6.22: Context Template for *foldWithStrongHandContext*

callWithDrawingHandContext Player calls a good *multiway* hand in order to see a flop. Good multiway hole cards include combinations such as small pairs (e.g. $4\clubsuit, 4\heartsuit$), *suited connectors* (e.g. $8\spadesuit, 9\spadesuit$) and suited aces (e.g. $A\heartsuit, 8\heartsuit$)

Attribute	**type**	**weight**
$playerAction = call$	boolean	6
$numPlayersInPot = 7$	how close to	3
$holeCardRanking = 5$	how close to	1

Table 6.23: Context Template for *callWithDrawingHandContext*

callToTrapContext Player calls with a strong hand (e.g. $A\heartsuit, A\spadesuit$) either in or out of position attempting to induce action in later rounds

Attribute	**type**	**weight**
$playerAction = call$	boolean	6
$numBetsToCall = 1$	how close to	1
$holeCardRanking = 1$	how close to	3

Table 6.24: Context Template for *callToTrapContext*

callWithMarginalHandContext Player makes a *loose call* with a hand that *tighter* players would likely fold. A loose call indicates a call when the player holds a relatively mediocre or poor hand (e.g. $K\heartsuit, 7\diamondsuit$). A tight player typically only plays with very strong hands and draws.

Attribute	**type**	**weight**
$playerAction = call$	boolean	6
$holeCardRanking = 9$	how close to	4

Table 6.25: Context Template for *callWithMarginalHandContext*

callWithWeakHandContext Player makes an extremely loose call with a very weak hand (e.g. $2\clubsuit, 7\spadesuit$)

Attribute	**type**	**weight**
$playerAction = call$	boolean	6
$holeCardRanking = 9$	how close to	4

Table 6.26: Context Template for *callWithWeakHandContext*

raiseWithDrawingHandContext Player makes a raise with a strong drawing hand, in an attempt to induce either folds or *free cards* in later rounds. A free card is when a player on the come acts strong and in a later round induces other players to 'check' around to him on the next round, allowing him to see a card (and possibly catch his draw) without betting or calling.

Attribute	**type**	**weight**
$playerAction = raise$	boolean	6
$numPlayersInPot = 2$	how close to	3
$numBetsToCall = 1$	how close to	1

Table 6.27: Context Template for *raiseWithDrawingHandContext*

raiseInPositionContext Player makes a raise based mainly on his position at the table

Attribute	**type**	**weight**
$playerAction = raise$	boolean	6
$distanceFromDealerButton = 0$	how close to	3
$numPlayersInPot = 2$	how close to	0.5
$numBetsToCall = 1$	how close to	1

Table 6.28: Context Template for *raiseInPositionContext*

raiseWithStrongHandContext - Player makes a raise with a strong opening hand (e.g. $A♣, K♣$)

Attribute	**type**	**weight**
$playerAction = raise$	boolean	6
$holeCardRanking = 1$	how close to	4

Table 6.29: Context Template for *raiseWithStrongHandContext*

raiseToBluffContext - Player makes a raise with a weak hand in order to induce the table to fold out

Attribute	**type**	**weight**
$playerAction = raise$	boolean	6
$holeCardRanking = 9$	how close to	2
$numPlayersInPot = 2$	how close to	1

Table 6.30: Context Template for *raiseWithStrongHandContext*

6.8.2 Scenario #5 Results

Scenario #5 proceeded in the same manner as the previous four scenarios. The results are tabulated below.

6.8.2.1 Predictive Accuracies of FAMTILE Across Values of $\bar{\rho_a}$

For each run, 100 patterns were used as testing patterns. The results in Tables 6.31 through 6.36 represent averages for 100 runs of each particular subject, number of training patterns and $\bar{\rho_a}$ value.

For this test, FAMTILE's predictive accuracy for *both* the inferred active context of the subject and the resultant action of the subject are tabulated.

Table 6.31: Average Predictive Accuracies of Subject *Contexts* for 100-run sets of FAMTILE, 300 training patterns

$\bar{\rho_a}$	Subject 1	Subject 2	Subject 3
0.05	61.27	52.85	61.48
0.10	62.15	52.62	60.76
0.15	63.63	53.47	59.76
0.20	61.26	53.32	61.01
0.25	62.58	54.01	59.26
0.30	61.79	52.96	60.16
0.35	61.65	53.09	59.52
0.40	61.35	53.41	60.44
0.45	61.94	53.35	59.76
0.50	61.35	52.95	59.41
0.55	62.20	54.14	60.37
0.60	62.45	52.37	59.90
0.65	63.42	53.31	60.91
0.70	63.30	53.72	60.77
0.75	63.17	55.17	61.78
0.80	65.00	55.28	62.90
0.85	64.87	55.96	63.43
0.90	**65.33**	55.85	**62.56**
0.95	64.82	**55.97**	62.19

Table 6.32: Average Predictive Accuracies of Subject *Contexts* for 100-run sets of FAMTILE, 600 training patterns

$\bar{\rho_a}$	Subject 1	Subject 2	Subject 3
0.05	64.10	56.31	61.49
0.10	62.67	55.45	61.91
0.15	63.63	54.62	61.71
0.20	63.02	54.56	61.92
0.25	62.71	55.75	62.82
0.30	63.35	55.52	61.95
0.35	63.63	55.57	61.75
0.40	62.97	54.86	62.68
0.45	63.16	55.06	61.50
0.50	63.61	55.43	62.33
0.55	63.77	55.32	62.24
0.60	63.98	55.81	61.40
0.65	63.72	54.90	62.84
0.70	64.52	56.57	62.79
0.75	65.06	56.33	64.19
0.80	65.06	57.53	64.66
0.85	66.07	**58.80**	64.86
0.90	**67.75**	57.99	**64.98**
0.95	67.24	58.66	64.30

Table 6.33: Average Predictive Accuracies of Subject *Contexts* for 100-run sets of FAMTILE, 900 training patterns

$\bar{\rho_a}$	Subject 1	Subject 2	Subject 3
0.05	64.91	56.87	63.48
0.10	64.34	57.60	63.60
0.15	64.53	57.26	62.45
0.20	64.29	56.36	63.84
0.25	65.07	57.12	63.06
0.30	65.12	57.73	62.78
0.35	64.87	57.58	62.76
0.40	64.61	56.96	63.40
0.45	64.74	57.54	63.22
0.50	64.16	57.64	62.98
0.55	64.52	57.30	62.70
0.60	65.23	57.51	63.37
0.65	65.12	56.93	63.73
0.70	65.86	57.97	63.23
0.75	66.91	57.32	63.79
0.80	66.02	58.44	64.82
0.85	67.30	59.03	66.09
0.90	**67.64**	**59.85**	**66.81**
0.95	67.29	59.61	65.62

Table 6.34: Average Predictive Accuracies of Subject *Actions* for 100-run sets of FAMTILE, 300 training patterns

$\bar{\rho_a}$	Subject 1	Subject 2	Subject 3
0.05	70.62	62.48	69.86
0.10	70.06	64.05	70.23
0.15	70.77	63.75	69.34
0.20	70.52	63.48	69.51
0.25	70.30	63.55	70.55
0.30	71.21	63.48	69.89
0.35	70.10	63.93	69.47
0.40	70.40	63.45	70.58
0.45	71.38	63.81	68.86
0.50	70.40	63.25	70.43
0.55	70.81	62.99	70.40
0.60	71.34	63.22	69.43
0.65	70.98	64.36	69.72
0.70	71.49	64.96	70.78
0.75	71.46	63.67	70.88
0.80	71.91	65.05	71.16
0.85	72.98	66.59	71.24
0.90	73.36	65.88	**72.04**
0.95	**73.58**	**66.85**	71.44

Table 6.35: Average Predictive Accuracies of Subject *Actions* for 100-run sets of FAMTILE, 600 training patterns

$\bar{\rho_a}$	Subject 1	Subject 2	Subject 3
0.05	71.63	65.86	70.70
0.10	72.02	64.83	71.88
0.15	71.78	64.89	71.48
0.20	71.82	66.75	70.93
0.25	72.24	64.52	71.05
0.30	72.02	65.59	71.24
0.35	71.79	65.75	71.45
0.40	72.07	65.48	71.26
0.45	71.74	65.38	71.17
0.50	72.16	65.94	70.55
0.55	72.06	65.45	71.20
0.60	72.02	65.17	71.13
0.65	72.23	65.05	70.84
0.70	72.58	65.71	71.64
0.75	73.79	66.21	72.79
0.80	73.14	66.20	72.93
0.85	74.49	66.92	**74.40**
0.90	74.71	67.48	73.61
0.95	**75.24**	**68.55**	74.11

Table 6.36: Average Predictive Accuracies of Subject *Actions* for 100-run sets of FAMTILE, 900 training patterns

$\bar{\rho_a}$	Subject 1	Subject 2	Subject 3
0.05	72.74	66.91	72.01
0.10	74.23	66.85	73.06
0.15	72.62	66.52	72.34
0.20	72.71	65.50	72.56
0.25	73.50	66.31	72.56
0.30	72.93	66.44	71.40
0.35	72.12	66.55	72.11
0.40	73.24	66.46	72.34
0.45	73.01	66.37	71.89
0.50	73.04	67.36	72.43
0.55	72.90	66.93	71.42
0.60	71.84	66.05	72.12
0.65	73.49	66.57	72.33
0.70	73.03	67.06	73.19
0.75	74.17	66.67	73.35
0.80	74.54	66.88	73.88
0.85	74.52	68.31	74.58
0.90	75.25	69.02	**75.68**
0.95	**75.57**	**69.12**	74.93

6.8.2.2 Obtaining a Best-Case Predictive Accuracy for FAMTILE in Scenario #5

The results are first tabulated across the baseline vigilance parameter, $\bar{\rho_a}$, in order to ascertain an acceptable value for the comparison test. For each subject, learning mechanism, and training pattern total, a value for $\bar{\rho_a}$ is chosen that maximizes the cumulative predictive accuracy FAMTILE on the validation set for each subject. That value is then set as a constant parameter. The systems are each re-tested (one for each expert) in three batches of 1000-run tests: one using 300 training patterns, one using 600, and the last using 900 of the 1000 patterns generated.

The purpose of these batches of runs is to compare the best-case predictive accuracies of FAM and FAMTILE using a value of $\bar{\rho_a}$ that produced the best results for each in the previous tests. The results of these runs are tabulated in full in Appendix F, and the mean values of each 1000-run test are tabulated in 6.37 - 6.39. The results of these tests of FAM are reproduced, for convenience, from testing Scenario #3.

It is noted that the predictive accuracy of FAMTILE did not improve drastically using increased numbers of training patterns. This was also the case in Scenario #3, and is observed for both the predictive accuracy of FAMTILE for both inferred contexts and actions.

Table 6.37: Average Predictive Accuracy for 1000-run Sets with 300 training patterns

Subject	FAM	FAMTILE (Actions)	FAMTILE (Contexts)
1	72.99	73.58	65.61
2	66.01	66.15	55.64
3	71.94	72.12	63.05

Table 6.38: Average Predictive Accuracy for 1000-run Sets with 600 training patterns

Subject	Fuzzy ARTMAP	FAMTILE (Actions)	FAMTILE (Contexts)
1	74.94	75.02	66.64
2	67.55	67.82	58.10
3	73.95	73.88	64.86

Table 6.39: Average Predictive Accuracy for 1000-run Sets with 900 training patterns

Subject	Fuzzy ARTMAP	FAMTILE (Actions)	FAMTILE (Contexts)
1	75.04	75.63	67.71
2	68.54	68.92	59.98
3	75.56	75.37	66.26

6.8.3 Analysis of Scenario #5 Results

There are several interesting things to note from these tables. In terms of the primary objectives of this research, the numbers in the third column are the most important - how well does FAMTILE predict the inferred context of the subject?

FAM is the baseline for these tests. Using a set of input patterns in the form described in Chapter 5, the neural network creates a set of clusters that maps similar patterns to one of three subject actions - call, raise, or fold. The numbers in the first column from Scenario #3 indicate how well FAM was able, on average, to predict the subject's action among the testing data given. No context inference is performed. The second column represents the ability of FAMTILE to make the same prediction. Rather than mapping input patterns to outputs, however, FAMTILE instead creates a mapping between the input pattern and the inferred active context for that pattern. For instance, consider the situation where the subject holds 5♠, 6♠ in the big blind. Four opponents call the blind, and the subject chooses to raise with his good multiway hand. Analyzing the attributes of the system, TBI infers that the subject is currently in *RaiseWithDrawingHand*. If this pattern was used to train FAMTILE, the network would choose the cluster most similar to the pattern that mapped to the *RaiseWithDrawingHand* context. During testing, if FAMTILE encounters a situation similar to this, it will choose the *RaiseWithDrawingHandContext*, and thus predict that the player will raise in that situation. Column 2 tabulates the predictive accuracy of the consequent action that follows from the predicted context.

A statistical analysis of the 1000-run batches for each subject was run to compare the best-case mean predictive accuracies of FAM, FAMTILE for actions, and FAMTILE for inferred contexts. These accuracies were tabulated in tables 6.37 - 6.39. For this analysis, a 2-tailed t-test was performed to compare these predictive accuracies. The results of this analysis is summarized in Table 6.40.

As the table above illustrates, these predictive accuracies of the subject's action for FAM and FAMTILE are nearly identical for each batch of runs and each subject. In the best case, for subject #1 with 900 training patterns, FAMTILE

Table 6.40: Tabulated 2-tailed t-tests on Best-Case Action Predictive Accuracies for Scenarios #3 and #5

Subject	# training patterns	μ_1	μ_2	$\mu_1 - \mu_2$	99%CI($\alpha = 0.01$)	p-value
1	300	73.58	72.99	0.587	(0.071,1.103)	0.003
1	600	75.02	74.94	0.080	(-0.410,0.570)	0.674
1	900	75.63	75.40	0.224	(-0.228,0.676)	0.201
2	300	66.15	66.01	0.143	(-0.409,0.695)	0.143
2	600	67.82	67.55	0.267	(-0.255,0.789)	0.187
2	900	68.92	68.55	0.372	(-0.135,0.879)	0.059
3	300	72.12	71.94	0.176	(-0.334,0.686)	0.374
3	600	73.89	73.95	-0.062	(-0.556,0,432)	0.746
3	900	75.37	75.56	-0.187	(-0.666,0.292)	0.315

outperformed FAM with an average of 75.63 correct predictions versus 75.04 for FAM. In the worst case, for subject #3 also with 900 training patterns, FAM narrowly outperformed FAMTILE with an average of 75.56 correct predictions versus 75.37 for FAMTILE. Neither of these margins are statistically significant.

In addition, FAMTILE is able to accurately predict the subject's active context an average of 67.71, 59.98, and 66.26 times for each of the three subjects observed, respectively, at optimum values for $\bar{\rho}_a$. Comparing these accuracies with those of FAM for predicting subject actions, we note that FAMTILE is an average of only 11.52% less effective at predicting contexts than FAM is at predicting actions.

The fact that FAMTILE is able to generate a competitive degree of context-predicting accuracy *without* disrupting the ability of FAM is a significant finding. In effect, therefore, we have created a system that adds the ability to predict context transitions to a neural network without affecting its ability to predict simple actions. The utility of such a system will be discussed in more detail in Chapter 7.

6.9 Testing Scenario #6

In Scenario #6, predictive accuracies for FAMTILE are collected and analyzed for Vignette *D* as they were for FAM in Scenario #4. Concluding this section will be a comparative analysis of the accuracies of the two systems, analogous to the analysis generated for the results of FAM and FAMTILE on Vignette *C*.

6.9.1 Template Descriptions for Vignette *D*

Below is the list of context templates used by FAMTILE for Vignette *D*. Each template description defines the situations under which the template is described along with each attribute and weight assigned to it.

There are 24 contexts used for Vignette D. The main reason for this increase versus Vignette *C* is that there are two additional actions available to the player (bet and raise).

The context templates developed for Vignette *D*, like in Vignette *C*, serve to partition the situations for which each of five possible actions (raise, call, fold, *check, bet*). Each template contains attributes from the following set:

1. playerAction (what the player did - fold, call, or raise)

2. handStrength (the strength of the player's hand as computed within Loki [BS98])

3. pPot (an index representing the player's potential to draw to a winning hand, as computed within Loki [BS98])

4. nPot (an index representing the opponent's potential to draw to a winning hand against the player, as computed within Loki [BS98]

Notice that the number of attributes used to infer context in Vignette *D* are fewer than the number used in the previous Scenario. The biggest motivation for this change was the results obtained in Scenario #5 for the test subject #2, discussed in the previous section. In those Scenarios, it was found that FAMTILE was less successful in predicting that subject's context or action than it was for the other two subjects. For Vignette *D*, therefore, the contexts created for Vignette were designed to be more basic. Concepts such as position and opponent aggression are removed from these templates, and the only attributes considered are the player's action, the strength of his hand, the strength of his draw, and the potential strength of his opponents' draws. By doing this, we remove the situations where TBI infers a context that the player is 'raising in position' in spite of the player's ignorance of that factor. All that is assumed in these template definitions is a basic concept hand strength during the game, i.e.

- strength of the player's hole cards
- strength of the player's hand based on the community cards
- likelihood the player can draw to a winning hand, based on the community cards

- likelihood the player has a winning hand but can be drawn out on by an opponent, based on the community cards

We consider the items above to be concepts known by all players of the game, not just the more experienced ones. By limiting the template attributes to these, therefore, each of the contexts become applicable. By contrast, contexts with template attributes that are not understood by novice players are inapplicable. For instance, it would not make sense to infer the a $RaiseInPosition$ context (used in Vignette C) for a novice player who does not know what position means or what it implies.

By setting up the context templates so that they refer to the most basic of game principles, we force the system to learn the more advanced knowledge involved in selecting a specific context when several are possible. For example, consider the contexts $RaiseWithDrawingHand$, $CallWithDrawingHand$ and $FoldWithDrawingHand$. Both refer to situations where the player is holding a drawing hand (e.g. if a player is holding $5\spadesuit, 6\clubsuit$ and the board is showing $4\heartsuit, 7\clubsuit, Q\diamondsuit$ - the player can draw an 8 or a 3 to make a straight), but the actions taken are different. There are several possible reasons to choose to raise versus a call, many of which involve advanced concepts not easily identified by more novice players. In fact, the identification and response to these situations is part of what separates an expert player from a novice player, and exactly the kind of knowledge we are interested in learning. For intelligence in a poker environment, this type of decision is what we consider to be a high-level decision: What conditions call for the $RaiseWithDrawingHand$ context versus the $CallWithDrawingHand$ context for a certain player?

foldWithWeakHandContext Player folds a weak hand relative to the board

Table 6.41: Context Template for *foldWithWeakHandContext*

Attribute	**type**	**weight**
$playerAction = fold$	boolean	50
$handStrength < 0.4$	boolean	40
$pPot = 0$	boolean	10

foldWithDrawingHandContext Player folds a hand that could draw to a winning hand

Table 6.42: Context Template for *foldWithDrawingHandContext*

Attribute	**type**	**weight**
$playerAction = fold$	boolean	50
$handStrength > 0.1$	boolean	10
$pPot > 0.2$	boolean	40

foldWithMediocreHandContext Player folds an average hand

Table 6.43: Context Template for *foldWithMediocreHandContext*

Attribute	**type**	**weight**
$playerAction = fold$	boolean	70
$handStrength < 0.6$	boolean	20
$pPot > 0.1$	boolean	10

foldWithStrongHand Player folds a strong hand

Table 6.44: Context Template for *foldWithStrongHand*

Attribute	**type**	**weight**
$playerAction = fold$	boolean	70
$handStrength > 0.7$	boolean	30

checkWithWeakHandContext Player checks a weak hand

Table 6.45: Context Template for *checkWithWeakHandContext*

Attribute	**type**	**weight**
$playerAction = check$	boolean	70
$handStrength < 0.4$	boolean	20
$pPot < 0.1$	boolean	10

checkWithMediocreHandContext Player checks a mediocre hand

Table 6.46: Context Template for *checkWithMediocreHandContext*

Attribute	**type**	**weight**
$playerAction = check$	boolean	70
$handStrength < 0.6$	boolean	20
$pPot < 0.15$	boolean	10

checkWithDrawingHandContext Player checks a drawing hand

Table 6.47: Context Template for *checkWithDrawingHandContext*

Attribute	**type**	**weight**
$playerAction = check$	boolean	70
$handStrength < 0.8$	boolean	5
$pPot > 0.2$	boolean	25

checkWithMonsterHandContext Player checks a very strong hand

Table 6.48: Context Template for *checkWithMonsterHandContext*

Attribute	**type**	**weight**
$playerAction = check$	boolean	70
$handStrength > 0.95$	boolean	20
$nPot < 0.05$	boolean	10

checkWithStrongButVulnerableHandContext Player checks a strong hand that could get drawn out on

Table 6.49: Context Template for *checkWithStrongButVulnerableHandContext*

Attribute	**type**	**weight**
$playerAction = check$	boolean	70
$handStrength > 0.7$	boolean	20
$nPot > 0.15$	boolean	10

betWithWeakHandContext Player bets a weak hand

Table 6.50: Context Template for *betWithWeakHandContext*

Attribute	**type**	**weight**
$playerAction = bet$	boolean	70
$handStrength < 0.4$	boolean	20
$pPot < 0.1$	boolean	10

betWithDrawingHandContext Player bets a drawing hand

Table 6.51: Context Template for *betWithWeakHandContext*

Attribute	**type**	**weight**
$playerAction = bet$	boolean	70
$handStrength < 0.8$	boolean	5
$pPot > 0.2$	boolean	25

betWithMediocreHandContext Player bets a mediocre hand

Table 6.52: Context Template for *betWithMediocreHandContext*

Attribute	**type**	**weight**
$playerAction = bet$	boolean	70
$handStrength < 0.6$	boolean	20
$pPot < 0.15$	boolean	10

betWithMonsterHandContext Player bets a very strong hand

Table 6.53: Context Template for *betWithMonsterHandContext*

Attribute	**type**	**weight**
$playerAction = bet$	boolean	70
$handStrength > 0.95$	boolean	20
$nPot < 0.05$	boolean	10

betWithStrongButVulnerableHandContext Player bets a strong hand that could get drawn out on

Table 6.54: Context Template for *betWithStrongButVulnerableHandContext*

Attribute	**type**	**weight**
$playerAction = bet$	boolean	70
$handStrength > 0.7$	boolean	20
$nPot > 0.15$	boolean	10

callWithWeakHandContext Player calls a weak hand

Table 6.55: Context Template for *callWithWeakHandContext*

Attribute	**type**	**weight**
$playerAction = call$	boolean	70
$handStrength < 0.4$	boolean	20
$pPot < 0.1$	boolean	10

callWithDrawingHandContext Player calls a drawing hand

Table 6.56: Context Template for *callWithWeakHandContext*

Attribute	**type**	**weight**
$playerAction = call$	boolean	70
$handStrength < 0.8$	boolean	5
$pPot > 0.2$	boolean	25

callWithMediocreHandContext Player calls a mediocre hand

Table 6.57: Context Template for *callWithMediocreHandContext*

Attribute	**type**	**weight**
$playerAction = call$	boolean	70
$handStrength < 0.6$	boolean	20
$pPot < 0.15$	boolean	10

callWithMonsterHandContext Player calls a very strong hand

Table 6.58: Context Template for *callWithMonsterHandContext*

Attribute	**type**	**weight**
$playerAction = call$	boolean	70
$handStrength > 0.95$	boolean	20
$nPot < 0.05$	boolean	10

callWithStrongButVulnerableHandContext Player calls a strong hand that could get drawn out on

Table 6.59: Context Template for *callWithStrongButVulnerableHandContext*

Attribute	**type**	**weight**
$playerAction = call$	boolean	70
$handStrength > 0.7$	boolean	20
$nPot > 0.15$	boolean	10

raiseWithWeakHandContext Player raises a weak hand

Table 6.60: Context Template for *raiseWithWeakHandContext*

Attribute	**type**	**weight**
$playerAction = raise$	boolean	70
$handStrength < 0.4$	boolean	20
$pPot < 0.1$	boolean	10

raiseWithDrawingHandContext Player raises a drawing hand

Table 6.61: Context Template for *raiseWithWeakHandContext*

Attribute	**type**	**weight**
$playerAction = raise$	boolean	70
$handStrength < 0.8$	boolean	5
$pPot > 0.2$	boolean	25

raiseWithMediocreHandContext Player raises a mediocre hand

Table 6.62: Context Template for *raiseWithMediocreHandContext*

Attribute	**type**	**weight**
$playerAction = raise$	boolean	70
$handStrength < 0.6$	boolean	20
$pPot < 0.15$	boolean	10

raiseWithMonsterHandContext Player raises a very strong hand

Table 6.63: Context Template for *raiseWithMonsterHandContext*

Attribute	**type**	**weight**
$playerAction = raise$	boolean	70
$handStrength > 0.95$	boolean	20
$nPot < 0.05$	boolean	10

raiseWithStrongButVulnerableHandContext Player raises a strong hand that could get drawn out on

Table 6.64: Context Template for *raiseWithStrongButVulnerableHandContext*

Attribute	**type**	**weight**
$playerAction = raise$	boolean	70
$handStrength > 0.7$	boolean	20
$nPot > 0.15$	boolean	10

6.9.2 Scenario #6 Results

Scenario #6 proceeded in the same manner as the previous five Scenarios. The results are tabulated below. 900 patterns were used to train FAMTILE in each test. It was found in Scenarios #3 and #5 that the number of training points did not significantly affect the predictive accuracy beyond 300 training points, though the systems performed better with the maximum amount of 900 points for testing. Therefore, the training point number was not varied in this testing Scenario.

6.9.2.1 Predictive Accuracies of FAMTILE Across Values of $\bar{\rho_a}$

A series of runs were executed to determine the value for $\bar{\rho_a}$ where FAMTILE achieves the best predictive accuracy for each subject. These runs are performed for each each subject using all but 100 of the total number of patterns generated for training. The remaining 100 patterns are used for the validation set, from which a predictive accuracy can be generated.

Both the training and validation sets used by FAMTILE in this testing Scenario were generated from the same subject patterns used by FAM for Scenario #4.

Table 6.65: Average Predictive Accuracies of Subject *Contexts* for FAMTILE

$\bar{\rho_a}$	Subject 1	Subject 2	Subject 3
0.05	40.2	41.0	34.2
0.1	40.5	40.0	34.8
0.15	40.9	40.1	37.2
0.2	42.6	**43.2**	**39.2**
0.25	41.6	40.0	38.0
0.3	38.7	41.7	36.3
0.35	**42.9**	41	35.0
0.4	39.1	40.5	36.6
0.45	40.7	40.3	36.9
0.5	39.2	37.2	35.8
0.55	39.1	40.0	37.8
0.6	41.3	40.3	36.4
0.65	39.3	39.8	37.2
0.7	38.1	39.0	37.3
0.75	39.6	41.5	38.5
0.8	37.9	38.0	36.6
0.85	40.1	39.0	37.8
0.9	40.6	42.5	37.0
0.95	40.8	40.9	37.1

Table 6.66: Average Predictive Accuracies of Subject *Actions* for FAMTILE

$\bar{\rho_a}$	Subject 1	Subject 2	Subject 3
0.05	54.5	59.2	50.8
0.1	53.8	59.4	52.0
0.15	55.5	58.3	**53.4**
0.2	54.4	56.8	50.2
0.25	53.9	57.7	50.1
0.3	55.5	58.9	49.7
0.35	54.0	57.7	50.5
0.4	55.6	59.8	51.4
0.45	53.6	57.9	49.7
0.5	55.1	56.5	51.1
0.55	53.3	59.2	51.0
0.6	54.9	57.3	49.3
0.65	53.0	58.1	50.2
0.7	55.6	58.8	49.2
0.75	56.0	59.0	49.9
0.8	56.2	**60.7**	50.3
0.85	55.3	55.4	51.0
0.9	55.9	57.1	49.3
0.95	**60.7**	57.9	49.9

6.9.2.2 Obtaining the Best-Case Predictive Accuracy of FAMTILE for Scenario #6

For each subject, the $\bar{\rho_a}$ the trials summarized in Tables 6.65 and 6.66 were used for a second set of 1000 runs. Across each of these 1000 runs, $\bar{\rho_a}$ was fixed at the value that produced the greatest average predictive accuracy of inferred contexts and actions for that subject.

Table 6.67: Average Predictive Accuracy of FAMTILE for Inferred Contexts and Actions over 1000-run Sets

Subject	$\bar{\rho_a}$	Context Predictive Accuracy	$\bar{\rho_a}$	Action Predictive Accuracy
1	0.35	43.22	0.95	60.25
2	0.20	47.79	0.8	60.14
3	0.20	39.99	0.15	54.07

6.9.3 Analysis of Scenario #6 Results

Just as in Scenario #5, FAMTILE achieved best-case predictive accuracies for each subject which were on par with those recorded by FAM during its evaluation. Here, the average best-case predictive accuracies reached (for subject actions) were 60.25, 60.14 and 54.07, respectively. These values are compared to values of 58.22, 60.18, and 55.32 achieved by FAM in Scenario #4.

Table 6.68 summarizes the results of a 2-tailed *t*-test on the best-case predictive accuracy means achieved in Scenarios #4 and #6 for each subject.

Table 6.68: Tabulated 2-tailed t-tests on Best-Case Action Predictive Accuracies for Scenarios #4 and #6

Subject	μ_6	μ_4	$\mu_6 - \mu_4$	99%CI($\alpha = 0.01$)	p-value
1	60.25	58.22	2.30	(1.253,3.347)	0.778
2	60.14	60.18	-0.04	(-0.460,0.380)	0.006
3	54.07	55.32	-1.25	(-2.38,-0.120)	0.572

The predictive accuracy of FAMTILE for predicting subject's inferred context also dropped considerably from the values achieved in Scenario #5. Whereas FAMTILE predicted contexts at rates of 67.71, 59.98 and 66.26 for Vignette C, those accuracies dropped by an average of over 28% across the two subjects who then also participated in Vignette D. One significant reason for this was the the increase in number of contexts. This number doubled from 12 to 24 contexts for Vignette D because two new actions needed to be accounted for (bet, check) along with representation of contexts potentially present after the pre-flop round of betting. Note that with 24 contexts, a random guess of the inferred active context could be expected to be correct slightly more than 4% of the time, which is ten times less than the accuracy achieved by FAMTILE.

Furthermore, Vignette D requires the player to reason about entirely new and more complex situations than faced in Vignette C. In addition to his/her hole cards, the player must also consider not only the community cards, but also the action of previous betting rounds and the possible responses of each opponent in response to a particular action.

6.10 Test Scenario #7

This section defines the purpose, motivation and results for the final testing Scenario used to evaluate both FAM and FAMTILE.

For this Scenario, the data collected from Vignette *D* is used. In this Vignette, the subject is placed at a random position at a poker table, and is asked asked to make decisions throughout entire hands and accumulate chips throughout the Scenario. As each hand is dealt, and each opponent makes an action on their cards until it is the subject's turn to act. When the subject acts, the betting round continues as well as the hand, and proceeds just as a standard round of Limit Hold'em. After each round, the dealer button rotates one chair to the left and a new hand is dealt. A *chip-count* is stored for the subject, which will reflect the amount of money won/lost during the sequence of hands played.

6.10.1 Scenario #7 Motivation

The final evaluation places both FAM and FAMTILE in the position of the very subject it observed. Both systems then perform the task of playing Limit Hold'em against a set of seven computer-generated opponents.

In doing this, the two systems must do the following at each decision point:

1. Observe the situation at the table
2. Transform situation into an input pattern
3. Choose an action/context based on the output of FAM/FAMTILE
4. Execute the action

In the general case, a CxBR model (whose individual contexts are written by a knowledge engineer externally to the learning process) controls an agent executing the desired behavior. The intelligence acquired by the two learning systems, therefore, drives the selection of actions made during the hand.

To perform this evaluation, FAMTILE observes an subject playing 1000 hands of Limit Hold'em. After this observation period, the transition logic learned by FAMTILE is used to determine actions during the Scenario. The model then itself plays 1000 hands, and the number of chips for both the subject during observation and the model after hand played are compared.

6.10.2 Scenario #7 Results

The purpose of Scenario #7 for testing the learning systems is to evaluate its ability to play Texas Hold'em and accumulate chips at a similar rate as the subject they observed. In the case of FAMTILE, this involves the following actions:

1. Observe an subject play 1000 hands of Limit Hold'em
2. Generate training patterns from each decision point the subject faced during observation
3. Use patterns to train Fuzzy ARTMAP, create a mapping between subject stimuli and output context
4. System executes the Vignette and uses learned knowledge to generate actions in response to the game

Consider the situation where FAMTILE is playing the role of the subject, and is involved in a hand in late position with the $10\clubsuit, J\clubsuit$ and the flop comes $7\clubsuit, 8\diamondsuit, 9\diamondsuit$. It has two opponents still in the hand, both acting first. The first opponent checks, and the second calls.

FAMTILE would first convert this observation into an input pattern for FAM. After being presented this pattern, FAM would return an output pattern that represents one specific context, e.g. *CallWithMonsterHandContext*. FAMTILE then executes the appropriate action as implied by the context - in this case a call.

The opponents would then react to that action. The first opponent can either call the bet (since he first checked), fold, or make a raise. The second opponent will only act again during this betting round if the first opponent makes a raise. Depending on those actions, a new situation will arise for FAMTILE. If the first player calls or folds, 4th street (the turn) will be dealt and a new round of betting will commence. Otherwise FAMTILE may have to call another raise to remain in the hand.

6.10.2.1 Chip Count Comparisons of Subject, FAM and FAMTILE

To achieve these comparisons, both FAM and FAMTILE must first be trained with subject data from Vignette D. Based on the results from Scenarios #5 and #6, each system was provided a value for $\bar{\rho_a}$ where it achieved its best case predictive accuracy. Each system was then repeatedly trained until it achieved the maximum score reached during their previous evaluations. As soon as it achieves the maximum accuracy on the testing set, the system is inserted into the simulation. 1000 hands are then simulated with each system FAMTILE acting in place of the subject. A running tally of remaining chips is kept.

To analyze the results of these tests, the 1000-hand sequence for both the subject used for training, FAM and FAMTILE was first broken down into a series of ten 100-hand sequences. From these ten sets, the best and worst-case results were truncated from the set. This was done to dampen the effect of luck that, while innate to poker, tends to even out over the long run.

For example, suppose FAM achieves the following chip gains and losses over ten 100-round sequences: -25, +50, +125, -5, -7, +12, -85, +80, -60, -18. Truncation would remove the sequence where the system won 125 and also the sequence where it lost 85 chips.

To generate the graphs in figures 6.9 - 6.11, the remaining 100-round sequences were averaged to generate a mean stack size across each round. These figures show the results of this average for both systems as well as the subject observed. Each graph represents the average chip count at each point in the 100-hand sequence.

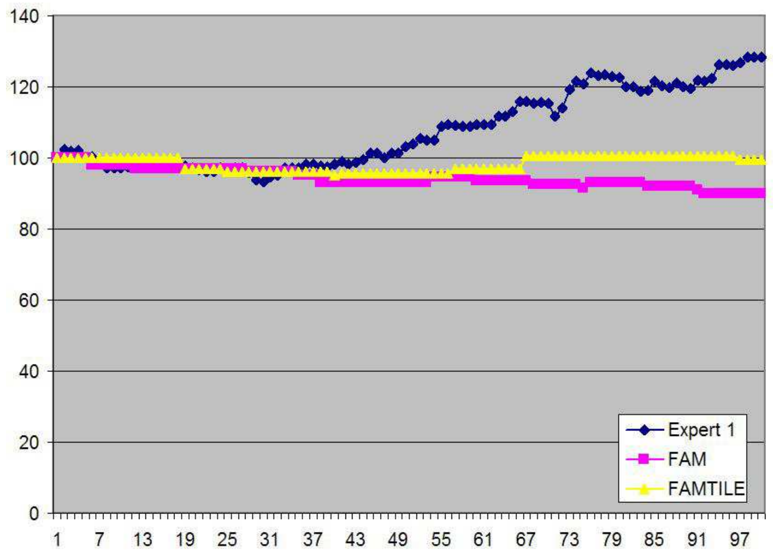


Figure 6.9: Chip Count Comparison of Subject #1 versus FAM and FAMTILE

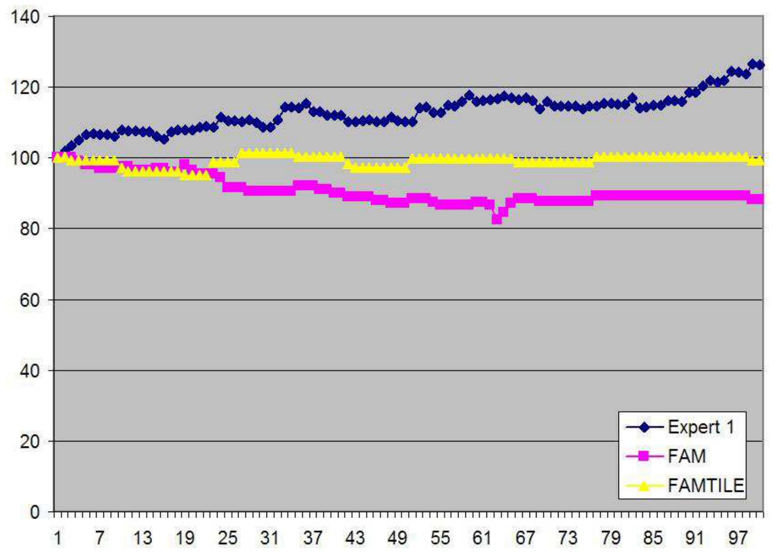


Figure 6.10: Chip Count Comparison of Subject #2 versus FAM and FAMTILE

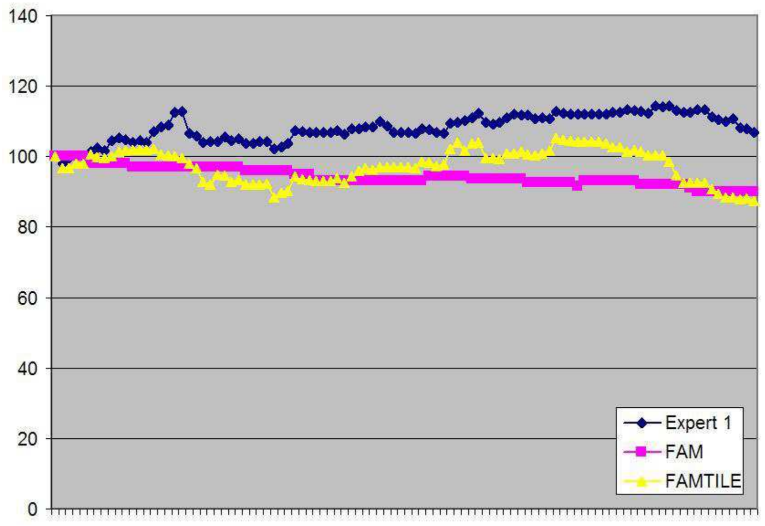


Figure 6.11: Chip Count Comparison of Subject #3 versus FAM and FAMTILE

6.10.2.2 Analysis of Scenario #7 Results

As expected based on the results of Scenarios #5 and #6, neither FAM nor FAMTILE was able to achieve a comparable increase in chip stack as the subjects they observed.

By only being able to predict around 60% of the subject's actions, both FAM and FAMTILE lack a great deal of the intelligence required to extrapolate decisions that do not closely match those learned during training. This problem is most visible in the decisions made after the pre-flop round of betting, where neither FAM or FAMTILE was able to choose correct contexts in post-flop betting rounds.

The main reason for this lack of 'post-flop intelligence' is the fact that the vast majority of training patterns were of pre-flop decisions. The subject was asked to play 1000 hands - in most of those hands, the subject did what basic Hold'em strategy would tell him/her to do - fold. Only in a small subset of those 1000 hands was the subject in the round for subsequent betting rounds so that training patterns could be generated with subject decisions from them.

To correctly train both networks so that they can attempt to fully model subject behavior in this Vignette, the poker simulation would need to be adjusted to set-up more (and varied) situations for post-flop decisions by the subject, instead of allowing the cards to be dealt realistically. This step is left for future research.

6.11 Summary and Conclusion of FAMTILE Evaluation

Throughout each of the seven testing Scenarios, the objective was to evaluate both FAM and FAMTILE using a sequence of Vignettes with varying degrees of complexity.

For the two simpler Vignettes (A and B), the basic FAM neural network was evaluated as to how well it could predict human actions in this setting. These evaluations were performed as testing Scenarios #1 and #2. FAM was able to achieve good predictive accuracies for each subject in both Vignettes, achieving over 95% accuracy in the best case.

With Vignette C, an attempt was made to create a significantly more complex tactical behavior that was also more dependent on the skill level and style of the subjects observed. Here, subjects were placed at a virtual Texas Hold'em table and asked to make a series of individual decisions (either to raise, call, or fold) based solely on his/her hole cards, his position at the table, and the actions of his opponents. Results from testing FAM in this environment were recorded as part of Scenario #3, and verified the increased complexity present in Vignette C. The network experienced over an 11% decrease in performance versus its recorded predictive accuracies from Scenarios #1 and #2. Nevertheless, FAM was able to achieve an average of over 70% predictive accuracy for each of the three players, and it is hypothesized that this accuracy could possibly be increased by slightly modifying the observations taken for the subjects during training. This suggestion is explored in more detail in section 7.5.

In Scenario #5, FAMTILE is asked to repeat the task performed by FAM in Scenario #3. Instead of learning subject actions, however, FAMTILE learns by converting each action to an inferred active context using a TBI engine and a listing of templates (provided in section 6.8.1, just as described in Chapters 4 and 5. Because there is a many-to-one mapping between context and action for Vignette C, the predicted context of FAMTILE can then easily be converted to represent the predicted action of the subject.

The results of Scenario #5 quantify the effectiveness of using FAMTILE to learn subject behaviors for turn-based tactical behaviors such as the game modeled by Vignette C. According to the tabulated results, FAMTILE is only 10% less effective at predicting the inferred context of the subject than FAM was in Scenario #3 at predicting the subject's resultant action. Furthermore, FAMTILE is able to then reproduce FAM's predictive accuracy by performing a simple conversion of those predicted contexts.

CHAPTER 7

SUMMARY, CONCLUSIONS AND FUTURE WORK

This chapter includes a summary of the work accomplished in this research, a list of the conclusions made from testing the system, and a discussions of possible directions that this research can be carried in the future.

7.1 Summary

The purpose of this research was the design of an algorithm capable of learning high-level, tactical behavior from the observations of an expert. In order to clarify the scope of 'high-level behavior', the modeling paradigm known as Context-Based Reasoning is introduced. In CxBR models, all knowledge is represented and activated within the abstraction of a context. A context is a set of environmental and physical conditions that may suggest a specific behavior or action [SG04]. Using the terms defined in this paradigm, high-level behavior is equivalent to the entire behavior to be represented by a CxBR model. CxBR contexts represent the lower-level 'sub-behaviors' that the expert executes while demonstrating a high-level behavior.

If it is assumed that these sub-behaviors are known and can be modeled by a knowledge engineer (KE) *a priori*, the task of the learning algorithm becomes that of identifying and learning the cues that result in an expert changing his low-level context. In terms of a CxBR model, this task is equivalent to learning the context-transition logic that determines the sequence of contexts executed by the model during the behavior.

The algorithm proposed to address this task is known as FAMTILE - Fuzzy ARTMAP Template-Based Interpretation Learning Engine. The FAMTILE engine includes two main components - a Fuzzy ARTMAP neural network and a Template-Based Interpretation engine. Developed = by Carpenter et al [CR92], Fuzzy ARTMAP is a type of neural network that learns by creating pattern *clusters* that group similar input and output patterns and creating a mapping between them. For FAMTILE, the input and output patterns will represent - respectively - the observation seen by the expert and the output context identified as his response to that cue.

Template-Based Interpretation is a technique developed by Drewes and Gonzalez [GG00]. TBI allows for context *templates* to be created that describe conditions and expert behavior conducive to his existence within that context. Using this technique, the low-level contexts active at each point during the expert's high-level behavior are inferred for each observation. Prior to observation-time, a KA develops templates for each low-level context possible for some high-level behavior. To do this, the defined context templates for each are referenced and marked when a certain attribute expected for that context is observed. When each attribute for each template has been reviewed, the context corresponding to the template with the highest 'score' observed is considered to be the most likely candidate for the expert's intent.

Using TBI, FAMTILE constructs a set of input/output patterns for presentation to Fuzzy ARTMAP. Each input pattern represents the expert's observation for a specific decision point, while each output pattern represents the inferred output context for that point. During training, the FAM within FAMTILE creates clusters matching similar input patterns that map to the same output context. These clusters can be converted into context transition rules using the rule-extraction technique introduced in [CT95].

Four training vignettes were constructed to generate training data for the learning systems. The first two (A and B involve a maze vignette, where experts are asked to navigate towards a goal position, given a small viewable window of his surroundings along with a distance vector in the direction of the goal. The purpose of these two vignettes was to confirm the ability of FAM to learn different expert's actions given a simple tactical scenario. As these vignettes were less complex than those posed in vignettes C and D, they also served to provide a ceiling for the kind of predictive accuracy that could be expected from FAMTILE in the best case.

The second pair of vignettes involved the game of Limit Texas Hold'em Poker. Here, a human expert is asked to reason about his/her own cards, position at the table, and the actions of other players. In response to these stimuli, the expert is asked to make an appropriate action - either to check, bet, call, raise, or fold - just as he would during a regular game of Limit Hold'em.

Seven testing scenarios were used to evaluate the performance of both FAM and FAMTILE in predicting expert decisions for each of the four vignettes.

7.2 Conclusions

Based on the results tabulated in chapter 6, it is concluded that FAMTILE is an adequate technique for learning high-level behaviors and, offers several promising characteristics that can be exploited in future research. Because it is able to learn low-level expert contexts without adversely affecting the clustering ability of Fuzzy ARTMAP, we feel that the FAMTILE system provides a significant tool for learning in systems where it is desirable to gain perspective of *why* the expert is doing what he is doing.

The results of the two maze scenarios provide a good indication as to Fuzzy ARTMAP's ability to predict expert responses to an observation. In scenario #1, the network is able to predict the expert's movement for an average of 86% on the validation set, achieving nearly a 95% average for one of the three experts. This scenario included input training patterns with 27 fields and 4 possible output patterns. The second maze scenario expanded the expert's viewing range, more than tripling the number of input-pattern fields to 88 (92 if the expert's previous action was recorded and considered). Nevertheless, Fuzzy ARTMAP is able to predict 85% of the validation set for the three experts, increasing to nearly 87% when the expert's previous action is considered.

While these are impressive numbers for predicting three different expert's actions, they only speak to the successes of Fuzzy ARTMAP and not to FAMTILE. These scenarios were executed and reported, for the most part, to justify the use of Fuzzy ARTMAP for doing the low-level learning task. Had these evaluations been a failure a different learning system would have been selected, one that performed better at predicting actions within these training scenarios.

Another reason for the documentation of the two maze vignettes was to generate a contrast with the more difficult task of predicting expert behavior in a poker environment. As shown by the results for scenario #3, Fuzzy ARTMAP has a much more difficult time predicting actions in this settings. With only 12 input pattern fields and 3 possible output patterns (compared with 88 and 4 for scenario #2), FAM is only able to achieve just over 73% prediction accuracy across all three experts, reaching a maximum of 75% for the 3^{rd} expert.

This variability in difficulty amongst the maze and poker vignettes seemed to create a good set of conditions for evaluating both FAM and FAMTILE. The first expert-prediction task was found to be relatively easy, and it reflects some variability amongst each of the three experts observed. The second two scenarios introduce a Poker scenario. These vignettes introduce a learning challenge that, while containing a comparable number of input-pattern fields and output possibilities, proved to be a more difficult task for both systems.

As described in chapters 4 and 5, FAMTILE requires the use of a completely separate TBI module that encodes *a priori* knowledge about the scenario within its context templates - while Fuzzy ARTMAP itself requires no such input - and fails to produce a worthwhile increase in predictive performance. A separate set of tests were run to evaluate FAMTILE's ability to correctly predict the inferred expert context for each decision point. While these tests resulted in lower predictive accuracies - certainly expected because the neural network must choose between 12 possible output patterns instead of only 3 when predicting actions - the results were promising. Using 900 training patterns, FAMTILE is able to correctly predict an average of 64.77 contexts out of a possible 100 across the three experts. As reported in chapter 6, FAMTILE's predictive accuracy for contexts is only around 11% worse than its accuracy for actions. This accuracy is achieved, furthermore, without affecting the accuracy of the network in predicting the expert's overall action.

Because of its ability to in-obtrusively predict contexts, we feel that the FAMTILE system is useful for learning tasks, specifically ones that:

- the behavior satisfies the characteristics of high-level tactical behavior, as defined in chapter 1
- the user is interested in creating models of the expert's behavior and is more interested in his resultant intentions and motivations than the actions observed at the lowest-level
- behaviors where the expert's ultimate action is more closely tied to his low-level behavior than to the raw observation presented at each decision point

The final testing scenario allowed both systems to replace the expert within the simulation and attempt to model his actions throughout a sequence of 1000 Texas Hold'em rounds. During each round, the chip-count of the expert was recorded and compared against the expert's chip-count at the round during the observation phase. While the first series of tests evaluated FAMTILE's ability to predict the expert's intent, this second phase of tests provided insight as to whether the knowledge acquired by the system during observation was enough to allow it to adequately execute a series of Hold'em rounds.

The results of this phase of tests were inconclusive, primarily due to the nature of the test. Since the game of Poker involves swings of good cards and bad cards, it was difficult to gain any perspective on the comparative abilities of FAMTILE and the expert it attempted to model using this chip-count comparison. The same can be said of any Poker player, for that matter. Even the top-tier professional players, for instance, have sessions where they lose money.

It was also noticed that players started catching on to the betting patterns and styles of the computer-generated opponents during the vignette. Each test subject remarked that he/she was able to identify patterns where it was known how the opponent would react. This tended to affect the action of the subject. Instead of making a play based on the parameters of the game, the player would instead make a move compatible with how he/she expected the opponent to react. Because of this, it is likely that the learning system could achieve better results using training patterns gathered from play against human opponents.

7.2.1 Lessons Learned

The most important lesson learned from this research is that learning and replicating human behavior is a difficult task to do well, specifically when constrained by architectures that do not represent the unknown mechanics of actual human decision-making. For instance, the central assumption made for this research was that high-level behavior can be represented by a sequence of lower-level behaviors that can be modeled by CxBR contexts. However, the trick then becomes defining and partitioning each context of a behavior in such a manner that they are truly atomic and identifiable independent of the specific expert being observed.

For example, consider the *RaiseWithStrongButVulnerableHand* context used in vignette D. This context was modeled to represent cases where the expert believes he has the best hand at the moment but also that his opponents can easily draw cards to beat him.

This context raises an interesting question - what if the expert doesn't recognize this? Obviously, then, the templates must be defined such that this context does not get inferred. But what if there are no contexts that accurately represent the low-level motivation and behavior of the expert?

High-level behaviors whose specifics are heavily dependent on human preference and expertise are equally difficult to represent. While a significant amount of *a priori* knowledge was encoded into the context templates used for Scenarios #3 and #4, that knowledge certainly does not represent the full range of motivations and contexts that constitute the entire task of playing Hold'em Poker. This is because these contexts are so dependent on the tendencies of the expert.

However, that is not to say that these assumptions serve only to doom the chances of success for the algorithm. On the contrary, these assumptions provide a means for motivating the directions that HBR research can progress. If we choose to learn a task where the modeling architecture, expert dependencies, and context topologies are all known, it is likely that the task modeled is too simple and not worth modeling. Texas Hold'em Poker, on the other hand, is an extremely complex game; and the number of techniques, strategies, and styles documented and used by advanced players suggest that the game is as much of an art as it is a science. As a supplemental testing procedure for FAMTILE, therefore, a separate vignette could be developed that has the following characteristics:

- Easily defined and identifiable contexts
- More direct correlation between expert skill and success within the scenario
- Individual performance parameters that can be compared between the FAMTILE algorithm and the expert modeled

With this vignette, we can attempt to eliminate the cases where expert intention is unclear. It is hypothesized that such a scenario would increase the predictive capacity of FAM within the FAMTILE architecture.

7.3 FAMTILE and Explainable AI

As reported earlier in this chapter, a significant result of this research was that FAMTILE was able to learn expert contexts to some extent without affecting the predictive accuracy of FAM to predict the expert's actions in that scenario. We feel that these results are most applicable in the arena of *Explainable AI*, which involves the concept intelligent agents not only performing but also *explaining* their actions in real-time during the execution of a behavior.

FAMTILE is applicable in this space because it provides the ability to learn the intelligence for an agent that is aware of its current contextual state. In practice, the agent can be used to execute that learned knowledge while explaining its context sequence (and motivations for selecting each particular context) to a third party in real-time.

For instance, consider the situation where FAMTILE is tasked to learn the behavior of a world-champion poker player. That knowledge is then used to generate a CxBR model that imitates the behavior of that player. Within simulation, the CxBR model could then be used as a teaching (or pedagogical) agent to aid novice players and to help them better understand the game. The player could 'sit beside' the agent and view his cards, while the agent detailed the current context he was in. The agent could also provide with a description of that context, and detail the conditions that triggered the selection of the current context it is in.

By contrast, the agent could be used to identify and correct differences in the play of a novice versus that of an expert. By training one FAMTILE system by observing a novice while training a second system using an expert, the systems could then be compared offline (and without the experts) to compare situations where the systems differ in their context selection. Those differences can then be used by the novice to identify the type of situations where he/she needs to improve his game.

7.4 Complexity and Scalability Analysis

To obtain a measure of the complexity of FAMTILE, we consider the complexities of both the TBI and FAM components that make up its primary functionality, along with the procedure required to convert an observation into patterns for both. Regarding TBI, Drewes [?] notes that, with N is the number of context templates and an average number of A attributes, the number of comparisons needed to infer a context for a single training pattern would be on the order of $N * A$, or O{N*A}. Assuming a finite number of attributes per template, therefore, this complexity reduces to O{N}.

For Fuzzy ARTMAP, multiple phases of the network must be considered. First, consider the complexity required to train the network with a single pattern. This complexity will be based on the number of bottom-up computations required for a single training pattern, which depends on (a) the number of fields present within the pattern and (b) the number of Computing the bottom-up weights

[GC01] to a single cluster within FAM in the input pattern implies a complexity of $O\{N_p\}$, where N_p refers to the number of fields in the input training pattern. Each cluster, therefore, will map to a certain output pattern templates in ART_B, and there is a many-to-one mapping between the number of clusters and the number of output pattern templates. In FAMTILE, each output pattern template corresponds to a unique context, and so we can refer to this number as N_c, the number of contexts present within the scenario used for training. The number of bottom-up weight comparisons, therefore, will be $(N_c + 1) * L$, where L is the average number of clusters that each map to a particular context ($N_c + 1$ is used to account for computing the bottom-up inputs to the uncommitted node). The complexity of FAM in training mode, therefore, is $O\{N_p * ((N_c + 1) * L)\}$, which reduces to $O\{N^2\}$. A similar calculation can be performed for FAM in testing mode.

The conversion of an observation into pattern usable by both TBI and FAMTILE are both linear operations performed on each field within the operation. The complexity of these conversions, therefore, reduces to $O\{N\} + O\{N\} \rightarrow 2O\{N\} \rightarrow O\{N\}$

For each observation pattern, one conversion must be performed along with one context inference and one Fuzzy ARTMAP training operation. The complexity of the FAMTILE algorithm, therefore, reduces to the complexity of the highest-ordered operation (FAM). In other words, $O\{N\} + O\{N^2\} + O\{N\} \rightarrow O\{N^2\}$.

This level of complexity, however, assumes scalable values for *both* the number of contexts required and the number of fields required to represent each observation. If either of these are held constant, this complexity then reduces to a more desirable $O\{N\}$.

7.5 Proposed Topics for Future Research

In this section, a list of future research topics is presented that would provide more perspective on the assumptions made for the construction, validation, and evaluation of FAMTILE.

- Introduction of the time variable into the observation sequence and context-transition identification process. In each training scenario for this research, all decision cues were turn-based, providing a clear identification of what observation induced what context-transition.
- Incorporation of the work by Gerber ([Ger01]) to create more robust context templates for use by FAMTILE
- Application of FAMTILE as a training tool
- Development of a new testing scenario where low-level behaviors are more easily defined and identified
- New training sessions of vignette D where the simulation intentionally places experts in situations to make more decisions after the flop. This could involve the development of a new vignette similar to C, where behavior is based on individual decision points. However, those decision points could be translated to points in a variety of situations occurring after the pre-flop round of betting.
- Incorporation of scenarios without forced transitions. For the poker scenarios used for context identification, transitions from one context to another were implied by the decision cues. Scenarios can certainly exist where low-level contexts 'remain active' at each decision point. The FAMTILE system should be expanded to represent logic for no context change.

- Expert familiarity with the context structure played a significant role in determining the predictive accuracy of FAMTILE. A modification of the procedure used in this algorithm could be performed where the context template definitions emerged from some sort of knowledge acquisition session with the expert. A comparison could then be done that evaluates FAMTILE with the more 'informed' templates versus the system using pre-defined templates developed independent of the expert.

- Additional training scenarios can be developed for FAMTILE where the defined contexts are centered more around the atomic action of the expert and not as much of the situation that surrounds it. This situation may allow for less of a reliability of the context templates to how closely they represent the expert's reasoning

- Fuzzy ARTMAP research can be done to explore ways to modify the algorithm to better suit the objectives of FAMTILE. This may include emphasis on pattern fields observed to have more important roles on determining context

- Use of the FAMTILE algorithm to learn high-level behavior in scenarios involving more complex low-level behavior. Here, extracted rules from the system can be used within a Norlander CxBR model architecture [Nor99] and evaluated in that mode

- The implementation of the full Loki model as computer-generated forces for the poker simulation

- The addition of psychological observations and player tendencies as observations within vignettes C and D

APPENDIX A

GLOSSARY OF TERMS

a priori Beforehand; prior to

Action In Poker, an action is a move made by a player - either a check, a bet, a raise, a call, or a fold.

Agent Any program or system that operates within a real or simulated environment

Agent Interface In CxBR, the module that connects the logic of the model with the methods of the agent to execute actions within some environment

Autonomous Agent Any program or system operating autonomously within a real or simulated environment

Big Blind A forced bet made by the player sitting two seats left of the dealer.

Bluff When a player represents a good hand by betting and raising when, in fact, he has a weak or mediocre hand

Board The name for the community cards that have been dealt on in a game of Texas Hold'em

Bot A computer-generated player/opponent

c-schema A frame-like abstraction that contains several pieces that define the parameters for entering the context behavioral knowledge to employ when a certain context is active

Call An action in poker where the amount of a given bet is matched by another player when it is his turn

Check Equivalent to a bet of 0 chips

Chip-count In poker, the amount of chips held by a player

Cluster In a Fuzzy ARTMAP neural network, a cluster is formed within both the ART_a and ART_b modules grouping similar input and output patterns with identical mappings

Clustering Ratio The average number of patterns included in an ART cluster during training

Community Cards In Poker, cards dealt face-up that are common to the hands of each player

Connectionist Learning A learning method that uses classified historical examples to establish the values of weights in an artificial neural network

Context A set of environmental and physical conditions that may suggest a specific behavior or action

Context-Based Reasoning A behavior modeling paradigm motivated by the idea that experts use only a fraction of their knowledge, at any given time, based on the context of his current situation

Context Moderator An abstract operator that has the ability to either affect decision-making after an active context has been selected (a functional moderator) or to affect the context-transition logic itself (a context-transition moderator)

Context Topology The term for the set of contexts, along with the set of possible transitions, that exists for a certain CxBR mission

Context-Transition Logic Any logic that defines parameters for switching the active context

Dealer Button In poker, the dealer button (or button) is a small white disc that rotates around the table and identifies the position of the dealer, who acts last in all rounds of Texas Hold'em

Drawn out on In poker, a winning hand is drawn out on if a card comes that makes an opponent's hand superior

Default Context In CxBR, the context that the agent will operate in (or *active context* at the beginning of the scenario

Elaboration Rule In Soar, rules that update the Soar agent's situational awareness by editing working memory with new information

Explanation-Based Learning A learning method where input/output exemplars are provided along with a sort of explanation that can be used to better learn the mapping

Flop In Texas Hold'em, the first three community cards dealt face-up at the same time after the first betting round

Flush Draw When a player holds four cards of one suit, meaning that one more card of that suit would make a flush

Fold When a player decides not to match the bet amount, forfeiting his hand for the round

Forced Bet Also called a blind, a forced bet is a bet made by the two players to the left of the dealer that ensure that there is money in the pot

Fuzzy Bit For Fuzzy ARTMAP, a value between 0 and 1.

Granularity The level of strictness applied by a Fuzzy ARTMAP cluster for accepting new training patterns. Highly strict FAM clusters are said to be of fine granularity, loosely strict clusters have a coarse granularity

High-Level Behavior For this research, a behavior that involves executing a sequence of identifiable, lower-level behaviors or actions

Hold'em See *Texas Hold'em*

Hole Cards In Texas Hold'em, the two cards dealt face-down to each player at the table

Inductive Learning A learning method that uses classified historical examples to develop an induction tree from which rules can be derived

Inside Straight Draw In Poker, when a player holds four cards to a straight but only one rank will make him a straight (i.e. 6-7-8-10)

Learning by Instruction A learning technique where the knowledge is provided directly from an expert

Learning from Observation The use of data acquired, through the act of observation, to assert knowledge from which an expert's behavior can be intimated

Limit Texas Hold'em A type of Texas Hold'em where all bet sizes and increments are fixed

Loose Call A call made by a player who holds a mediocre or poor hand and draw.

Low-Level Behavior For this research, a behavior that is identifiable and used in conjunction with other low-level behaviors to constitute a learnable high-level behavior

Mission In CxBR, the mission consists of a goal, a context topology, and a set of constraints. Missions are assigned to CxBR models prior to execution-time

Model a construct that defines the behavior of a some autonomous agent executing some behavior

Multiway Hand A hand that, if made, will likely make a winning hand regardless of the number of players in the hand.

Neighbor Merging In a HMM, states that are connected by a transition and share a common label are merged into one state with a self-transition loop

On a Draw When a player holds four cards to a good hand with more cards to come

On the Come See *On a Draw*

Outside Straight Draw When a player holds four consecutive cards such that cards of two ranks will make a straight (i.e. 4-5-6-7, a 3 or an 8 will make the player a straight)

Operator Structures within Soar models that are responsible for allowing the agent to react and make actions either directly or indirectly in response to his environment

Operator Application Rule In a Soar model, a rule that executes the functionality of a selected operator

Operator Proposal Rule A Soar rule that allows the agent to select from (or set preference values to) a list of possible operators

Pot Represents all the chips bet during a particular hand that are awarded to the winners of that hand

Predictive accuracy For Fuzzy ARTMAP, the number of correct predictions made out of the total number of predictions

Proceduralized Context From Brézillon [BS97], a part of context knowledge that is invoked, structured and situated according to a certain scenario-specific focus

Q-Learning A reinforcement learning method intended for domains that can be modeled using a Markov model

Raise In Poker, a raise is when a player increases the amount of the current bet

River In Texas Hold'em, the river is the 5^{th} and

Reading In Poker, when a player can assess the strength of an opponent's hand by observing and identifying patterns in their betting, their mannerisms, or personality while at the table

Rounds In Poker, betting is done by players in rounds, usually after a card or set of cards are dealt. In a round of betting, each player makes an action - either a check, a bet, a fold, a raise, or a call. A round ends when every player has either folded or called the final bet or raise made final community card dealt

Semi-Bluff A semi-bluff is a type of bluff where the player has a weak hand but is on the come to a stronger hand

Sentinel Rule In CxBR models, a rule that defines the conditions for a certain context transition

Showdown In poker, when all players show their hand, with the player with the best hand winning the pot

Slowplay In Poker, when a player feigns weakness in the opening rounds of betting with a superior hand, in an attempt to trap their opponents in later rounds

Small Blind In Poker, a forced bet made by the player immediately to the left of the dealer button

Soar A Rule-based cognitive architecture for developing intelligent models and systems

Stone Cold nuts In poker, a hand that cannot be beaten no matter what cards are drawn by your opponent

Sub-Context Context-like structures that encompass a small functional section of a context not directly critical to the mission objectives

Substate In Soar, substates decompose the action/behavior space into goal-defined structures from which the agent can execute more specific operators relevant to that goal

Suit In Poker, each card in the deack is assigned one of four suits - hearts, spades, diamonds, and clubs

Supervised Learning a technique by which the learning system is controlled, in terms of what it learns, by an outside party or system

Template A list of attributes, each of varying relevance, that denote traits of a certain context for a Template-Based Interpretation engine evaluated by a TBI engine to determine

Texas Hold'em A variety of 5-card Poker. In Texas Hold'em, players use two hole cards along with 5 community cards to make their best hand

Transition In CxBR, a transition is a switch from one active context to another

Turn In Texas Hold'em, the fourth community card

Unsupervised Learning A form of learning where the system must decipher its own input-output mapping, which is not presented to it by a third party during learning

V-Merging In a HMM, V-Merging merges states that share a transition either to or from the same state, and also share the same label

APPENDIX B

GLOSSARY OF ACRONYMS

ACT-R	Adaptive Control of Thought - Rational
ADF	Automatically Defined Function
AI	Artificial Intelligence
ART	Adaptive Resonance Theory
CGO	Computer-Generated Object
CHMM	Coupled Hidden Markov Model
CI	Confidence Interval
CMB	Context Mediated Behavior
CVQ	Continuous Valued Q-learning
CxBR	Context-Based Reasoning
EBL	Explanation-based Learning
FAM	Fuzzy ARTMAP
FAMTILE	Fuzzy ARTMAP / Template-Based Interpretation Learning Engine
GA	Genetic Algorithm
GP	Genetic Programming
HMM	Hidden Markov Model
ILP	Inductive Logic Programming
KA	Knowledge Acquisition
KE	Knowledge Engineer

LFO Learning from Observation

MDP Markov Decision Process

MDM Markov Dynamic Model

MEBN Multi-Entity Bayesian Network

MODSAF Modular Semi-Automated Forces

NN Neural Networks

POMDP Partially Observable Markov Decision Process

RPDM Recognition-Primed Decision Making

SME Subject-Matter Expert

TBI Template-Based Interpretation

WMD Weapons of Mass Destruction

WSE Weighted Sum of Entropies

APPENDIX C

POKER AND TEXAS HOLD'EM

Poker is a class of card-games where players use the rank and suit of their cards, and attempt to make the strongest hand. Poker cards hold ranks that determine their value. The Ace holds the highest rank, followed by the King, the Queen, the Jack, and then ranks 10 down to 2. Each card in the 52-card deck also holds one of 4 *suits* - clubs, diamonds, hearts, or spades.

In a poker game, players are dealt cards and attempt to make a *hand* that is stronger than that of each of the other players. Poker hands are named, and ranked based on the following criteria. Each hand is listed below in descending order of strength.

Royal Flush: A Royal Flush is the 10, Jack, Queen, King, and Ace where each card is of the same suit. If no wild-card exists in the game, which is the case in Texas Hold'em, a Royal Flush is the best possible hand.

Figure C.1: A Royal Flush

Straight Flush: A Straight Flush is cards of consecutive rank where each card is of the same suit. The highest card present in the hand distinguishes the relative strength of the straight flush - i.e. a straight flush to the 9 outranks one that runs to the 6.

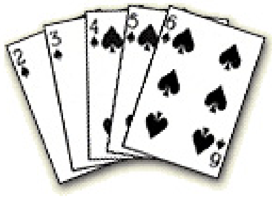


Figure C.2: A Straight Flush

Four of a Kind: Four of a Kind is a hand where four cards have the same rank. As with the other hands, the rank of these four cards determine the relative strength.

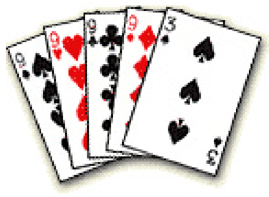


Figure C.3: Four of a Kind

Full House: A Full House, or 'Full Boat' consists of five cards where three cards are of one rank and two cards are of a second rank. The highest rank of the 'three-of-a-kind' determines the relative strength of the Full House. For example, a King-King-King-Seven-Seven outranks a Jack-Jack-Jack-Seven-Seven.

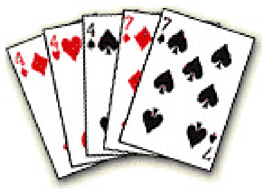


Figure C.4: A Full House

Flush: A Flush consists of any five cards of the same suit. The highest card in this hand determines the relative strength of the flush against an opposing flush.

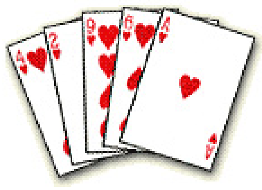


Figure C.5: A Flush

Straight: A Straight consists of five cards with consecutive rank. The highest-ranking card determines the relative strength of the straight.

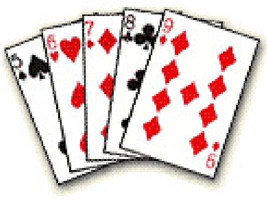


Figure C.6: A Straight

Three of a Kind: Three of a Kind, or 'Trips', is any five-card hand where three cards out of the hand are of the same rank. The relative strength of such a hand is determined by the rank of the trips.

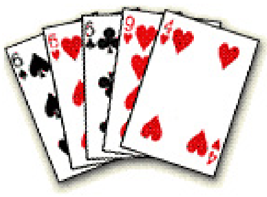


Figure C.7: Three of a Kind

Two Pair: Two Pair is a five-card hand with two distinct 'pairs', or two cards of the same rank. The hand with the highest pair determines the relative strength of the two pair. This hand is also referred to as 'X's up' where X is the rank of the highest pair in the hand.

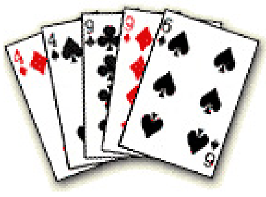


Figure C.8: Two Pair

Pair: A Pair is a five-card hand with two cards of the same rank.

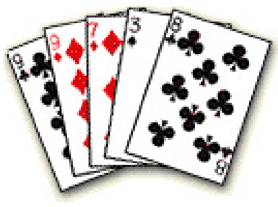


Figure C.9: A Pair

High Card: High Card is defined by the highest-ranking card in the two-card hand. This is also referred to as 'X-High', where X is the highest-ranking card.

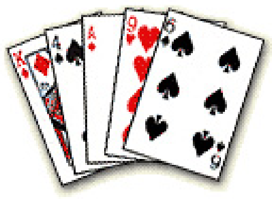


Figure C.10: High Card - Ace High

The feature that makes Poker such a popular game is the concept of betting. Depending on the game, each hand of Poker consists of *rounds* of betting. In a betting round, players can make a bet by placing a certain amount of chips into a *pot*. All the other players must either match this bet (referred to as *calling*) or must fold their hand. In addition, players have the option to *raise* the bet, forcing all other players to increase the size of their call to stay in the hand. At the end of the hand, all players who have not folded must show their hand to the table. The player with the best hand is declared the winner and takes all of the chips in the pot for that hand.

C.1 Texas Hold'em Poker

Texas Hold'em is a variation of 5-card poker that has become enormously popular over the past few years. Often referred to as 'The Cadillac of Poker', this Poker variation is a very easy game to learn. However, as with most well-designed games - it is nearly an impossible game to master. The main event at the 'World Series of Poker', in fact, features a variation of Texas Hold'em.

To begin a Texas Hold'em hand, players are first dealt two cards face down, and a round of betting ensues. These cards are dealt in a clockwise direction starting to the left of the player holding the *dealer button* or *button* - a chip that rotates clockwise around the table after each hand. The player immediately to the left of the dealer posts an automatic or *forced bet* known as the *small blind*. The player to the left of the small-blind bet posts another forced bet known as the *big blind*, whose value is twice that of the small blind. Players then act clockwise around the table (starting with the player to the left of the big blind), and have the option to either call the big-blind bet, make a raise, or fold their hand. In *Limit Hold'em*, which is the variation of Texas Hold'em used for these poker training scenarios, all bets and raises must be of a fixed amount. When the *action* (player's turn to act on their hand) comes back around, the player on the small-blind is only required to post half the amount of the big blind to call (unless the pot has been raised), and has the option to raise. The player on the big-blind does not have to post any amount in order to call (again assuming no raise) and also has the option to raise.

After this round of betting, three *community cards* are dealt face up in the center of the board. These cards are referred to as community cards because each player uses the 5-card combination of the community cards and his two down-cards that make his best possible hand. A round of betting follows the presentation of these three cards, known as the *flop*, and the action begins with the first player to the left of the *dealer button* who did not fold in the previous betting round. The dealer button is a white disc that rotates clockwise around the table after each hand, indicating the order in which the players must act. When this betting round finishes, a fourth card (the *turn* card) is placed on the board and a third round of betting then takes place. In this round of betting, the amount of the fixed-bet amount is doubled (now equal to twice the amount of the big blind). All bets and raises are now equal to this amount for the remainder of the hand. Finally a fifth card (the *river* card) is presented to the group of community cards. After a round of betting, all players who have not folded proceed to show their two down-cards, indicating their best possible hand. This is called the *showdown*. The player who can make the best possible hand with his two down-cards is declared the winner and takes the pot (all the chips bet by the players during the betting rounds).

APPENDIX D

POKER EXPERT QUESTIONNAIRE AND RESPONSES

1. How long have you been playing Poker?
2. How long have you been playing Poker seriously/professionally?
3. How often do you play poker?
4. Do you consider yourself to be a tight or loose player? Explain.
5. Do you consider yourself to be a passive or aggressive player? Explain.
6. Do you read literature on the game of Poker? If so, how does it affect the strategies you employ during a game?

Please answer the following questions as they pertain to ***Limit Texas Hold'em*** against an intermediate-skilled opponents:

1. List all conditions where you would consider a *check* to be a legitimate action. Give card-by-card examples for each.
2. List all conditions where you would consider a *bet* to be a legitimate action. Give card-by-card examples for each.
3. List all conditions where you would consider a *call* to be a legitimate action. Give card-by-card examples for each.
4. List all conditions where you would consider a *raise* to be a legitimate action. Give card-by-card examples for each.
5. List all conditions where you would consider a *fold* to be a legitimate action. Give card-by-card examples for each.
6. In the cases where multiple actions are appropriate, what other factors do you consider?

D.1 Questionnaire Response #1

1. 3 years
2. 0 years
3. weekly (sometimes)
4. varies
5. aggressive
6. Yes. It affects them greatly
7. (none)
8. (none)
9. (none)
10. (none)
11. (none)
12. I don't have a set of tactics that I apply to every given situation. I play the people in the hand at that time. There are certainly some heuristics to follow, based on good game mechanics.

D.2 Questionnaire Response #2

note: this player was Expert #2 in Scenarios #3, #4, #5, #6 and #7

1. 9 months
2. N/A
3. 1/week
4. tight
5. passive
6. no
7. when I am first to act with an okay hand
8. when my cards are good, or when slowplaying a great to unbeatable hand
9. when I think I have a hand that could beat the other players
10. AA, when I am first to act with a good to unbeatable hand or last to act with weak players
11. 72, when I don't think I could win
12. number of people playing, who's playing (aggressive versus passive players), money in pot, money in my hand

D.3 Questionnaire Response #3

note: this player was Expert #3 in Scenarios #5, #6 and #7

1. 4 years
2. 2 years
3. bi-weekly
4. tight player, don't tend to stay in on bad hands, don't play on-tilt, etc.
5. aggressive, in that strong hands are raised aggressively
6. yes (Sklansky), but that predominantly shaped my game early on. Now past readings primarily help me recognize and classify the way other people play
7. check with the plan to check-raise [strong hands]; with a weak hand that I would fold but when checked to me.
8. on just about any hand worth staying in on; very frequently in late positions.
9. semi-bluff scenario, drawing hands, occasional deceptive play for table image
10. strong hands; medium and strong hands in late positions if in late position and called to me.
11. on all weak hands, except occasionally; when I deem that I've been beaten by somebody else, based on their betting actions; when raised to me by an infrequent bluffer when I have a medium hand, or I have a hand not likely to be the strongest

12. nature of the table–loose vs tight; skill of the players (e.g., will they call no matter what I do? Are they skilled enough to interpret my own action as intended and to take the appropriate action?) Do I have a drawing hand? [Is what I'm drawing to the same as what everyone else who stayed in might be drawing to?] Would it be beneficial to advertise a bit, perhaps staying in on a hand, or showing that I bluff when the end cost would be minimal?

D.4 Questionnaire Response #4

1. In general for fun since I was 7 or 8 years old, but more this past year for money.
2. Maybe the last few months I have played for money.
3. A few times a week
4. Both Depending on how much money I have compared to other players at the table.
5. Again depends on how I am doing compared to others at the table.
6. No, I learn sometimes from word of mouth, or watching different strategies on TV.
7. If you have put in for the big blind and everyone else has just called, then obviously I would check. If you have nothing going head to head another opponent hoping you have high card. If you do not have a strong hand and everyone else has checked.
8. Any time you feel you have the strongest hand. If everyone else has checked and you are the last one with action. If you are given a strong pocket hand, or if you hit the hand anytime for example a flush, straight or full house.
9. If you know you have the best hand and you are suckering the other person to keep betting. Or if you think the other player may be bluffing.
10. Any time you know you have the best hand. Pre flop if you are given a good pocket. Big card turned over if you want to try and bluff to show that you might have a good hand. For example hitting a flush, or straight.

11. To high of a bet by another player. Have nothing on the flop. Too many people are already in the hand with playable cards.

12. (none)

D.5 Questionnaire Response #5

1. 18 months
2. never
3. live 1.2 days a week, online 4 days a week
4. (tight or loose) heavly depends on the game and situation, players etc..
5. aggressive - same as above
6. yes - ?
7. trap - after flop, very strong hand, draw - my hand needs help and I can get it for free blinds - 7/2 in the BB
8. bluff - when the flop comes and I see weakness, good hand - at any point see what I'm up against - at any point
9. same as number 1
10. bluff - at any time, nuts - at any time when I think I'm winning, find out information when I'm setting up a table image - advertising
11. when I feel I have the worst hand and low pot odds
12. Stack size, position, opponents, table image

D.6 Questionnaire Response #6

note: this player was Expert #1 in Scenarios #3, #4, #5, #6 and #7

1. I have been playing for four years.

2. I have been playing seriously for two years.

3. I play poker four to six times a week.

4. (tight or loose) Succinctly, I am a very tight tournament player but in a ring game I will adjust my tightness to counter that of my opponents. The general image I project is that of a player that loosens up as a session progresses, but in reality I utilize the mood of the table to my advantage.

5. I am a passive player with less than the best hand but aggressive with the best or second best depending on the number and ability of my opponents.

6. I have read many of the David Sklansky books on Hold'em, and tournament play. Upon review of the books, I find myself helped against skilled opponents, but hurt against weaker opponents due my lack of adaptation to their bad play. Experience and further reading should eliminate this weakness.

7. A check is a legitimate action when:

   a. You are unsure about the strength of your hand and sure your opponent will call your bet (instead of folding)... An example of this is having A8o with a flop of A23. If there were no preflop raises, you could face a straight or someone with A9-AQ may have seen it fit to merely call preflop.

b. You are fairly certain someone acting after you will bet. (Waiting for information about other hands).. Given a small pocket pair in first position, it is advisable to check in order to see how many others enter the pot and thus determine the appropriate action.

c. Your hand is so strong that giving opponents free cards is unlikely to hurt you, but betting will drive others out the pot...

Given a wired pair of aces and a flop of AT2(o), the set of aces should check in the early seats in order to let other players enter the pot. It is almost certain that the other ace and several other hands will call other bets, but if you started the betting preflop, a betting the flop from early position is almost certain to drive other players out of the hand and consequently lower the profit to be made. If the turn happens to complete a straight for someone holding KQ, QJ, QT, or other cards, a decision can be made to lay the hand down or continue since your outs to a full house have increased.

d. You have top pair with a weak kicker and want to see how other people react to the flop... If a player has A2(o) and flops a pair of aces, there is little incentive to bet in any seat but the blinds due to the fact that anyone that paired their ace can only tie with the player with A2 in order for that player to have a chance at winning. If the second player is not tied with the first, they are winning and thus, in any situation where a player has weak kicker, and the player is in early position, a check must be made in order to gauge their opponents.

8. Flop: If a player is in one of the early positions and there have been no bets, it may be correct to bet in order to knock out those who missed the flop entirely. An example of this would be if the first player to act is holding JdJs with a flop of 2s7c9d. If in late position and there have been no bets, a player may bet to see if there were any players slowplaying their hands.

   Turn: A bet on the turn is appropriate from any position if the player has been slowplaying a strong hand, needs more leverage to knock out players with weak hands that didn't improve or to put in a semi bluff bet. An example of the first situation is having AA preflop when the flop is A29(o). At that point, the player with AA has the best hand possible and should check. Once the turn presents itself, a player in early position with a set of aces should check, but a bet is also legitimate. In late position they should bet their hand if it has been checked to them.

9. In any situation where a player has a draw to a greater hand and has the pot odds to call, the call should be made. For example, if a player has A9(d) and the flop is AsT94d, the player in early position should call all bets in order to encourage others to enter the hand with weaker draws. If the same number of people are in the pot on the turn, unless the board is paired, a call is warranted. On the river, if not improved and the board is not paired, the player should call if they believe there will only be one bet to call.

10. A raise is used to knock out players or to build a pot. The closer you are to the early positions, the more effective a raise can be at eliminating opponents. In later positions, a raise forces other players that have already called to put more money in the pot, thus giving players in earlier positions

the incentive to call. Traditionally, if a player has a low pocket pair (22-JJ) they should raise in early position to reduce the competition. Players in middle position should raise preflop with high pocket pairs (AA-QQ), AK(s)-AJ(s), KQ(s)-KJ(s) and in loose games, AK-AT in order to eliminate the later players. On the flop, if a player in early position has the highest set possible or a draw to the best straight possible or the best flush possible, the player should raise. If the player in late position with similar draws, a raise can be done to slow down betting on later streets.

11. A fold is legitimate from any position when a player has little or no chance to win. For example, a player with a hand 9s9d should fold if the flop is Ac2cKc because a flush, two overpairs and two straight draws are present. Even if the player gets a nine on the turn, the player is still hoping for another nine or for the board to pair. Unless the player gets another nine, the paired board gives the person with a set of kings or aces and thus a better full houses than that of the player with a set of nines. In general, if a player is drawing to a hand that is second or third best, the player must, at the very least, consider folding. Examples of this are drawing to a flush with a paired board or drawing to the smaller end of a straight when larger straights are possible and probable. An example of this is a player with 9T with a flop of JQA. A king will give the player a straight to the king, but any player with a ten will have the best straight possible. If the straight is made and the player is raised after betting their straight, a fold is acceptable.

12. In cases where multiple factors are appropriate, first the player must recall the categories or groups of hands another player would play in their respective positions. Once determined, the player must calculate how strong their

current hand or draw is in opposition to the various hands hypothesized. If the current hand or draw is layed proper odds by the pot and the number of possible winning hands of the opposition is less than the number of losing hands, the player must continue to play. In short, pot size, hand history, pattern of betting (table position with respect to hands and frequency of bets with types of hands), table position (the player) and overall mood of the other players must be considered in order to be successful.

APPENDIX E

TESTING DATA

E.1 Scenario #1 Results

Table E.1: Results for Scenario #1: Average Number Correct of 100 Testing Patterns

$\bar{\rho_a}$	$\bar{\rho_a}(test)$	Subject 1	Subject 2	Subject 3
0.1	0	93.2	84.2	77.4
0.1	0.1	95.2	81.6	75.8
0.2	0	96.6	85.8	78.4
0.2	0.2	93.8	83.2	79
0.3	0	93.6	86.2	77.8
0.3	0.3	95.6	81.8	75.8
0.4	0	94.4	82.8	78.6
0.4	0.4	94.8	83.8	78.4
0.5	0	94.2	86.2	76.4
0.5	0.5	93.2	84.4	78.4
0.6	0	**95.8**	84	77.2
0.6	0.6	94.4	86.4	76.4
0.7	0	**95.8**	87.2	78.6
0.7	0.7	92.4	86.8	79.8
0.8	0	94.8	90	**82.4**
0.8	0.8	92.2	86.8	81.2
0.9	0	94.8	**88.2**	80.4
0.9	0.9	92	85.8	81.8

Table E.2: Summarized Results for Scenario #1: 1000 Runs for each subject, Using $\bar{\rho_a}$ and $\bar{\rho}_{atest}$ Values that Yielded Best Accuracy (see Table 6.3)

	Number of Runs	$\bar{\rho_a}$	$\bar{\rho}_{atest}$	$\bar{\mu}$	$\bar{\sigma}$	99%CI	p-value
Subject 1	1000	0.6	0	94.7	2.38	(94.5524,94.9416)	1.00
Subject 2	1000	0.8	0	87.3	3.27	(87.055,87.589)	1.00
Subject 3	1000	0.8	0	80.6	3.76	(80.336,80.950)	1.00

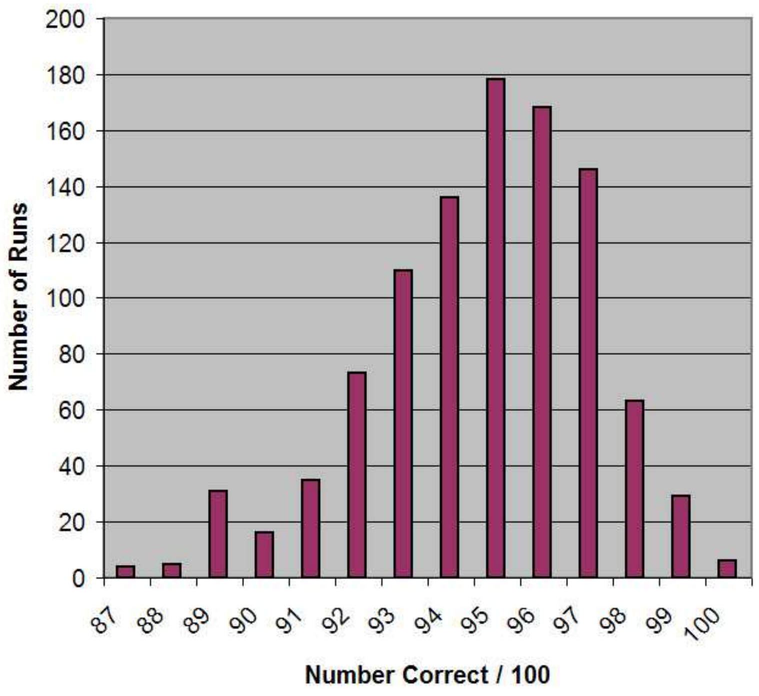


Figure E.1: Scenario 1 Results: Frequency of Correct Predictions over 1000 Runs for Subject 1

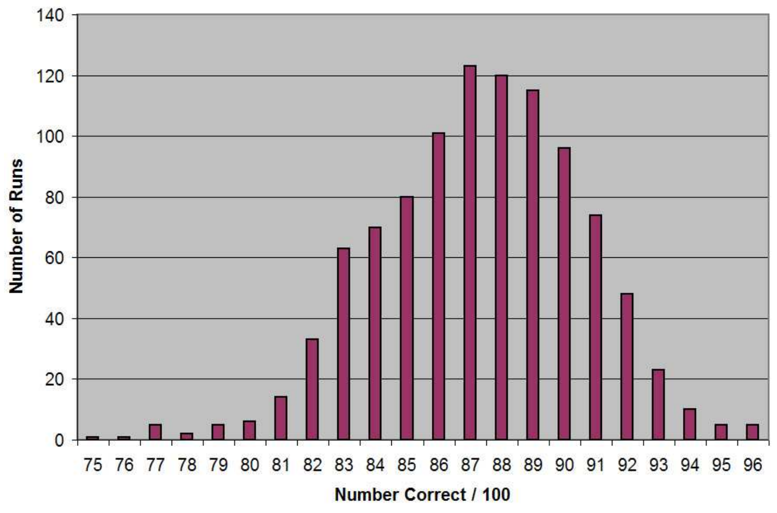


Figure E.2: Scenario 1 Results: Frequency of Correct Predictions over 1000 Runs for Subject 2

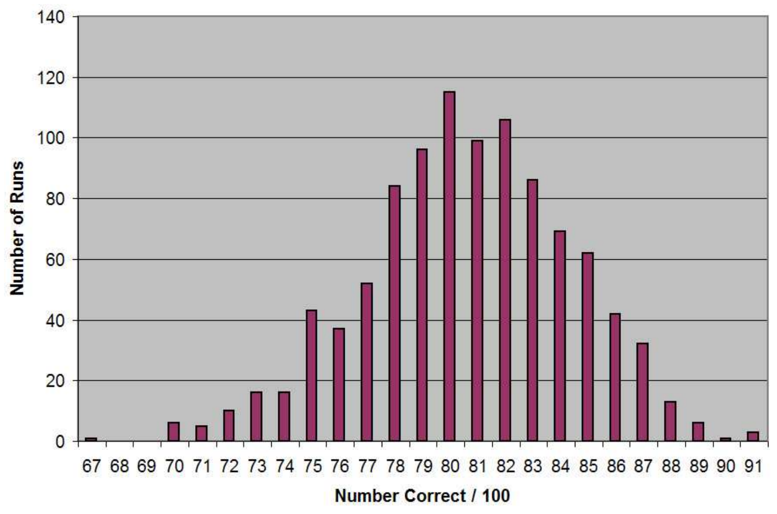


Figure E.3: Scenario 1 Results: Frequency of Correct Predictions over 1000 Runs for Subject 3

E.2 Scenario #2 Results

Table E.3: Results for Scenario #2: Average Number Correct of 90 testing patterns

$\bar{\rho_a}$	$\bar{\rho_a}(test)$	Subject 1	Subject 2	Subject 3
0.1	0	90.8	85.2	82.2
0.1	0.1	85.6	84.4	82.2
0.2	0	91.0	81.6	81.0
0.2	0.2	88.0	81.8	84.6
0.3	0	89.2	82.0	82.0
0.3	0.3	90.8	82.0	82.0
0.4	0	90.6	80.4	84.8
0.4	0.4	91.2	82.4	86.4
0.5	0	**93.6**	84.8	83.4
0.5	0.5	89.2	84.6	86.6
0.6	0	91.0	85.2	84.4
0.6	0.6	91.4	79.4	84.6
0.7	0	93.2	82.4	86.6
0.7	0.7	91.6	81.8	**88.4**
0.8	0	**93.6**	**85.8**	85.6
0.8	0.8	91.0	84.6	85.2
0.9	0	93.4	82.2	87.8
0.9	0.9	91.4	81.8	80.2

Table E.4: Summarized Results for Scenario #2: 1000 Runs for each subject, Using $\bar{\rho_a}$ Values that Yielded Best Accuracy (see Table 6.7)

	Number of Runs	$\bar{\rho_a}$	$\bar{\rho}_{atest}$	$\bar{\mu}$	$\bar{\sigma}$	99%CI	p-value
Subject 1	1000	0.8	0	92.5	2.63	(92.3074,92.7366)	1.00
Subject 2	1000	0.8	0	84.5	3.42	(84.181,84.739)	1.00
Subject 3	1000	0.7	0	85.6	3.31	(85.308,85.848)	1.00

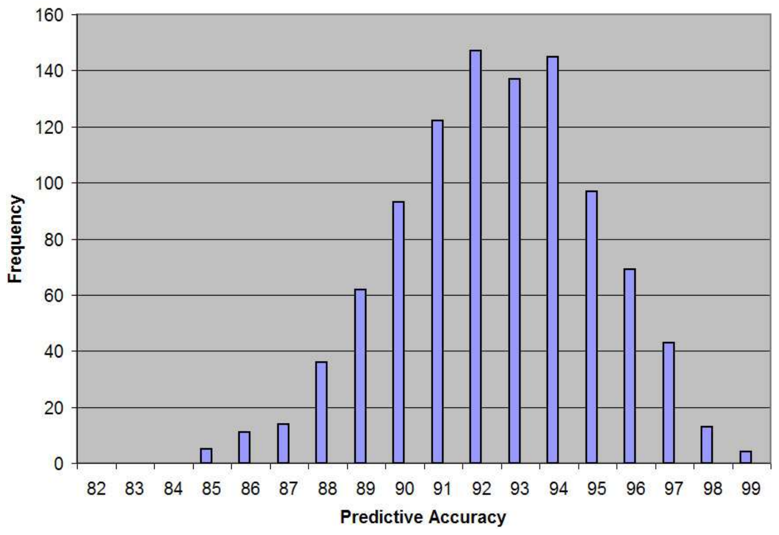


Figure E.4: Scenario 2 Results: Frequency of Correct Predictions over 1000 Runs for Subject 1

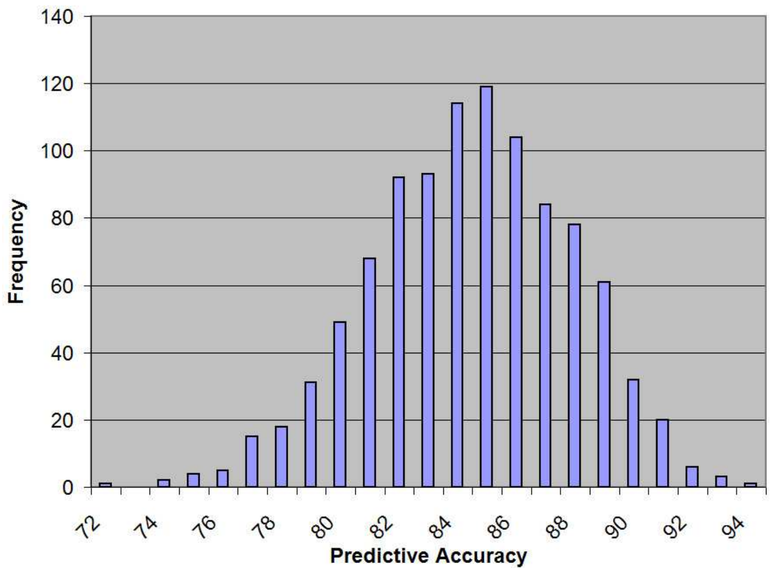


Figure E.5: Scenario 2 Results: Frequency of Correct Predictions over 1000 Runs for Subject 2

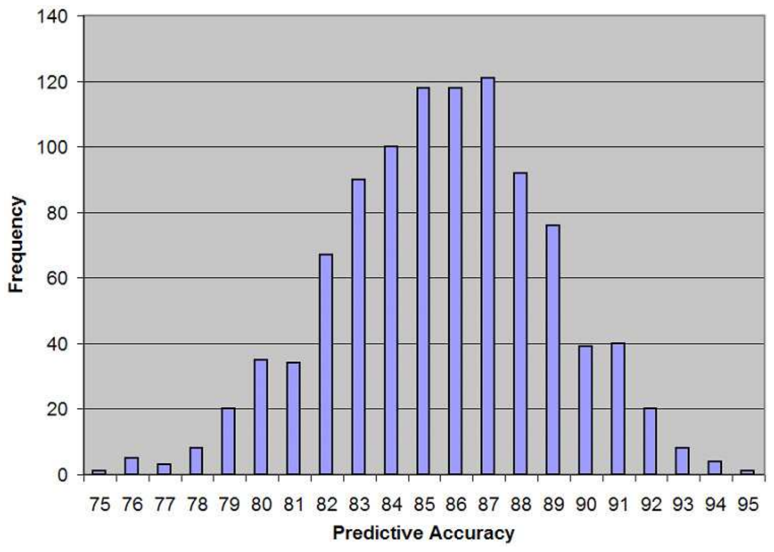


Figure E.6: Scenario 2 Results: Frequency of Correct Predictions over 1000 Runs for Subject 3

E.3 Scenario #3 Results

Table E.5: Scenario #3: Average Predictive Accuracies of Subject *Actions* for 100-run sets of FAM, 300 Training Points

$\bar{\rho_a}$	Subject 1	Subject 2	Subject 3
0.05	66.98	59.91	66.67
0.10	66.08	60.68	65.78
0.15	66.62	60.64	67.39
0.20	66.16	60.06	66.74
0.25	65.97	60.25	66.47
0.30	66.48	60.90	66.25
0.35	67.22	60.64	66.25
0.40	67.97	59.40	65.69
0.45	66.89	60.48	66.16
0.50	66.58	60.76	66.74
0.55	65.95	59.47	66.22
0.60	67.50	60.14	65.86
0.65	67.22	60.20	66.91
0.70	67.67	60.25	68.24
0.75	68.58	61.38	69.12
0.80	70.78	61.77	69.75
0.85	71.46	64.13	71.36
0.90	**74.00**	66.28	**72.40**
0.95	73.47	**66.64**	72.11

Table E.6: Scenario #3: Average Predictive Accuracies of Subject *Actions* for 100-run sets of FAM, 600 Training Points

$\bar{\rho_a}$	Subject 1	Subject 2	Subject 3
0.05	67.93	61.34	67.51
0.10	69.03	61.08	67.65
0.15	67.79	60.03	67.47
0.20	67.80	61.90	67.95
0.25	68.16	62.06	66.92
0.30	67.69	61.37	67.69
0.35	68.38	61.30	66.40
0.40	68.38	61.89	66.87
0.45	67.92	61.96	68.07
0.50	68.21	60.78	67.20
0.55	69.11	61.92	67.04
0.60	68.39	60.71	67.81
0.65	68.39	62.36	67.28
0.70	69.10	61.10	68.28
0.75	70.63	61.57	69.53
0.80	70.69	62.88	70.35
0.85	72.05	64.80	72.58
0.90	74.94	66.29	74.24
0.95	**75.62**	**67.18**	**74.30**

Table E.7: Scenario #3: Average Predictive Accuracies of Subject *Actions* for 100-run sets of FAM, 900 Training Points

$\bar{\rho_a}$	Subject 1	Subject 2	Subject 3
0.05	69.59	61.65	67.52
0.10	69.67	63.12	68.68
0.15	68.79	62.51	68.01
0.20	70.35	62.06	67.71
0.25	69.12	62.69	68.19
0.30	69.32	63.09	69.45
0.35	69.99	62.47	68.01
0.40	69.54	62.45	68.69
0.45	69.26	62.49	68.00
0.50	70.45	61.80	68.22
0.55	69.30	62.68	68.57
0.60	69.02	61.87	67.71
0.65	69.58	62.26	67.71
0.70	70.37	62.72	68.96
0.75	71.20	62.37	69.99
0.80	72.10	63.72	72.07
0.85	72.56	65.44	72.51
0.90	74.53	67.52	74.79
0.95	**75.23**	**68.74**	**74.95**

Table E.8: Scenario #3: Average Predictive Accuracy for 1000-run Sets for Scenario #3 Using Optimal Values for $\bar{\rho_a}$

Subject	300	600	900
1	72.99	74.94	75.04
2	66.01	67.55	68.54
3	71.94	73.95	75.56

E.4 Scenario #4 Results

Table E.9: Scenario #4: Average Predictive Accuracies of Subject *Actions* for FAM

$\bar{\rho}_a$	Subject 1	Subject 2	Subject 3
0.05	55.7	59.3	52.7
0.1	56.4	56.7	**54.9**
0.15	55.8	57.5	53.2
0.2	53.4	57.3	50.7
0.25	56.3	**60.9**	53.8
0.3	57.6	58.6	50.9
0.35	54.6	56.8	53.1
0.4	56.2	57.6	52.3
0.45	52.1	57.6	49.8
0.5	54.3	57.9	51
0.55	55.9	59.6	51.8
0.6	55.5	58.1	52.6
0.65	56.9	58.7	53.8
0.7	54	57.1	51.8
0.75	55.3	55.6	51.2
0.8	54.8	58.6	53.3
0.85	57.7	57.1	53.8
0.9	57.4	56.6	52
0.95	**58.3**	57.7	51.2

Table E.10: Scenario #4: Average Predictive Accuracy for 1000-run Sets

Subject	$\bar{\rho_a}$	Predictive Accuracy
1	0.95	58.22
2	0.25	60.18
3	0.10	55.32

E.5 Scenario #5 Results

Table E.11: Scenario #5: Average Predictive Accuracies of Subject *Contexts* for 100-run sets of FAMTILE, 300 training patterns

$\bar{\rho_a}$	Subject 1	Subject 2	Subject 3
0.05	61.27	52.85	61.48
0.10	62.15	52.62	60.76
0.15	63.63	53.47	59.76
0.20	61.26	53.32	61.01
0.25	62.58	54.01	59.26
0.30	61.79	52.96	60.16
0.35	61.65	53.09	59.52
0.40	61.35	53.41	60.44
0.45	61.94	53.35	59.76
0.50	61.35	52.95	59.41
0.55	62.20	54.14	60.37
0.60	62.45	52.37	59.90
0.65	63.42	53.31	60.91
0.70	63.30	53.72	60.77
0.75	63.17	55.17	61.78
0.80	65.00	55.28	62.90
0.85	64.87	55.96	63.43
0.90	**65.33**	55.85	**62.56**
0.95	64.82	**55.97**	62.19

Table E.12: Scenario #5: Average Predictive Accuracies of Subject *Contexts* for 100-run sets of FAMTILE, 600 training patterns

$\bar{\rho_a}$	Subject 1	Subject 2	Subject 3
0.05	64.10	56.31	61.49
0.10	62.67	55.45	61.91
0.15	63.63	54.62	61.71
0.20	63.02	54.56	61.92
0.25	62.71	55.75	62.82
0.30	63.35	55.52	61.95
0.35	63.63	55.57	61.75
0.40	62.97	54.86	62.68
0.45	63.16	55.06	61.50
0.50	63.61	55.43	62.33
0.55	63.77	55.32	62.24
0.60	63.98	55.81	61.40
0.65	63.72	54.90	62.84
0.70	64.52	56.57	62.79
0.75	65.06	56.33	64.19
0.80	65.06	57.53	64.66
0.85	66.07	**58.80**	64.86
0.90	**67.75**	57.99	**64.98**
0.95	67.24	58.66	64.30

Table E.13: Scenario #5: Average Predictive Accuracies of Subject *Contexts* for 100-run sets of FAMTILE, 900 training patterns

$\bar{\rho_a}$	Subject 1	Subject 2	Subject 3
0.05	64.91	56.87	63.48
0.10	64.34	57.60	63.60
0.15	64.53	57.26	62.45
0.20	64.29	56.36	63.84
0.25	65.07	57.12	63.06
0.30	65.12	57.73	62.78
0.35	64.87	57.58	62.76
0.40	64.61	56.96	63.40
0.45	64.74	57.54	63.22
0.50	64.16	57.64	62.98
0.55	64.52	57.30	62.70
0.60	65.23	57.51	63.37
0.65	65.12	56.93	63.73
0.70	65.86	57.97	63.23
0.75	66.91	57.32	63.79
0.80	66.02	58.44	64.82
0.85	67.30	59.03	66.09
0.90	**67.64**	**59.85**	**66.81**
0.95	67.29	59.61	65.62

Table E.14: Scenario #5: Average Predictive Accuracies of Subject *Actions* for 100-run sets of FAMTILE, 300 training patterns

$\bar{\rho_a}$	Subject 1	Subject 2	Subject 3
0.05	70.62	62.48	69.86
0.10	70.06	64.05	70.23
0.15	70.77	63.75	69.34
0.20	70.52	63.48	69.51
0.25	70.30	63.55	70.55
0.30	71.21	63.48	69.89
0.35	70.10	63.93	69.47
0.40	70.40	63.45	70.58
0.45	71.38	63.81	68.86
0.50	70.40	63.25	70.43
0.55	70.81	62.99	70.40
0.60	71.34	63.22	69.43
0.65	70.98	64.36	69.72
0.70	71.49	64.96	70.78
0.75	71.46	63.67	70.88
0.80	71.91	65.05	71.16
0.85	72.98	66.59	71.24
0.90	73.36	65.88	**72.04**
0.95	**73.58**	**66.85**	71.44

Table E.15: Scenario #5: Average Predictive Accuracies of Subject *Actions* for 100-run sets of FAMTILE, 600 training patterns

$\bar{\rho_a}$	Subject 1	Subject 2	Subject 3
0.05	71.63	65.86	70.70
0.10	72.02	64.83	71.88
0.15	71.78	64.89	71.48
0.20	71.82	66.75	70.93
0.25	72.24	64.52	71.05
0.30	72.02	65.59	71.24
0.35	71.79	65.75	71.45
0.40	72.07	65.48	71.26
0.45	71.74	65.38	71.17
0.50	72.16	65.94	70.55
0.55	72.06	65.45	71.20
0.60	72.02	65.17	71.13
0.65	72.23	65.05	70.84
0.70	72.58	65.71	71.64
0.75	73.79	66.21	72.79
0.80	73.14	66.20	72.93
0.85	74.49	66.92	**74.40**
0.90	74.71	67.48	73.61
0.95	**75.24**	**68.55**	74.11

Table E.16: Average Predictive Accuracies of Subject *Actions* for 100-run sets of FAMTILE, 900 training patterns

$\bar{\rho_a}$	Subject 1	Subject 2	Subject 3
0.05	72.74	66.91	72.01
0.10	74.23	66.85	73.06
0.15	72.62	66.52	72.34
0.20	72.71	65.50	72.56
0.25	73.50	66.31	72.56
0.30	72.93	66.44	71.40
0.35	72.12	66.55	72.11
0.40	73.24	66.46	72.34
0.45	73.01	66.37	71.89
0.50	73.04	67.36	72.43
0.55	72.90	66.93	71.42
0.60	71.84	66.05	72.12
0.65	73.49	66.57	72.33
0.70	73.03	67.06	73.19
0.75	74.17	66.67	73.35
0.80	74.54	66.88	73.88
0.85	74.52	68.31	74.58
0.90	75.25	69.02	**75.68**
0.95	**75.57**	**69.12**	74.93

Table E.17: Average Predictive Accuracy for 1000-run Sets with 300 training patterns

Subject	FAM	FAMTILE (Actions)	FAMTILE (Contexts)
1	72.99	73.58	65.61
2	66.01	66.15	55.64
3	71.94	72.12	63.05

Table E.18: Average Predictive Accuracy for 1000-run Sets with 600 training patterns

Subject	Fuzzy ARTMAP	FAMTILE (Actions)	FAMTILE (Contexts)
1	74.94	75.02	66.64
2	67.55	67.82	58.10
3	73.95	73.88	64.86

Table E.19: Average Predictive Accuracy for 1000-run Sets with 900 training patterns

Subject	Fuzzy ARTMAP	FAMTILE (Actions)	FAMTILE (Contexts)
1	75.04	75.63	67.71
2	68.54	68.92	59.98
3	75.56	75.37	66.26

E.6 Scenario #6 Results

Table E.20: Scenario #6: Average Predictive Accuracies of Subject *Contexts* for FAMTILE

$\bar{\rho_a}$	Subject 1	Subject 2	Subject 3
0.05	40.2	41.0	34.2
0.1	40.5	40.0	34.8
0.15	40.9	40.1	37.2
0.2	42.6	**43.2**	**39.2**
0.25	41.6	40.0	38.0
0.3	38.7	41.7	36.3
0.35	**42.9**	41	35.0
0.4	39.1	40.5	36.6
0.45	40.7	40.3	36.9
0.5	39.2	37.2	35.8
0.55	39.1	40.0	37.8
0.6	41.3	40.3	36.4
0.65	39.3	39.8	37.2
0.7	38.1	39.0	37.3
0.75	39.6	41.5	38.5
0.8	37.9	38.0	36.6
0.85	40.1	39.0	37.8
0.9	40.6	42.5	37.0
0.95	40.8	40.9	37.1

Table E.21: Scenario #6: Average Predictive Accuracies of Subject *Actions* for FAMTILE

$\bar{\rho_a}$	Subject 1	Subject 2	Subject 3
0.05	54.5	59.2	50.8
0.1	53.8	59.4	52.0
0.15	55.5	58.3	**53.4**
0.2	54.4	56.8	50.2
0.25	53.9	57.7	50.1
0.3	55.5	58.9	49.7
0.35	54.0	57.7	50.5
0.4	55.6	59.8	51.4
0.45	53.6	57.9	49.7
0.5	55.1	56.5	51.1
0.55	53.3	59.2	51.0
0.6	54.9	57.3	49.3
0.65	53.0	58.1	50.2
0.7	55.6	58.8	49.2
0.75	56.0	59.0	49.9
0.8	56.2	**60.7**	50.3
0.85	55.3	55.4	51.0
0.9	55.9	57.1	49.3
0.95	**60.7**	57.9	49.9

Table E.22: Scenario #6: Average Predictive Accuracy of FAMTILE for Inferred Contexts and Actions over 1000-run Sets

Subject	$\bar{\rho_a}$	Context Predictive Accuracy	$\bar{\rho_a}$	Action Predictive Accuracy
1	0.35	43.22	0.95	60.25
2	0.20	47.79	0.8	60.14
3	0.20	39.99	0.15	54.07

E.7 Scenario #7 Results

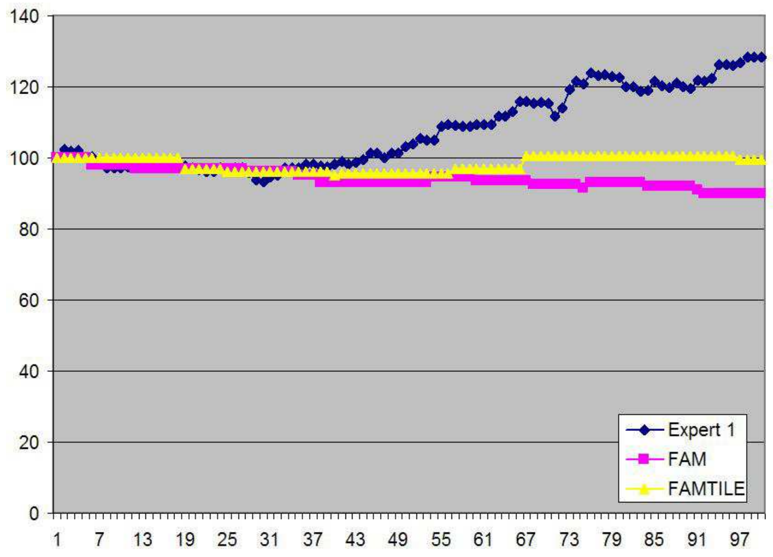


Figure E.7: Chip Count Comparison of Subject #1 versus FAM and FAMTILE

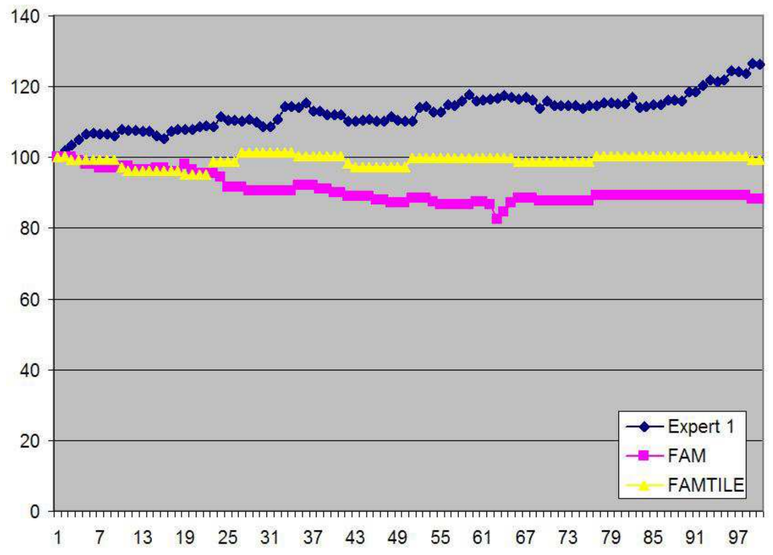


Figure E.8: Chip Count Comparison of Subject #2 versus FAM and FAMTILE

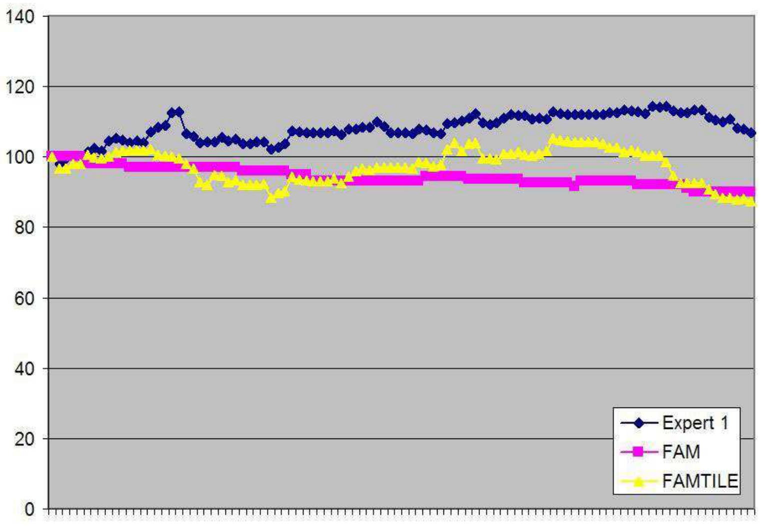


Figure E.9: Chip Count Comparison of Subject #3 versus FAM and FAMTILE

List of References

[And96] J.R. Anderson. "ACT: A Simple Theory of Complex Cognition." *American Psychologist*, **51**(4):355–365, April 1996.

[Bru79] D. Brunson. *Doyle Brunsonś Super System.* Cardoza Publishers, 3 edition, 1979.

[BS97] Gentile C. Saker I. Brézillon, P. and M. Secron. "SART: A system for supporting operators with contextural knowledge." In *Proceedings of the International and Interdisciplinary Conference on Modeling and using Context (CONTEXT-97)*, 1997.

[BS98] Papp D. Schaeffer J. Billings, D. and D. Szafron. "Poker as a Testbed for AI Research." In *Proceedings of AI '98, The Twelfth Canadian Conference on Artificial Intelligence*, 1998.

[CK02] Crandall B.W. Calderwood, R. and G.A. Klein. "Expert and Novice Fireground Command Decisions." Technical report, Klein Associates, Inc., 2002.

[CR92] Grossberg S. Markuzon N. Reynolds J.H. Caarpenter, G.A. and D.B. Rosen. "Fuzzy ARTMAP: A Neural Network Architecture for Incremental Supervised Learning of Analog Multidimensional Maps." In *IEEE Transactions on Neural Networks*, volume 3, September 1992.

[CT95] G.A. Carpenter and A.H. Tan. "Rule Extraction: From neural architecture to symbolic representation." *Connection Science*, **7**:3–27, 1995.

[FG01] H.K. Fernlund and A.J. Gonzalez. "An approach towards building human behavior models automatically by observation." 2001.

[GA96] A.J. Gonzalez and R.H. Ahlers. "Context-Based Representation of Intelligent Behavior in Simulated Opponents." In *Computer Generated Forces and Behavior Representation Conference*, 1996.

[GC01] M. Georgiopoulos and C. Christodoulou. *Applications of Neural Networks in Electromagnetics.* Artech House Publishers, 2001.

[Ger01] W.J. Gerber. *Real-Time Synchronization of Behavioral Models with Human Performance in a Simulation.* PhD thesis, University of Central Florida, Orlando, FL, 2001.

[GG00] Drewes P.J. Gonzalez, A.J. and W.J. Gerber. “Interpreting Trainee Intent in Real Time in a Simulation-based Training System.” In *Transactions of the Society for Computer Simulation International*, volume 17, pp. 135–147, September 2000.

[gH01] S.M gustafson and W.H. Hsu. “Layered Learning in Genetic Programming for a Cooperative Robot Soccer Problem.” Technical report, ASAP Group, School of Computer Science and Information Technology, University of Nottingham, Nottingham, UK, 2001.

[Hen01] A. Henninger. *Neural Network Based Movement Models to Improve the Predictive Utility of Entity state synchronization methods for Distributed Simulations.* PhD thesis, University of Central Florida, Orlando, FL, May 2001.

[HM97] Sikka P. Hovland, G.E. and B.J. McCarragher. “Skill Acquisition from Human Demonstration Using a Hidden Markov Model.” In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 3, pp. 2706–2711, 1997.

[JD02] McGinnis M. Mollaghasemi M. Johnson, M.J. and T. Damarly. “Methodology for Human Decision Making Using Fuzzy ARTMAP Neural Networks.” In *Proceedings of the International Joint Conference on Neural Networks*, 2002.

[JK99] Laird J.E. Nielsen P.E. Coulter K.J. Kenny P.G. Jones, R.M. and F. Koss. “Automated Intelligent Pilots for Combat Flight Simulation.” *AI Magazine*, **20**(1):27–41, 1999.

[JL96] R. Jones and J. Laird. “Constraints on the Design of a High-Level Model of Cognition.” Technical report, University of Michigan, Ann Arbor, MI, 1996.

[Kha98] R. Khardon. “Learning to Take Actions.” Technical report, Department of Computer Science, University of Edinburgh, June 1998.

[KK95] Oztemel E. Uludag M. Kocabas, S. and N. Koc. “Automated Agents that Learn and Explain their own Actions: A Progress Report.” In *Proceedings of the fifth Conference on computer generated Forces and Behavior Representation*, Orlando, FL, 1995.

[KL04] T. Konik and J. Laird. "Learning goal Hierarchies from Structured Observations and Expert Annotations." In *14th International Conference on Inductive Logic Programming*, 2004.

[LL99] M. van Lent and J.E. Laird. "Learning Hierarchical Performance Knowledge by Observation." In *International Conference on Machine Learning*, 1999.

[LL01] M. van Lent and J.E. Laird. "Learning Procedural Knowledge by Observation." In *First International conference on Knowledge Capture (K-CAP 2001)*, pp. 179–186, Victoria, British Columbia, Canada, October 2001.

[LP01] G. Laurent and E. Piat. "Parallel Q-Learning for a Block-Pushing Problem." In *IEEE International Conference on Intelligent Robots and Systems*, Maui, HI, October-November 2001.

[LR87] Newell A. Laird, J. and P. Rosenbloom. "Soar: An Architecture for General Intelligence." *Artificial Intelligence*, **33**:1–64, 1987.

[LR95] Laird J. Lehman, J.F. and P. Rosenbloom. "A Gentle Introduction to Soar, an Architecture for Human Cognition." Technical report, University of Michigan, Ann Arbor, MI, 1995.

[LW04] Alghamdi G. Laskey, K. and X. Wang. "Detecting threatening Behavior Using Bayesian Networks." In *Proceedings of the Behavior Representation in Modeling and Simulatoin Conference*, May 2004.

[MM86] Carbonell J.G. Michalski, R.S. and T.M. Mitchell. *Machine Learning: An Artificial Intelligence Approach*, volume II. Morgan Kaufmann Publishers, Los Altos, CA, 1986.

[Mug95] S. Muggleton. "Inverse Entailment and Prolog." *New Generation Computing 13*, pp. 245–286, 1995.

[Nor99] L. Norlander. *"A Framework for Efficient Implementation of Context-Based Reasoning in Intelligent Simulation."*. Master's thesis, University of Central Florida, Orlando, FL, 1999.

[OP00] N. Oliver and A.P. Pentland. "Graphical Models for Driver Behavior Recognition in a SmartCar." In *Proceedings of the IEEE Intelligent Vehicles symposium 2000 (IV2000)*, pp. 7–12, 2000.

[PK03] Liao L. Fox D. Patterson, D.J. and H. Kautz. “Inferring High-Level Behavior from Low-Level Sensors.” Technical report, University of Washington, Seattle, WA, 2003.

[PL99] A. Pemtland and A. Liu. “Modeling and Prediction of Human Behavior.” In *Neural Computation*, pp. 11, 229–242, 1999.

[PT02] J. Pineau and S. Thrun. “High-level robot behavior control using POMDPs.” Technical report, Carnegie Mellon University, Pittsburgh, PA, 2002.

[Rab89] L.R. Rabiner. “A Tutorial on Hidden Markov Models and Selected Applications in speech recognition.” In *Proceedings of the IEEE*, volume 77, pp. 257–286, 1989.

[SG00] T.A. Sidani and A.J. Gonzalez. “A Framework for Learning Implicit Expert Knowledge through Observation.” In *TRANSACTIONS of the Society for Computer Simulation International*, volume 17, pp. 54–72, April 2000.

[SG04] B.S. Stensrud and A.J. Gonzalez. “Context-Based Reasoning: A Revised Specification.” In *Proceedings of the Florida Artificial Intelligence Research Society (FLAIRS) Conference*, Miami Beach, FL, May 2004.

[SH01] T. Stanard and R. Hutton. “A Computational Model of Driver Decision Making at an Intersection Controlled by a Traffic Light.” Technical report, Micro analysis and Design, Boulder, CO, 2001.

[Skl89] D. Sklansky. *The Theory of Poker.* Two Plus Two Publications, 3 edition, December 1989.

[SM03] D. Sklansky and M. Malmuth. *Hold’em Poker for Advanced Players.* Creel Printing Company, 21st century edition, September 2003.

[SR99] McCallum A. Seymore, K. and R. Rosenfeld. “Learning Hidden Markov Model Structure for Information Extraction.” In *AAAI ’99 Workshop on Machine learning for Information Extraction*, July 1999.

[sta90] *Elementary Statistics in a World of Applications.* Harper Collins Publishers, 3 edition, 1990.

[TA00] Nakemura T. Imai M. Ogasawara T. Takeda, M. and M. Asada. “Enhanced Continuous Valued Q-learning for Real Autonomous Robots.” In *Proceedings of International Conference of the Society for Adaptive Behavior*, pp. 195–202, 2000.

[TG04] Stensrud B.S. Trinh, V.C. and A.J. Gonzalez. "Implementation and Analysis of a Context-Based Reasoning Model on a Physical Platform." In *Proceedings of the Swedish-American Workshop in Modeling and Simulation*, 2004.

[TK74] A. Tversky and D. Kahnerman. "Judgments Under Uncertainty: Heuristics and Biases." *Science*, **185**:1124–1131, 1974.

[Tur98] R.M. Turner. "Context-Mediated Behavior for Intelligent Agents." In *International Journal of Human-Computer Studies: Special Issue on Using Context in Applications*, volume 48, pp. 307–330, March 1998.

[WJ02] Laird J.E. Nuxoll A. Wray, R.E. and R.M. Jones. "Intelligent Opponents for Virtual Reality Training." In *Inter-service/Industry Training, Simulation, and Education conference (I/ITSEC)*, Orlando, FL, December 2002.

[WM] W. Warwick and S. McIlwaine. "Developing Computational Models of Recognition-Primed Decisions: Progress and Lessons Learned." In *Computer Generated Forces and Behavior Representation Conference.*

[WP99] Ganapathiraju A. Wu, Y. and J. Picone. "Baum-Welch re-estimation of Hidden Markov Model." Technical report, Mississippi State University, Starkville, MS, June 1999.

[YN00] Hori K. Yairi, T. and S. Nakasuka. "Autonomous Reconstruction of State Space for Learning of Robot Behavior." In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 891–896, 2000.

[ZM00] M.X. Zhou and S. Ma. "Toward Applying Machine Learning to Design Rule Acquisition for Automated Graphics Generation." 2000.