Development of Functional Requirments for Cognitive Motivated Machines

A dissertation presented to
the faculty of
the Russ College of Engineering and Technology of Ohio University

In partial fulfillment
of the requirements for the degree
Doctor of Philosophy

James T. Graham
April 2016

This dissertation titled

Development of Functional Requirements for Cognitive Motivated Machines

by

JAMES T GRAHAM

has been approved for

the School of Electrical Engineering and Computer Science

and the Russ College of Engineering and Technology by

Janusz A. Starzyk

Professor Emeritus of Electrical Engineering and Computer Science

Dennis Irwin

Dean, Russ College of Engineering and Technology

ABSTRACT

GRAHAM, JAMES T., Ph.D., April 2016, Electrical Engineering

Development of Functional Requirements for Cognitive Motivated Machines

Director of Dissertation: Janusz A. Starzyk

Machine Intelligence, and all of its associated fields and specialties, is a wide and complex area actively researched in laboratories around the world. This work aims to address some of the critical problems inherent in such research, from the most basic neural network structures, to handling of information, to higher level cognitive processes. All of these components and more are needed to construct a functioning intelligent machine. However, creating and implementing machine intelligence is easier said than done, especially when working from the ground up as many researchers have attempted. Instead, it is proposed that the problem be approached from both bottom-up and top-down level design paradigms, so that the two approaches will benefit from and support one another. To clarify, my research looks at both low level learning, and high level cognitive models and attempts to work toward a middle ground where the two approaches are combined into a single cognitive system. Specifically, this work covers the development of the Motivated Learning Embodied Cognition (MLECOG) model, and the associated components required for it to function. These consist of the Motivated Learning approach, various types of memory, action monitoring, visual and mental saccades, focus of attention, attention switching, planning, etc. Additionally, some elements needed for processing sensory data will be briefly examined because they are relevant to the eventual creation of a full cognitive model with proper sensory/motor I/O. The development of the Motivated Learning cognitive architecture is covered from its initial beginnings as a simple Motivated Learning algorithm to its advancement to a more complex architecture and eventually the proposed MLECOG model. The objective of this research is to show that a cognitive architecture that uses motivated learning principles is feasible, and to provide a path toward its development.

DEDICATION

This work is dedicated to my parents Andrew T. Graham and Marilyn S. Graham, and everyone else who encouraged over the course of completing this work.

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Janusz A. Starzyk for working with and helping me complete my dissertation research. I would also like to acknowledge the support given to me via grant payments supported by the Polish National Science Center under grant DEC-2011/03/B/ST7/02518.

I would also like to thank Dr. Mehmet Celenk, Dr. Jeffrey Dill, Dr. Savas Kaya, Dr. Jeff Vancouver, and Dr. Annie Shen for serving on my committee and aiding me in the completion of my research and dissertation.

Finally, I would like to thank my family for supporting and encouraging me during all this time.

TABLE OF CONTENTS

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER 1: INTRODUCTION

1.1 Background

Artificial Intelligence (AI) research began in earnest in the late 1950s, largely as a result of a conference at Dartmouth College in 1955 [1] and lead to a significant effort on such research until problems surfaced in the mid-70s. These problems and the resulting lack of progress led to a dry spell in AI research until the creation and commercial success of "expert systems" in the early 1980's. Although expert systems of this time were useful, they emphasized the collection and codification of data and were not suited toward AI development. AI research, based on expert systems, tended to include the assumption that the AI agent's "understanding" would eventually develop as it accumulated knowledge and an artificial mind would be "born". ("Agent" is used to refer to the software/hardware under consideration.) More recently, in the 1990s and up to now, AI research has evolved and taken increased inspiration from knowledge on how the human mind/brain works. Research examining cognition has increasingly investigated embodied intelligence (e.g. how the body shapes the mind [2]) and used developmental robotics as a source of inspiration. Over these years, AI research has greatly benefitted from research in robotics, developmental psychology, and study of the brain.

AI research has many applications in a variety of domains, for example, logistics, data mining, finance, medical diagnosis, robotics, customer service and many other areas. Research on AI is often applied to problems of object recognition (character, speech, face, etc.) and control (placing the system in a desired state, automation, game AI, etc.), often with significant overlap between both research and application areas. For example, the manufacturing industry would like to have semi-intelligent robots that are capable of constructing its products and controlling the process of production. AI for that purpose would not need to be conscious, or even particularly intelligent. However, AI for process control would need to recognize and correct deviations from a desired pattern. This means the AI agent would have complex motor control, sensory recognition (e.g. temperature, pressure, mass or flow measurement, visual recognition and elements of computer vision), and possibly some level of data mining and automated reasoning.

As artificial intelligence systems become more complex, we start to think of them as "intelligent machines." But how do we define intelligence? What about cognition? Definitions for these terms tend to be vague, or broader than we might prefer, presumably because we are still trying to understand how intelligence and cognition work. Most dictionaries define an individual's intelligence as an encompassing term describing several intermeshing abilities such

as the capacity for abstract thought, reasoning, planning, problem solving, understanding of symbols, and so on. Our knowledge of how the brain and intelligence in general work is, in many respects limited. Hence, these definitions tend to rely on philosophy to bridge the gaps between knowledge and expectation. We have learned a great deal about how the brain works via testing in the various medical fields (mostly neuroscience and psychology), but there is still a great deal that remains unknown.

Cognitive AI is AI that is capable of acquiring knowledge and understanding through, experience driven by actions and supported by senses. Cognitive AI agents (particularly those with some level of autonomy) have assistive potential in many areas, and *research on cognitive AI has additional benefit in helping us understand our own intelligence and intelligence in general*. The more autonomy or freedom to act given to a cognitive AI, the more independent and capable that AI can become. However, it also becomes more difficult to control effectively. (Greater autonomy also tends to imply more complexity/capability in the agent's design as well.) There are ways to direct the agent to do what the designer needs without eliminating its autonomy.

One approach toward doing this is to design the agent with certain initial state/setting so that it learns in the desired way. However, making these initial settings too complex could cripple the agent's ability to learn as a result of the constraints that force the agent to learn in a specific way. On the other hand, if one or more simple requirements are specified, for example, the need for a certain food or nutrition element, then the agent can learn on its own how to get it, provided it has the needed tools to learn how to operate. The agent, while taking longer to do so, will have the ability to complete the task while potentially learning other useful and related skills that it would not otherwise attempt.

Another method for guiding an autonomous agent is to have the agent observe the teacher and/or accept commands; however, this is also difficult, as it requires the agent to have reached a certain level of complexity to be able to take this type of direction. To learn by observation, the agent has to be able to understand and put itself "in the shoes" of whomever or whatever it is observing and try to imitate their actions if desirable. Furthermore, the very concept of autonomy means the agent may decide to do something completely different than what is desired by its designer/teacher/controller. The fact that an agent is autonomous means that it makes decisions on its own, and this implies that it can do things that are unexpected by the designer. This can be good in the sense that the agent may actually learn to perform tasks better or differently than the designer anticipated.

As AI models increase in capability, they are also expected to increase in complexity. The additional complexity that comes hand-in-hand with more capability means that as we move from more basic applications (such as assembly line robotics) toward increasing the intelligence and autonomous capability of our agents, the added complexity makes it more difficult to see into their inner workings, or to propose further improvements. In other words, the more complex something gets the more difficult it can be to untangle its inner workings to track down problems, test its abilities or to justify and insert new functionality.

1.2 Research Overview

This research explores the development of autonomous cognitive AI. Autonomous learning agents are expected to be useful because they can learn and adapt. This type of AI is suited for things like companion or assistive robots and working in environments where we cannot or do not want to go. Space exploration is another example of a situation where intelligent autonomous robots could be very useful, because the distances involved make communication between earth and the robot very slow. Even when separated by great distance, extra-planetary robots have to operate in real time and can easily encounter obstacles or events that were unplanned and where significant command delay could prove catastrophic, like when a rover fails to recharge solar batteries before nightfall or when it does not avoid falling rock. However, by adding a capable AI, the risk associated with long communication delays can be reduced or eliminated. And this is only one application of autonomous learning AIs. There are many potential applications, some of which may have not even been thought of yet.

Studying how to create a cognitive agent aids in the realization of next-generation AI/robot capabilities from the ability to automatically perceive, monitor and map surroundings; to improvements in the ability to locate, detect, and identify objects of interest; to making decisions and taking action based on an understanding of the environment and various user inputs. Research in cognitive AI also requires investigation into areas such as the role of an agent's interaction with the environment in developing perception, memory organization, prediction, internal value systems, internal goals, and the use of memory in development of motor skills and machine intelligence.

The construction of learning mechanisms and the creation of memory structures is essential for the development of autonomous robots. A system with such memories can learn and achieve its objectives autonomously, without continuous supervision. Hence, it is important to develop knowledge about the role of an agent's interaction with the environment and resulting

memory organization, prediction, internal goals, and the use the motor actions. These are all things that other cognitive models and this dissertation attempt to address, particularly through the Motivated Learning Embodied Cognition (MLECOG) model (see Chapter 7).

There have been several cognitive agent systems proposed, such as CLARION [3][4], ICARUS [5][6], LIDA [7][8], Soar [9][10], SASE [11][12], Charisma [13][14], and others (see Section 2.3). Although these approaches have advanced the field of AI, the approaches embodied in these agents also have weaknesses. These weaknesses consist of varying levels of reliance on pre-specified goals, lack of symbol grounding, inability to handle complex environments, limited learning capabilities, and more.

Symbol grounding is a common problem that is the result of an agent manipulating symbols that lack "meaning" or grounding in reality. Man also manipulates symbols. However, these symbols are grounded by experiences and the associations attached to them via memory structures. This means, that in order for an agent to be symbol grounded, the symbols with which it works need to be related to its sensory input(s) and developed associations.

Similarly, overreliance on pre-specified goals can mean that an agent cannot learn, or has difficulty learning outside of a rigidly specified environment. An agent with limited learning capabilities may be adequate if the agent is designed for a specific domain. For more complex domains and environments, reduced learning capabilities can lead to the inability of an agent to perform. For example, a social agent may be designed to engage in social cooperative activities, but this predisposition may only reduce its ability in other areas, e.g., in individual thought capabilities. One philosophy of the work involved in this dissertation is that it is preferred to give the agent the underlying structure/tools needed to develop, rather than overly predispose it to any particular application area. This may mean the development process is slower, but may also lead to a more long-term effective and capable AI.

The agent's development is also impacted by its learning environment. The environment's responses to the agent, the information the agent can pull from the environment, and the agent's ability to interact with the environment all help to determine how it develops. However, investigating the interaction of the agent with the environment is complex. There are many factors involved, both in the agent and the environment, hence, another facet of this work is to investigate approaches to simplify interaction with the environment, while still allowing for increasing the complexity if needed/desired.

In a simulation, the designer has total control of both the agent and the environment and can make them as simple or as complex as desired, making simulations a good answer to the

problem of overly complex environments. The simulation-based approach also has the advantage of being easier to develop and prototype due to its flexibility and low cost, as it is not necessary to purchase and maintain an expensive robotic platform.

Constructing an environment, even in simulation, is not simple. First, there is the consideration of what to include in the environment. Beyond that, the agent's ability, or inability, to interact with the environment needs to be considered. For example, if there are other actors in the environment, can the agent interact with them, and if so, how? For example, in order to deal with other actors in its environment, an agent needs to be capable of recognizing their presence, tracking what they do, and reacting in some way to their actions. In other words, the agent has to be capable of developing motivations in response to the other actors' impact on its environment, whether that impact is good or bad. This ability to consider the actions of others and their impact on an agent's motivation likely sets the groundwork for social behavior, cooperation with other agents, following other agents' example, and accepting other agents as teachers or leaders. This may also invoke putting society's interests ahead of its own and developing emotions like empathy for others.

Motivation in autonomous cognitive intelligences is of primary importance. Without some form of motivation, the agent will have no reason or drive to learn or to act. In the last several years, there has been increased focus on what motivates machines and on intrinsic motivation, in particular. Intrinsic motivation is motivation that is internal to an agent, and not set by an outside source. Curiosity can be considered a type of intrinsic motivation, and has been used as such by other researchers, such as in [15]. The Motivated Learning (ML) research, investigated in this dissertation, however, focuses on intrinsic motivations beyond curiosity and on the agent's ability to develop and add to its motivations. The foundations of ML will be covered in this dissertation.

1.3 Dissertation Objectives

As mentioned in Section 1.2, there is much utility to be found in autonomous agents and AI research in general, however, there are also many obstacles to overcome in the form of processing data, memory, object recognition, problems solving, directing (or not directing) an agent, etc. This dissertation describes the development of a Motivated Learning agent and subsequent improvements to the agent that are capable of creating new motivations as the agent learns how to satisfy its most basic needs while interacting with its environment. The objective is

to show that a cognitive architecture that uses motivated learning principles is feasible and to provide a path toward its development.

"Motivated Learning" and the new Motivated Learning Embodied COGnitive (MLECOG) model address many of the shortcomings previously mentioned, such as how an agent can provide itself with its own drives and motivations without relying extensively on designer or other external directives or rewards, how to deal with the "reality gap" by ensuring there is a form of symbol grounding, using attention switching and saccading to deal with large amounts of input and memory information, etc. Also looked at is how the agent integrates elements from the basic concepts of self-organizing networks, reinforcement learning (RL), hierarchical (or deep) learning, invariant recognition, and symbol grounding. The agent model uses episodic and semantic memory to store "knowledge" (data and associations) and uses attention switching for focusing itself on specifics of the environment and switching between cognitive tasks and associations in its memory, both of which have not previously been done in conjunction with motivated learning.

The goals of this work are as follows:

1) Design and implement a Motivated Learning (ML) Scheme
    a. Create a flexible environment for testing
    b. Ensure the ML agent can function in a changing environment
    c. Implement and test the ML agent in a NeoAxis [16] environment
2) Compare ML to different RL algorithms as a way to evaluate its performance and capabilities
    a. Design a simple "Black Box" environment
    b. Get and test several other RL algorithms in the environment.
3) Design and implement a ML based cognitive model
    a. Determine necessary components of cognitive model
    b. Design and implement components (as feasible)
    c. Implement a reduced version of the cognitive model
    d. Test implementation in NeoAxis

These goals have all been met or exceeded as discussed in the next section in terms of the scope of this work and as shown in the following chapters.

1.4 Scope

This work covers the development of the MLECOG model, and the associated components required for it to function. These components consist of things such as the Motivated Learning approach, various types of memory, action monitoring, attention switching, planning, etc., and are discussed in more detail throughout the dissertation. Some elements needed for the model to be able to process sensory data (i.e. invariant object recognition) will be examined because they are relevant to the eventual creation of a full cognitive model with proper sensory/motor I/O.

I/O handling is an extensive and complex area. However, there are common requirements for I/O in a cognitive agent. First, there needs to be a way to consistently recognize features in the environment. For that purpose, this dissertation also explores hierarchically invariant object recognition as a potential method for performing visual recognition of features and objects. Currently, there are several "deep learning" algorithms (a category to which many hierarchical learning algorithms belong). However, they have their flaws. It is hoped that investigation into the area of deep learning would aid in utilizing similar algorithms for the proposed cognitive agent.

ML systems, like most cognitive agents, utilize neural network (NN) inspired architectures because they try to reflect the architecture of the human brain. It is generally accepted in neuropsychology that the brain is a self-organizing network, albeit, a very complex one [17]. Therefore, self-organizing maps (SOMs) tend to be present in some form in these architectures as a method for streamlining the "unsupervised learning" aspect of the agent. However, SOMs can also have issues with parameterization (too many parameters or high dependence on parameter values), and high computational overhead. There are also advantages to SOMs. Because of their self-organizing nature, SOMs can operate without oversight and explicit supervised training. Alternatively, they can take specific input for teaching, since they can align themselves to external or *internal* feedback (internal feedback being generated by the agent itself). Considering the utility of such networks, some investigation into them for potential use in a cognitive architecture is warranted. While not a direct part of this dissertation, some research was done in this area [18], with attempts to minimize some of the aforementioned problems such as parameterization.

By investigating learning mechanisms such as the previously mentioned object recognition and neural gas networks, insight will be gained into how to better structure the MLECOG model. However, as research time is limited, and the MLECOG model has not been

fully realized with all modules functioning; for the purpose of this work, the model is partially implemented (some modules are bypassed for now, e.g. episodic memory, and other modules such as sematic memory are emulated rather than fully implemented) in both Matlab and C++. The C++ implementation is integratable with the already existing NeoAxis implementation used with the simpler, already realized, Opportunistic Motivated Learning model (see Chapter 4). Chapter 2 provides an explanation of the cognitive models in general and briefly examines several of them. Chapter 3 covers the underlying structures, organization, and principles of Motivated Learning.

Chapter 4 discusses several modifications and additions to the original ML algorithm. One of the most important chapters is Chapter 5, which presents results and simulation data pertaining to the material discussed in Chapter 3 and 4. Chapter 6 provides some comparisons between the Motivated Learning algorithm and a selection of reinforcement learning algorithms. Chapter 7 covers the architecture of the more complex MLECOG model, based around the ML algorithm, as well as initial simulation results thereof. And Chapter 8 concludes the dissertation with a discussion on what has been accomplished and what may be done to further advance this work. Chapter 8 also provides a brief discussion of some related supplementary research work that was undertaken over the development of this material, such as the development of a Hybrid Neural Gas algorithm that is useful for unsupervised learning applications, and a self-organizing hierarchal invariant object recognition algorithm that had possible applications for use in the MLECOG algorithm as a means for reliably recognizing objects on the sensory inputs.

CHAPTER 2: COGNITIVE AGENTS

2.1 Introduction

In the Introduction of Chapter 1, the primary focus of this research is described as the development of a cognitive architecture based on Motivated Learning (ML). This chapter presents a short overview of several basic assumptions of the work on ML and presents the background of some other alternative cognitive architectures.

2.2 Motivated Learning versus Classical Reinforcement Learning

The idea behind Motivated Learning (ML) is that it should not rely on external control but instead move towards self-determining agents, albeit with a set of broad constraints and guidelines for performance as determined by the embodiment (with primitive pains and sensor/motor capabilities) and limited by the environment. In ML, motivation is expressed through needs or drives present in the agent. “Drive” and “need” will be used in this dissertation interchangeably, as one leads to the other. At its initialization, an agent starts with one or more primitive needs that it must learn to fulfill (the levels of the needs are measured by sensors). From these needs the agent creates additional abstract needs based on its solutions to the primitives and the knowledge it has gained. In ML, the agent’s motivations, created internally by the agent (known as abstract needs), compete with and can take precedence over the needs specified by the designers (known also as primitive needs). The ML agent creates new needs as a result of learning how to satisfy existing needs.

An unfulfilled need has an associated pain signal. The ML agent has motivations to fulfill its needs and reduce associated pain signals. For example, if it has a “primitive” need to keep itself charged, and discovers that the solution to doing so is to use batteries, it will want to maintain a supply of batteries and, thus, create a need to do so. In other words, as the agent moves around, it depletes its charge, increasing its “lack of charge pain”, directly related to its primitive need. It will try to resolve that need and eventually discover that it can use new batteries to power itself. This will create a motivation to maintain a supply of batteries, as well as an associated “lack of batteries” pain. (In more advanced models, a ML agent can learn to override primitive motivations in a situation where a combination of attention, memory stimulation, and other motivations lead to a higher priority on the abstract motivation and related action. This is discussed in more detail in Chapters 4 and 7.) Through this process, the ML system reinforces useful behaviors and calls upon them when needed. (The following publications, [19]–[22],

presented the idea of motivation as the underlying force behind a cognitive agent's operation.) See Chapter 3 for a more detailed description of how the Motivated Learning system functions.

While the ML approach serves as the basis for the MLECOG architecture, it also shares several similarities to Reinforcement Learning (RL) [21]; for example, both approaches have a "reward" element, learn value-functions (functions that determine the "value" of a specific action given the input state), possess objectives, etc. However, there are also several significant differences (see Table 2-1). Most notable is that a Motivated Learning agent tends to build a "network of needs" as it discovers them and adds abstract pains, and evaluates the interdependencies of these needs to determine the best actions to take. It performs its actions based on the state of the environment and its current internal state. Reinforcement Learning (RL), on the other hand, chooses its next action based only on the state of the environment and its known responses.

Table 2-1. Reinforcement Learning vs. Motivated Learning

Reinforcement learning	**Motivated learning**
Single value function	Multiple value functions
Measurable rewards	Externally immeasurable rewards
Predictable	Unpredictable
Objectives set by designer	Sets its own objectives
Maximizes the reward	Solves minimax problem
Potentially unstable	Always stable
Always active	Acts only when needed

A Reinforcement Learning agent tries to learn value functions that maximize externally set rewards. Thus, needs in RL are defined by the designer and are strictly reinforced by external rewards. In contrast, the Motivated Learning based approach chooses its own internal motivations, and hence determines its own rewards (with the caveat that satisfying "primitive" pains is analogous to RL based rewards). Furthermore, in ML, each "need" has its own "value function", so depending on the state of its needs a ML agent may choose different actions for the same environment state.

Because rewards are internal to the ML agent, total reward is not externally measureable, and the ML agent cannot be optimized since the internal state is not only a function of the state of the environment but depends on its entire history, its sensory and motor system, and the ability to

process incoming information and to learn from it. As a result, the ML agent's behavior can be unpredictable due to its ability to set its own goals and needs. ML solves a minimax problem and, thus, is well equipped to deal with multiple needs. This differs from RL algorithms, which tend to be based on maximization of a reward, and thus have a preference for goals directly related to the reward signal. RL algorithms generally have no upper limit on the amount of the reward received. Therefore, they maximize their rewards, neglecting other needs that the system may have or ignoring damage to the environment that their actions may cause (and thereby potentially impacting other needs). This greedy approach to reward may even lead to the destruction of the system itself as we all know from the destructive results of drug use or any action that continues to stimulate pleasure centers in the brain. Thus, we call such systems unstable.

A ML agent is always stable in terms of its reward seeking, since its reward is always constrained (limited by the level of a corresponding pain signal) and it only acts when needed as indicated by the need/pain level (when all needs are below threshold, the agent may decide to do nothing).

Additionally, to assist the exploratory process, many RL algorithms include methods by which they can deviate from value function maximization. However, as time passes the frequency of this behavior tends to decrease. The ML algorithm can also explore how to best implement its goals. However, most of its searches are goal oriented and not only based on curiosity and increasing maximum information as in typical implementations dealing with intrinsic rewards implemented by RL researchers.

The ML agent chooses its goal oriented action depending on the strength of the interconnection weight between the goal and a specific action. During the learning process the agent adjusts these interconnection weights. A later modification of the ML algorithm changed how the agent selects actions based on probabilities rather than the interconnection weights (see Chapter 4). The idea was to allow the agent to explore alternative goal oriented actions, even if it does so infrequently. Furthermore, the only time ML currently engages in pure "curiosity" or exploratory behavior is when it has no other active motivation (pain).

As previously noted, in order to develop a human-like mind, a system that is capable, at some level, of operating on its own initiative is required. While ML is always controlled by its motivations, the architecture is designed such that the agent is capable of defining its own motivations to guide its interaction with the environment.

Initial models for ML used concurrent processing with symbolic representation of sensory inputs and motor actions. However, the concurrent symbolic implementation of the

Motivated Learning concept by itself is not sufficient, as it requires "filtering" of its inputs through object recognition and representation building systems, as well as translation of its symbolic outputs to time domain sequences for motor control. In order to properly leverage ML, a full cognitive model is needed with memory, planning, and sensor/motor I/O.

As mentioned in Section 1.4, several modules are to be added to the ML algorithm to enhance its capabilities. Implementation of these additional components as part of a comprehensive cognitive model allows the agent to shift its attention between different objects and concepts within memory, plan complex actions, and recall previous events for later reexamination. The proposed version of this cognitive model is covered in Chapter 7.

It should be noted that the implemented version of the model is a reduced version with some components removed and others reduced in scope (such as the use of a pseudo-semantic memory rather than actual semantic memory, as mentioned in the Scope section). That said, our co-researchers are working to create a working equivalent for some of the reduced or missing modules, which may, in time, be included in the implementation.

2.3 Embodied Cognitive Systems

Many models for intelligent machines exist, but they are often either very limited or exist only as theoretical models and are not implemented. Several of these models were mentioned in the Introduction. These systems consist of various cognitive architecture models that affect how each machine is designed and constructed and most follow the concept of embodied cognition [23]. (Note, a somewhat reduced variation of the following discussion is present in [24].)

There are several paradigms to which these models belong. Vernon et al. [25] provides a survey of these paradigms of cognition, consisting of cognitive (symbol based) approaches, emergent system approaches (consisting of connectionist, dynamical, and enactive - according to Vernon), and their various hybrid combinations. LIDA, CHARISMA, and the Motivated Learning approach belong to the emergent approach (part of embodied cognition), in which the agent gradually learns to become effective in its environment.

Embodied cognition has developed to challenge the earlier philosophy that an artificial intelligence can develop largely independently of the body, which houses it [26], [27]. This approach parallels similar developments in robotics that state that intelligence has to be embodied, situated in its environment, and cannot develop without embodiment [2], [28]. The general idea of embodied cognition is that the embodiment of an intelligent system influences the directions and methods by which the system develops. Sensory and motor abilities, as well as

natural and social environment, have a direct impact on the development of an intelligent system, and we can see examples of the effect of embodiment in human development. For example, someone who is born blind only understands the concept of color in an abstract manner. And just as a robot without the same senses as humans possess would have trouble relating to us, we would have trouble relating to robots who "see" in radio, "smell" via gravity, and lack a sense of taste.

It has been claimed by some that natural and social environments become integrated as part of the cognitive system [29], [30], and that the brain initially developed to guide motion and basic tasks before abstract thinking became relevant, and that it eventually evolved to its current point in humans. Therefore, the structure of a cognitive agent mind should reflect this development of the brain by "evolving" more advanced features on top of lower level functions. By instituting this type of "natural" development an agent can "infer control decisions" rather than have them programmed [31].

The goal of creating cognitive systems is to develop machines with human-like intelligence. Since they would possess a level of intelligence, different from, but also similar to humans, due to the similarities in embodiment, such machines would, like all creatures with some level of intelligence, process their environment, decide upon actions autonomously, and learn by observing the results of their actions. The increasing interest in the embodiment of intelligences over the last couple of decades has led to a corresponding upsurge in the utilization of biological inspiration and processes in the development of cognitive architectures. A cognitive architecture tries to model cognitive processes using either functional or structural implementation approach. The level of biological inspiration differs from model to model. Some, like the Ikaros architecture [32] are designed specifically around biological principles, while others, like Leabra [33], try to model biological understanding in cognitive models to prove/disprove their validity. Other architectures, such as Soar [9], one of the most widely known cognitive systems, started out in a more traditional manner (purely symbolic and with significantly less biological inspiration than most of the newer models), but has added numerous additional biologically inspired capabilities since it was first developed.

Soar is one of the oldest architectures discussed here and began as a purely symbolic system, but has been significantly improved with specialized modules for supporting a wide variety of capabilities [10]. The fact that it is based on a production system (rule based system) and uses production rules to govern its behavior means that it shares some similarities with expert systems. It uses many useful functional blocks like semantic, procedural, and episodic memory,

working memory, and sub-symbolic processing of the sensory input data. It uses decision cycles to search for knowledge relevant to its goals, and uses production rules and activation mechanism in conjunction with the episodic memory. More recently, Soar has even been extended for non-symbolic reasoning [34], however, while non-symbolic modules have been added, they remain somewhat separate from the Soar's core reasoning which is purely symbolic. Additionally, Soar does not use an effective attentional mechanism, limiting use of attention to the perceptual stage only; therefore it does not contain real-time regulation of its processes and assumes that it has enough processing power to compute everything that it needs to decide about its action.

In the same area as Soar, there is also the ACT-R/E architecture [35], which is an "embodied" variant of the traditional ACT-R [36], [37]. ACT-R is a symbolic system very much like Soar, however, it incorporates sub-symbolic functions, and its knowledge can be divided in two types: declarative and procedural. ACT-R/E tries to faithfully model people's behavior as they perceive, think about, and act on the world around them. By closely modeling human processing, the creators of ACT-R/E believe that it will be more capable of interacting with humans in assistive situations, such that their agent will leave people alone when they are doing something they are good at, or help them when they need help and provide them relevant knowledge that they do not have.

In contrast to Soar and ACT-R, the DIARC architecture [38] features asynchronous processing, such that each component has its own "cognitive" cycle, possibly with multiple threads. There is no centralized controller, and goals are explicitly represented as pre- and post-operating conditions. Action selection is distributed and priority based (via goal priorities), and different components can use different learning mechanisms.

The MDB architecture takes an evolutionary approach to provide robots with lifelong adaptation [39]. It allows for intrinsic change of goals or motivations by introducing a satisfaction model and permitting fast reactive behavior. The MDB system does so while preserving deliberative characteristics and considering the selection of behaviors instead of simple actions. In the MDB model, motivation exists to guide the agent's behavior based on the degree of fulfillment of the motivation using both the internal and external perceptions of the agent. This is similar to how MLECOG (discussed in Chapter 7) handles motivations and action choices based on pain/need (and other factors such as distance, availability, etc.). Additionally, MDB possesses long-term and short-term memories. However, it currently lacks any kind of attention switching mechanism to assist with perception (or memory processing).

The SASE architecture [11], [12], proposed by Weng, is based on Markov decision processes (MDP), which are an extension of Markov chains, and includes the concept of autonomous mental development. It strives to be a self-aware, self-effecting architecture, and has been tested on autonomous developmental robotic systems.

It is generally accepted that the brain's cortex is organized in mini-columns [40] that engage in top-down bottom-up feedback both among themselves and other parts of the brain. This top-down feedback is often in the form of (usually unconscious) prediction as to what is on the feed-forward "input". When we observe a scene or perform an action, certain observations or occurrences are expected and we tend to react when the expectations are not met. The PASAR architecture [41], based on the original DAC model, uses this understanding and attempts to create top-down bottom-up mechanism to drive attention. Its goal is to explore how a data association mechanism can integrate bottom-up sensory information and top-down knowledge.

Another architecture that takes a similar approach is TRoPICALS [42], which proposes general principles about the brain mechanisms underlying compatibility effects, in particular the idea of using top-down bias from the prefrontal cortex, and whether it conflicts or not with the actions afforded by an object, plays a key role in such phenomena. One of the interesting features of the model is that it can provide testable predictions of the possible consequences of damage to volitional circuits such as in patients with Parkinson's disease [42].

The NARS system developed by Wang [43], [44] has many features that make it a good candidate for human-like intelligence. The system grounds its knowledge in experience, and is therefore an embodied situated system. It also uses fuzzy reasoning and learning, operates in real-time, and prioritizes its tasks based on urgency that depends on its internal state and durability. The tasks it generates compete for attention in dynamic memory in terms of resource management and context dependent resource allocation. In other words, tasks gain attention via their proposed resource usage/consumption and the current agent/environment context. While NARS does have externally set goals, which are persistent, other goals developed by the system are prioritized by their determined urgency and durability.

Although, it is very appealing to formalize an intelligent system through language and inference rules that support its development, this kind of framework is not necessary. Not all decisions people make are logically justified, or consciously explained, and decision making depends on a continuously changing emotional state. Instead, in the MLECOG architecture presented in this dissertation, self-organizing networks are used to build distributed representations and support decisions. Perception in NARS is considered as a special case of

action, where such action results in a new judgment about the perceived objects according to the goals of the agent. This is in agreement with the approach taken in MLECOG.

The ICARUS architecture [5], [6] is primarily concerned with behavior in embodied agents and emphasizes perception and action over finding solutions to abstract problems. ICARUS uses symbolic representation of knowledge, pattern matching, and has an interpret-act cycle. Its behavior is driven mainly by the utility value of its potential actions. Icarus has been used to develop a number of synthetic characters for simulated environments, as well as more traditional AI tasks.

CLARION, a hybrid cognitive architecture presented in [3], [4], [45] combines sensory perception with symbolic representation through neural networks, reinforcement learning, and higher-level declarative knowledge. It uses a motivational system for perception, cognition and action. Action can either be applied to the external environment or to manage its memory and goals, which is an important function in cognitive systems. Time management is focused on real-time response of individual modules with a meta-cognitive subsystem dynamically modifying system action management. Unlike other architectures, CLARION attempts to discover explicit rules via inductive analysis of what has been implicitly learned, which typically means extracting rules from an artificial neural network.

Higher level motivations in CLARION are pre-trained using a back propagation neural network. Sun suggests that some motivations may be derived through conditioning [45]. However, there is no demonstration of how such motivations can be derived by the CLARION agent, nor any practical example in which the learning agent derives its motivations. This approach to derived drives is similar to the ML approach discussed in this dissertation, however CLARION uses arbitrarily set constants to determine the maximum strength of the motivation [46]. This implies that motivations are known in advance and therefore cannot be learned autonomously by the agent. In ML no such arbitrary setting is used or needed to develop an agent's motivations.

The LIDA architecture [7], [8], [47], [48], is based on the Global Workspace Theory (GWT) [49], [50], and draws on the concept of belief-desire-intention (BDI) [51]. It uses several types of memory blocks and operates based on cognitive cycles during which the system senses, attends and acts. Global Workspace Theory helps explain the serial nature of consciousness despite the parallel nature of the brain, while BDI provides a mechanism for agents to balance the process of deliberating plans and executing them. To summarize, in LIDA a single dominating

idea that is considered the "most important" idea is broadcast to the many specialized unconscious networks.

LIDA works on the hypothesis that human cognition is based on cognitive cycles, and attempts to model a broad spectrum of cognition in biological systems, varying from low-level perception and action to high-level reasoning. In LIDA perception memory is activated by sensory data and associations between declarative and episodic memory are generated by using the local workspace. There is also an attention mechanism that works via attention "codelets" that compete for selection in the global workspace. Once selected, the codelet is broadcast throughout the system, affecting memory, learning, and action selection. By utilizing an attention mechanism, LIDA is able to manage real-time operation, reducing the amount of information processed. MLECOG's organization (see Chapter 7) is similar to LIDA's except it does not use cognitive codelets but a simpler mechanism of mental saccades. It uses intrinsic motivations to manage goals and stimulate the agent's behavior. The mental saccade mechanism is simpler than the coalition of cognitive processes used in LIDA, since it establishes a cognitive process only after the focus of attention is shifted to the selected part of associative memory. Other competing activations of concepts and their associations are not cognitively recognized in MLECOG.

Similar to LIDA, is the CAMAL [52] architecture. It makes use of affect and affordance based learning derived from the concept of BDI [51] and uses a "motivational blackboard." CAMAL makes its decisions via a control language that is grounded in the theory of affect, which Davis argues subsumes emotion. While similar in how it handles motivations, MLECOG, does not rely on a control language or motivational blackboard, but instead, as discussed in Chapter 7, upon the interplay generated by semantic memory, attention switching and motivational blocks. CAMAL has a type of attention switching in the form of its affect and affordance system. Although this allows it to weigh its beliefs, processes and control the economics of its processing, it lacks a distinct attention switching mechanism. CASE [53] is a similar architecture; however, it makes use of distinct perceptual systems, context data extrapolation, a working memory, and unconscious processing of information. However, it lacks some key features of MLECOG, such as memory saccading and semantic, procedural and episodic memories.

CHARISMA [13], [14] also shares some similarities to LIDA, as both models make use of GWT. However, CHARISMA, relies on a bottom-up Alife (artificial life) approach, in which users don't need to predefine skills and knowledge for the agent to acquire, but that the knowledge/skill acquisition is determined by the simulated scenario and what the agent needs to do to satisfy its own motivations. CHARISMA is also designed with the social cognitive

development in mind, which can be considered essential for learning in open-ended environments.

The Ikon Flux architecture [54] represents knowledge by using predictive models grounded in the machine's operations. It works by treating goals and desired states as target models (e.g. models that must be learned or approximated as best as possible), and uses anticipation and competitive reactive planning at multiple levels. The focus of attention and importance of objects is handled via a limited attention control with thresholds. The attention control also processes data from the environment and the agent itself. Self-growth is implemented at the structural level and in [55] it was evaluated as one of the most promising solutions to build a system with human-like intelligence.

Cox suggested that to advance cognitive architecture a full integration of perception, cognition, action, and meta-cognition defined as thinking about thinking is needed [56]. In his MIDCA architecture he builds a symmetrical structure for managing goals in the mental domain (meta-level) to implement goals in the physical domain (object level). Goals can arise from the sub-goals or discrepancies between observations and expectations. New goals are inserted to explain these discrepancies. MIDCA uses cognitive action-perception cycles that contain perception, interpretation and goal evaluation followed by intention, planning, and action evaluation. These cycles monitor the planning process, goal accomplishment, internal state of the system, or reasoning on the meta-level. Although meta-cognition has a potential to introduce new motivations, there is no clear mechanism to do so. Goal insertion creates new goals but only if there is disagreement between system expectations and experience. This is not much different from curiosity learning, which as demonstrated in MLECOG, is not as effective as it could be in environments that change due to the agent's actions [57].

Dorner's theory [58] uses demands (related to urges) to motivate the agent's action in his MicroPsi architecture. These demands include the need for things like food and energy or even socialization and novelty. It tries to implement some basic emotions in the form of pleasure and displeasure (similar to the concepts of pain and pleasure, but with more emotional overtones), to reflect how well the agent's urges are satisfied. An unsatisfied urge will deviate from its targeted range. The agent's ability to satisfy urges and its knowledge about how to function is measured in the form of competence and certainty. Emotions in MicroPsi are mental responses of the system to certain urges and are influenced by goal selection thresholds, certainty, resolution level of perception, and readiness for action. Objects, events, categories, actions, scenes and plans are represented as hierarchical networks in memory. Emotions modulate intrinsic aspects of cognition

and action selection. MicroPsi agents use reinforcement learning based on pleasure/distress signals related to the satisfaction of drives. Like ML, MicroPsi uses pain signals to indicate the level of satisfaction of needs. MicroPsi uses pre-specified motivations to develop, such as physiological motivations (fuel, intactness), cognitive motivations (certainty, competence, aesthetics), or social motivations (affiliation) [59]. The ML approach discussed in this dissertation differs from this as it can introduce new motivations and needs not pre-specified by the designer.

Another architecture that uses emotions, and is, in fact, designed around them, is EmoCog [60]. It works to integrate emotion generation and emotional effects in the context of cognitive processes, and tries to unify various models of computational emotions while fully integrating with work done in cognitive architectures. It provides a general framework to reconcile and unify existing computational models of emotion into a cognitive architecture. EmoCog's creators hope to be able to reproduce human behavior with greater fidelity by considering both when emotions can aid us in decision making and when emotions can lead us astray.

It is generally considered by most researchers that an autonomous system should be capable of surviving in diverse environments without human intervention. Since the definition of autonomy essentially implies this, the idea is in agreement with this dissertation. In fact, in this dissertation, an autonomous intelligent system is defined as being capable of surviving in a hostile environment. This is thought necessary, since a hostile environment stimulates the development of the cognitive capabilities of an intelligent system.

While autonomous systems do not have to be intelligent, autonomy is useful for better learning and effective performance in an intelligent system. In [55] an autonomous system was described as one that performs tasks in a diverse environment through learning and adaptation that help the system accomplish its high-level goals. MALA [61] tries to achieve autonomy by combining symbolic reasoning and planning capabilities with specialized perception and actuation. It tries to bridge the gap between sensor-motor and cognitive components by obtaining information from the world without labeling objects, and learning to manipulate nontrivial objects and navigate difficult terrain.

Resource management is often at the core of problems for an autonomous intelligent system to address. Another highly important goal for autonomous intelligent systems is survival, which includes maintaining "fuel" levels, avoiding hazardous situations, and avoiding "damaging" actions by other agents. The ML approach and associated MLECOG architecture

(discussed in detail in Chapter 7) presented in this work provides a motivational development mechanism to engage such goals.

Goertzel's CogPrime architecture [62] is built around the idea of "(achieving) complex goals in complex environments using limited resources", via the execution of procedures it believes have the best chance of succeeding in a given state of the environment. The architecture represents knowledge using generalized hypergraphs in which links can point to links or nodes, nodes contain embedded hypergraphs, and atoms are used to represent percepts procedures or concepts. CogPrime also utilized a global memory that is simultaneously localized at distributed.

ISAC [63] is another architecture that focuses on the problem of complex goals in a complex environment by providing a framework and architecture for robots to complete complicated tasks in situation where several task constrains may have been changed from what was previously observed.

Hawes recently did a survey of progress made in cognitive systems [64], in which goal oriented behavior and the generation of goals was the main focus. He argued that developing a management system or framework for machine motives should be explored. His survey reviewed a number of approaches that included some form of motive management. These approaches included system like Soar, Goal and Resource Using architecture (GRUE) [65], and Motivated Behavior Architecture (MBA) [66]. Systems with motive management usually either set their goals independently from one another, as in the "distributed integrated affect cognition and reflection" architecture (DIARC) [67], or they have an arbitration step as in GRUE. Most of the systems perform some form of representation of resources and set priorities for goal selection. DIARC, for example, considers temporal and behavioral dependencies when scheduling (several) goals and assigning implementation urgency.

Several approaches to goal generation have been presented in literature. Goals can be subject to chance, divided into subgoals, abandoned or generated from scratch. The MIDCA architecture can choose from two goal generation algorithms [68]. In one approach [56] goals arise from detecting discrepancy between observed sensory input and expectations derived from previous observations. The agent generates goals to remove the perceived discrepancy. Although the MIDCA system is still under development it is an important extension of goal-driven cognitive architectures.

One of the more significant deficiencies in current efforts toward creating an autonomous cognitive intelligence is the lack of true autonomy. Many efforts focus more on implementing goals designed by humans rather than enabling the nascent intelligence to create the agent's own

motivations and goals. Focusing on human designed goals is believed to be too restrictive to attain human-like intelligence in a machine. Agents cannot be fully autonomous when they are constrained by waiting for commands or by overly strict built-in directives. A much more viable prospect is if the "predefined" goals are only those needed to "survive" and the agent is self-motivated to generate and pursue its own goals full autonomy. This approach gives the agent only as much forced direction as is needed to "bootstrap" its learning process.

Although goals may be derived in other systems, only in the motivated learning approach presented in this work do they result in new motivations. This makes a motivated learning agent better fitted to work in complex environments where it must learn dependencies between events, actions and resources. For instance, hierarchical reinforcement learning (RL), which may also derive and use subgoals, is not as efficient as a motivated learning agent that creates abstract motivations and related goals, as demonstrated using comparative tests (see Chapter 5). The main difference is that in hierarchical RL the subgoals are created only when the system works on a goal that refers to this subgoal as needed for the completion of the main goal. In existing architectures either goals or motivations (or both) are given and the agent only implements these goals. In the AGI domain there are efforts to equip the agent with mechanisms for goal creation (e.g. [69]) but they are not implemented in embodied agents.

Motivations and goals are both generated and managed as part of system operation and are the foundation of the MLECOG architecture. While motivations are important for driving the system and providing it with a basic learning capabilities, there are other issues to be addressed as well, such as symbol grounding and input processing. However, these issues are currently beyond the scope of this work. While they are briefly touched upon here and in Chapter 7, the lack of a fully functioning memory structure for the MLECOG model (see Chapter 7) made it impossible to properly investigate symbol grounding. Input processing is simply beyond the scope of this work's investigation, since it deals with areas such image processing and other sensory related processing, which in spite of a great deal of work already done requires more investigation.

Symbol grounding is a common problem that is the result of an agent manipulating symbols that lack "meaning" or grounding in reality. Humans manipulate symbols, however, they are grounded by experience and associations, which are attached to them in our memory structures. Therefore, to ground a cognitive agent, a memory structure related to its inputs needs to be developed and associated with motor actions. However, processing all sensory input simultaneously is not feasible, hence mechanisms to direct an agent's attention and focus processing capabilities are also important.

Additionally as discussed briefly in the Research Overview (Section 1.2), relying too much on pre-specified goals could limit the agent's ability to learn. And while this might be sufficient for a specific goal or domain, it will cause significant issue for a more general autonomous unit. One of the significant philosophies behind the MLECOG architecture is that it is preferred to give the agent the underlying structures and tools needed to develop, rather than predispose the agent to particular task of environments. Doing this may slow the rate of development, but it may also lead to more effective and capable AI.

One of the biggest hurdles to creation of a complete and working cognitive model has been the lack of understanding of how humans think. How individual neurons work in the brains is reasonably well understood; however, how neurons work together to achieve functionality is still a mystery. We know that groups of neurons form mini-columns and columns; however, the "algorithm" by which they operate is unknown. There has even been some study of the mini-columns in the neo-cortex [40]; however, understanding of how the mini-columns interact with one another and the rest of the brain is limited.

2.4 Architectural Requirements for Embodied Cognitive Systems

Work on cognitive systems requires a working definition of intelligence, hopefully general enough to characterize agents of various levels of intelligence, including human. (Note: portions of this section are drawn from a recent journal publication [24] of ours.)

Goertzel defined intelligence as "the ability to achieve complex goals in complex environments, using limited resources" [62]. Goertzel's definition does not require that the system learns, and may imply that goals are given to the system. It is argued that while some goals can be predefined, an intelligent system must be able to generate and pursue its own goals. Wang defines intelligence as the "ability for an information system to achieve its goals with insufficient knowledge and resources" [43]. Wang's definition implies a need for learning new knowledge, however, it focuses only on insufficiency of knowledge and resources, and neglects potential threats to the agent inflicted by other agents. Our definition of embodied intelligence is similar and can be formulated as follows [19]:

Definition:

> Embodied intelligence (EI) is defined as a mechanism that learns how to survive in a hostile environment.
>
> A mechanism in this definition applies to all forms of embodied intelligence, including biological, mechanical or virtual agents with fixed or

> variable embodiment, and fixed or variable sensors and actuators. Implied in this definition is that an EI interacts with the environment and that the results of actions are perceived through its sensors. Also implied is that the environment is hostile to the EI.

The survival aspect of our definition is implemented in system motivation to avoid predefined external pain signals (which include not only limited resources but also direct aggression from the outside).

It is argued that a hostile environment stimulates development of the cognitive capabilities of an intelligent system. The system learns based on the results of its actions, thus, it must be able to adapt and change its behavior to improve its chances for survival. Many researchers believe [43], [45], [48], [55] that an intelligent autonomous system should be able to survive without human designer intervention in diverse environments. Therefore, this research accepts autonomy as one of the requirements for a cognitive system.

Cognitive systems can be grouped into three categories [70]: symbolic, emergent, and hybrid. Symbolic systems are just what they seem: they rely on symbolic representation to manipulate data. Alternatively, emergent systems are systems whose intelligence "emerges" from processing tremendous amounts of data, have long training time, and a genetically modified interconnection structure with the goal of reaching human-like intelligence. This work focuses on a hybrid approach, where we use a combination of symbolic manipulation and emergent development. To clarify, symbol grounding is used to obtain representations of objects in the system's associative memory (upon which it can later perform symbolic manipulations), and we have predetermined functional blocks that correspond to regions of the human brain in an attempt to emulate the functionalities of regions like the hippocampus, amygdale, and thalamus. Hence, knowledge, skills, and drives emerge in such a system through interaction with its environment.

Designing a cognitive architecture requires both knowledge and understanding of the level of importance of artificial mind components and their functionalities, and which components are most essential for intelligent cognitive processes. In the human brain, these building blocks were gained through the process of evolutionary development. Many of the architectures proposed over the last couple of decades take some inspiration from the human brain's organization and psychology. Both neuroscience and behavioral psychology have been providing knowledge about various regions of the brain, how they react during various cognitive tasks, and how we think and behave in general. While full understanding of the brain and human thought processes may never occur, some structural elements can be derived for use in different

cognitive architectures as a result of our understanding of the brain's architecture. And, while it is not required that we follow the organization of the brain in developing artificial cognitive intelligence, it is the only source of information in which we have both an internal and external view (external in the sense that we can observe from the outside, and internal via observation of our own thought processes) of its operation. Essentially, the brain is the only known implementation of a fully conscious, intelligent mind; hence, using the organization of the brain as a basis for developing the proposed cognitive architecture makes sense.

The brains of living organisms evolved to satisfy their needs based around survival and procreation, however, a human designed artificial intelligence system will have its basic needs defined by humans (largely via its embodiment). Artificial sensors and actuators placed within the agent's embodiment will replace biological sensor and motor abilities, and they will be connected via wires and data buses rather than the axons and dendrites of neurons. Unfortunately, technology cannot match the complexity and density of the sensory inputs available to the human body, nor can it match the level of complexity and overall flexibility provided by the human skeleton's muscle and overall motor structure. Furthermore, the material used to construct an embodied agent will differ in terms of strength, durability, sensitivity to the environment, energy consumption, etc., and hence, will put different constraints on the machine's ability to interact with the environment than those of the human body.

It is highly unlikely that we will ever duplicate the human mind with high fidelity and given that the physical embodiment will have its effect on the development of an artificial mind, we will likely never see a machine that acts, feels, and thinks like a human does. Therefore, long-term goal should not be to develop a machine that mimics human behavior exactly, but one with human-like capabilities to learn, accumulate knowledge, think, plan its actions, anticipate, be able to solve problems, and have a natural form of intelligence with a conscious, self-aware mind. Despite the almost certain "alien-ness" and different needs and motivations, we can hope that the created intelligences will be sufficiently relatable and willing to collaborate with humans in problems solving, knowledge discovery, knowledge distribution, etc.

When setting the requirements for a cognitive architecture, the roles the brain (or nervous system in general) plays in living organism should be considered. These roles should be identified as a way to control functions within agents' embodiments, but also to provide motivations, set goals, perform learning, and control actions. Some roles are easier to identify in the form of functional blocks that have to be present in all architectures. This includes blocks responsible for perception and semantic memory, motor control, and reactionary response to pain

signals. Other blocks, such as working memory, episodic memory, pain management, planning, attention, etc., involve dynamic interaction between various blocks. How exactly these blocks coordinate, cooperate, and compete with one another is less certain and difficult to determine.

There is an increasing amount of evidence [71] that cognition is a result of interaction between distributed brain areas in dynamically established large-scale functional networks (LFSN). Large-scale functional networks are defined as interconnected brain areas that interact to perform specific cognitive functions. Advances in medical imaging have made the discovery and analysis of the LFSNs possible. Imaging techniques such as fMRI and diffusion magnetic resonance imaging (DTI [72] and DSI [73]), help to investigate what happens in the brain and observe and conceptualize the organization of the LSFNs and their dynamics. Different areas within the brain play different roles, with some acting as controllers to engage other areas, while others provide specific sensory or conceptual content. The more we learn about how the different parts of LFSNs interact with each other and different LFSNs, the easier it will be to develop human-like machine intelligence.

It is argued that LSFNs are involved in many functions of the human brain, such as emotional, social, and cognitive capabilities, which gives rise to the domain-general functionality of the human brain [74]. This argument points towards constructionist cognitive architecture, since such architecture uses distributed mappings between the brain structure and its functionalities. A recent meta-analysis [75] demonstrated that several brain regions (such as the amygdala, anterior insula, anterior cirgulate cortex, and orbitofrontal cortex) are activated to different degrees in different emotional situations, contradicting earlier convictions that emotions (such as fear, disgust, sadness, and anger) are situated in distinct neural modules in the brain. Additionally, certain areas (like the amygdala) are not only involved in emotional responses, but are also involved in learning, social cognition, perception, and attention.

Hence, based on what we have learned about the brain's organization in terms of both "individual" components and how they interconnect via LFSNs, this work tries to organize the functional blocks of the MLECOG model to better satisfy what is known of the functional requirements for an intelligent cognitive system. We try to provide the necessary desired functionality and show how the artificial mind functionality can be accomplished through interaction between the various functional blocks. It is expected that as in the real brain, more advanced functional blocks will, in the future, be built on the lower level, easier to implement functionalities. In other words, we hope future work can "evolve" the design by building upon and improving MLECOG functionality. To some degree, this functional organization will

resemble the subsumption architecture proposed by Brooks [BRO86], but it will be arrived to by functional analysis, rather than emerging through the developmental process.

Individual functional blocks can be studied separately and then integrated into the cognitive architecture. Granted, because of the level of interconnectivity, changes in one block will likely necessitate changes in other blocks and vice versa. This type of structural partitioning facilitates the development and adaptation of industrial hardware and software solutions, such as the hierarchical temporal memory developed by Numenta [76] (considered as a cortical learning algorithm by its creators), and IBM's neural network chip with one million programmable neurons, 256 million programmable synapses and 46 billion synaptic operations per second per single watt of dissipated power [77]. Using a modular approach to development helps accelerate research and testing of the cognitive architecture proposed in this work. The functional blocks of the proposed architecture are described in Chapter 7.

The model developed in this research takes a hybrid approach where self-organization and learning play a role in each functional block. Thus, the various elements of cognitive intelligence emerge from interaction with the environment in the form of various motor skills, motivations and needs, cognitive reasoning and memories. This emerging property is considered necessary for the system to be able to work within different environments and perform a variety of tasks using the sensor and motor actuators it is provided with. In addition, it is assumed that embodied systems use raw percepts to recognize objects and observe results of their actions. Thus, concepts developed and represented by a cognitive system must be grounded in perception using sensory inputs as described in [78].

CHAPTER 3: BASIC MOTIVATED LEARNING AND GOAL CREATION

3.1 Introduction

Goal Creation and Motivated Learning have their roots in Reinforcement Learning, since a Motivated Learning agent receives reinforcement when it successfully deals with predefined needs. Reinforcement Learning, which has become main-stream in machine learning since the 1990s, covers a range of algorithms and applications [79], as discussed below. For example, Morimoto and Doya [80] showed that using reinforcement learning it was possible to estimate the dynamics of a pendulum via estimation of hidden-variables in the pendulum "swing up task." Xiao and Tan produced the Reinforcement Learning variant TD-Falcon [81] (Temporal-Difference Learning, Cognition, and Navigation), which applies Reinforcement Learning to navigation as a generalization of adaptive resonance theory. OReilly's [82] primary value and learned value (PVLV) approach is an alternative to the traditional TD based Reinforcement Learning in that it directly associates rewards and stimuli using a Pavlovian conditioning scheme. Work by Bakker and Schmidhuber [83] on hierarchical Reinforcement Learning (HRL) and HASSLE [84] with subgoal discovery has tried to alleviate some of the pitfalls of traditional Reinforcement Learning, such as the "credit assignment problem," [85] by assigning hierarchically discovered "subgoals" for the agent to pursue. The credit assignment problem arises from the temporal difference (TD) mechanism that Reinforcement Learning uses to spread the value of a reward to earlier stages. In other words, it spreads a reward to all actions following the last reward, since it has no way of knowing which actions were useful toward acquiring the reward. Despite subgoals offering improved performance by essentially subdividing the steps needed to receive the reward, they are still serve predefined goals and this work believes that it is important for an agent to be able to develop/define its own *goals* and *motivations*, not merely pursue predefined goals. (Please note that much of this chapter is taken from a previous work, [20], regarding the initial development of the ML algorithm.)

The concept of Motivated Learning also has distinct similarities to BDI (belief-desire-intention) agents [51] such as the need to create a kind of internal representation or belief of the state of the environment. In the case of a ML agent, its representation is grounded mostly in the link between its perception and its semantic knowledgebase of the world around itself. Desires in a BDI agent correspond most closely to the concept of motivations as discovered by the agent in ML, and lastly, intention corresponds to the action of choosing a goal and its desired effect. The

significant difference is that BDI agents have predetermined desires, while ML develops them through interaction with the environment and learning.

Both Reinforcement Learning and Motivated Learning use artificial curiosity. Artificial curiosity is a kind of undirected motivation to explore and learn. It has mainly been used to drive agents to explore outside their "comfort" area; to cause them to explore the "state space" without a particular reason. One such example of a curiosity implementation, an Intelligent Adaptive Curiosity algorithm (IAC) was proposed by Oudeyer et al. [86], [87]. The IAC algorithm attempted to allow a robot to function based on curiosity in continuous, noisy, inhomogeneous environments, and allowed autonomous self-organization of behavior toward increasingly complex behavior patterns. Roa et al. [88] took Oudeyer's work further and attempted to combine curiosity in Reinforcement Learning via a combination of active learning and reinforcement using both intrinsic and extrinsic rewards, with the intrinsic portion based on Oudeyer's work. The problem with IAC is that while it eliminates some of the issues with purely curiosity based learning, it is still learning without purpose. The agent will try to learn everything new, rather than optimize for specific skill or higher levels of performance.

Pfeifer and Bongrand [2] believe that an agent's motivations should emerge as part of the development process. And Merrick pointed out that RL agents do not have internal drives to maintain their resources within an acceptable range, and that the designer cannot realistically determine all goals [89]. Merrick introduced motivated reinforcement learning (MRL) and used motivated exploration in video games [89]. Motivated learning based on the need for resources was used to develop a coordinated learning strategy in a multi-stage stochastic game in [90].

3.2 The Goal Creation System

Much of the material discussed in this chapter was initiated and grew from the initial research done on Goal Creation and Motivated Learning (see the associated publications [20], [91]). Goal creation is described as a simplified variation of the motivated embodied intelligence idea as described by [19]. It takes the basic concepts of goal creation via simple pain based motivation and attempts to learn how to resolve its basic needs. Goal creation differs from other methods, such as goal inference [92] or sub-goal generation as discussed in [93] and [94]. Sub-goal generation is typically an adoption of existing goals rather than creation of new goals by the agent. Such limitations in an agent's ability to set the goals for itself would limit its autonomy and may compromise its performance.

3.2.1 Pain Based Goal Creation

The general motivation of a ML agent is to succeed in an unknown environment. We define motivations as follows:

Definition:

An agent's *motivations* are to satisfy its needs, which means that the agent must reduce the associated pains below threshold.

The view of motivations in ML is in agreement with views of psychologists like Maslow [95]. This approach, although based on predefined "physiological" needs, can lead to higher level needs like safety, friendship etc. Maslov predefines these needs in his hierarchy of needs; however, the ML agent's hierarchy of needs evolves automatically as the agent learns. We refer to these "physiological" needs as primitive pains.

Definition:

An agent has predefined *needs* (for instance need for shelter, food, or energy) inherent or built-in to its design. *Needs* are the conditions that the agent must meet to satisfy its motivations. Motivations represent the desire (or need) to fulfil *needs*. As an agent adds new needs, it adds new motivations corresponding to those needs.

Definition:

A *primitive pain* is related to each predefined need and is defined as a measure that reflects how far the agent is from satisfying its need. The pain is larger if the degree of satisfaction of a need is lower. The agent acts on its need only if the pain is greater than a prespecified threshold. *Pain reduction* in ML is loosely equivalent to the generation of a *reward* in RL.

Primitive pains are defined as part of the embodiment of an agent and are generally treated as external signals, as they exist the time of an agent's creation. The environment in which the agent is embodied (and the embodiment itself) operates based around some number of rules that define its interaction and perception of the environment. Learning to resolve its needs correspondingly leads to the agent learning how to use the rules of the environment to its advantage. A ML agent learns to interact with and change its environment to its advantage, rather than simply respond to the environment's state.

The machine's development is driven by a simple built-in pain based mechanism. The primitive pain (equivalent to a negative reward signal) comes from the hostile environment, and

forces the machine to respond. A primitive pain leads to a primitive goal and its satisfaction through proper action triggers the development of higher-level pain centers and creates higher-level motivations. By this approach, the agent discovers the rules that govern relationships between the objects that affect its perception and its pain signals. A motivated learning system related its goals to desired conditions for the environment.

The motivated learning mechanism uses basic pain detection and learning units shown in Figure 3-1. In this figure sensory and motor neuron activities, are represented by single neurons (*S* and *M*), although, distributed representations of sensory objects or motor actions are more effective and can be used in this method. A similar simplification is used to describe the neural network organization in Section 3.3.4.

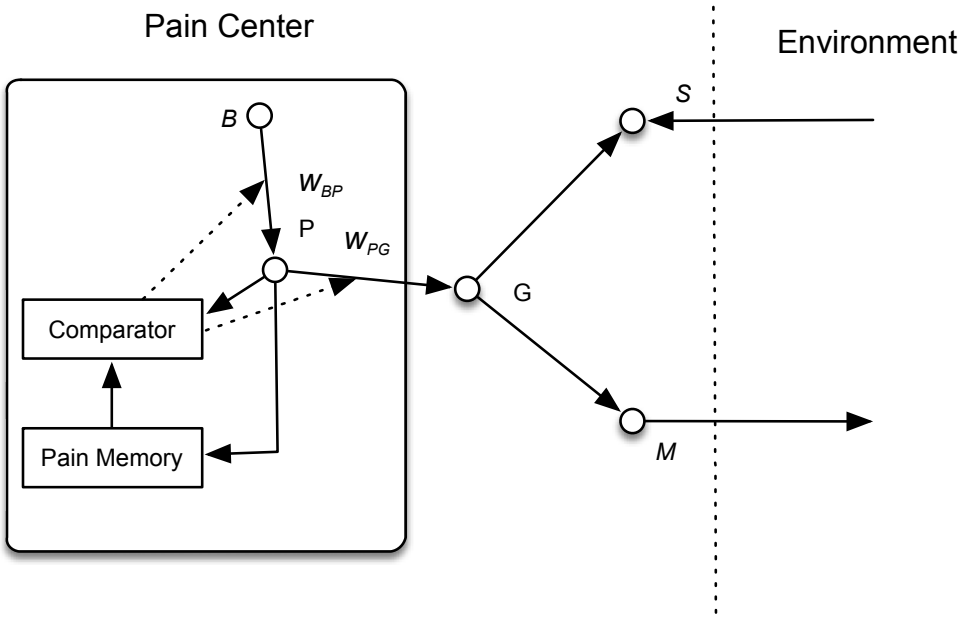


Figure 3-1. Basic pain detection and learning unit.

The pain detection center responds to the input pain signal and represents the negative stimulus that the machine needs to minimize. If the pain exists due to the absence of a certain resource that the machine may need, then a "correct" action (or series of actions) that results in finding such a resource in the environment will reduce this pain signal. In Figure 3-1, the actual pain level *P* is controlled by a bias *B* (which is linked to the resource's level) times w_{BP} weight. A pain memory center stores the delayed pain level. The newly measured pain signal is compared

with the previous pain signal via a comparator mechanism and $w_{BP}$and w_{PG}weights are updated accordingly.

Increasing pain signals force the machine to explore various motor actions by stimulating the goal neurons, *G*, through initially random connection weights (as represented by w_{PG}). The machine searches for the proper action starting from the one with the strongest activation (strongest weights connecting to the pain stimuli). All goal neurons and pain neurons compete with each other using Winner-Take-All (WTA) competition (see Figure 3-3 in section 3.3).

3.2.2 Representation Building

The winning pain signal forces the machine to explore its environment to reduce the pain. A solution can be found through exploration or observation of another entity performing a desired task [96]. In doing so, the machine discovers relationships between objects observed through its sensory inputs and actions it performs. Observed concepts are not predefined but emerge as a result of successful operations. Thus, the concept of an object is related to useful and predictable properties the object may have with respect to the machine's objectives and its ability to fulfill them with the proper action(s). In connectionist networks, objects are recognized mostly through correlation and self-organization of similar features, while feature invariance building is accomplished through continuous observations and correlation through time. Reliable perception and invariant representation building are active research topics and their full discussion is beyond the scope of this work. Thus, in the description of the ML mechanism and in simulation experiments pre-defined concepts and motor operations are used for simplicity. This however does not constrain ML's ability to learn new concepts and skills.

For the optimum development of concepts and related skills, an agent operates best in a protective environment that gradually increases its complexity. Thus, the developmental process must be monitored and the learning environment structured to facilitate the machine's learning. An important observation is that representation building, (which results from the association of observed actions with the internal or external reward), comes from the motivation of the machine to act, whereas motivations to act come from representation building. New representations may yield new motivations to protect or acquire desired resources while new motivations force the machine to discover new ways of solving related problems.

3.2.3 Creation of Abstract Pains and Motivations

As soon as the agent discovers a valid action, any inability to perform this action in the future (lack of resources or deprivation of motor actions) will result in an abstract pain. For instance, if the agent needs a certain resource to satisfy its primitive pain, and the resource is not available, this creates an abstract pain signal. This abstract pain motivates the machine to explore how to obtain the missing resource. An abstract pain center uses a similar organization to trigger this motivation (as shown previously in Figure 3-1). However, an abstract pain center is not stimulated from a physical pain sensor; it only symbolizes an internal pain from not having sufficient resources to lower its primitive or abstract pain. Thus, a mechanism is needed to generate this internal pain signal.

At any given time, the machine may experience a number of different pains, each one triggering different goals. Changing pains change the machine's motivation for action, concentrating its efforts on reducing the winning pain. The same mechanism that created the response to a lower level pain will govern learning how to respond to abstract higher level pains. It will result in the emergence of a complex system of drives, values, and concepts about the observed environment. In addition, this motivating mechanism will stimulate the machine to interact with its environment and to develop its skills.

For instance, an agent may suffer from a primitive pain when it is hungry. When "food" is available and the agent "eats", the primitive pain is relieved. An abstract pain center responding to lack of food is created. An inhibitory link is developed between the sensory signal representing the presence of "food" and the abstract pain center, and detection of "food" can inhibit the abstract pain. When "food" is not available, the agent tries to find a solution to reduce the "abstract pain" – lack of food. Thus, the agent may feel the abstract pain (no food) without feeling the lower level pain (not hungry).

Motivated by a dominant abstract pain, the agent is forced to explore to reduce this abstract pain. Eventually, the reduction in the abstract pain of no food may result from the action "open" combined with the sensory object "refrigerator". This indicates that the abstract pain triggered by the absence of "food" will be associated with the sensory-motor pair "refrigerator"-"open". In the case of the machine opening the refrigerator and seeing food, the abstract pain "no food" is suppressed. Strengthening the interconnection weight between the abstract pain "no food" and the goal that alleviates this pain –"refrigerator"-"open" by successfully finding food in the "refrigerator" will reinforce the performed action. "In addition, an expectation link from the motor action "open" to the sensory neuron "food" is built; thus "food" will be expected as the

result of the action “refrigerator”-“open”. This expectation link will be used for planning future actions with an expected response from the environment. [19]” This process is illustrated using Figure 3-2.

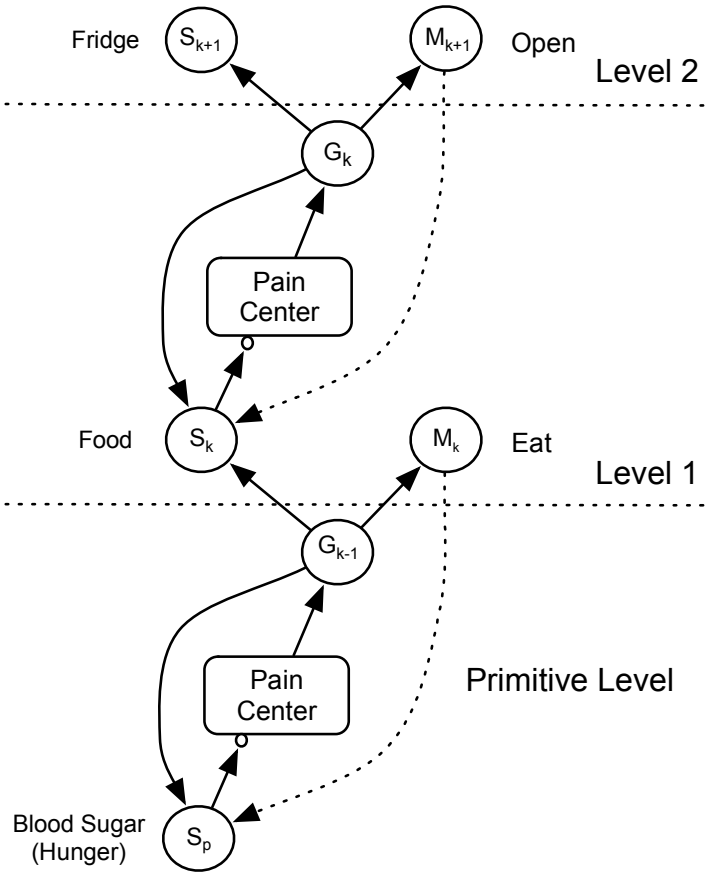


Figure 3-2. Development of abstract pain signals.

This abstract pain and related goal hierarchy can be further expanded. If the agent “opens” the “refrigerator”, but the “food” is not found, the machine needs to try other options to suppress this abstract pain. It may explore the environment or use instruction. At this stage either RL or random search may be used. Once the machine “spends” some “money” (in a store) to buy “food”, “food” becomes available and the level 1 abstract pain (“no food”) is reduced. Such an action is rewarded by an internal reward signal that depends on the effectiveness of the pain reduction. So, in the future, the action will be strongly stimulated by the abstract pain center “no food in refrigerator”. However, when “money” is not available, an abstract pain center on level 3

is activated with an inhibitory link from the sensory percept of “money”. Subsequently, the machine needs to learn how to solve the abstract pain on level 3 related to lack of “money”, etc.

In motivated learning, at every step, the machine finds an action that satisfies its goals, and this action and the involved representations may result in creating further motivations and abstract goals. Therefore, via this simple mechanism, the machine simultaneously learns to match the goals with deliberate actions, the expected results of actions, the means to represent and obtain objects, and relations among objects. It learns which objects are related to its motivations. The machine governs the execution of actions to satisfy its goals and manages the goal priorities at any given time.

3.3 Network Organization

The simple implementation of ML schema uses a neural network where each sensory neuron represents an object and each motor neuron represents an action.

The ML system’s neural network, in addition to sensory S and motor M neurons, contains pain neurons P that register the pain signals, and goal neurons G responsible for pain reduction. Selected pain neurons are connected to the external reward/punishment signals. In RL these neurons receive a reward or punishment signal according to the training algorithm, and in ML they receive primitive pain signals that directly increase or decrease their activation level. In ML abstract pain centers are created through the goal creation mechanism [19], [97], and are activated via an interpretation of sensory inputs. Through these inputs the agent perceives the state of the environment, results of its actions, and the obtained reward/punishment. A goal is an intended action that involves a sensory-motor pair. To implement a goal the agent acts on the observed object. All goal neurons and pain neurons are subject to Winner-Take-All (WTA) competition. Once the highest pain above threshold is determined, the winning goal is established by evaluating w_{pg} weights for a given pain.

There is one-to-one correspondence between sensory percepts and pain centers as the agent may generate an abstract pain related to each observed resource, but there are no direct links between *S* and *P* neurons. There are feedforward connections between the pain and the goal neurons, between the goal and the motor neurons, and feedback connections from the goal to the sensory neurons. The feedforward connections to motor neurons activate motor responses, while feedback sensory connections help to focus attention on the object on which the agent acts, and provide sensory expectation signals. All abstract pain neurons have a bias input *B* that depends on the state of the environment. Figure 3-3 shows symbolically the structure of interconnections,

between *S*, *P*, *B*, *G* and *M* neurons. In Figure 3-3 an abstract pain center P_k connections to its sensory, bias, goal and motor neurons are shown.

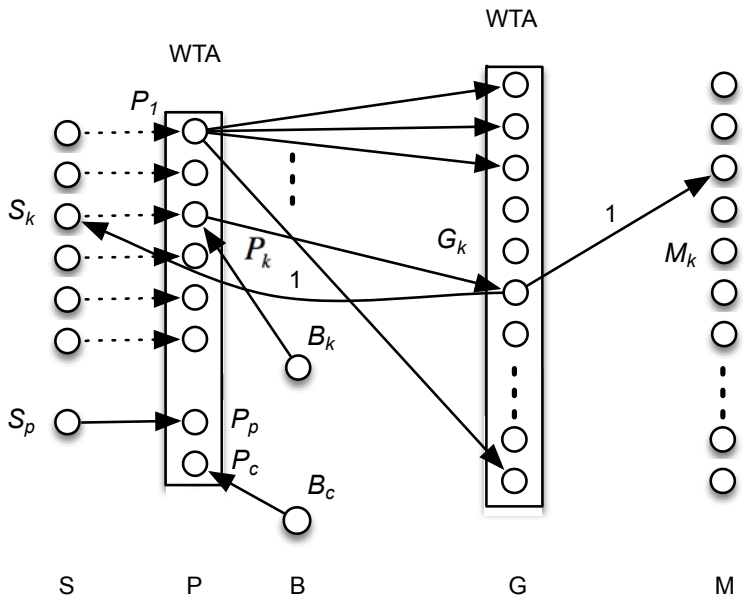


Figure 3-3. Connections between sensory, motor, bias, pain and goal neurons.

A single curiosity neuron is added as a pain signal neuron to indicate machine's desire to explore. It is triggered by the constant pain value equal to threshold. The curiosity pain neuron is linked to all the goal neurons with randomly set weights. As the machine learns, the weights from the curiosity pain to various goals decrease. We can set a threshold for curiosity weights to regulate the agent's curiosity. Thus, after initial learning and exploration of its environment, when all curiosity weights are below threshold, the agent may not act out of curiosity any more.

G neurons are connected via untrainable weights equal to 1 to corresponding *S* and *M* neurons, *P* and *G* neurons are fully connected with trainable w_{PG} weights. There are no direct connections from the pain center neurons *P* to the motor neurons *M*. Figure 3-4 shows trainable connections between bias, pain, and goal neurons as well as additional inhibitory neurons (unavailable resource (UR) and unsuccessful action (UA) neurons) that fire depending on the environment conditions. A UR neuron inhibits goal selection if a resource required to perform the action is not observed on the sensory input, while a UA neuron inhibits goal selection when a desired action could not be completed (due to motor malfunction or adverse environment conditions). UA neurons are normally inhibited by an inhibitory link from an action completed

(AC) neuron, indicating that a motor function can be performed if needed. The AC neuron forms a bi-stable pair with a UA neuron and only one of them is active at any given time. When a desired action (*A*) cannot be completed a UA neuron is activated.

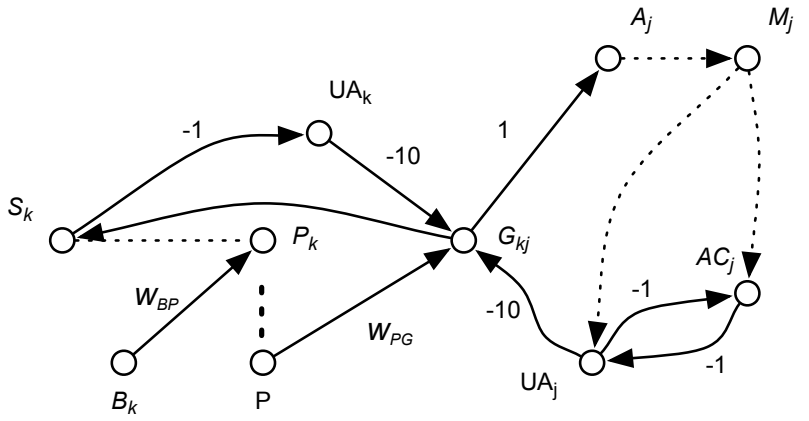


Figure 3-4. Trainable connections between pain, bias, and goal neurons.

In ML, each time an action based on selected sensory-motor pair (S_k-M_k) results in decrease of a dominant pain *P*, there is an increase in the connection weights between this pain neuron and the selected action neuron G (w_{PG} in Figure 3-4). In addition, the bias link strength w_{BP} of the abstract pain neuron P_k associated with the desired sensory input S_k is increased. In a similar way the bias link strength w_{BP} of the abstract pain neuron P_k associated with the desired action G_j is increased. These weights increases gradually establish the need for resources and actions that have helped the agent to reach its goals. At the same time, the lack of these resources or an inability to perform needed actions will become abstract needs (pains) of the learning agent.

The only exception to this is in a curiosity triggered action (when no other pain exceeds the pain threshold). Is this situation, the w_{PG} connection weight is always decreased signifying a decrease in curiosity for this goal. Thus, when the machine can predict the response to its actions, no curiosity action will take place (even if no pain other than curiosity is detected). Furthermore, the connection weights for the other non-curiosity pains are all increased or decreased depending on the effect of the curiosity based action on the associated pain centers.

Sensory-motor sub-networks of the goal creation mechanism include unavailable action neurons (UA) whose role is to inhibit the action associated with neuron *G* from firing if a sensory input required for this action is not present (see Figure 3-4) or, in general, if the selected action

cannot be currently performed. Each UA neuron fires automatically unless it is inhibited by the sensory neuron activation. In addition, the network has an abstract pain center neuron associated with each sensory input. Finally, a fully connected network of *P*-*G* links completes the network configuration.

When pain is decreased, the bias link strength of the abstract pain neuron P_k associated with the selected sensory input S_k is increased. The weights of other links to activated neurons are slightly decreased. The bias signal is associated with probability of the corresponding sensory input activation indicating availability of the resource;

$$B = -log_2(\text{estimated probability}) \quad (1)$$

Other methods of obtaining bias signals are discussed in Section 4.3. They represent a more general environment with desired or undesired resources and actions by other agents. The probability based approach to resource representation in the environment was too limited for us to utilize when trying to represent a more varied environment. It worked by representing the probability as to whether a resource would be visible at a particular time. For example, if food were present 50% of the time, the agent would detect it 50% of the time when checked its food sensor. While this may make sense probabilistically, realistically, it does not, because it would allow for food to be detected one cycle and for it to disappear in the next one. Realistically the food is either present or not. Hence, we moved toward representing objects and resources in the environment with actual quantities. We also included distance between resources and time to complete actions, both of which would have made the issue with probability based resource handling evident.

Reduction of the bias signal reduces the associated abstract pain P_k and triggers learning. However, if the dominant pain increases as result of the selected action, then the interconnection weight between corresponding *P* and G_k neurons is reduced. All pain and action related weights might be subject to a small systematic reduction over time.

Using the bias signal, abstract pain value is estimated from:

$$P(s_i) = B(s_i) * w_{BP}(s_i) \quad (2)$$

where w_{BP} is a bias to pain weight for a given pain center associated with sensor s_i. The w_{BP} value is computed incrementally based on pain change signals that resulted from the action taken as follows:

$$w_{BP} = \begin{cases} w_{BP} + \Delta_{b+} * (\alpha_b - w_{BP}) & if\ associated\ pain\ changed \\ w_{BP} * (1 - \Delta_{b+}) & if\ there\ was\ no\ change\ in\ pain \\ w_{BP} * (1 - \Delta_{b-}) & if associated\ percept\ was\ not\ used \end{cases} \quad (3)$$

where $\alpha_b = 0.5$, sets the ceiling for w_{BP}; $\Delta_{b-} = 0.0001$, sets the rate of decline for w_{BP} weights; $\Delta_{b+} = 0.08$, sets the rate of increase for w_{BP} weights. The pain calculation is based on the "traditional" NN approach. If the item associated with a pain is present, we consider it active, and calculate the value based on the bias level and the aforementioned w_{BP} weight.

Initial weights between *P-G* neurons are randomly selected in a 0-α_g interval. Assume that the weights are adjusted upwards or downwards by a maximum amount α_g. In order to keep the interconnection weights within pre-specified limits ($0 < w_{PG} < \alpha_g$), the value of the actual weight adjustment applied can be less than α_g and is computed as

$$\Delta_a = \mu_g min\left(\left|\alpha_g - w_{PG}\right|, w_{PG}\right) \qquad \text{where } \alpha_g \leq 1. \quad (4)$$

This weight adjustment produces weights that slowly saturate towards 0 or 1. (For quick learning set μ_g is set to 0.5). No other weights from other pain centers to this specific action are changed, so the sum of weights incoming to the node G is not constant. However, all w_{PG} weights from the selected pain center P to all actions A are adjusted to have a constant sum. The actual adjustment of w_{PG} weights is done via

$$w_{PG} = w_{PG} + \delta_p \cdot \Delta_a \quad (5)$$

where δ_p represents how the last action affected the pain. δ_p is 1 if pain was reduced or -1 if it was not affected or worsened.

At the start all B_i-P_i weights are set to 0. The machine initially responds only to the primitive pain signal P directly stimulated by the environment. Each time a specific pain P is

reduced the weight $w_{BP}$of B_k-P_k bias link increases. However, if the action activated by the pain center P is completed and does not result in a reduction of pain P, then the weights $w_{BP}$are reduced.

Since the bias weight B_k-P_k indicates how useful it is to have a desired S_k, bias weight adjustment parameter Δb must be properly selected to reflect the rate of stimuli to a higher order pain center. This rate reflects how often a given abstract pain center P_k was used to reduce the lower order pain P.

Bias links w_{BP} are adjusted to indicate a significance of each abstract pain. Each time an abstract pain is reduced as a result of an action its bias weight is automatically reduced according to $w_{BP1} = w_{BP1}\ (1\text{-}\Delta_{b\text{-}})$ and the bias of the associated abstract pain is increased as $w_{BP2} = w_{BP2} + \Delta_{b+}\ (\alpha_b - w_{BP2})$. This adjustment takes place in two different pain centers as illustrated in Figure 3-5.

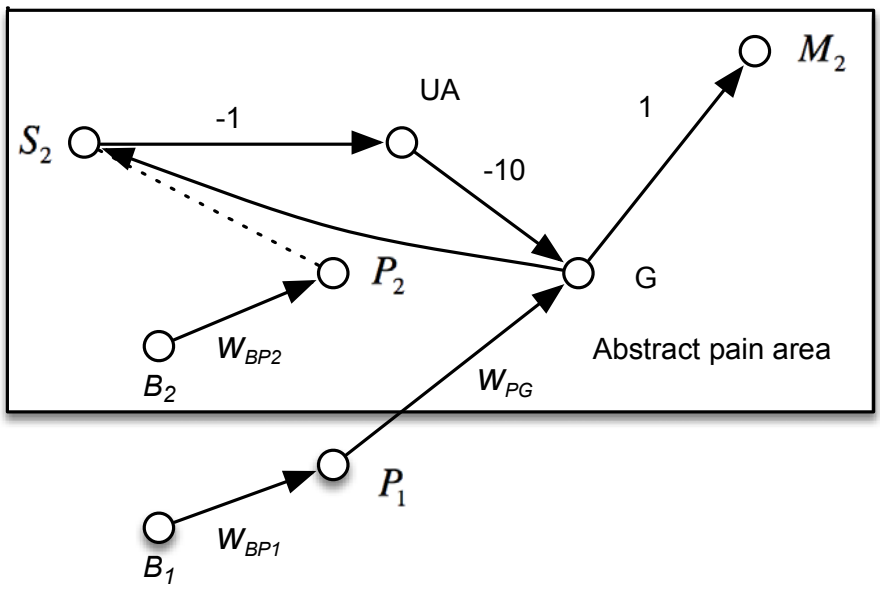


Figure 3-5. Bias weight adjusted after action.

Suppose that as a result of an action A involving sensory-motor pair (S_2 - M_2) the resource on sensory input S_1 is obtained and reduces the pain P_1, then w_{BP1} is reduced and w_{BP2} is increased. The increase of w_{BP2} is due to the increase of its relevance to the agent, while the decrease of w_{BP1} is simply the result of gradual weakening of unused weights. Hence, all unused

w_{BP} weights experience a gradual decline when unused (as indicated in the final segment of equation 3).

When, at the end of the training session, all the inputs to a goal *G* are less than μ_g, the corresponding action is removed together with its entire set of incident links. This case typifies a useless action (like eating money) that did not reduce any pain. Since each pain neuron can be connected to S×M action neurons learning to remove an action neuron A may take on the order of S×M steps.

If a specific action is not invoked for a long period of time its importance in satisfying a lower level pain is gradually reduced. A similar reduction of B_k-P_k links indicates a gradual decline in importance of an abstract pain P_k. This mechanism of lowering the weights to an abstract pain center prevents the machine from overestimating its abstract pain importance by adjusting the relevance of this abstract pain to the lower level pain that was responsible for its creation. Otherwise, the machine can generate higher-level goals even if they are no longer required to support its lower level goals. For instance, if making money is necessary to support living, an internal stimulus may force the machine to make more money even though the machine no longer needs it (or has a sufficient amount to cover its needs for a long period of time).

Evaluation of one's goals may require a more complex mechanism than a constant rate of diminishing an importance of goals that are not activated. However, for now we use this simplifying approach.

The machine uses its goal creation approach to learn what to do and to adjust to changing environment conditions. It is doing so by adjusting pain biases and weights between the pain signals and actions.

We can summarize the procedure by which the Motivated Learning algorithm (MLA) works as follows [22]:

3.3.1 The Motivated Learning Algorithm

1. The agent reads the current state of the environment and establishes biases (eqn.1).
2. Based on the sensory inputs and its internal state, the agent evaluates the internal pains (eqn. 2).
3. The agent checks if the pains were reduced and adjusts the bias-to-pain weights accordingly (eqn. 3).
    a. If a pain was reduced, the agent learns proper behavior using classical reinforcement learning and creates or reinforces an abstract need. Otherwise, it

learns that the action is not useful and reduces the chance of repeating it in a similar state of the environment and the agent.

b. Depending on the outcome of pain reduction, pain-to-goal weights are adjusted (eqns. 4-5).

4. The pains are updated using the degree of satisfaction of its needs (eqn. 2). If a new need was established due to a successful action, the related pain is evaluated as well.
5. A goal is selected based on the existing pains and the state of the environment.
6. An action is performed to accomplish the goal (i.e. reduce the dominant pain).
7. Repeat 1-6.

Notice that the ML algorithm works in real time and the agent interprets inputs from the environment at each iteration. Thus, a goal may be changed if the conditions in the environment change.

3.4 Curiosity and Certainty Learning

How is curiosity learning organized in this neural network implementation of ML? It operates similarly to regular pain-based action with a few significant differences. A curiosity-based action will occur when none of the other abstract or primitive pains is above threshold and the machine still feels curious. In section 3.3 it was mentioned, that curiosity based w_{PG} weights decrease in value to indicate that something was learned. The decrease in w_{PG} weights also indicates an overall decrease in curiosity. When all curiosity-based action weights have fallen below a predefined threshold, the machine will no longer perform curiosity-based actions, unless new concepts that need to be explored are identified.[91]

In existing ML implementations curiosity is implemented as a constant low-level pain just above the pain threshold. This allows the machine to explore the environment when not performing any other pain based actions such as eating food, or working. That's not to say that the machine might not try to eat food out of curiosity.

Let us consider such a situation. Assume that based on its curiosity, the agent observed that eating food reduces its primitive hunger and depletes its food supply. It would then adjust the appropriate pain-action weights, but would not adjust its bias-pain weights, since it did not perform the action based on a pain.

In addition to the continually decreasing curiosity w_{PG} weights, there is another factor that can help in establishing a winning curiosity action referred to as "certainty". Certainty C_G is a measure indicating how certain the agent is about a particular goal. For example, if any of the

w_{PG} weights associated with a specific goal approaches one, then the goal's certainty approaches one. Conversely, if all the values of w_{PG} weights for a particular goal approach zero, we can say that the certainty for that goal also approaches 1.0.

When calculating the goal's curiosity value the w_{PG_C} value is multiplied with one minus the certainty (1 - C_G), because if the agent is certain about a particular action, there is no reason to be curious about it. In summary, certainty C_G and the associated curiosity goal strength G_S are determined as:

$$C_G = 1 - min\{\min_P(w_{PG}), 1 - \max_P(w_{PG})\} \quad (6)$$

$$G_S = P_C \cdot (1 - C_G) \cdot w_{PG_C} \quad (7)$$

where P_c is the curiosity pain and w_{PG_C} is curiosity to goal weight. This means actions that have been determined to be either useful of useless for a given pain will be assigned a higher certainty and therefore not further investigated, while actions with indeterminate capabilities will continue to be examined via the curiosity function. Note that in current ML implementations this only applies to curiosity triggered actions, and certainty does not play a direct role in determining actions selected based on motivations other than curiosity.

3.5 Extensions Toward Subgoals

Some goals need a sequence of steps to implement them. These steps can be treated as subgoals and each subgoal may require specific conditions to implement. This section discusses how a series of goals, each with its own prerequisites, may be implemented within the basic ML algorithm with only minor changes. As previously mentioned, the concept of subgoals is well understood in RL. Thus, this discussion is intended to show the difference between abstract goals in motivated learning and subgoals in reinforcement learning, as abstract goals may in some cases be viewed as subgoals needed to implement a complex goal.

The neural network architecture to manage sequential goals slightly extends the preceding structures shown in Figure 3-3-5 to allow for the creation of sequence of sub-goals. Figure 3-6, below, depicts this modified structure.

The main difference between this structure and the earlier examples is the inclusion of the intention (*I*) and subgoal motivation (*P′*) neurons. The *I* neuron acts as a gate for the *G* neuron,

such that there is one *I* neuron for every *G* neuron, while subgoal motivation motivates machine to implement a subgoal. Unlike the competition between action neurons *G* in Figure 3-3, *I* neurons compete within the second Winner-Take-All (WTA) block to select the current intention.

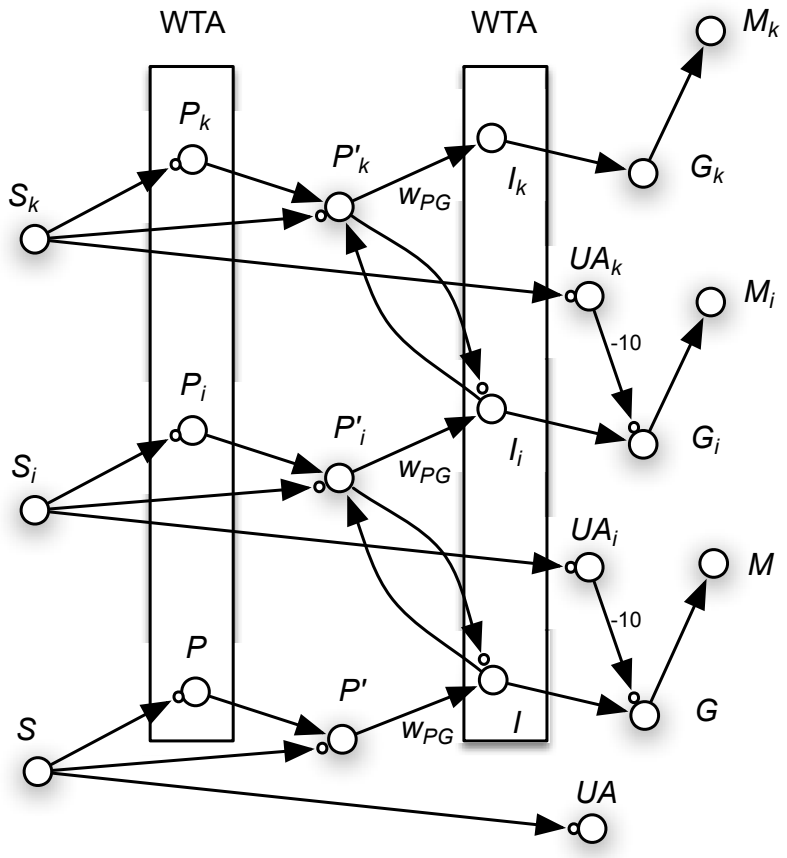


Figure 3-6. Subgoal capable network.

To illustrate the situation, let us assume a network that has already learned all of the necessary associations and that we have a winning pain *P*. This *P* will subsequently activate the associated *P′* and *I* neurons (where the active I neuron is determined by a WTA event among all the available *I* neurons as determined by the weights w_{PG} from *P′* to the set of *I* neurons). Let us assume that the resource required to implement this winning intention is not available. In this scenario, the *I* neuron will attempt to activate the associated action *G*, only to find that it cannot because a UA neuron, indicating that the needed resource or event is unavailable, blocks it. This unavailability of resource S_i will cause the *I* neuron to activate the P_i' associated with the UA

neuron, which will in turn inhibit the current *I* neuron. Notice that P_i' was not initially active since P_i lost the WTA competition to *P*. The now active P_i' neuron will then attempt to activate its own *I* neuron, which we'll refer to as I_i. However, what if the G_i associated with I_i is unavailable as well? The process simply repeats with I_i activating a P_k' neuron, which in turn will cause an attempt to resolve the lack of resources needed for I_i. Notice that several subgoal motivation neurons *P′* can be simultaneously active. They are deactivated on the completion of a corresponding subgoal.

Once G_k successfully completes, the resource associated with the sensory input S_k will become available, which in turn, will deactivate the UA_k and P_k' neurons, allowing for I_i and G_i to return to being active. A_i will then be able to execute, which will allow the original *I* and *A* neurons to become active once again. Finally, the original action may be completed. This example considered only a simple situation with two subgoals; however, it can be easily applied to a significantly larger series of subgoals.

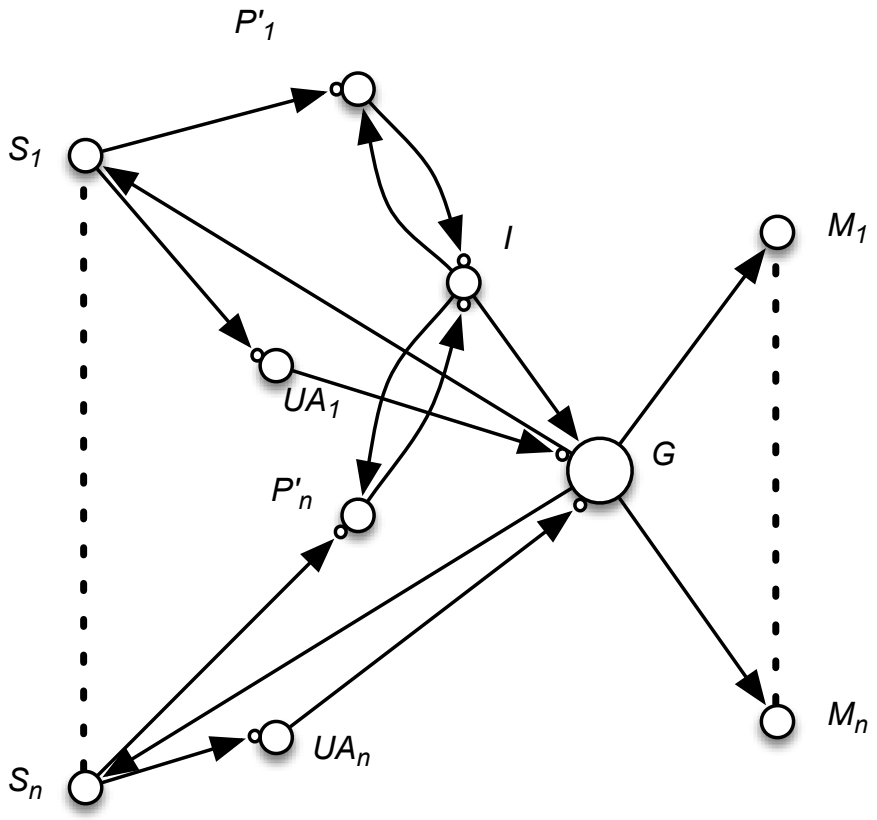


Figure 3-7. Multiple resource requirements.

However, this also leads to questions, such as what happens in a circular system? For example, what happens if you need to patch a hole in a bucket, and through a series of subgoals you end up back where you started, in need of a good bucket? Fortunately, this is a non-problem for the proposed network. Because the *I* neuron associated with using the bucket will have been inhibited earlier in the sequence, it will not be available later on for selection, meaning the network will be forced to find some other solution.

Another potential situation to be examined, is what happens when an action *G* requires multiple resources? The answer is to have multiple UA neurons attached to, and multiple links from, the *G* neuron back to the associated P_i' neurons (see Figure 3-7). Only these P_i' neurons that correspond to tasks not yet completed will be activated. These activated P_i' neurons will trigger the corresponding actions in the activation order that results from their corresponding w_{PG} weights. Once all the required resources are obtained, then the *I* neuron will turn on the goal neuron *G* to generate the associated action and complete its task.

3.6 Conclusion

This chapter has introduced the basics of the Motivated Learning algorithm. It discussed the roles of pains and biases in the system and how motivations are formed. Also discussed were the basic neural network related structures of the system and how they fit together. The basics of curiosity and the implementation of subgoals were also briefly handled. In the next Chapter, several enhancements to the basic Motivated Learning algorithms are discussed. These changes include the addition of non-agent characters (NACs) to the ML agent's environment, changes in bias calculations, and several other adjustments.

CHAPTER 4: ENHANCEMENTS TO MOTIVATED LEARNING

The ML agent implemented in Chapter 3 responds to the state of specific resources and needs in an environment, and thus, showcases a specialized instance of an intelligent agent that learns how to survive in the environment, where resources are limited and can be renewed by the agent's actions. However, the agent did not face other difficulties, nor did it react to changes in the environment while it was undertaking an action. In this chapter I present enhancements to ML that prepare the agent to respond to actions by other agents. This extension of ML capabilities was initially introduced in [21] to help determine which goals of the agent will be implemented and in what order.

4.1 Making Use of "Opportunity"

In general, numerous potential situations the agent may face in a hostile environment need to be considered. The resources in the environment, about which the agent cares, can be either used by the agent itself to its benefit, or they can be used by other agents. Finally, the environment could present the agent with resources not previously observed or available. In the first case (when the resources are beneficial to the agent) the agent must learn how to restore these resources. In the second case (when the beneficial resources are used by another agent) the agent may learn how to prevent competing agents from using these resources. In the third case (when the resources are no longer available), the agent needs to be able to adapt to changing circumstances. For example, there may be a situation where a targeted resource disappears from the environment or previous resource known to be useful is rediscovered during the ongoing execution of an action, such that if the agent were to use this change it could potentially improve its state. However, to do so, the agent would have to interrupt its current task and set another goal for itself. This redirection of goals is referred to as "opportunistic behavior." Opportunistic behavior is formally defined as actions of the agent aimed at the largest reduction of the average overall pain considering the current state of the environment and the agent, and not necessarily the most pressing need.

For example, let us examine a situation where an individual is on his way to the bank to cash a check. While driving to the bank he passes by a supermarket and realizes that he also needs groceries. He decides to stop and buy groceries before continuing on, thus reducing a (less critical) need for food. It was a reasonable decision since he otherwise would have had to go back to the supermarket after cashing his check. In performing opportunistic actions, the agent has to balance several factors, such as, current need levels, travel distances and time involved, time

needed to perform various actions, and predicted (or known) changes in pains/needs. In opportunistic behavior the agent has to continuously reevaluate its options, since its pain levels change dynamically both as a result of its actions and changes within the environment.

Section 4.1.1 provides a mathematical treatment of opportunistic behavior, reducing it to an optimization problem with equations (13) and (14), below, being different instances of the optimized functions. (Much of the material in this chapter was previously presented in our journal publications [98], [99] and conference papers [57].)

In a standard implementation of the ML algorithm from Chapter 3, the agent selects a goal in step 5 of the ML algorithm choosing the maximum pain that can be resolved considering current conditions in the environment. This is obtained by simply using winner-takes-all competition between pain signals with inhibition to those goals that cannot be currently performed. Although such implementation of ML works very well, it can be improved by using the opportunistic behavior explored in this chapter.

The opportunistic motivated learning agent implements step 5 of the ML algorithm (see Section 3.3.1) differently by first determining the effort to reduce each pain based on the resource location and the required task time and then heuristically selecting the next action by considering both the pain level and the cost of performing the action. The main focus point is now the choice, organization, and properties of the heuristic action selection.

Although pains (and goals) may change over time, the agent makes its opportunistic decision based on the current state of the environment and its needs. As conditions of the environment change, the agent reevaluates its previous choices dynamically adjusting to changing environment and changes to its internal needs.

In humans, opportunistic behavior involves attention switching to evaluate various opportunities in a changing environment. We adopted a similar approach to design cognitive agents. Figure 4-1 indicates how attention switching fits into our cognitive model described in [100]. Implementing this cognitive model is our long-term objective.

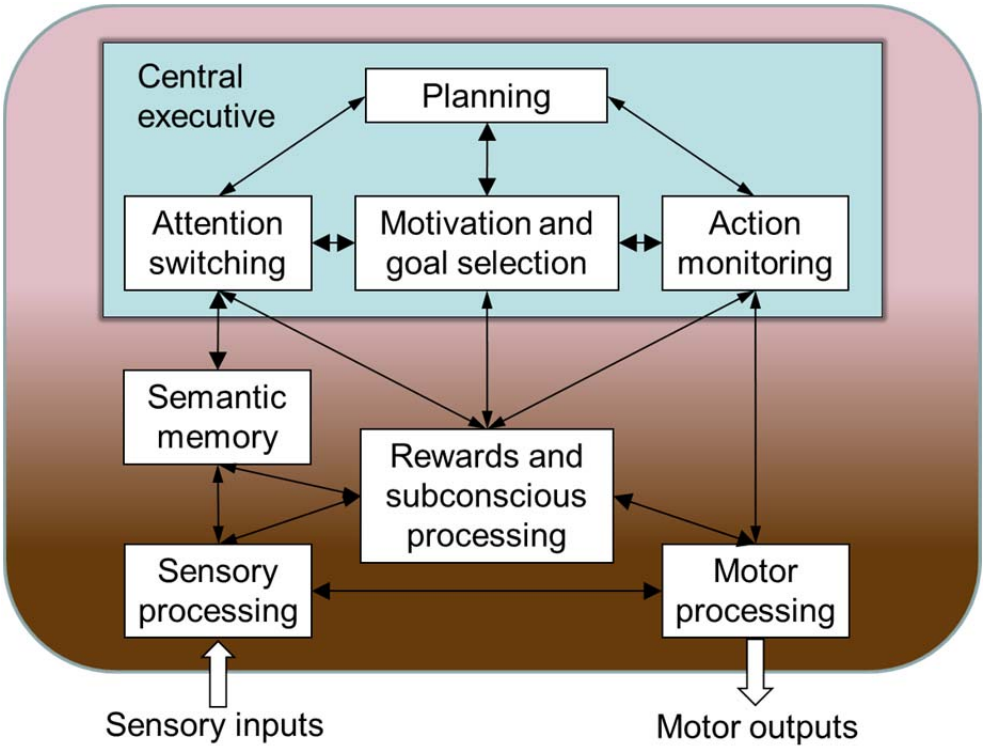


Figure 4-1. Cognitive model pursued in this work.

4.1.1 Math of Opportunistic Behavior

The opportunistic motivated agent (OML) considers all pains (needs) and the total effort to reduce them, trying to minimize its cumulative pain. The method we chose and the reasons for the choice will be discussed in this section. It is based on our need to limit computation time while improving overall pain reduction. (Note: much of the material in this chapter, and this section in particular, was previously presented in an earlier journal publication [99].)

Opportunistic behavior is based on the interplay of motivations that the agent develops while learning the behaviors needed to function in its current environment. Using ML the agent develops needs and related pains that must be satisfied at any given moment. Opportunistic behavior is used to tell the agent which of these needs should be considered first by prioritizing them as we discuss in this section.

Let us consider multiple pains that the agent can reduce by taking proper actions. Assume that the agent must spend some time performing each action to reduce a pain and additional time to travel to the proper destination to perform these actions. Also assume that the agent does not anticipate future pains, so it can minimize the expected cumulative pain based on the current state of various pains. Finally assume that the agent knows the time required to travel between

different destinations and the effort required to finish each job. Dynamic changes that take place in the environment may affect these travel times and the efforts, but we assume that the agent knows their current values.

To find the minimum cumulative pain the agent choses the travel route and estimates the cumulative pain along this route. This cumulative pain P_c can be estimated from

$$P_c = \min_r\{\sum_{k=1}^{n} \mathrm{p}_k \cdot (\sum_{i=1}^{k}(\mathrm{t}_i^{tr} + \mathrm{t}_i^{w}))\} \tag{8}$$

or as expressed in an equivalent form

$$P_c = \min_r \left\{\sum_{k=1}^{n} \left((\mathrm{t}_k^{tr} + \mathrm{t}_k^{w}) \cdot \sum_{i=k}^{n} \mathrm{p}_i\right)\right\} \tag{9}$$

where r is the traveled route between various destinations that the agent selects, n is the number of destinations, t_k^{tr} is time to travel from point k to k+1 along the route, t_k^{w} is the working time to complete a task at the k^{th} destination along the route, and p_i is the pain at point $i, i = 1, \dots, k$. Notice that at the beginning of the route ($t_1^{w} = 0$) the agent may not be at the job location, so in such a case the corresponding p_1=0. Also notice that each individual pain effect is cumulative over the whole task until the pain is reduced or eliminated. This is particularly obvious from (8) where each individual part of the route that ends with performing a task at location k is multiplied by the sum of the pains in all remaining locations not yet visited by the agent.

Equations (8) and (9) estimate the cumulative pain that the opportunistic agent will experience, assuming that pain is known ahead of time and is fixed. We can generalize this result and assume that pain at each location changes over time. In such case we can modify (9) as follows:

$$\begin{aligned} P_{av} &= \min_r \left\{\sum_{k=1}^{n} \left((\mathrm{t}_k^{tr} + \mathrm{t}_k^{w}) \cdot \sum_{i=k}^{n} \left(\frac{1}{\mathrm{t}_k^{tr} + \mathrm{t}_k^{w}} \int_0^{\mathrm{t}_k^{tr} + \mathrm{t}_k^{w}} \mathrm{p}_i dt\right)\right)\right\} \\ &= \min_r \left\{\sum_{k=1}^{n} \left((\mathrm{t}_k^{tr} + \mathrm{t}_k^{w}) \cdot \sum_{i=k}^{n} P_{i\,av}\right)\right\} \end{aligned} \tag{10}$$

where $P_{i\,av}$ is the average pain level at location $i, i = 1, \ldots, k$.over the duration of time to travel k^{th} section of the route and complete k^{th} task.

In a similar way we can generalize (8) as follows:

$$P_c = \min_r\{\sum_{k=1}^{n} P_{kav}^{c} \cdot \left(\sum_{i=1}^{k}\left(\mathrm{t}_i^{tr} + \mathrm{t}_i^{w}\right)\right)\} \tag{11}$$

where P_{kav}^{c} is the average pain level at location k over the duration of time to travel all sections from section *1* to k of the route and complete all corresponding tasks along these sections. Although (8) and (9) are equivalent, (10) is easier to compute than (11) since P_{kav}^{c} depends on the pain change over previously traveled sections, so it cannot be precomputed even if these changes are anticipated and can be predicted over any specific time interval.

Lemma:

Finding the minimum P_c is an NP complete problem.

Proof:

We can prove this by reducing this problem to the travelling salesmen problem (TSP), which is known to be NP complete. The simplest case of the cumulative pain minimization is when the agent is at a specific target destination and plans its traveling route and all $t_k^w = 0$. In addition, let us assume that all p_i are constant and $p_i = 1/n$. In such a case (9) is reduced to:

$$P_{av} = \min_r \sum_{k=1}^{n} \mathrm{t}_k^{tr} w_k \tag{12}$$

where $\mathrm{w_k} = 1$. The traveling salesman problem is a special case of (12) with all $\mathrm{w_k} = 1$ and $\mathrm{t_k^{tr}}$ replaced by the distance between points k and k+1.

□

Since finding the optimum solution to the opportunistic behavior problem is NP complete, we can use heuristics to approximate the minimum. Although many heuristic algorithms exist for the TSP, none of them can be directly used in this problem since weights w_k in (12) depend on the path selected.

Next we present two heuristic algorithms that the opportunistic agent may use to control its behavior.

4.1.1.1 First OML Algorithm: Linear Heuristic (LH) algorithm: (pain reduction rate)

One of the simplest heuristics for TSP is a greedy strategy in which the agent goes to the closest location. We can adopt a similar strategy in the opportunistic agent, but instead of selecting the nearest location the agent selects the location where it can get the largest pain reduction with the smallest effort. In this algorithm the agent determines the pain reduction rate defined as:

$$P_{rr} = \max_k \left(\frac{\mathrm{p}_k}{(\mathrm{t}_k^{tr} + \mathrm{t}_k^{w})^2} \right), k = 1, \dots, m \tag{13}$$

where k is one of the possible locations reachable from the current position of the agent. The linear heuristic algorithm for the opportunistic agent is as follows:

1. Iterate with $i = 1, \dots, n$, where n is the number of pain locations that the agent considers. Set the initial location as the current location of the agent, and set m=n, and $d_{LH} = \emptyset$.
2. At the current location of the agent, which is equal to pain reduction rate (13), find the maximum where t_k^{tr} is a distance from the agent at the current location to the pain at location k.
3. Append node d_r to the selected path d_{LH}=$\{d_{LH}, d_r\}$ where r corresponds to k with the minimum value P_{rr} in (13).
4. Change the current location of the agent to the new location r, remove location r from pain locations, and repeat steps 2 and 3.
5. The resulting optimized route is d_{LH}=$\{d_1, d_2, \dots, d_n\}$.

This algorithm is simple to implement, has linear complexity, and, thus, is useful for a complex environment where the agent faces many choices and needs to update them in real time. Moreover, in spite of its simplicity the algorithm delivers near optimum performance in tests.

4.1.1.2 OML Example 1

Assume that an agent is placed on a 2D grid and the starting location of the agent is (x, y) = (10,7). Assume that at current iteration of the ML algorithm, the agents determined its needs, related pains, and observed the locations of the resources it can use to satisfy its needs. Assume that there are seven resource location points with x and y coordinates and the pain levels to be reduced at these points as shown in Table 4-1.

Table 4-1. Locations and Pain Signal Values

Location Number	1	2	3	4	5	6	7
x Coordinate	2	2	15	8	12	18	12
y Coordinate	2	19	8	14	9	2	18
Pain Level	10	15	7	30	15	16	20

Distances between different locations are as shown in Table 4-2.

Table 4-2. Distances between Locations that Opportunistic Agent Needs to Visit.

Distance	1	2	3	4	5	6	7
1	0.00	17.00	14.32	13.42	29.73	17.46	16.40
2	17.00	0.00	17.03	7.81	14.87	18.87	10.77
3	14.32	17.03	0.00	9.22	22.20	3.16	7.62
4	13.42	7.81	9.22	0.00	16.49	11.18	4.12
5	29.73	14.87	22.20	16.49	0.00	21.84	15.00
6	17.46	18.87	3.16	11.18	21.84	0.00	8.49
7	16.40	10.77	7.62	4.12	15.00	8.49	0.00

For the given example the pain reduction rates computed using (13) are as shown in Table 4-3, and following the linear heuristic algorithm, the opportunistic agent selects to go to location 5 – its nearest location. Notice that standard ML agent would go to location 4 with the largest pain (as shown in the last row of Table 4-1). Here, we assume that t_k^{tr} can be replaced by the traveling distances shown in Table 4-2, and that all values of t_k^w are equal to 1. Using this heuristic strategy the opportunistic agent will chose the route through the following nodes: d_{LH}={5, 4, 7, 2, 1, 6, 3}, while the route of the ML agent will have nodes: d_{ML}={4, 7, 6, 5, 2, 1, 3}. This will result in a 25.9 % reduction in the total pain of the opportunistic agent over the pain suffered by the ML agent. The work time in this experiment was set to 1 for all tasks.

Table 4-3. Computed Pain Reduction Rates

Location	1	2	3	4	5	6	7
P_{rr}	0.96	0.97	1.15	3.62	3.92	1.53	1.64

Obviously, the advantage of opportunistic behavior over ML is greater if the agent would be forced to go back and forth chasing the biggest pain rather than taking the opportunity offered by nearby pain reduction.

Notice that the agent reevaluates its decision after each iteration of the ML algorithm, so it is likely that it will not complete the entire path. Nevertheless, the advantage of selecting the goal to be implemented based on the LH algorithm can be demonstrated in many simulated environments.

4.1.1.3 Second OML Algorithm: Quadratic Heuristic (QH) algorithm

Locally the quadratic heuristic (QH) approach is similar to the greedy algorithm, however, since the search is along all possible path segments, rather than from the current agent location the agent has a chance to find a better total path that minimizes its cumulative pain (10). The main idea behind QH algorithm is to be able to paste together the entire path planned by the agent from the locally dominating section of the path. In this algorithm the agents finds the minimum normalized effort reduction in the entire table

$$P_{rr} = \min_{ik}\left(\frac{(\mathrm{t}_{ik}^{tr}+\mathrm{t}_{ik}^{w})^{4}}{\mathrm{p}_{k}}\right). \tag{14}$$

The quadratic heuristic algorithm for an opportunistic agent is as follows:

1. Compute combined effort matrix$(n+1)\times n$ M weighted with pain at each destination location:

$$M = \begin{bmatrix} (a_1+\mathrm{t}_1^w)^4 & (a_2+\mathrm{t}_2^w)^4 & \dots & (a_n+\mathrm{t}_n^w)^4 \\ (d_{11}+\mathrm{t}_1^w)^4 & (d_{12}+\mathrm{t}_2^w)^4 & \dots & (d_{1n}+\mathrm{t}_n^w)^4 \\ \vdots & \vdots & \ddots & \vdots \\ (d_{n1}+\mathrm{t}_1^w)^4 & (d_{n2}+\mathrm{t}_2^w)^4 & \dots & (d_{nn}+\mathrm{t}_n^w)^4 \end{bmatrix}. \\ \begin{bmatrix} 1/p_{11} & 0 & \dots & 0 \\ 0 & 1/p_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1/p_{nn} \end{bmatrix} \tag{15}$$

where: a_j represents agent distance, t_j^w is the working time at the j-th location, d_{ij} represents distances between pain locations and p_{jj} is the pain value at location j.

2. Create square matrix W from matrix M by adding a new column with ∞ values and change diagonal elements to ∞.

$$W_{(n+1)x(n+1)} = \begin{bmatrix} \infty & m_{11} & m_{12} & \cdots & m_{1n} \\ \infty & \infty & m_{22} & \cdots & m_{2n} \\ \infty & m_{31} & \infty & \cdots & m_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \infty & m_{n+1,1} & m_{n+1,2} & \cdots & \infty \end{bmatrix} \quad (16)$$

The ∞ value represents connections, which will never be chosen, as the agent must start from its current location and cannot revisit any location.

3. Normalize matrix W by dividing each element w_{jk} by the sum of finite elements in row j and column k of matrix W. In addition, change all ∞ values to 1 to obtain the normalized effort matrix V:

$$V = \begin{bmatrix} 1 & v_{12} & v_{13} & \cdots & v_{1,n+1} \\ 1 & 1 & v_{23} & \cdots & v_{2,n+1} \\ 1 & v_{32} & 1 & \cdots & v_{3,n+1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & v_{n+1,1} & v_{n+1,2} & \cdots & 1 \end{bmatrix} \quad (17)$$

where

$$v_{jk} = \left[\frac{w_{jk}}{\sum_{i \neq j}^{n+1} w_{ji} + \sum_{i \neq k}^{n+1} w_{ik}} \right] < 1, \quad j = 1, \ldots, n+1,$$

$$k = 2, \ldots, n+1, \quad j \neq k,$$

4. Iterate with i=1,…,n

   4.1. *Find the sma*llest element v_{be} of matrix V and store the row index of this element in the resulting rows vector R (setting r_i=b-1), and columns vector C (setting c_i=e-1).

   4.2. Replace all elements of row b and column e by 1. Since all elements of the matrix V are not larger than 1 this will eliminate possibility of selecting this element again.

5. To obtain the selected path, iterate with m=1,…,n

   5.1. Start with r_i=0, set m=1, and use b=c_i as the first visited location (d_m=b).

   5.2. Next, find r_k=d_m, increase m by 1, and use b=c_k as the next visited location (d_m=b).

6. The resulting route is d_{QH}={d_1, d_2, …, d_n}.

4.1.1.4 Example 2

To illustrate the steps of transforming the QH algorithm to a TSP problem let us consider the 1D case in which the agent is located at coordinate $x = 5$, with resource locations (x coordinates) and the pain levels that can be reduced at these locations as shown in the Table 4-4.

Table 4-4. Pains at Various Locations

Location number	1	2	3	4	5	6
x coordinate	2	10	20	38	80	93
Pain level	30	18	20	7	31	19

First, we formulate the effort matrix M (10) and reduce this matrix to the normalized form V (17). Matrix M is equal to (only three significant digits are shown):

$$M = 10^4 * \begin{bmatrix} 0.00 & 0.01 & 0.33 & 19.1 & 108 & 330 \\ 0 & 0.04 & 0.65 & 26.8 & 126 & 377 \\ 0.02 & 0 & 0.07 & 10.1 & 82 & 262 \\ 0.43 & 0.08 & 0 & 1.86 & 44.7 & 158 \\ 6.25 & 3.93 & 0.65 & 0 & 11.0 & 51.8 \\ 130 & 141 & 69.2 & 48.8 & 0 & 0.20 \\ 239 & 27.7 & 150 & 140 & 0.124 & 0 \end{bmatrix} \tag{18}$$

and the normalized V matrix is:

$$V = \begin{bmatrix} 1 & 1E\text{-}06 & 8E\text{-}06 & 5E\text{-}04 & 0.027 & 0.130 & 0.202 \\ 1 & 1 & 3E\text{-}05 & 9E\text{-}04 & 0.034 & 0.139 & 0.221 \\ 1 & 3E\text{-}05 & 1 & 1E\text{-}04 & 0.017 & 0.113 & 0.171 \\ 1 & 0.001 & 1E\text{-}04 & 1 & 0.004 & 0.078 & 0.114 \\ 1 & 0.014 & 0.008 & 0.002 & 1 & 0.025 & 0.041 \\ 1 & 0.170 & 0.174 & 0.114 & 0.077 & 1 & 1E\text{-}04 \\ 1 & 0.202 & 0.225 & 0.146 & 0.133 & 1E\text{-}04 & 1 \end{bmatrix} \tag{19}$$

4.2 Handling Non-Agent Characters and Desired and Undesired Events

Prior to this section, the ML agent has only dealt with resources in the environment, and only those which are beneficial. However, how do we handle resources that "hurt" the agent (e.g. undesired resources), and how do we manage other agents? What if another agent (hereafter referred to as a non-agent character or NAC) instigates an undesired (or desired) event? In this

case, the ML agent can either remove the resource that the NAC agent needs to perform the undesired action (for instance remove its weapon), or prevent the NAC agent from performing the action (by blocking its action or removal of the NAC). (Please note that material from Sections 4.2-4.4 was previously submitted for publication in [98] and conference paper [57].)

More specifically, the resources in the environment can be used either by the agent itself to its benefit, or by other agents. In the first case the agent must learn how to restore these resources, while in the second case the agent may need to learn how to prevent the competing agents (NACs) from using the resources that the agent needs. The resources can be either desirable, in which case the agent needs to protect or restore them, undesirable, in which case the agent needs to remove or destroy them, or neutral to the agent.

Initially, all resources and NAC actions are unknown to the agent, so their desirability is unknown.

Definition:

An agent finds a resource *undesirable* when the resource increases the agent's pain. A resource is found *desirable* once its use reduces the agent's pain.

A more general approach is to also consider desired and undesired events caused either by the agent itself or by other agents. We want the agent to learn how to avoid undesirable events and encourage those that benefit it. Typically an event desired by the agent requires it to act on a resource. Any inability to do this is a source of pain to the intelligent agent. One such situation is a lack of the desired resource, but equally important is the inability to perform a desired action. Suppose that the agent needs to bring coal to heat its home and has found a huge coal slab. Although it has a desired resource, if it cannot transport it home, its need will not be satisfied. A similar situation is when the agent wants to remove an undesired resource from its environment – it must identify the undesired resource and have the ability to remove it. A slightly different situation is when a NAC instigates an undesired event. In this case, the ML agent can either remove the resource that the NAC agent needs to perform the undesired action (for instance remove his weapon), or otherwise prevent the undesired event (for instance by stopping the NAC's action).

An interesting case is when both desired and undesired events are related to the same resource (or another agent). Consider a beehive. It is desired to have more bees, so the agent can get more honey, on the other hand, the more bees an agent has around the more likely that they will sting him causing a pain increase. Removal of bees from the environment will prevent them from doing harm to the agent. However, this may increase an abstract pain related to the lack of

honey. A practical solution in this case, is to prevent bees from stinging the agent by wearing protective gear. Thus, the agent must consider all consequences of its actions, and chose an action that minimizes its overall pain. The agent–environment interaction can be summarized by Table 4-5, which shows several possible ways to induce pains and trigger goal creation along the lines of the preceding examples.

All of these environment conditions, and observed events allow for the further generalization of ML. To this point, how the agent uses its observations to develop biases, abstract pains, needs, motivations, and goals has been described. Since a ML agent relies on its motivations to act, it must create them by paying attention to resources and actions. It generates an internal pain signal when it has too few desired resources or too many undesired ones. The agent also generates an internal pain signal if it cannot perform a desired action or cannot stop an undesired one. The following subsections discuss how the ML agent has been adjusted to take these things into account. Next, setting bias signals and adjusting weights associated with pain reduction is described.

Table 4-5. Agent's Learning and Goal Creation Principles

Relations	**Pain changes**	**Resource**	**Abstract pain**	**Agent's goal**
The agent acts on the environment	Reduced	Desirable	Low resource amount or inability to act on the resource	Increase the resource amount
	Increased	Desirable	None	Do not perform this action
		Undesirable	High resource amount	Reduce the resource amount or do not perform this action
	No change	Unknown	None	None
The environment acts on the agent	Reduced	Desirable	Low resource amount or no desired NAC's action	Increase the resource amount or learn how to encourage NAC's action
	Increased	Desirable	Inability to perform defensive action	Learn to prevent the NAC's action
		Undesirable	High resource amount or inability to perform defensive action	Reduce the resource amount or learn to prevent NAC's action
	No change	Unknown	None	None

4.3 Bias Signals, Weights, and Associated Pains

As has been previously stated, the ML agent is motivated to learn to minimize its internal pains. Three types of internal pains are differentiated in ML: a) pain based on availability of resources; b) action related pain; c) pain caused by the inability to perform a desired action. A bias signal triggers an abstract pain and depends on the perceived situation. The introduction and proper use of bias signals is perhaps the most differentiating factor between RL and ML schemes. (The material in this section was previously published as part of a conference paper [57].)

In the ML algorithm bias signals are used to determine the level of pain or need associated with a particular concept (resource availability or an action performed by another agent). Biases indicate the likelihood of running out of resources needed or of facing a hostile action by another agent. There are several ways in which bias can be calculated. One method is a simple approach,

$$B_i = \gamma \cdot \frac{\varepsilon + R_d(S_i)}{\varepsilon + R_c(S_i)} \tag{20}$$

where R_c is the current resource value, R_d is the desired (or initial) resource value, s_i represents the resource under consideration, and ε is a small positive number. This method was used for the simple ML implementation once we moved beyond representing resource availability as probabilities and began using actual quantities. Equation 20 is useful because it produces a large bias as a desired resource approaches zero. On the down side it does not go to zero when $R_c = R_d$, but instead approaches γ.

Originally, the initial state was considered to be the desired state of the agent; however, the initial state of a resource is not necessarily the optimum state for the agent. The agent should be able to set the desired state by itself. How this might be done is discussed in Section 4.6.

Another way to calculate bias is using:

$$B_i = \left| log_2 \left(\frac{\varepsilon + R_c(S_i)}{\epsilon + R_d(S_i)} \right) \right| \tag{21}$$

which works well when resource probability is used (as was the case in our initial experiments). It works with both probability based resource values and the discrete value in more recent research, but has some minor issues due to the use of the ε value.

A third variant uses slightly more complex computations to deliver a smooth transition around the desired resource level:

$$B_i = -ln\left(\frac{R_c(S_i)}{R_d(S_i)}\right) + \frac{R_c(S_i)}{R_d(S_i)} - 1 \tag{22}$$

The three aforementioned approaches for bias calculations were all originally used for resource calculation, but were deficient in handling both resources and the actions of other agents. A method for calculating bias for both resources and agent actions was needed. Prior to this, the bias calculations for each were to be handled separately. The following subsections describe how this issue was handled.

4.3.1 Resource Related Pains

If the autonomous agent needs to maintain a certain level of resources, a resource related pain is triggered. To generate this pain, a bias signal that reflects how difficult it is to obtain this resource, is used. In order to introduce a resource related bias signal, the agent must first use the resource to determine if the resource is desired or undesired. The resource bias signal is calculated from the quantity of resources and its desired/undesired status as follows:

$$B(s_i) = -(1 + \delta_i) \cdot (lnA(s_i) + 1) + 2 \cdot A(s_i) \tag{23}$$

where

$$A(s_i) = \frac{\varepsilon + R_c(s_i)}{\varepsilon + R_d(s_i)} \tag{24}$$

represents availability of the resource s_i , R_c is the current resource amount, and R_d is the desired resource amount. If the resource is undesired R_d is the limit value of the resource perceived as painful to the agent (for instance a painful amount of pollution); ε is a small positive number to prevent numerical overflow, and $\delta_i = 1$ when the resource is *desired*, $\delta_i = -1$ when it is *not desired*, and $\delta_i = 0$ when the character of the resource is unknown.

R_d is used as a normalizing factor for the resource level. If $R_c(s_i) = R_d(s_i)$ the corresponding resource pain is zero for a desirable resource; for undesirable resources the smaller $R_c(s_i)$ is the smaller the resource pain is. Pain reaches significant levels when the agent is out of a desired resource or has an excess of an undesired resource.

Figure 4-2 shows changes in the bias signal values for desired and undesired resources. Bias value also increases when a desired resource exceeds the desired level. This is a useful feature of the assumed bias function as it penalizes hoarding. (There is usually minimal utility in having too much of a resource, particularly if it prone to spoilage or is not used very often.)

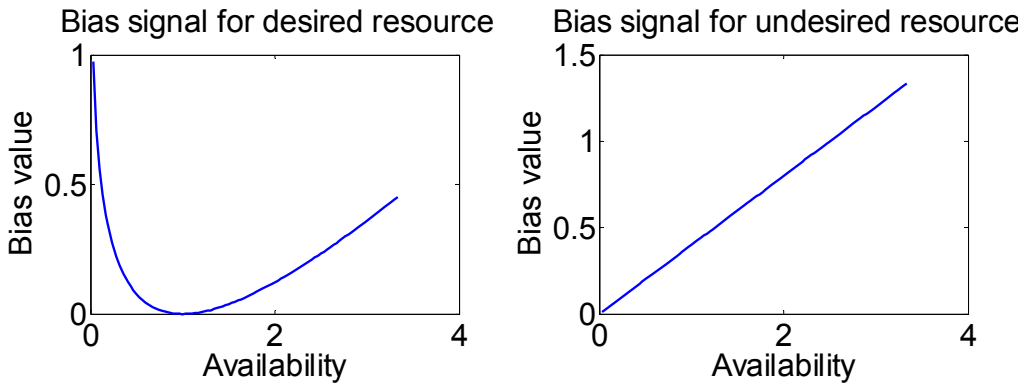


Figure 4-2. Bias signal for a) desired resource b) undesired resource.

In special cases, when the resource was found to be both desired and undesired we first set $\delta_i = 1$ and learn how to prevent it from harming the agent by proper action. When this fails, we set $\delta_i = -1$.

While other functions can be used to compute biases, we found that the function (23) appropriately correlates availability and desirability of resources and actions, and can be conveniently applied in all situations, as discussed next.

4.3.2 Action Related Pains

The biases related to a NAC action, shown in Figure 4-3 as 'NAC Action Biases', are activated through the link from the likelihood of the specific action. 'NAC Action Pain' is first learned when the NAC's action is observed in correlation to an increased pain. Subsequently it is triggered by the 'NAC Action Bias'. Setting 'NAC Action Pain' is important since this motivates the agent to properly respond to a NAC action, rather than to just avoid the associated pain inflicted by such action. This provides the agent with the ability to beneficially interact with other agents.

Likewise, if based on the observed beneficial result of a NAC's action (reduced pain or reward received), the agent would like to encourage such an action. Such biases are created as the

result of pain reduction as indicated by the link from 'Pain' to 'NAC Action Biases' and create a need to encourage the NAC to act in the desired way.

The 'NAC Action Bias' is calculated from (23) with the exception that $A(s_i)$ represents action availability computed from:

$$A(s_i) = \frac{1}{\frac{d_c}{d_d} + \frac{1+\delta_i}{2}} \tag{25}$$

where d_c is current and d_d is a desired (a comfortable) distance to another agent and $\delta_i = 1$ when the NAC action is *desired*, $\delta_i = -1$ when it is *not desired*, and $\delta_i = 0$ otherwise. Figure 4-3 shows changes in the bias signal values for desired, undesired and neutral NAC actions both as a function of availability and a relative distance $\frac{d_c}{d_d}$.

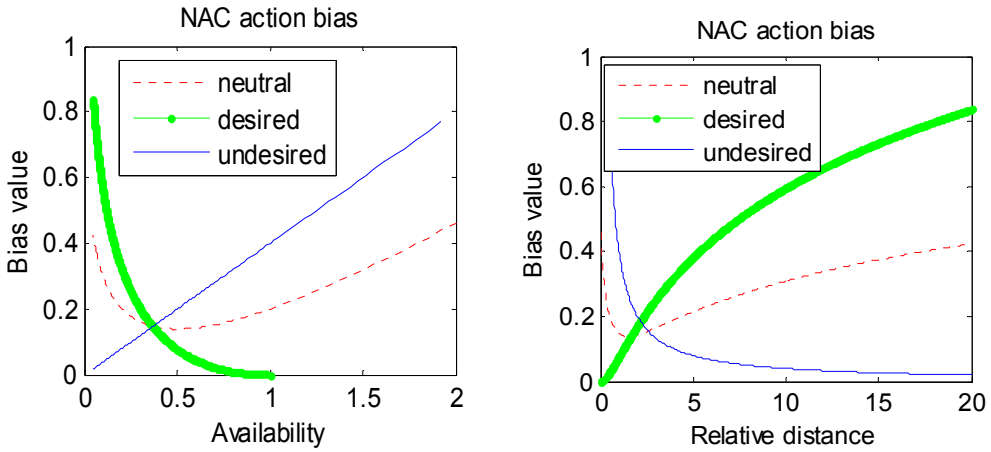


Figure 4-3. Bias signal as a function of a) availability b) relative distance.

Distance based availability is poor estimate of a likelihood of NAC action. We used this in our simulation to show that the agent learns correct responses to NAC actions. A more elaborate approach would replace $A(s_i)$ with likelihood that NAC agent will perform its action. This may depend on specifics of the implementation, and does not pose a significant limitation to ML development.

4.3.3 Inability to Perform an Action

The third kind of pain – the inability to perform a desired action is generated after the agent learns that an action is useful but it is unable to perform it in spite of available resources. This pain may be caused by a motor malfunction or restriction on the agent moves. In both cases it produces an abstract pain, which grows in proportion to the need to perform the action. Thus, the third kind of pain depends on the first two, and is triggered by them. However, once it is triggered the agent does not depend on a current need to obtain a resource or to perform the action.

The bias signal in this case is obtained from:

$$B = \gamma \cdot \overline{P(s_i)} \tag{26}$$

where $\overline{P(s_i)}$ is the mean value of the lower level pain that requires the given action to be performed in order to reduce this pain.

Pain biases calculated in this section were set using arbitrary functions. They were introduced to illustrate setting of internal pain signals that represent the motivational state of the agent. Other approaches to compute the internal state of the agent are possible. Figure 4-4 schematically shows block dependencies between perceptions, biases, pains motivations, goals, and actions.

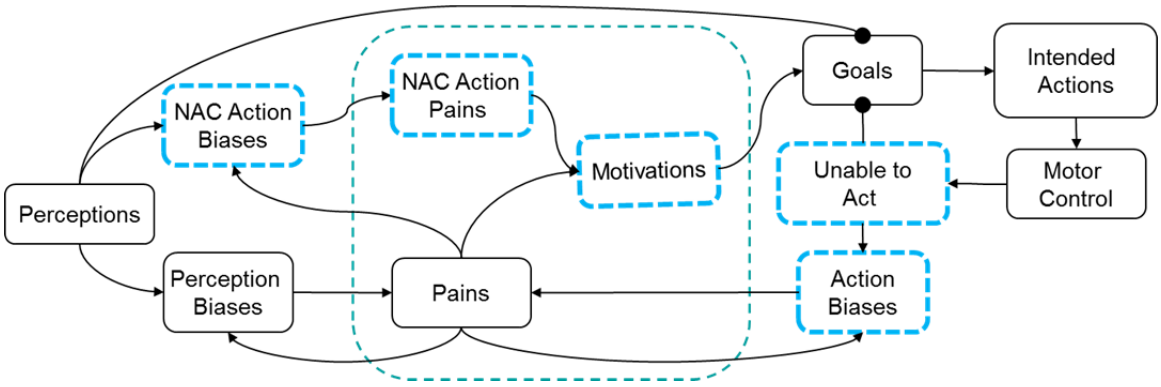


Figure 4-4. Dependencies between perceptions, biases, pains, motivations, goals and actions.

Links in Figure 4-4 that end with a black dot are inhibitory, while these that end with an arrow are excitatory.

Blocks marked with dashed lines extend the original motivated agent that was only able to create resource related pains [19], [97]. The new blocks contain perception based NAC Action Bias that correlates the observed action by a NAC and the pain it inflicted on the ML agent. Subsequently, an observed attack will trigger the NAC Action Pain related to the action by the NAC.

This, in turn, triggers a motivation block to prevent or stop the attack and the proper action that accomplishes this is learned. "Unable to Act" and "Action Biases" are new functional blocks related to agent's inability to perform useful actions either to restore/destroy resources or to act defensively.

From the point of calculating bias, the agent next needs to determine the pain levels. This is done using the process described in Section 3.3 with equations (2) and (3). While procedures for calculating the bias and updating weights have changed, pain is still calculated by multiplying bias with the associated bias-to-pain weights.

4.4 Learning Recommendations

A motivated agent learns when its actions satisfy one of its needs, which translates to a reduction in one of its pain signals. In the previous implementations of ML discussed so far in Chapter 3 and Section 4.1 [20], [21], pain reduction was related to the case of a passive environment in which the agent only used resources to its advantage. Here we extend this to an active environment, where other NACs may act on the ML agent affecting its pains.

A correct action may not result in the reduction of pain, but instead it may stop the increase of pain. Thus, a successful action cannot only be measured by pain reduction alone.

In the general case of an active environment, learning based solely on the agent's own actions may not suffice. In this case the agent learns how to respond depending on the outcome of interactions with NACs as described in Table 4-6.

Table 4-6. Determination of w_{PG} Weight Adjustments

Pain	δ_a	δ_n	$\delta_a \delta_n$
-1	(1,0)	(0,1)	(1,1)
0	(-1,0)	(0,0)	(1,-1)
1	(-1,0)	(0,-1)	(-1,-1)

In Table 4-6 there are 9 distinct situations depending on the action taken and its outcome. We consider three types of actions: agent acts δ_a, NAC acts δ_n, both agent and NAC act $\delta_a\delta_n$. The outcome of each action can be pain reduction (-1 in pain column), neutral (0 in pain column) and pain increase (1 in the Pain column).

Pairs (*x,y*) in Table 4-6 indicate *learning recommendations* related to reinforcement and desirability. If *x=1* the action taken by the agent should be reinforced by increasing w_{PG} weights, if *x=0* no change in w_{PG} weights is recommended, and when *x=-1* then w_{PG} weights should be reduced.

Desirability of NAC actions is determined based on *y* values. When *y=1* this action is desirable and should be encouraged by the agent, when *y=0* the action is neutral and the agent should not care about it, and when *y=-1* this action is undesirable and should be discouraged.

Recommendations related to NAC actions mean that the agent must learn how to influence the NAC's behavior. This means that any action by the ML agent that results in the desired behavior by NAC will be reinforced.

An interesting case is the second row in Table 4-6 when the pain does not change. If the pain did not change after the agent's own action, such action should be discouraged, since it is unnecessary. If the agent acted in response to an action by a NAC, this action is deemed useful since it is possible that it stopped an "attack" on the agent. This is a special case in which the agent's own action should be encouraged but that of the NAC agent should be discouraged. Even though the action by the NAC agent would not do any harm, it requires a defensive action by the learning agent, possibly taking away precious time and resources.

4.4.1 Fight or Flight

If the NAC acts and "hurts" the agent, the agent must respond to protect itself or its resources. The correct response depends on the agent's ability to observe the NAC's action and prevent it from harming the agent. Thus, detecting an action by the NAC, and determining if the action is desired or undesired, is critical.

When the agent detects a NAC action, it activates a potential pain based on the bias signal (23) for the NAC action. This potential pain competes for the agent's attention and motivates it to act defensively. At the same time, the potential pain is not registered as a pain increase, since it is only the motivating factor and not the real pain. Likewise, if the agent acts defensively and stops the attack, it learns the proper action by observing that the real pain did not increase. This lack of pain increase is sufficient reason to reinforce such behavior. However, if

the real pain increased, this indicates that the agent did not use an appropriate response to the attack, and the corresponding w_{PG} weight must go down.

This action related potential pain signal is removed as the observed NAC action stops. If it stopped as a result of the agent action a reinforced learning takes place. If it stopped without the agent's action (for instance when the attacker walks away), the learning agent gradually reduces the potential pain signal.

If the agent is attacked, it can resolve to either defend itself by attacking the enemy or running away from it. The agent can learn which action is a better choice, depending on its experience. The cognitive agent will be able to estimate its chance of running away from the enemy based on its understanding of its own and the enemy's embodiment and limits on its ability to prevent the enemy's action.

This process can be observed in Figure 5-22 of Section 5.4.3.

4.4.2 Setting Goals

As previously mentioned, the ML agent sets its goals based on the pain signals and its ability to control them. In a simple neural network implementation of ML, the goal is selected based on the strength of interconnection weights w_{PG} shown in Figure 3-4 in Chapter 3. A given pain signal *P* is multiplied by w_{PG}weights and a goal neuron with the strongest activation is selected. (This discussion on weight adjustment was previously published as part of a conference paper [101].)

To control w_{PG}weights we established the *pain reduction parameter* δ_p based on the reduction or increase of pain and actions performed. δ_p is negative when pain increases**,** and is positive if the pain decreases. The pain reduction parameter δ_p represents the learning recommendations from Table 4-6 and is obtained using the formula:

$$\delta_p = 2 \cdot x + y \tag{27}$$

where (*x,y*) are obtained from Table 4-6. δ_p is between -3 and +3. Larger weight is given to the agent's own actions, represented by x, since the agent controls them better than NAC actions.

Initial weights between *P*-*G* neurons are randomly selected in the 0-α_g interval (a good setting is between 0.49 and 0.51 of α_g for faster learning). Assume that the weights are adjusted upwards or downwards by a maximum amount μ_g. In order to keep the interconnection weights

within prespecified limits ($0< w_{PG}<\alpha_g$), the value of the actual weight adjustment applied can be less than μ_g and is computed as

$$\Delta_a = \mu_g \cdot \operatorname{atan}\left(\frac{ds}{d\hat{s}}\right) \cdot \min\left(\left|\alpha_g - w_{PG}\right|, w_{PG}\right) \tag{28}$$

where $\alpha_g \leq 1$, $\frac{ds}{d\hat{s}}$ is the rate of change of lower level resource s as a result of using higher level resource $\hat{s}$ observed by the agent, and

$$\mu_g = \mu_0 \left(1 - \frac{2}{\pi}\operatorname{atan}(10/B)\right), \qquad \text{where } \mu_0 = 0.3. \tag{29}$$

Using (28) produces weights that slowly saturate towards 0 or α_g (for quick learning set $\mu_0 = \alpha_g / 2$) and μ_g gradually changes from μ_0 towards 0 when B changes from a large value towards 0.

If, as a result of an action taken, the pain that triggered the action increases (as determined by pain reduction parameter δ_p), then the w_{PG} weight will be decreased, and if the pain decreases, then the w_{PG} weight is increased as follows:

$$w_{PG} = w_{PG} + \delta_p \cdot \Delta_a \tag{30}$$

where δ_p is computed from (27) depending on the *(x,y)* value in Table 4-6. A new pain signal is created to either discourage or encourage the NAC action depending on the *y* value. The agent learns how to reduce NAC action pain, adjusting corresponding w_{PG} weights.

Additionally, the curiosity weight associated with the action is reduced

$$w_{PG_C} = w_{PG_C} - \Delta_{ac}. \tag{31}$$

As discussed in section 3.4, curiosity helps the machine to explore the environment and learn when not performing any specific pain based actions.

4.5 Probabilistic Goal Selection

Our Motivated Learning agent selects which actions to take based on pain levels and pain-goal weights (w_{PG}). These weights are adjusted dynamically through learning as discussed in Section 3.3 and [20], most currently using the procedure discussed in 4.4.2 (equations 28-30). (Material in this subsection and Section 4.6 was taken from an earlier conference paper [57].)

However, there is a problem with the preceding approach. While the agent does a good enough job of learning, in many cases it doesn't learn all of the available options very well (or at all). This can leave the agent in trouble if it runs out of a resource at an inopportune time and without an alternative option to fall back upon. Hence, in this subsection, we consider a slightly different probabilistic approach to selecting actions, which requires a few additional changes to how actions are learned.

In the probabilistic goal selection approach, goals are selected using probability values based on the current w_{PG} weights:

$$p_{PGi} = \frac{w_{PGi}}{\sum_{i=1}^{n_{PG}} w_{PGi}}. \tag{32}$$

In this case there is no need to limit weight values as in equation 34 since we can assume (after normalization)

$$w_{PGi} = p_{PGi} \leq 1, \ \ and \ \ \sum_i p_{PGi} = 1\,. \tag{33}$$

Using (28)-(30) for probability based goal selection faced the problem of weight saturation to α_g, which resulted in all probabilities approximating the same value for successful goals independently of the actual usefulness of the associated resource (other than the fact that it was considered useful).

A modification of (30) differentiates between useful actions depending on their efficiency. Initial w_{PG} weights are set to $1/n_{PG} \pm \epsilon$, where n_{PG} is the number of w_{PG} weights (instead of a $0.5 \pm \epsilon$). After each time a goal is triggered by a specific pain signal its corresponding w_{PG} weight is adjusted as follows:

$$w_{PG} = min\left(\frac{3}{\pi}\text{atan}\left(\frac{ds}{d\hat{s}}\right), w_{PG} \cdot \mu_g \cdot \frac{4}{\pi}\text{atan}\left(\frac{ds}{d\hat{s}}\right)\right), \tag{34}$$

where $\frac{ds}{d\hat{s}}$ is the rate of change of the lower level resource s as a result of using higher level resource $\hat{s}$ observed by the agent.

$$\mu_g = 1 + \left(1 - \frac{2}{\pi}\text{atan}(10/B)\right) \cdot \left(\alpha_f{}^{-\delta_i} - 1\right), \tag{35}$$

where $\alpha_f < 1$ (recommended value is around 0.7). As a result w_{PG} weights saturate at $(3/\pi)\text{atan}(ds/d\hat{s})$. Thus more efficient ways are more likely to be chosen in the probabilistic goal selection.

4.6 Determining the Resource Consumption

To perform a successful action and remove its pain, the agent must estimate the total effort required, for instance estimating the quantity of resources it must use or the force it must apply to accomplish its goal. To calculate the required amount of resource consumption, it is necessary to first determine the change in pain resulting from a *successful* action (with observed pain reduction).

$$\Delta P_{ob} = P_{i+1} - P_i \tag{36}$$

and to assume that the desired pain decrease is equal to the previous pain value

$$\Delta P_{de} = -P_i. \tag{36}$$

The desired pain ratio is computed using

$$R_P = \frac{\Delta P_{ob}}{\Delta P_{de}} = 1 - \frac{P_{i+1}}{P_i}. \tag{38}$$

By observing the environment the agent can estimate the environmentally determined resource exchange rate R_{er} using

$$R_{er}(s_i) = \frac{\Delta R_o(s_i)}{\Delta R_u(s_i)} = \frac{R_{o(i+1)}(s_i) - R_{oi}(s_i)}{R_{ui}(s_i) - R_{u(i+1)}(s_i)} \tag{39}$$

where R_o is amount of resource obtained as a result of a useful action by the agent, while R_u is the amount of resource used that agent spent to produce the desired effect.

To estimate the amount of resource to be used in a useful action, the agent must know the desired amount of the resource $R_d(s_i)$ it needs. If the agent knows its computational model, it can estimate the bias of its abstract pain from (2) and determine the desired amount of resource using (22-23). Alternatively, it can use the rate at which it depletes the desired resource in a fixed amount of time to find out the desired amount of resource.

In the first case, the agent first evaluates the desired resource $R_d(s_i)$ from

$$R_d(s_i) = R_c(s_i) \cdot 2^{\frac{P_i(s_i)}{w_{BP}(s_i)}} \tag{40}$$

then it uses this desired value to find needed amount of resource he must use from

$$\Delta R_u(s_i) = \frac{R_c(s_i)}{R_{er}(s_i)}\left(2^{\frac{P_i(s_i)}{w_{BP}(s_i)}} - 1 \right). \tag{41}$$

In the second case, the agent evaluates the desired resource $R_d(s_i)$ on the basis of how quickly it uses the resource in a given time interval $\frac{\Delta R_d(s_i)}{\Delta t}$. Then the desired amount of resource will be estimated from

$$R_d(s_i) = min\left(\frac{\Delta R_d(s_i)}{\Delta t} \cdot T, \widehat{R_d}(s_i)\right) \tag{42}$$

where $\widehat{R_d}(s_i)$ is the maximum storage capacity devoted to this item. Time period T in (44) can, for instance, be an expiration time for the resource the agent uses. From there, it can subsequently estimate the amount of resource it must use from:

$$\Delta R_u(s_i) = \frac{\Delta R_o(s_i)}{R_{er}(s_i)} = \frac{R_d(s_i) - R_c(s_i)}{R_{er}(s_i)} \tag{43}$$

Beta is defined as the amount of (higher level) resource used to reach a desired value of the obtained resource (typically of the lower level), so

$$R_{u(i+1)}(s) = R_{ui}(s) - \beta \tag{44}$$

and resource obtained is

$$R_{o(i+1)}(s) = R_{oi}(s) + \beta R_{er}. \tag{45}$$

If it is not known how to produce a useful result, we assume $\beta = 1$, otherwise we compute beta from:

$$\beta = min\left(\frac{R_c(s_i)}{2}, \Delta R_u(s_i)\right). \tag{46}$$

In action related efforts, the ML agent estimates all necessary parameters of its actions (e.g. force applied, speed to run away from danger, etc.) experimentally. Too large a force may break the target object destroying it; too slow a pace of escape may end up with the agent being hurt.

4.6.1 Setting Desired Resource Levels

Desired values should be set according to the agent's needs. This was touched upon briefly in [101], and later covered in greater detail in [57]. To set desired resource levels, we start with the need for the primitive resource (e.g. the energy level). This level R_{dp} is known and is set by the environment. Another environment set variable is the rate of use of the desired resource Δ_p(for instance how quickly the energy is lost by the agent). All desired levels for other resources (related to abstract pains) are set by the agent. These levels are established by the agent only for those resources that are useful to the agent.

In order for the agent to be successful in all of its tasks the frequency of performing various tasks cannot be too great as its time is limited. In the simplest case, consider that each task takes one unit of time (equivalent to a single iteration). If a primitive resource will be exhausted in n_s iterations the agent must perform an operation to restore this resource with the frequency $f_s = \frac{1}{n_s}$. Considering all actions that agent performs its operations can be successful only if

$$\sum_i f_{si} < 1 \tag{47}$$

where the summation is for all resources that the agent needs to restore and all actions it must perform to avoid hostile activities by other agents.

For all primitive resources the restoration frequency is predetermined from

$$n_s = \frac{R_{ds}}{\Delta_s} = \frac{1}{f_s}, \tag{48}$$

restoration frequencies of other resources can be set arbitrarily provided that (49) is satisfied. Notice that in order to obtain the restoration frequency of a higher level resource we must consider the restoration frequency of the lower level resource it helps to restore, thus we have

$$f_{\hat{s}} = f_s \cdot \frac{\Delta_{\hat{s}}}{R_{d\hat{s}}} < f_s. \tag{49}$$

The largest restoration frequency for the higher level resource equals the restoration frequency of the resource it helps to restore, and in such case $R_{d\hat{s}} = \Delta_{\hat{s}}$ and the higher level resource needs to be restored each time it is used. This seems to be an extreme situation, although the required resource level will be very low. However, when the agent has multiple goals we may easily violate (47). Keeping the restoration frequency equal or close to the frequency of the resource it is restoring is useful if we want to teach the agent proper responses in a shortest possible time.

If (47) is satisfied for all primitive needs we can always set desired values of higher-level resources $R_{d\hat{s}}$ in such a way that (47) holds. This may require high values of $R_{d\hat{s}}$. In general to set $R_{d\hat{s}}$ in an optimized fashion (such that the agent does not maintain overly large supplies of a resource), we need to solve the optimization problem:

$$F(\alpha_1, \alpha_2, \dots, \alpha_m) = min \sum_{i=1}^{m} \frac{1}{\alpha_i}. \tag{50}$$

Subject to constraints

$$0 \le \alpha_i \le 1, \qquad i = 1, \dots, m \tag{51}$$

and

$$\sum_{i=1}^{p} f_{si}\left(1+\alpha_{i1}\left(1+\alpha_{i2}\left(1+\ldots+\alpha_{n_i}\right)\right)\right)<1 \tag{52}$$

where p is the number of primitive, α_{i1} represents the resource $\hat{s}_{i1}$ that is needed to restore s_i and

$$\alpha_{i1}=\frac{\Delta_{\hat{s}_{i1}}}{R_{d\hat{s}_{i1}}} \tag{53}$$

is a relative frequency of the higher level resource to be determined, which tells us how many times we can use the resource before it is gone and needs to be replenished.

In a similar way α_{i2} represents the resource $\hat{s}_{i2}$ that is needed to restore $\hat{s}_{i1}$ and so on, and where $\hat{s}_{ni},\ldots,\hat{s}_{i2},\hat{s}_{i1},s_i$ is a dependence path from the most abstract resource $\hat{s}_{ni}$ to the primitive resource s_i.

Once α_i is known, we can determine the desired level of each higher-level resource using

$$R_{d\hat{s}_i}=\frac{\Delta_{\hat{s}_i}}{\alpha_i} \tag{54}$$

The environment set rules establish derivative $\frac{ds_i}{d\hat{s}_i}$ which can be used to compute $\Delta_{\hat{s}_i}$ in order to restore the level of a resource to its desired value R_{ds}

$$\Delta_{\hat{s}_i}=\frac{R_{ds_i}}{\frac{ds_i}{d\hat{s}_i}} \tag{55}$$

thus

$$\mathrm{R}_{\mathrm{d\hat{s}_i}}=\frac{\mathrm{R_{ds_i}}}{\alpha_{\mathrm{i}}*\frac{\mathrm{ds_i}}{\mathrm{d\hat{s}_i}}}. \tag{58}$$

To solve the optimization problem (50) the Matlab function *fmincon* with 'sqp' or sequential quadratic programming algorithm, which rigidly respects bounds, and is robust in handling *inf* and *nan* values, is used.

To illustrate setting the desired resource level as discussed in this section let us consider the following example:

4.6.1.1 Setting desired resource level: Example 1

Let us assume that we have 3 primitive resources with the need for the primitive resources equal to:

$$R_{d1} = 20, \quad R_{d2} = 12, \quad R_{d1} = 21,$$

and the rate of use of the primitive resources equal to

$$\Delta_1 = 4,\ \Delta_2 = 2\ ,\ \Delta_3 = 3.$$

Thus, primitive resource restoration frequencies are

$$f_{s1} = \frac{\Delta_1}{R_{d1}} = \frac{1}{5},\ f_{s2} = \frac{\Delta_2}{R_{d2}} = \frac{1}{6},\ f_{s3} = \frac{\Delta_3}{R_{d3}} = \frac{1}{7}.$$

In addition, there are 7 higher level resources that can be used to restore these primitive ones and we would like to set desired resource values in such way that the agent learns in an optimized way, and yet is capable of handling all the tasks.

Also assume the following dependence paths from the most abstract resource to the primitive resource s_i:

$$s_6, s_5, s_4, s_1$$

$$s_8, s_7, s_2$$

$$s_{10}, s_9, s_3$$

And the lower level resource restoration rates with respect to the utilization rate of the higher level resource are as follows:

$$\frac{ds_1}{ds_4} = 3, \qquad \frac{ds_4}{ds_5} = 2.5, \qquad \frac{ds_5}{ds_6} = 4,$$

$$\frac{ds_2}{ds_7} = 2, \quad \frac{ds_7}{ds_8} = 5, \quad \frac{ds_3}{ds_9} = 5, \quad \frac{ds_9}{ds_{10}} = 1.5$$

Thus, our optimization problem can be formulated as follows:

$$F(\alpha_4, \alpha_5, \dots, \alpha_{10}) = min \sum_{i=4}^{10} \frac{1}{\alpha_i}$$

$$\alpha_i \leq 1, \qquad i = 4, \dots, 10$$

and

$$f_{s1}\left(1 + \alpha_4\left(1 + \alpha_5(1 + \alpha_6)\right)\right) + f_{s2}\left(1 + \alpha_7(1 + \alpha_8)\right) + f_{s2}\left(1 + \alpha_9(1 + \alpha_{10})\right) < 1.$$

To solve this, we wrote an objective function and a constraints function in Matlab and invoked a constrained optimization routine to obtain the optimized parameter values equal to:

$$x = [0.398 \quad 0.686 \quad 1.00 \quad 0.477 \quad 0.969 \quad 0.511 \quad 1.00]$$

and the objective function value equal to 11.0673.

In this solution α_{3+i} corresponds to *x(i)*, thus using (56) we have

$$R_{d4} = \frac{R_{d1}}{\alpha_4 \cdot \frac{ds_1}{ds_4}} = \frac{5}{0.3965 \cdot 3} = 16.8138.$$

In a similar way we will obtain

$R_{d5} = 9.8082, R_{d6} = 2.4521, R_{d7} = 12.5865,\ R_{d8} = 2.5984, R_{d9} = 8.2208,\ R_{d10} = 5.4805.$

Thus, we can determine the desired resource values by using the provided hierarchy and other provided data in conjunction with the optimized values to calculate the desired resource levels.

4.6.2 Network Dependencies

4.6.2.1 An acyclic case

The previously discussed case possessed strict hierarchical dependencies in which abstract resources were stacked against lower level resources. In a more general case we may consider network dependencies between resources as shown on Figure 4-5.

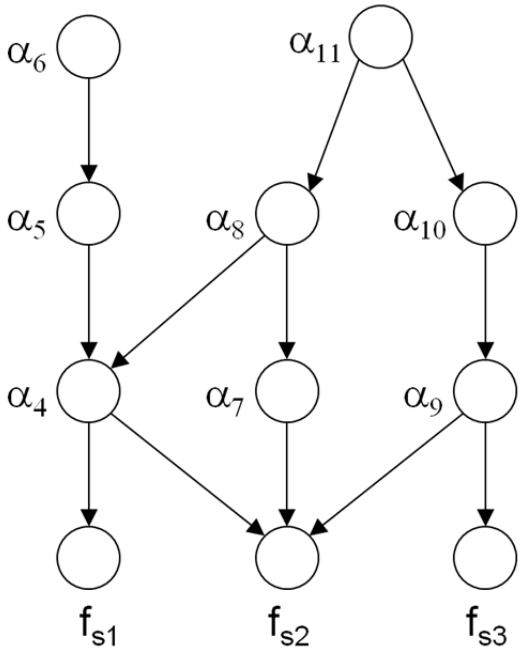


Figure 4-5. Abstract resource dependencies.

Figure 4-5 illustrates an acyclic case where there are 3 primitive resources that are used with frequencies f_{s1}, f_{s2}, and f_{s3} respectively and 8 abstract resources with dependencies illustrated by downward pointing arrows.

To establish the optimum level of desired abstract resources we can solve the optimization problem as before with only slightly modified constraints

$$F(\alpha_1, \alpha_2, \dots, \alpha_m) = min \sum_{i=1}^{m} \frac{1}{\alpha_i} \tag{57}$$

Subject to constraints

$$0 \leq \alpha_i \leq 1, \qquad i = j+1, \dots, m \tag{58}$$

and sum of all frequencies is less than 1

$$\sum_{i=1}^{d} f_{si} < 1 \tag{59}$$

where j is the number of primitive resources and each abstract node frequency $f_{\hat{s}}$ is obtained by multiplying $\alpha_{\hat{s}}$ by the sum of frequencies of lower level nodes that directly depend on it

$$f_{\hat{s}} = \alpha_{\hat{s}} \cdot \sum_{i=1}^{\hat{s}} f_{si} \tag{60}$$

Since in this case

$$\alpha_{\hat{s}_i} = \frac{\Delta_{\hat{s}_i}}{R_{d\hat{s}_i}} = \frac{\sum_{i=1}^{\hat{s}} \frac{R_{ds_i}}{ds_i}}{R_{d\hat{s}_i}}, \tag{61}$$

where summation is over all lower level resource that are dependent on resource $\hat{s}_i$, so the desired level of abstract resource can be computed from

$$R_{d\hat{s}_i} = \frac{\sum_{i=1}^{\hat{s}} \frac{R_{ds_i}}{ds_i}}{\alpha_{\hat{s}_i}}. \tag{62}$$

4.6.2.2 A general case

In general, a resource dependency graph such as that shown in Figure 4-5 may have cyclical dependencies and the way the desired resource level is established needs to be modified. In addition, when a resource can be restored in several different ways, we may use probabilities for selecting actions that lead to restoration of the resource. The two considerations can be combined in a general case of desired resource optimization.

To establish the optimum level of desired abstract resources we can solve the optimization problem as before

$$F(\alpha_1, \alpha_2, \dots, \alpha_m) = min \sum_{i=1}^{m} \frac{1}{\alpha_i}. \tag{63}$$

Subject to constraints

$$0 \le \alpha_i \le 1, \qquad i = 1, \dots, m \tag{64}$$

and sum of all frequencies is less than 1

$$\sum_{i=1}^{d} f_{si} < 1 \tag{65}$$

where each abstract node frequency $f_{\hat{s}}$ is obtained by solving the following linear equation:

$$diag\left(\frac{1}{\alpha_{\hat{s}}}\right) \cdot F_{\hat{s}} = P_{\hat{s}} \cdot F_{\hat{s}} + P_s \cdot F_s. \tag{66}$$

where $F_{\hat{s}}$ is a vector of abstract resource frequencies that need to be evaluated, $diag(1/\alpha_{\hat{s}})$ is a diagonal matrix of relative frequencies for higher level resources to be determined in the optimization process, $P_{\hat{s}}$ is a $m \times b$ matrix of action related resource selection probabilities to restore abstract resources on a lower level, where m is the total number of abstract resources, and b is a bottom part of abstract resources (excluding top level inexhaustible resources). Similarly, F_s is a known vector of primitive resource frequencies and P_s is $m \times p$ matrix of action related resource selection probabilities to restore the primitive resources and p is the number of primitive resources.

Each element p_{ij} of column j of P_s and $P_{\hat{s}}$ contain sum of probabilities of actions using resource i to restore resource j. Since sum of probabilities of all actions to restore resource j is equal to 1, then each column of P_s and $P_{\hat{s}}$ adds to 1.

We can further divide (63) to separate top level (inexhaustible resources) from other resources.

$$\begin{bmatrix} diag\left(\frac{1}{\alpha_{\hat{s}_t}}\right) & 0 \\ 0 & diag\left(\frac{1}{\alpha_{\hat{s}_b}}\right) \end{bmatrix} \cdot \begin{bmatrix} F_{\hat{s}_t} \\ F_{\hat{s}_b} \end{bmatrix} = \begin{bmatrix} P_{\hat{s}_t} \\ P_{\hat{s}_b} \end{bmatrix} \cdot F_{\hat{s}_b} + \begin{bmatrix} P_{s_t} \\ P_{s_b} \end{bmatrix} \cdot F_s \tag{67}$$

Where $diag(1/\alpha_{\hat{s}_t})$ and $diag(1/\alpha_{\hat{s}_b})$ are diagonal matrices of relative frequencies for top and bottom higher level resources, respectively. And $F_{\hat{s}_t}$and $F_{\hat{s}_b}$are vectors of frequencies of use for top and bottom resources, respectively. In a similar way we defined $P_{\hat{s}_t}$ and $P_{\hat{s}_b}$as well as $P_{s_t}$and P_{s_b} as submatrices of $P_{\hat{s}}$ and P_s that correspond to the top and bottom part of abstract and primitive resources. In (65) only F_s is known and $F_{\hat{s}_t}$ and $F_{\hat{s}_b}$ must be evaluated.

This linear equation can be solved in two steps, first to obtain $\mathrm{F}_{\hat{\mathrm{s}}_\mathrm{b}}$ using

$$diag\left(\frac{1}{\alpha_{\hat{s}_b}}\right) \cdot F_{\hat{s}_b} = P_{\hat{s}_b} \cdot F_{\hat{s}_b} + P_{s_b} \cdot F_s \tag{68}$$

so

$$F_{\hat{s}_b} = \left(diag\left(\frac{1}{\alpha_{\hat{s}_b}}\right) - P_{\hat{s}_b}\right)^{-1} \cdot P_{s_b} \cdot F_s \tag{69}$$

after which $F_{\hat{s}_t}$can be obtained directly from

$$diag\left(\frac{1}{\alpha_{\hat{s}_t}}\right) \cdot F_{\hat{s}_t} = P_{\hat{s}_t} \cdot F_{\hat{s}_b} + P_{s_t} \cdot F_s. \tag{70}$$

Since we can estimate that

$$\Delta_{\hat{s}_i} = \sum_{i=1}^{\hat{s}} p(\hat{\mathrm{s}}_\mathrm{i})_{\mathrm{s}_\mathrm{i}} \cdot \frac{\mathrm{R}_{\mathrm{ds}_\mathrm{i}}}{\frac{\mathrm{ds}_\mathrm{i}}{\mathrm{d}\hat{\mathrm{s}}_\mathrm{i}}} \tag{71}$$

where $p(\hat{\mathrm{s}}_\mathrm{i})_{\mathrm{s}_\mathrm{i}}$ stands for the probability of using resource $\hat{\mathrm{s}}_\mathrm{i}$ to restore resource s_i.

Then after solving the optimization problem, the desired resource level of various abstract resources can be obtained as:

$$R_{d\hat{s}_i} = \frac{\Delta_{\hat{s}_i}}{\alpha_i}. \tag{72}$$

Since the solution of (71) may depend of $R_{d\hat{s}_i}$that needs to be determined from (72), we have to solve this linear problem in a similar way as (67) by subdividing it into smaller parts. Let us first define a matrix

$$\chi(\hat{\mathrm{s}}_\mathrm{i}, \mathrm{s}_\mathrm{i}) = \frac{p(\hat{\mathrm{s}}_\mathrm{i})_{\mathrm{s}_\mathrm{i}}}{\alpha_i * \frac{\mathrm{ds}_\mathrm{i}}{\mathrm{d}\hat{\mathrm{s}}_\mathrm{i}}} \tag{73}$$

Desired resource values can be obtained from

$$R_{d\hat{s}_i} = \chi(\hat{\mathrm{s}}_\mathrm{i}, \mathrm{s}_\mathrm{i}) \cdot \mathrm{R}_{\mathrm{ds}_\mathrm{i}}. \tag{74}$$

Next, we partition (74) to separate top level resource $R_{d\hat{s}_t}$ (inexhaustible) bottom level abstract resources $R_{d\hat{s}_b}$ and primitive resources $R_{d\hat{s}_p}$ (with given desired resource levels):

$$\begin{bmatrix} R_{d\hat{s}_t} \\ R_{d\hat{s}_b} \end{bmatrix} = \begin{bmatrix} \chi(\hat{\mathrm{s}}_\mathrm{i}, \mathrm{s}_\mathrm{i})_{tb} & \chi(\hat{\mathrm{s}}_\mathrm{i}, \mathrm{s}_\mathrm{i})_{tp} \\ \chi(\hat{\mathrm{s}}_\mathrm{i}, \mathrm{s}_\mathrm{i})_{bb} & \chi(\hat{\mathrm{s}}_\mathrm{i}, \mathrm{s}_\mathrm{i})_{bp} \end{bmatrix} \cdot \begin{bmatrix} R_{d\hat{s}_b} \\ R_{d\hat{s}_p} \end{bmatrix} \tag{75}$$

We can solve

$$R_{d\hat{s}_b} = [\chi(\hat{\mathrm{s}}_\mathrm{i}, \mathrm{s}_\mathrm{i})_{bb} \quad \chi(\hat{\mathrm{s}}_\mathrm{i}, \mathrm{s}_\mathrm{i})_{bp}] \cdot \begin{bmatrix} R_{d\hat{s}_b} \\ R_{d\hat{s}_p} \end{bmatrix}. \tag{76}$$

To obtain $R_{d\hat{s}_b}$from $R_{d\hat{s}_p}$

$$R_{d\hat{s}_b} = (1 - \chi(\hat{\mathrm{s}}_\mathrm{i}, \mathrm{s}_\mathrm{i})_{bb})^{-1} \cdot \chi(\hat{\mathrm{s}}_\mathrm{i}, \mathrm{s}_\mathrm{i})_{bp} \cdot R_{d\hat{s}_p}. \tag{77}$$

And then compute $R_{d\hat{s}_t}$

$$R_{d\hat{s}_t} = [\chi(\hat{s}_i, s_i)_{tb} \quad \chi(\hat{s}_i, s_i)_{tp}] \cdot \begin{bmatrix} R_{d\hat{s}_b} \\ R_{d\hat{s}_p} \end{bmatrix}. \tag{78}$$

4.6.2.3 Resource dependency setting: Example 2

To illustrate this general case let us consider the resource dependencies graph shown in Figure 4-6.

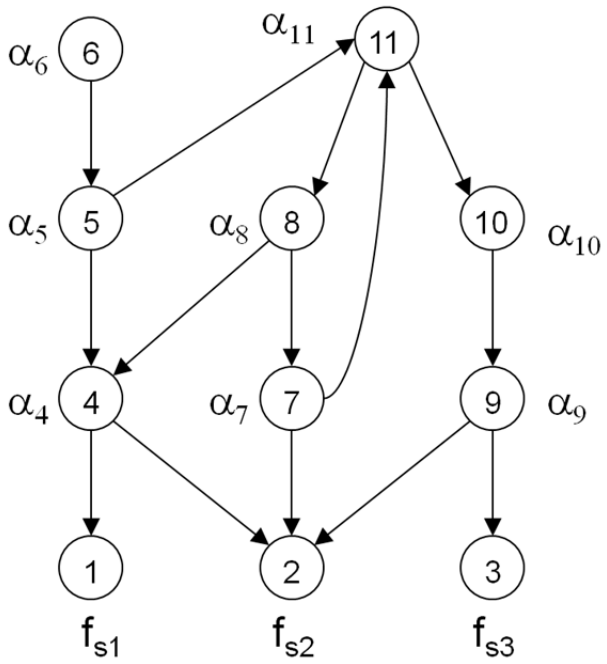


Figure 4-6. Graph with cyclical resource dependencies.

Assume that related resource-to resource probability matrix *P* is as follows:

$$P = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0.2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.4 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0 & 0.6 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.3 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

where P_{ij} shows probability of using resource *i* to restore resource *j*, and the matrix of derivatives of the dependent lower level resource over the higher level resource DS is

$$DS = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 2.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2.2 \\ 0 & 0 & 0 & 5 & 0 & 0 & 1.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 7 & 0 & 0 & 0 & 0 & 0 & 2.4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 2.7 & 0 \end{bmatrix}$$

$DS(i,j)$ indicates the derivative of *j* wrt. *i*. Notice that this order was flipped as compared to matrix *P* for easier calculation of $\chi(\hat{s}_i, s_i)$. In the resource dependencies graph resources 1, 2 and 3 are primitive resources, and resource 6 is a top-level resource. The desired level of primitive resources is known

$$R_{d\hat{s}_p} = \begin{bmatrix} 20 \\ 12 \\ 21 \end{bmatrix}$$

and with $\alpha_{\hat{s}_t} = [0.3686 \quad 0.7620 \quad 0.5446 \quad 0.5391 \quad 0.3934 \quad 1.0000 \quad 0.6581 \quad 0.5318]$

We can solve (74) and (75) to get the desired level of all resources $R_{d\hat{s}_b} = [\ 20.691 \quad 9.380 \quad 14.238 \quad 30.443 \quad 4.200 \quad 2.659 \quad 30.472\ ]'$

and

$$R_{d\hat{s}_t} = [\ 8.6127].$$

4.7 Conclusion

In this chapter we have examined the numerous improvements that have been made to the basic ML algorithm presented in Chapter 3. The improvements consist of adding 'opportunistic' action selection heuristics, the addition of Non-Agent Characters (NACs), modification to bias, weight adjustment changes, and pain handling to support NACs and improve learning. An alternative, probabilistic, method for selecting actions in a more diverse manner has also been covered. Additionally, section 4.6 was allocated to detailing how to determine how much of a resource or resources to use when performing an action, and how to set the target (or desired) level of a resource when calculating resource usage or restoration.

The next chapter presents the results of various tests performed using the ML algorithms defined in Chapters 3 and 4. It begins with simulation results using the basic Motivated Learning algorithm from Chapter 3, extends this to present results from the more complex ML implementation in Chapter 4, and a NeoAxis based implementation of the same algorithm (e.g. a graphical interface and environment for the model). However, results and simulation of the more complex simplified MLECOG model are not presented until Chapter 7.

CHAPTER 5: SIMULATIONS AND RESULTS

5.1 Introduction

This chapter presents and discusses the simulation results of the motivated learning algorithm, its implementations, and associated architectures, as discussed in Chapters 3-5. Section 5.2 elaborates on the simulation methodology and testing environment. It starts by outlining basic ML scenarios and discusses how the scenarios are implemented in both Matlab and NeoAxis for the various implementations simulated and discussed in the successive subsections. The rest of the chapter presents simulation results from the basic ML algorithm as presented in Chapter 3 and a more complex implementation of ML with the enhancements that were discussed in Chapter 4. Chapter 6 will present a comparison of ML to similarly configured RL algorithms using a black box environment. The MLECOG algorithm, its simplified implementation and results are discussed in Chapter 7. The next section starts by outlining a basic ML scenario and presents the results from a motivated learning simulation on that scenario.

5.2 Creating the Simulation Environment

This section describes the basics of the environments simulation used to test the various ML based implantation discussed in this work. This is only to be a general overview of what goes into the environment implementation. More detail about the specifics of the environment used in each simulation is provided alongside the associated results. It is important to have a good general understanding of the methodology that was used for the creation of the various environments, what was needed to create them, and what was taken into consideration when doing do. (This discussion on using Matlab and NeoAxis for simulating environment was previously included as part of a journal paper [99].)

Some “simulation” environments readers of this work might be familiar with may come from games like The Sims, Call of Duty, and Halo, or social networking environments like Second Life. They can be thought of as simulation environments because they simulate, with caveats, facets of real life, in which their players can immerse themselves and even use them to learn. However, the environments of this dissertation have more in common conceptually with simulators. For example, a flight simulator simulates actual flight and provides feedback in the form of various gauges and a simulated external view in order to train a pilot on “good” and “bad” reactions to a controlled situation (environment). A good simulator can immerse the user

well enough that they react realistically to perceived changes and issues within the simulated environment.

While the simulated environments used in this work are below the complexity of a flight simulator, they provide a necessary starting point and act as a base from which to build more complex and advanced environments. The principal factor keeping the ML agents from more complex simulation is that they are not yet ready to handle that level complexity and the environments have to be coded to "pass" the data to the agents. For example, in order for an ML agent to handle environment complexity as desired (by using primarily visual input), the agent has to have the tools to process and recognize the input properly, and this requires the implementation of a semantic memory. Unfortunately, only the pseudo-semantic memory is currently available. The creation of a semantic memory for the MLECOG model is a significant undertaking, and it will be some time before it is available. However, it is being worked upon [102]. Next, is a brief discussion about the development of the Matlab and NeoAxis simulation environments, consisting of how they progressed from their initial states to their state at the time of writing. While the specifics of the individual simulation runs are not discussed, the development, capabilities, and basic assumption of each environment type are covered.

5.2.1 MATLAB Environment

The Matlab simulation environment was the first to be developed due to the ML model's initial design via Matlab. This subsection describes the development of the Matlab based environment and how it reached its current form. The initial implementation had very little separation between the environment and the agent. In a modified ML agent implementation, the environment and agent were separated into two structures.

The first implementation of the ML environment only considered the initial capabilities of the ML agent, which, at the time, were designed to work only with the probabilities of finding resources in the environment. Therefore, the initial environment simply tracked how often each resource was observed and generated a probability for that resource to show up on the sensory input. This information was then fed to the bias equation (1).

This approach worked well for the initial experiments, but proved insufficient for more complex behaviors, which led to the introduction of two significant changes. First, resource probabilities were replaced with discrete quantities, leading to more complex bias equations discussed in Section 4.3. Additionally, resources now had resource restoration ratios associated with them. For example, in a valid action consuming one unit of resource A might restore two (or

more) units of resource B, then the corresponding resource restoration ratio was 2. This allowed for wider flexibility in resource utilizations (there could be multiple resource for a specific action but each resource could possess different restoration ratios) and increased the complexity of the agent, since the agent now had to determine how much of a resource to use to resolve its pain.

The second significant change to the Matlab implementation was the introduction of "motion" to the agent. This was done by creating a virtual grid in which the agent could move and by placing the resources in (usually) random positions in the grid. Doing this introduced the concept of "travel time" to the agent and a need for the introduction of "action time" or the time the agent needs to perform an action once it reaches the target resource location. The introduction of motion also brought about the possibility and eventual realization of the OML approach to the ML algorithm.

There have been other additions to ML implementation. For example, NACs, introduced in Chapter 4, added competing pseudo-agents to the environment, which the agent needed to learn how to handle. Another change was how the "movement grid" of the environment was implemented. Initially, it was kept as a discrete space, as depicted in Figure 5-1. The environment is divided up into a grid, where the agent can move between cells (spaces defined by the grid lines), and resources and NACs can reside within them. In this space, the agent could only move from one "cell" in the grid to another. This was modified so that the agent could move more freely, up to one unit or cell in any direction, allowing for more realistic motion of the agent. This change was made to keep parity with the NeoAxis implementation (discussed next) whose 3D graphical environment defaults to floating point distances and position values.

Figure 5-1 represents an example of a Matlab environment configuration. In the 10x10 grid configuration, there are 8 resources, represented by the numbers 1-8, positioned at various locations in the grid. The agent is represented by an 'A', while the arrows indicate the cells to which the agent can move in the next iteration. Using this approach an ML agent is capable of moving in the simulated environment, interacting with other agents, selecting how much of a resource to consume, and engaging in opportunistic based behavior. While the Matlab implementation remains relatively simple, it was decided that more complexity will be added using a more advanced graphical and simulation engine, NeoAxis, which is discussed in the next subsection.

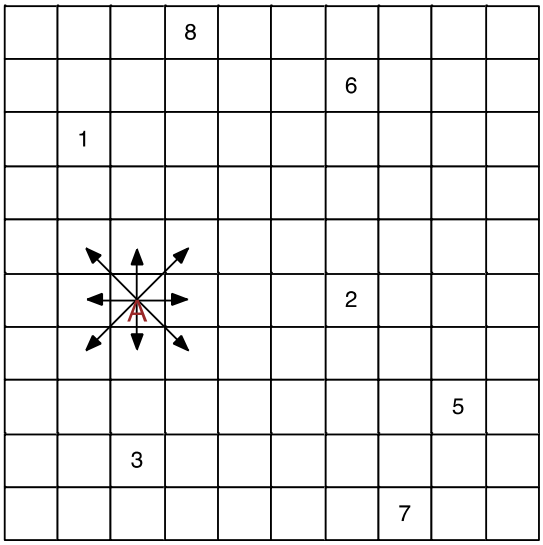


Figure 5-1. Graphical representation of Matlab environment “space”.

5.2.2 NeoAxis Environment

The implemented ML learning agent needs to be tested in a real time application. This can be done either via robotics, where the agent controls a robotic body, or via simulation, where the agent exists within a virtual environment and possesses a virtual “body.” Both of these approaches require an embodied agent central to Embodied Cognition, initiated in the early-90s with the work by Brooks [28] and Varela [103]. The premise behind Embodied Cognition is that the machine’s interaction with its environment is predetermined by its embodiment and that intelligence cannot develop without embodiment.

This approach adheres to the idea that the agent’s body shapes its development, because the motor capabilities and sensory apparatuses of the body together with the abilities of its mind, intimately effect how the agent functions.

Robotics platforms can be costly to design, build and operate, can cost upwards of $100,000, and are more suited to latter stages and implementation of Intelligent Agents (IA) than they are to the early stages of development of an IA scheme. However, it is still necessary to create an environment in which the robot (simulated or otherwise) can operate. In a simulation there is much more freedom to design the agent and its environment, more freedom to change things during development, and physical hardware costs are limited to the costs of computers. Simulation and virtual environments were used to develop PSI agent in [58], use learning agents as non-player characters in computer games [89], and build robotic training platforms, [104],

[105]. For these reasons, we chose to use simulated virtual agents. Using a simulation approach makes it easier to demonstrate ML agents' improved learning capabilities when the agent adapts and changes its behavior, and allows for easier examination, tracking, and recording of agents' performance. Also, a simulated environment allows both greater and finer control of simulation parameters.

The NeoAxis SDK [16] was used to expand the agent's capabilities and to test the ML agent in an alternative environment [106]. This was accomplished by implementing the ML agent functionality in C++ (as discussed in 5.2.2).

While Matlab has the potential capability to simulate just about anything needed, the one thing it is not very good at is presenting a graphical environment. Hence, a search for potential graphical simulation environments was undertaken. Several such environments were investigated, however, NeoAxis[16] was chosen because it offers an accessible 3D simulation environment, comes with existing graphical assets (decreasing the amount of work needed), and has a robust developer community. It is available as a free SDK (software development kit) with existing demo assets and full set of editing programs that allow the creation of new assets and maps with which to create and define new environments. Additionally, a programmer can make modifications to the code itself. The only thing off limits (unless a paid version is acquired) is the inner workings of the graphical engine. The presence of several terrain maps also helps to further reduce the work in developing a 3D environment.

The process of embedding the ML agent into NeoAxis started with determining how complex the initial environment needed to be. At the time, the Matlab based implementation had both discrete resource and movement components, but lacked the NAC based changes. Therefore, it was decided to implement a similar system in NeoAxis. A scenario was designed with appropriate graphical objects and resources. Figure 5-2 shows the view of a scenario from within the map editor. A new class of objects, "resources" was created and appropriate features were added to it. Figure 5-3 provides an example of one of these resources, in this case it is an 'apple', an example of a 'food' resource for the ML scenarios. The features present in a resource object consist of the objects' current quantity, desired and/or initial quantities, and a unique resource ID. While this set of features is limited, more can be easily added if it becomes necessary in the future. The 'resource id' is the equivalent of the numerical resource identification in the Matlab implementation. While an image recognition based approach would be more appropriate for a graphical representation, the main focus of the work was the ML algorithm itself and the graphical object recognition approach was left for the future research. However, at the time of

writing it is being worked upon in the form of a semantic memory based recognition algorithm [102].

Embedding the ML agent into NeoAxis was simple, but non-trivial. Since at the time of porting the most recent version of the ML code was OML based, the OML Matlab code was used for the port. The OML agent was ported with only minor changes to the code, mostly dealing with associating environment object IDs with their equivalents in the agent's structure. It did, however, take a significant amount of time to port, simply due to the fact that the code had to be ported from Matlab to C++, with the added fact that the Boost Library [107] had to be used to allow the implementation of matrices within C++.

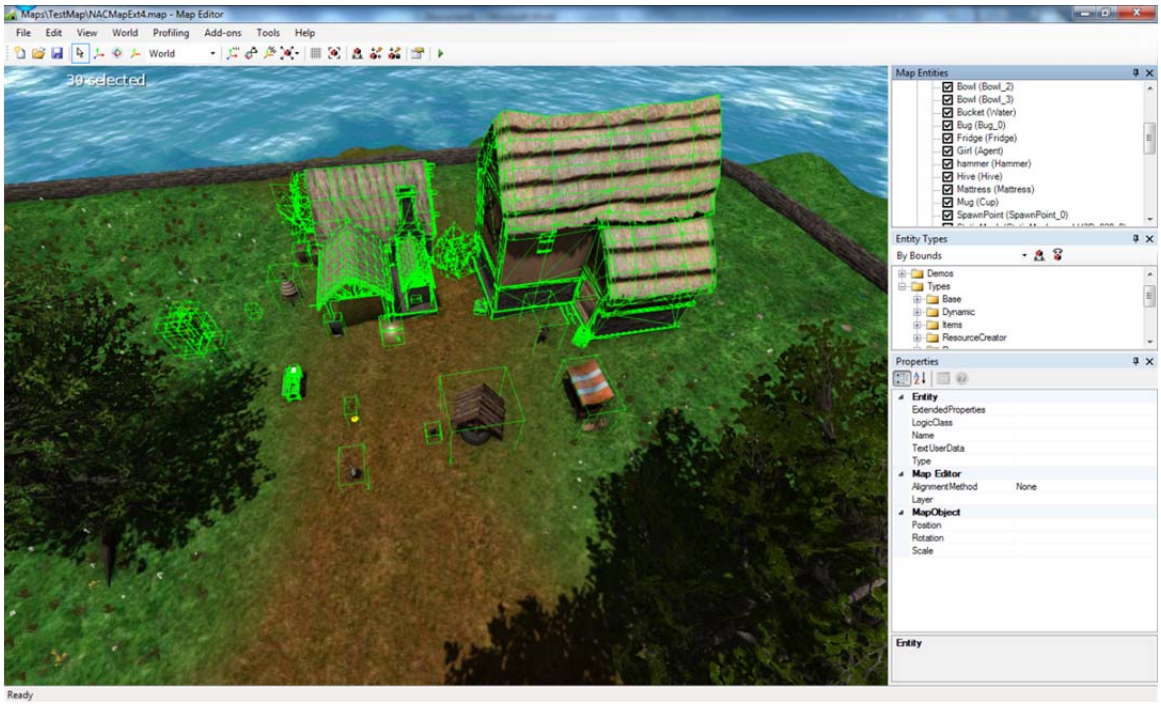


Figure 5-2. A NeoAxis environment from within the MapEditor.

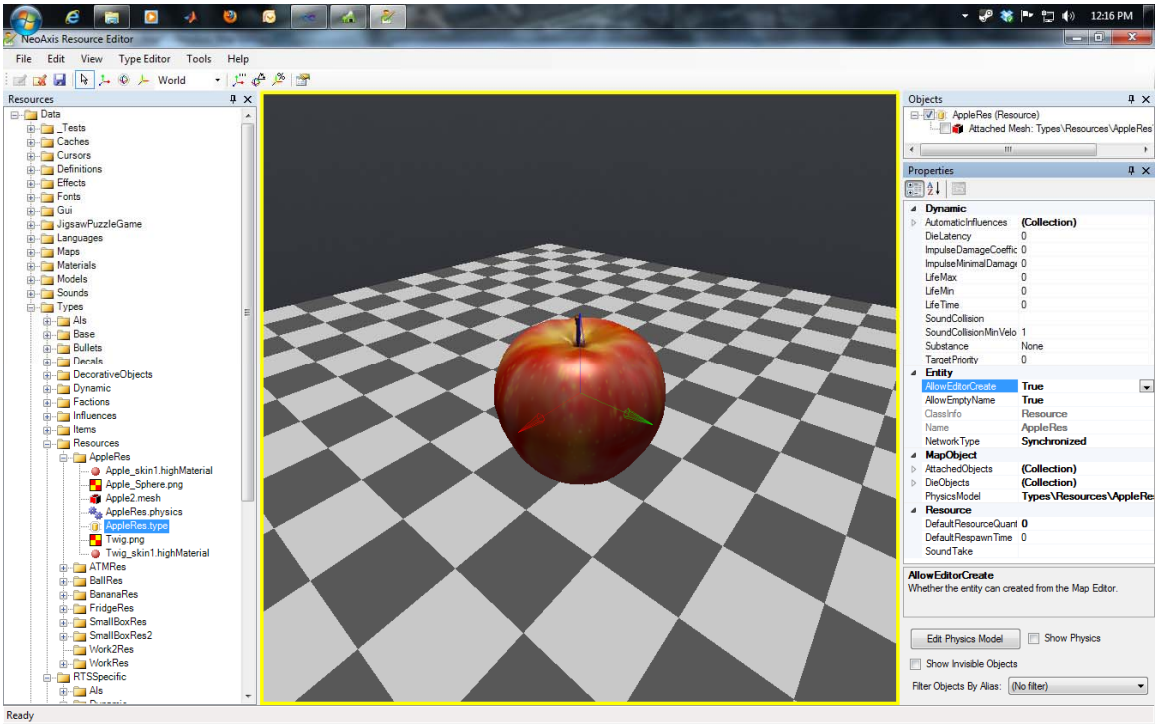


Figure 5-3. Example of an ‘Apple’ resource object.

Aside from porting the agent to NeoAxis, several changes had to be made to the NeoAxis code base to run the agent. These changes involved three main things, first, interfacing the ML agent’s code with a NeoAxis AI and modifying the AI code to send and receive data from the ML code and interpret the results, second, provide an environment definition for the AI and ML agent to work from, and third, provide a visual interface to the user (to indicate the ML agent’s status). This is important, since the NeoAxis AI acts as the embodiment of the ML agent in the NeoAxis environment. It provides sensory input to the ML code and actually performs the actions selected by the ML agent. Conceptually speaking, the ML agent code is the mind and the NeoAxis AI code is the body, while the rest of the NeoAxis code is the environment that it all runs in.

Figure 5-4. Early implementation example with debug lines showing agent navigation.

Figure 5-5 shows the user interface developed for the environment. It is primarily for the user's convenience so that the agent's state can be observed. It consists of a list of currently discovered resources, the task the agent is attempting, the current winning pain, and a list of known pains. It is worth noting that as the agent discovers new resources and their desirability, the ML algorithm creates the associated pains. This NeoAxis implementation and user interface is of particular importance, because, aside from the information it provides to a user, it provides visual proof of the ML agent's capabilities. And visual proof of the agent being able to operate successfully in a 3D environment is more intuitive and valuable than the plots and charts generated by the previous Matlab simulation.

Figure 5-5. Example of environment interface in NeoAxis.

5.3 Basic ML Simulation Results

In this section, some early results from the baseline ML algorithm covered in Chapter 3 are presented and discussed. The more advanced implementations discussed following Chapter 3 maintain many of the same characteristics as the following figures since those implementations also possess the same ML properties. (Some of the material in this section was drawn from a prior journal publication [20].)

The environment used in this simple experiment is shown in Table 5-1. Table 5-1 represents a simple linear hierarchy of resources, where each resource can be restored by the resource "above" it until the "top" level is reached. The resource at the top level is usually unlimited. The "Increases" column describes the result of the motor action on the state of the environment or internal state of the machine and indicates an improvement in the state of one of the available resources and corresponding decrease in the pain associated with that resource. The "Decreases" column describes the result of the motor action on the reservoir of goods related to that motor action. Thus, when the supply of a particular item in the "Sensory" field is low, the

machine will attempt to take the appropriate "Motor" action to increase the supply. For example, as time passes, the machine's sugar level decreases (meaning the hunger pain grows), which leads the machine to take action to reduce this primitive pain. It will learn to do this by eating food. However, while eating food alleviates the pain, it decreases the available food supply, leading to a corresponding increase in the "lack of food" abstract pain.

Table 5-1. Meaningful Sensory-Motor Pairs and their Effect on the Environment.

SENSORY	MOTOR	INCREASES	DECREASES
Food	Eat	Sugar level	Food supplies
Food at grocery	Buy	Food supplies	Money at hand
Money from bank	Withdraw	Money at hand	Spending limits
In the office	Work	Spending limits	Job opportunities
At school	Study	Job opportunities	-

This basic implementation of ML calculates Bias as a result of estimated resource probability (see eqn. 1 from Chapter 3) and, hence, the agent will know the likelihood of finding a resource in the environment. Figure 5-6, depicts the probability that a resource is available to the agent. For ease of visualization, this, and several of the other figures have been zoomed in upon.

w_{BP} weights vary during the learning process as shown in Figure 5-7, and are used to calculate pains represented in Figure 5-8. Figure 5-7 shows that when the agent discovers that a resource is useful then its associated w_{BP} weight goes above zero. One should be able to see the correlation between resource levels in Figure 5-6 and pain in Figure 5-8. (The Primitive Hunger pain is hidden is this figure so that it does not obscure the other signals.)

Figure 5-9 depicts the frequency at which specific goals are selected. For example, for the Primitive Hunger pain Goal 1 is selected with the highest rate and is selected roughly 50% of the time out of all actions. Note that "higher level" actions tend to be selected less frequently as can be seen from the varying axes scales in Figure 5-9. It is apparent in Figure 5-9 that the agent was able to learn correct actions for each pain as indicated by the bars in the figure. Each pain has a single dominant action (as indicated by the longest bar associated with each pain) that it has learned to resolve that particular pain.

Figure 5-10 shows pain over time averaged over a 100 simulation runs. In this figure we can see the time instance when a pain is discovered. We can also see that the agent learns to

manage the various pains, by observing that each pain, after the initial peak, eventually reaches equilibrium. In Figure 5-10, the primitive pain is at the very bottom, while abstract pains are plotted in consecutive rows. Figure 5-11, is a scatter plot of the goals selected by the agent, and is a convenient visual method for observing what actions the agent chooses. In this plot, green dots represent correct actions, while red dots represent invalid or incorrect actions. After the first couple of hundred iterations it is noticeable that the agent has learned the various correct solutions, since the frequency of red dots decreased and the green dots form distinct lines.

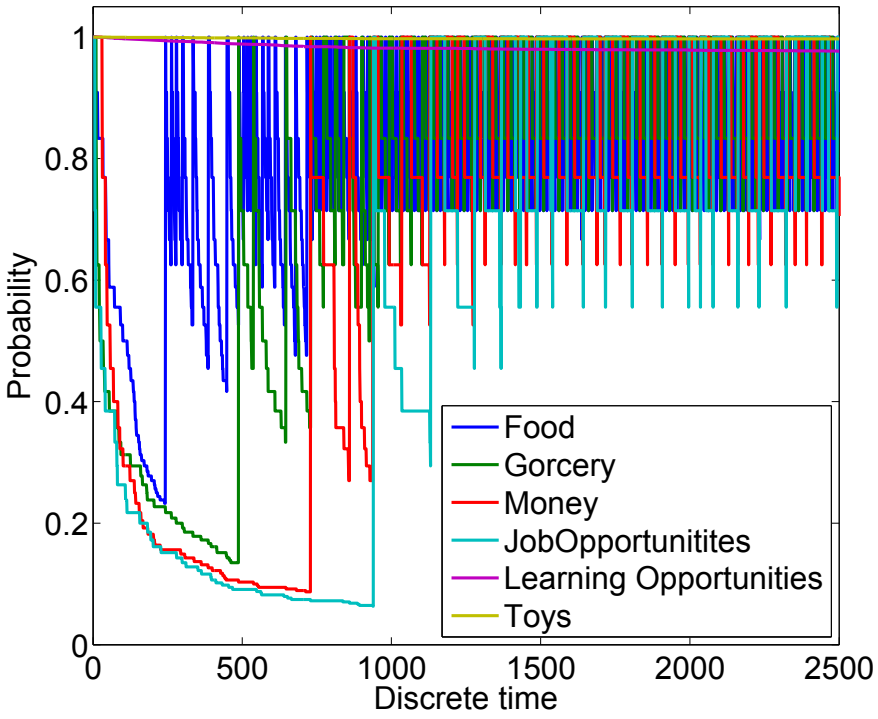


Figure 5-6. Resource probabilities.

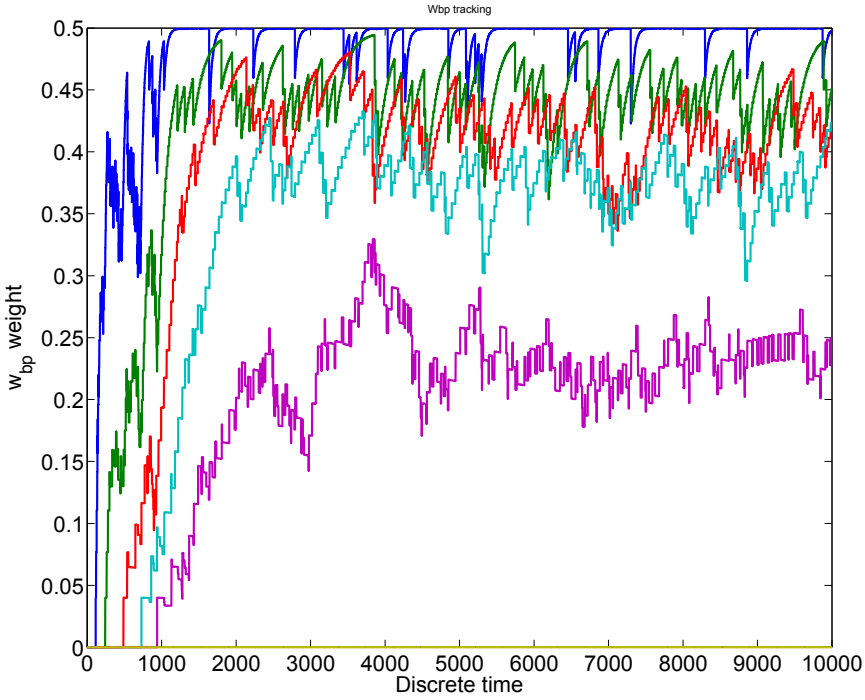


Figure 5-7. Changes in bias to pain weights over time.

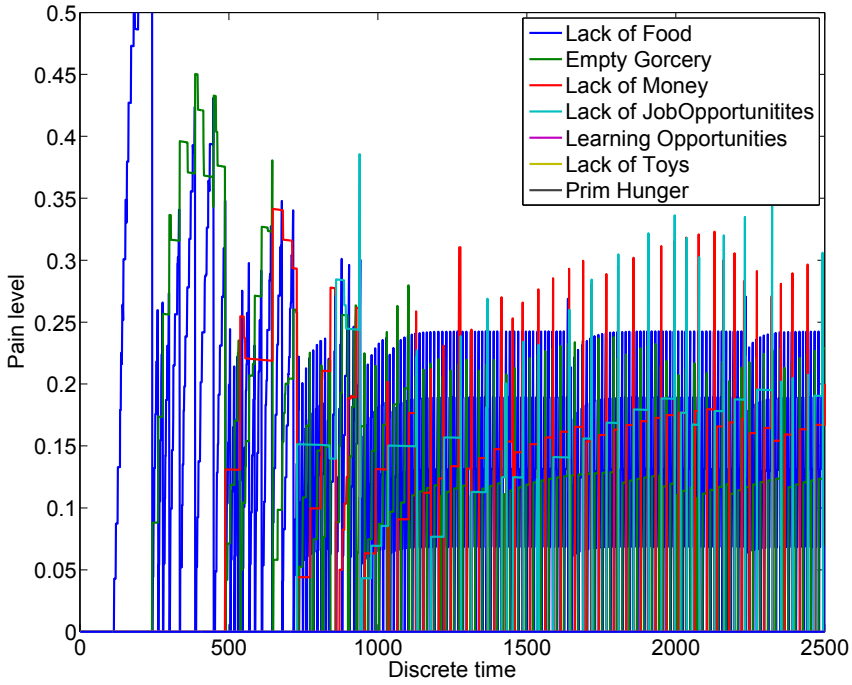


Figure 5-8. Pain levels with respect to simulation time.

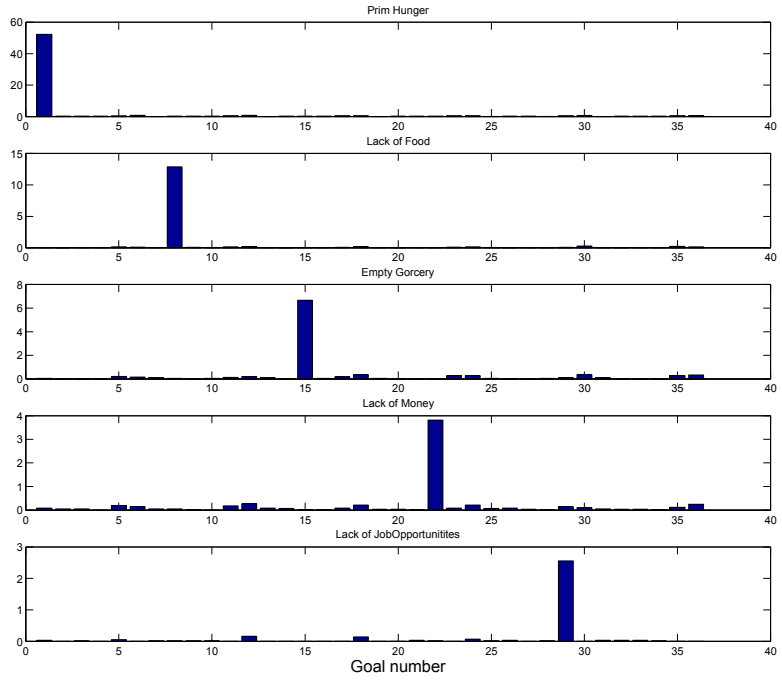


Figure 5-9. Goal selection frequency.

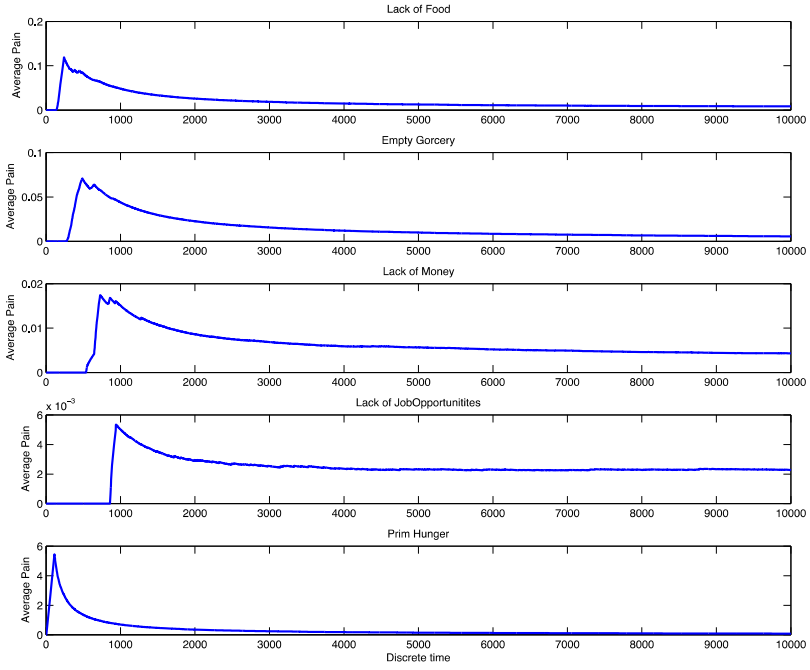


Figure 5

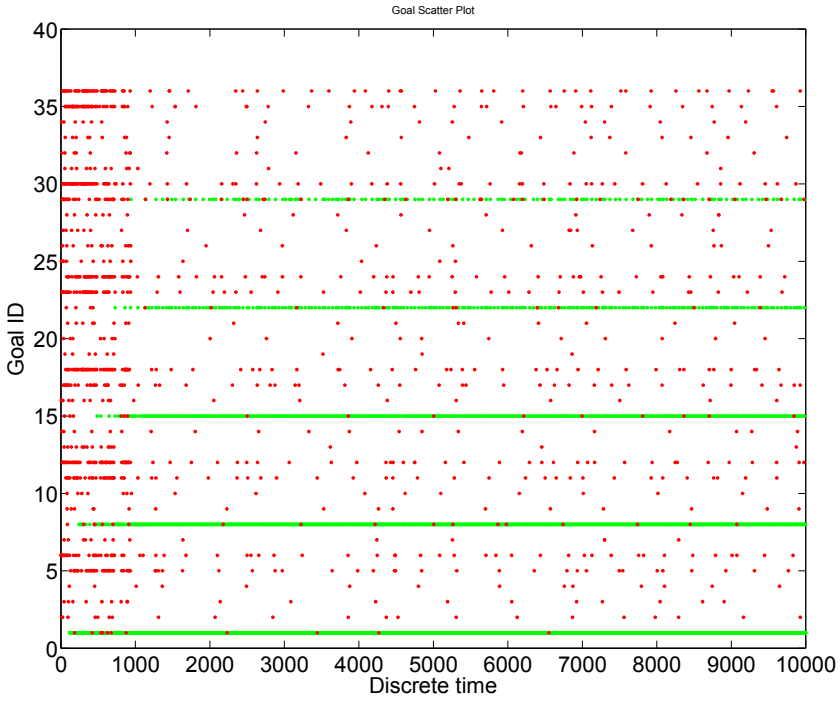


Figure 5-11. Goal scatter plot.

5.4 OML Simulation

Section 4.1.1 and [99] discussed the reasoning and mathematical underpinning behind the use of Opportunistic Motivated Learning (OML). This section discusses the computational efficiency of the OML algorithm vs. the non-OML alternatives, as well as the few variations of OML presented in Section 4.1.1 and [99]. (Note: the results in this section were previously presented in our work in [99].)

5.4.1 OML Computational Efficiency

To compare the efficiency of the proposed solution to opportunistic agent behavior the description of the problem is transformed to a format compatible with the asymmetric travelling salesman problem (TSP) described by a directed graph. Since the TSP is one of the most researched problems in discrete optimization and has a large number of heuristics solutions, both efficiency and computational cost of the OML algorithms can be compared. This section proceeds to do just that by looking at the effect each algorithm has on pain levels, computational time, etc.

Let us consider the Quadratic Heuristic (QH) procedure applied to the problem described in Example 2 in Section 4.1.1 (see the location of Table 4-4). This procedure results in the

directed graph shown in Figure 5-12. The matrix V from (18) and graph in Figure 5-12 are used for the standard TSP. According to the QH algorithm a combined distance matrix with average pain at each location is included in the graph through the calculation of matrix M (14). Another important point is that the agent always starts from node 1, visits every node once, and does not go back to the starting point.

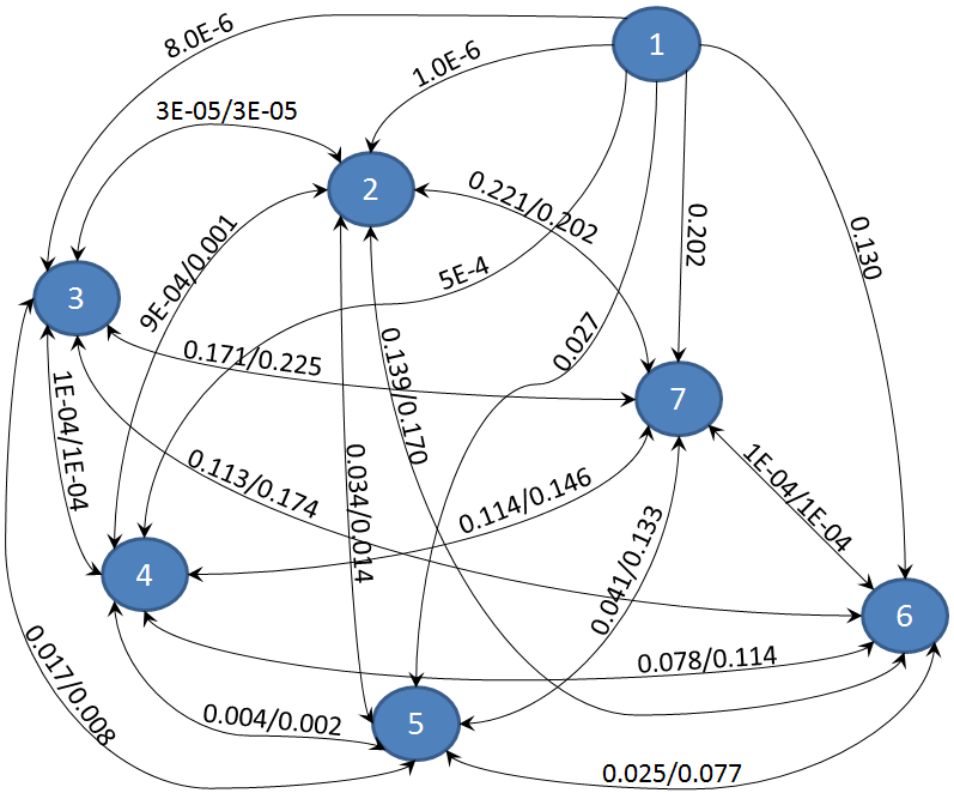


Figure 5-12. Directed graph with 7 nodes and 36 edges corresponding to matrix V (18).

The goal of the TSP is to find a minimum total path length that a salesman will have to travel starting from the initial node (node 1) to visit all nodes in the graph. Christofides showed [108] a polynomial time algorithm for TSP. The total length obtained by Christofides' algorithm L_a is less than 3/2*L_{min}, where L_{min} is the global minimum that can be found by using a heuristic process. In our case the shortest total path for the TSP is determined by an exhaustive search of the graph shown in Fig. 5-12, which results in a length of 5701. This means that a polynomial time algorithm can find a solution with the total path (equivalent to the total accumulated pain signal) less than the Christofides limit $L_a < 3/2 * 5701 = 8551.5$.

In addition, in [109] authors demonstrated a tighter bound for Christofides' algorithm $L_a \leq \frac{3\mathrm{m}-1}{2\mathrm{m}} L_{min}$ where m is the largest integer not greater than $n/2$ and that the bound is tight (reaching equality) for $n>6$. In our case with $n=6$, $m=3$, Christofides algorithm gives $L_a = \frac{3*3-1}{2*3} 5701 = 7601.32$.

As can be seen from Table 5-2, both heuristic algorithms described in this dissertation produce cumulative pain results below the limits of algorithmic solutions as specified by Christofides. By running the QH algorithm for Example 2 we got a route visiting nodes d_{QH}={1, 2, 3, 4, 6, 5} and total pain of 6519, which was better than 99.58% of all cases. The Linear heuristic algorithm did even better in this case with total pain equal to 5701 (the global optimum).

According to the linear heuristic (LH) algorithm, the opportunistic agent will chose the route visiting nodes d_{LH}={1, 2, 3, 4, 5, 6}, while the "regular" ML agent will visit nodes in the order of decreasing pain signal values d_{ML}={5, 1, 3, 6, 2, 4} and will suffer 4.15 times larger cumulative pain than the opportunistic agent. Using exhaustive search for this example, all cumulative pain solutions are shown with the histogram in Figure 5-13 with a mean pain value of 19229 and standard deviation of 5728. The minimum total pain was 5701, and the maximum was 32159. Total accumulated pain for the opportunistic agent was 5701, while for the motivated learning agent it was 23701. The opportunistic agent was equal or better than 99.86% of all cases, while ML was better than only 24.58% of cases.

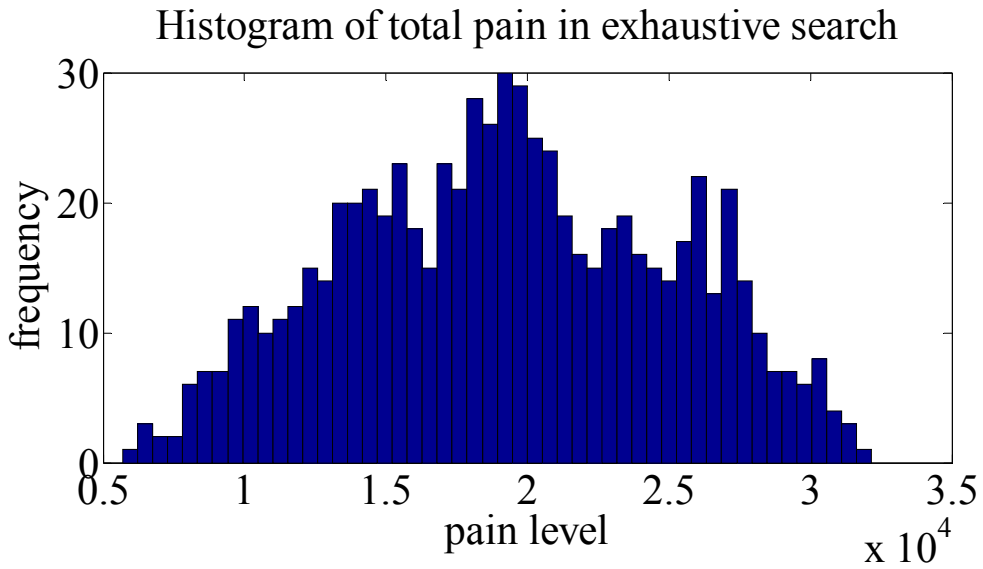


Figure 5-13. Histogram of all solutions obtained by using an exhaustive search

The LH and QH algorithms have been tested against Christofides' bounds on a number of graphs starting from the graph in Figure 5-12 and gradually increasing the problem complexity by adding more nodes (pain centers). For each size of the graph the test was repeated with randomly generated graph structure. The average results with standard deviations obtained after 20 runs of algorithms are shown in Table 5-2.

Table 5-2. Comparison of Total Pain to Christofides Limits

Number of Nodes	**Christofides Limit [35]**	**LH**	**QH**
12	$L_a \leq 18389.8$	12981 (±1680.3)	16118 (±3812.5)
11	$L_a \leq 14964.6$	10689 (±1067.2)	13519 (±3135.2)
10	$L_a \leq 14501.2$	10358 (±1470.3)	11376 (±1674.8)
9	$L_a \leq 11655.1$	8476.4 (±1044.7)	9144.8 (±1287.6)
8	$L_a \leq 10308.1$	7496.8 (± 1241.5)	7946.1 (±1933.3)
7	$L_a \leq 9147.9$	6860.9 (± 708.7)	7498.9 (±1080.8)
6	$L_a \leq 7438.8$	5719.1 (±795.5)	5845.5 (±1034.6)

The approach presented in Example 2 was used to formulate an equivalent TSP problem for opportunistic agent behavior and applied several popular TSP algorithms comparing both algorithmic complexity and quality of the final results. The results of analysis are shown in Table 5-3 and Figure 5-14 in ascending order of run time. Figure 5-14 shows the logarithm of the run-time of the compared algorithms as a function of the number of nodes in the resulting directed graph.

As can be seen from the presented results, the linear and quadratic heuristic algorithms are both fast and effective. Notice that although, for small graphs, the exhaustive algorithm is more efficient than ant colony, random search, or genetic algorithms, it is the most computationally expensive once the graphs have more than 9 nodes. All other algorithms maintain their run time level for tested size of graphs.

Table 5-3. Comparison of Opportunistic Agent with Several TSP Algorithms

Method	Minimum total pain	Standard Dev.	Run time in seconds	Total pain for nodes 4-12
LH Algorithm	5701	1144	0.000733	68286
QH Algorithm	6519	1994	0.007729	68971
Simulated Annealing	12425	1289	0.076700	164414
Greedy Search	13159	455	0.024973	106429
Exhaustive Search	5701	1479	0.060810	62102
Ant Colony Algorithm	8624	626	0.350591	83343
Random Search	12827	1359	0.917594	168471
Genetic Algorithm	5701	95	5.372666	62102

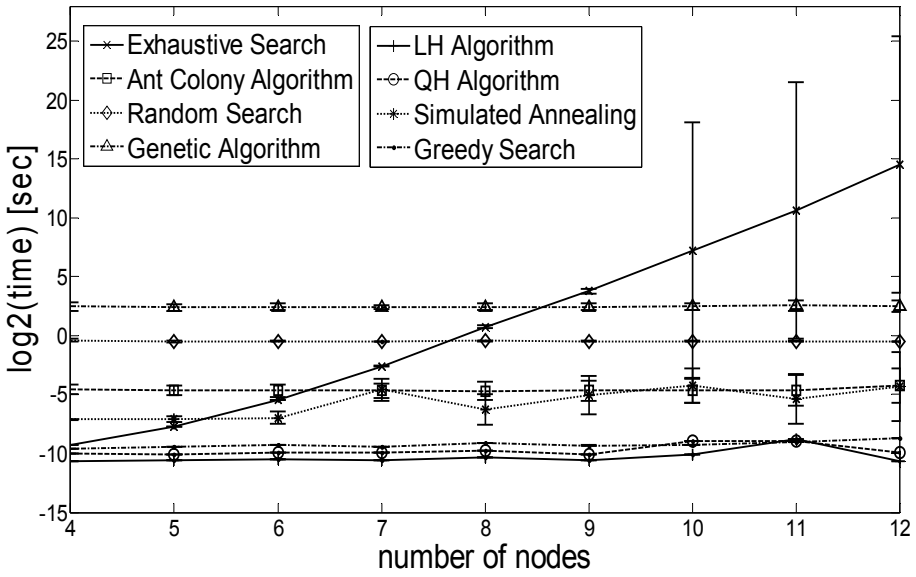


Figure 5-14. The run time for different algorithms (with standard deviation – shown as vertical lines).

According to Figure 5-14 the linear and quadratic algorithms require the least simulation time, while the exhaustive search is the most expensive one with an exponentially growing time requirement.

Figure 5-15 shows the overall pain of the compared algorithms as a function of the number of nodes. Each data point on Figure 5-15 is the average result obtained after 20 runs. Results demonstrate that both the linear and quadratic algorithms were almost as efficient as the genetic and exhaustive search algorithms that yielded the best results.

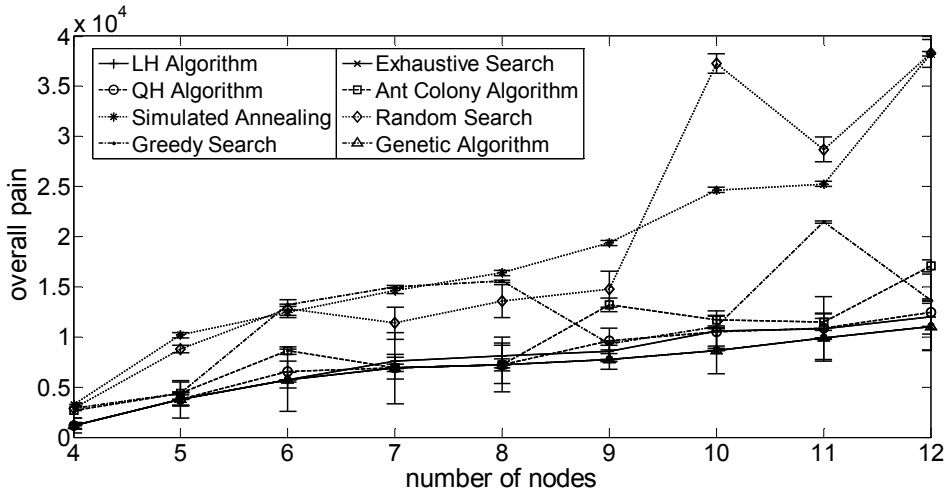


Figure 5-15. Overall pain for different algorithms (with standard deviation).

Since there is a large variability of the overall pain in various algorithms, total pain in all experiments was summarized with the number of nodes changing from 4 to 12 and presented in the last column of Table 5-3. This confirms that the proposed algorithms (QH and LH) are almost as efficient as the most computationally demanding ones, and are superior to simulated annealing, greedy search, ant colony, and random search algorithms in both their simulation time and total pain obtained.

Since, in a typical situation observed in a simulated environment an opportunistic agent must choose from 3 to 7 options (represented by the number of active pains), using the LH algorithm to control the behavior of the opportunistic agent can be recommended. The quadratic algorithm has been useful to formulate opportunistic behavior in terms of the well-studied travelling salesman problem and helped to make the final recommendation regarding the use of the linear heuristic algorithm.

The examples considered represent what an opportunistic agent may encounter while dealing with its pains. An agent may have only a small number of pains that are relevant at any given moment. Thus, the algorithm selection is specific to ML needs and corresponds to a small number of choices a cognitive agent considers in its working memory. This work does not claim

that these algorithms are effective for solving large scale TSP problems, since large scale problems are beyond its scope and would very likely take too long to test.

To summarize, the opportunistic agent uses the lowest cost path to improve every step of the motivated learning process. The learning mechanism still uses abstract goal creation and internal motivations as described in Chapter 3. Note that the heuristic algorithms used in the OML algorithm discussed here and in Section 4.1.1 are to help the agent decide how to manage the set of needs that it develops during its learning process. This should not be confused with the learning process itself. The agent learns what is good for it by interacting with the dynamically changing environment and observing results of actions, but the planning process it uses to decide what to do involves solving problems similar to TSP using one of the presented heuristic algorithms. The opportunistic extension to ML only solves a search problem, while the action taken by the agent after the search is evaluated is the basis for learning within the motivated learning approach. Specifically, actions that result in pain reduction will be reinforced, and, in addition, may lead to changes in the agent's motivations and result in new goals.

5.4.2 OML Simulation Results

OML implementation of the learning agent combines motivated learning (providing pain signals and motivations) with a heuristic algorithm (to choose the current goal). In this section four implementations of the ML algorithm are compared. The first one denoted here as ML is a standard implementation of ML where the action is chosen based on the maximum pain signal, the second and third algorithms, denoted as Linear OML and Quadratic OML, are opportunistic ML algorithms where the choice of action is based on the linear and quadratic heuristic algorithms from Section 4.1.1, and the fourth one, denoted by Exhaustive OML, is an opportunistic ML where the choice of action is based on an exhaustive search.

The performance metric in this comparison is defined as the minimum total average pain suffered by each agent that is subject to the same environment. It is assumed that the smaller the agent's total pain, the better the agent's performance.

Figure 5-16 shows a third person view of the opportunistic agent that has been implemented in NeoAxis. It shows the agent about to act on a resource object in front of itself. The depicted opportunistic agent, detects a useful resource within its view range, evaluates it against its needs and decides whether to deviate from its current objective or not. In this case, the agent's choice is determined by the linear heuristic algorithm discussed in Section 4.1.1.

Figure 5-16. Agent walking toward a target resource.

The OML algorithm chosen for use with the NeoAxis implementation was the linear heuristic algorithm. While this algorithm is not as effective as the Exhaustive Algorithm, it still provides noticeable benefits, as shown in results discussed in the following paragraphs. In the following tests, a simple "farming" scenario was used where the agent had to farm for food and keep itself fed. There are five primitive pains in the base scenario that are linked to predefined needs of the agent. The agent can satisfy its primitive needs by using various resources and create up to eight abstract needs.

Figure 5-17 shows a comparison of total average pains for four algorithms (ML, Linear OML, Quadratic OML and Exhaustive OML) using 20,000 iterations for each algorithm in a simulated scenario. The "Total Average Pain" is the cumulative sum of all pains, divided by the iteration number, and is used as the performance measure of the particular run. The agent's objective is to lower this total average pain.

As can be observed in Figure 5-17, after the agent learns how to manage new abstract pains, Total Average Pain stabilizes and moves toward equilibrium. (The dashed lines around each solid line correspond to a 95% confidence interval, determined by 2x the standard deviation of each average pain and represent results obtained after 10 simulation runs.) As can be seen, the

Exhaustive OML algorithm provides the best results with the lowest average pain at the end of the simulation, although, the difference between the Linear, Quadratic and Exhaustive algorithms is, in this case, negligible. However, when the simulations are run with a more complex environment consisting of 8 primitive and 14 abstract needs (in comparison to 5 primitive and 8 abstract needs in the baseline scenario) a greater performance benefit of the Linear OML algorithm over the other algorithms can be observed in terms of computation time (see Table 5-4). Considering that the Exhaustive algorithm is NP complete, the high effectiveness of the linear algorithm eliminates the need for a more advanced decision making process in most practical situations.

Table 5-4 shows how time requirements for the ML and OML algorithms depend on the number of pains used in the simulation. For the simulation results shown in Figure 5-17, the Exhaustive algorithm took on average 74.1 seconds compared to the 37.3 seconds needed for the Linear OML implementation, and 32.6 seconds for the ML algorithm. In such a simple environment the Exhaustive OML algorithm did not take a large amount of time due to only a few pains at a time being above threshold (3 on average), thereby keeping the computational depth of the Exhaustive algorithm's factorial level complexity low. However, in the more complex environments, more pains reside above threshold, increasing the computational time of the Exhaustive algorithm as seen in last three rows in Table 5-4. While the ML and Linear OML runs have a linear dependency on the number of pains in the environment, the factorial dependency of the Exhaustive OML on the number of pains above threshold observed in Table 5-4 is definitely a problem for real time applications.

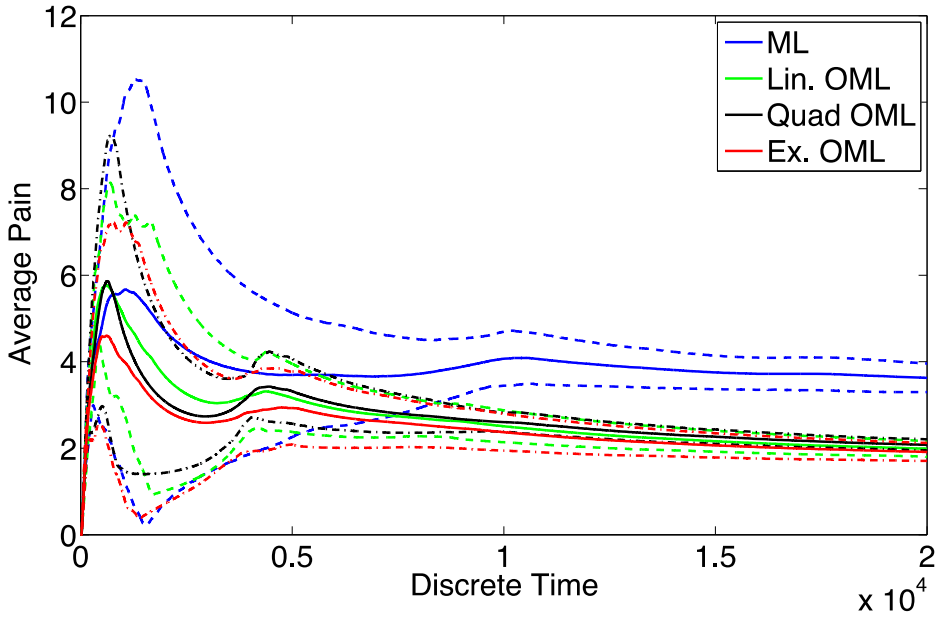


Figure 5-17. Total average pain comparison for standard ML and all 3 OML algorithms.

Table 5-4. Time Needed (s) vs. Number and Type of Pains

# and Type of Pains	ML	LIN. OML	QD. OML	EX. OML
5 Prim, 8 Abstract	32.6	37.3	40.3	74.1
6 Prim, 10 Abstract	34.4	39.2	47.0	217.0
7 Prim, 12 Abstract	38.0	45.0	52.3	312.9
8 Prim, 14 Abstract	43.7	54.7	67.3	4633.0

It is clear that many more pains ended up above the threshold in the last three rows in Table 5-4 than in the baseline scenario (row one), increasing the time needed for exhaustive OML. Figure 5-18 provides a pair of histograms showing the frequency for each number of pains above threshold (in the base line scenario with 5 primitive pains and 8 possible abstract pains) for the standard ML and Linear OML runs, respectively. From these charts there is a clear indication that the OML algorithm does a better job managing the pains as it is able to keep more pains below threshold. While the peak of the number of pains above thresholds for ML is 5, OML has no more than 5 pains above threshold and peaks at 3 pains above threshold. This shows that the tradeoff for computation time increase between ML and Linear OML is justified. While additional performance may be gained by using Quadratic or Exhaustive OML the increased

computation time required would tend to offset the benefits, particularly as the environment complexity increases, as shown in Table 5-4.

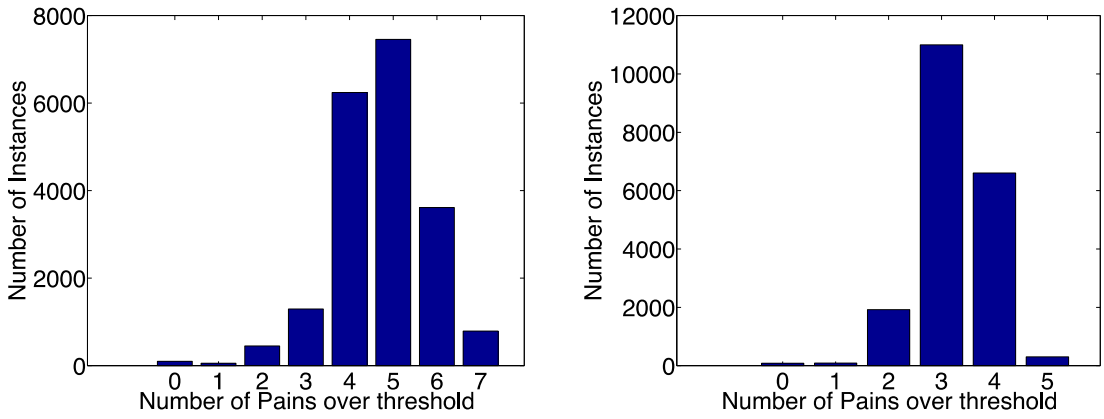


Figure 5-18. Frequency of pains above threshold. (ML-left, Lin. OML-right)

In the simulations the agent does not automatically know how long the execution of each type of action will take before trying it out, so the agent has to estimate. The agent assumes constant values to estimate the unfamiliar task's working time (shown as the estimated motor effort in Table 5-5). As it performs each task, the agent learns the real working times replacing the initial estimates. The real working times in this simulation follow a rough Gaussian distribution with mean and standard deviation equal to 3 and 1, respectively. Table 5-5 chronicles how the effort time values estimated by the agent affect the performance of the algorithm by showing their respective average pains levels at the end of simulation runs.

Table 5-5. Total Average Pains at Run Completion for Varying Estimated Motor Efforts

Estimated Motor Effort	Final Average Pain				Delta Lin-Ex.
	ML	Lin. OML	Quad. OML	Ex. OML	
1	6.8147	1.7688	4.4014	1.5250	0.2438
2	3.5543	1.8897	2.0354	1.8780	0.0117
3	3.5474	1.9224	2.0927	1.9103	0.0121
4	3.6028	1.9373	2.0935	1.9250	0.0123
5	3.6330	1.9039	2.0387	1.8860	0.0179
6	3.6231	1.8876	2.0199	1.8517	0.0359
10	3.6286	1.9112	2.0261	1.8819	0.0293
15	3.6536	1.9285	2.0699	1.9057	0.0228
20	3.6441	1.9152	2.0939	1.9125	0.0027

From Table 5-5 it can be seen that the best results generally occur when the agent matches or underestimates the working time (except with an estimated effort of one). This has the effect of biasing the agent toward more exploration. However, we can see that underestimating the necessary motor effort too much can have a deleterious effect, as seen when we chose a value of 1 in Table 5-5. Because the value for unexplored actions is so low, the ML (and Quadratic OML) agent was overly biased toward the exploration and tended to engage in such behavior even if it would have been more advantageous to resolve a known pain.

So far, only the clear advantages of implementing OML have been examined, however, are there any disadvantages? The most obvious one to consider is the increase in computational overhead associated with the OML algorithms. However, when observing the significant average pain reduction between ML and Linear OML, the small increase in computational time is of negligible importance. Another possible disadvantage of OML is its lack of analysis of past events or estimation of future pains in order to better explore its opportunities. However, it is anticipated that the MLECOG model, once it has a semantic memory implementation, will be capable of this type of evaluation. In any event, it has been demonstrated in this section that OML gives improved performance compared to standard ML in both the Matlab and NeoAxis environments.

5.4.3 OML with NACs

This section discusses implementation of opportunistic motivated learning in NeoAxis when NACs are present in the environment. These implementation is more complex, because, as previously discussed, adding NACs to the simulation requires adjustments to the agent to handle and react to NAC actions, as well as modifications to the environment to include the NACs and support their actions. While an agent selects the resource it wants to act on, the inclusion of NACS complicates things further. As mentioned in Section 4.4.1, if a NAC “attacks” an agent, the agent should respond to the attack. The correct response depends on the agent’s ability to observe the NAC's action and prevent it from harming the agent. For this reason, the agent focuses on the most dangerous NACs or situations in order to avoid the expected increase in pain. Also for that reason, detecting an action by a NAC is critical. When the agent detects an undesired action, it activates a potential pain bias (23) based on the previous pain it suffered when

the NAC was acting on the agent in a similar way. (Note these results with NACs were originally reported as part of [98] with the associated environment scenario initially presented in [110].)

In order to test the NAC implementation a more complex environment was implemented within NeoAxis. In this environment, a number of resources that the agent could use were created (as presented in Table 5-6), and the agent was given the ability to act on these resources. The desired agent's motor actions are listed in column 1 of Table 5-6, and its actions are driven by pains listed in column 3. Only the pains listed in Table 5-6 as primitive pains are predefined. They are: Hunger, Thirst, and Sweet Tooth, all of which increase over time, Bee Sting which results from the action of the Bee NAC, and Curiosity. The curiosity pain causes the agent to explore the environment when no other pain is detected and it lasts until all useful actions are learned. The "World Rules", which describe what the results of the agent's actions are, are listed in Table 5-6 under Outcome).

Non-agent characters like a Bug and Bees were introduced to the scenario to illustrate the NAC action pains. The NACs interact with the agent or with the resources. Bees produce Honey from Flowers, the Bug eats Food, and Bees can also sting the agent. We wanted to create characters that do useful or harmful actions for the agent. The Bug only engages in harmful actions by steeling food, which causes the agent's 'Lack of Food' pain. Bees do both useful and harmful actions. They produce honey and they sting the agent. This example shows the ability of the model to easily accommodate characters, which can perform both useful and harmful actions.

A graph representation of the scenario can be observed from Figure 5-19. All resources in the figure are represented by ovals and actions by rectangles. Acting on a resource that inhibits a pain is indicated by inhibitory links (with a solid black circle). An excitatory link (with an arrow) triggers a NAC action pain or a NAC appearance. In Figure 5-19, only simplified relations are shown to avoid clutter. Each resource symbol can be interpreted as an inhibitory interaction on the amount of resource and the lack of resource pain as shown in Figure 5-20. Similarly, a related action on the resource causes a small inhibitory feedback link to the resource being utilized as shown on Figure 5-20b from the action 'Plant Flowers' to the resource 'Flowers'.

As illustrated in Figure 5-20, the top inhibitory link inhibits the abstract pain 'Lack of Flowers'. When the 'Lack of Flowers' is inhibited the Flowers resource is automatically activated. A resource is restored through proper action by the agent, in this case by buying flowers. As the resource is used up by frequent action (planting flowers) the inhibition from Flowers to the Lack of Flowers is weaker and the abstract pain 'Lack of Flowers' increases. In Figure 5-20, the resource nodes and lack of resource nodes are automatically activated unless inhibited. The

forward arrow links are excitatory, so the Flowers resource activates the non-agent character (bees).

Table 5-6. List of Resources, Useful Resource-Motor Pairs and Their Outcomes

Motor action	Resource name	Agent's pains	Outcome		
			Increase	Decrease	Pain reduced
Eat food from	Bowl	Lack of Bowls		Bowls	Hunger
Drink water from	Cup	Lack of Cups		Cups	Thirst
Eat honey from	Honeycomb	Lack of Honeycombs		Honeycombs	Sweet tooth
Smoke	Cigar	Lack of Cigars		Cigars	Bee sting
Take food from	Fridge	Lack of Fridges	Bowls	Fridges	Lack of Bowls
Pour water from	Bucket	Lack of Buckets	Cups	Buckets	Lack of Cups
Plant	Flowers	Lack of Flowers	Honeycombs	Flowers	Lack of Honeycombs
Buy food with	Money	Lack of Money	Fridges	Money	Lack of Fridges
Pull water from	Well	-	Buckets	-	Lack of Buckets
Buy flowers with	Money	Lack of Money	Flowers	Money	Lack of Flowers
Buy cigars with	Money	Lack of Money	Cigars	Money	Lack of Cigars
Work for money with	Tools	Lack of Tools	Money	Tools	Lack of Money
Study for job with	Book	Lack of Books	Tools	Books	Lack of Tools
Play for joy with	Beach ball		Books		Lack of Books
Kick	Bug	-		Likelihood	Bug eating food
		Hunger - primitive pain			
		Thirst - primitive pain			
		Sweet tooth - primitive pain			
		Bee sting - primitive pain			
Any		Curiosity - primitive pain			Curiosity

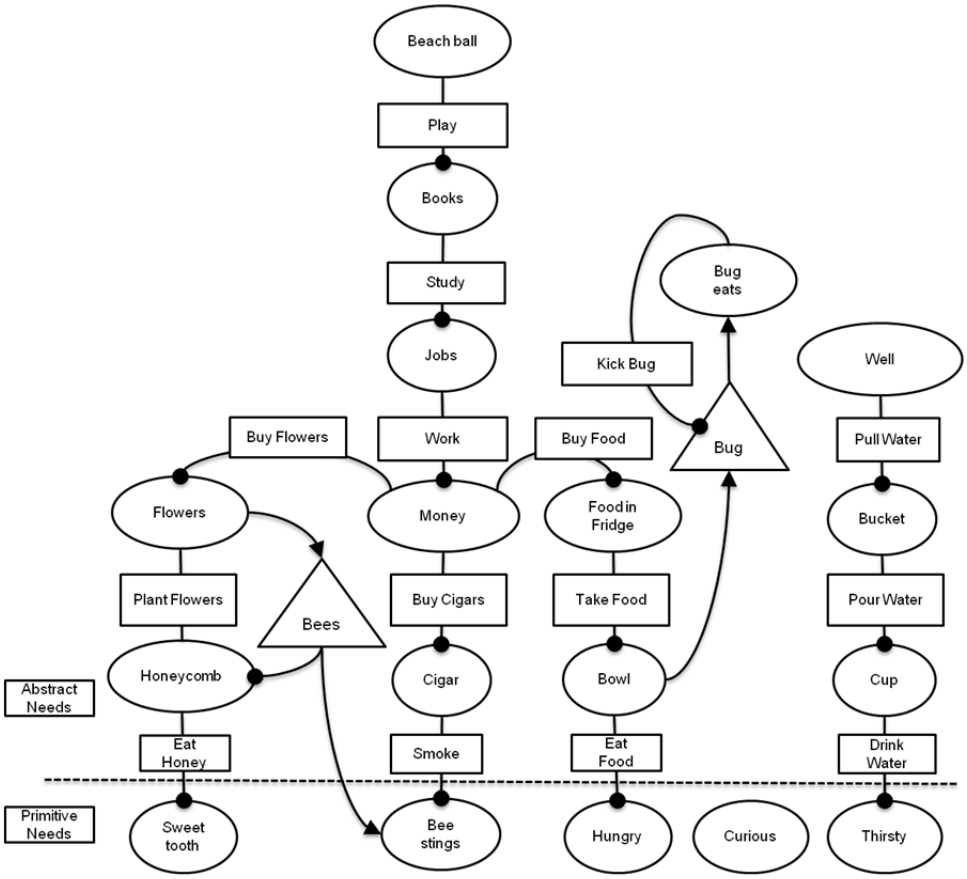


Figure 5-19. Scenario diagram for NAC OML testing [SGP14].

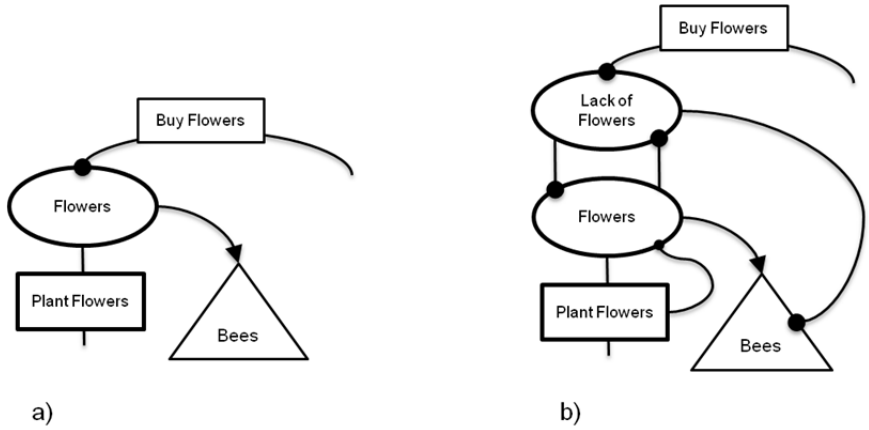


Figure 5-20**.** a) Simplified relations from Figure 5-19 between the resource (flowers) and the action on the resource (plant flowers), b) detailed relations

The agent's actions result in various outcomes like increasing and decreasing resources quantities, and reducing abstract pains. To visualize the resource quantities, the current task, pains

levels and the agent's memory, simulation windows were added to display the ML agent and the current state of the environment, as shown in Figure 5-21. When a pain level is above threshold it is displayed in red. In this screenshot the agent action is driven by NAC action pain.

The agent tries to learn useful actions and their outcomes. Sometimes the agent performs a nonsense action like "Play for joy with hammer", but even such “useless” actions are used to learn. The memory window on the left part of Figure 5-21 displays the memory state of various sensory-motor pairs. When the color is gray, then it means that the agent didn't learn the value of the corresponding sensory-motor pair. When the color is white then it means that action is useful for the agent, and if color is black then the action is known to be useless.

Figure 5-21. Main simulation view with displayed simulation state in windows.

The environment parameters must be appropriately set to give the ML agent a chance to learn the “correct” behavior (e.g. learn to survive in the environment before it “dies”). When resources are sparse, the agent doesn’t learn all useful actions because it runs out of resources to test new actions. On the other hand, when resources, are plentiful, the agent does not bother to learn anything new about the environment besides using the resources that it needs to satisfy its pains. Also, when action times are too long the agent is unable to satisfy all of its pains. When appropriate simulation parameters are selected (e.g. the environment isn’t too “hostile”), the ML

agent proves able to learn and survive even in a complex and changing environment. Multiple simulations have been run where resource quantities and motor action times were modified. There was even an attempt to insert a human controlled character to disturb the ML agent by getting in its way or moving resources to different locations. The attempt had minimal impact, however, we were able to get the agent to do something else by delaying it from reaching its destination long enough for a primitive pain to take precedence and cause the agent to change its action. In all these simulations the ML agent learned to manage the changes in the environment and minimize its average pain.

Figure 5-22 illustrates changes in various pain levels, with particular emphasis on the pain triggered by a NAC action labeled by the solid line as "NAC Action" pain. This pain is related to the consumption of food by the NAC. Food is directly related to the primitive hunger pain (not shown in this figure), so any pain related to food reduction is discovered quickly. Initially, the pain remains below threshold. However, as the NAC consumes food, it causes a buildup of bias against the NAC action (23) and related pain. Eventually the pain passes threshold, and the agent learns to scare the NAC away. Figure 5-22 illustrates various pains including NAC action pain. This shows that the agent correctly interrupts an undesirable NAC behavior, and reduces the pain associated with that behavior by forcing the NAC away.

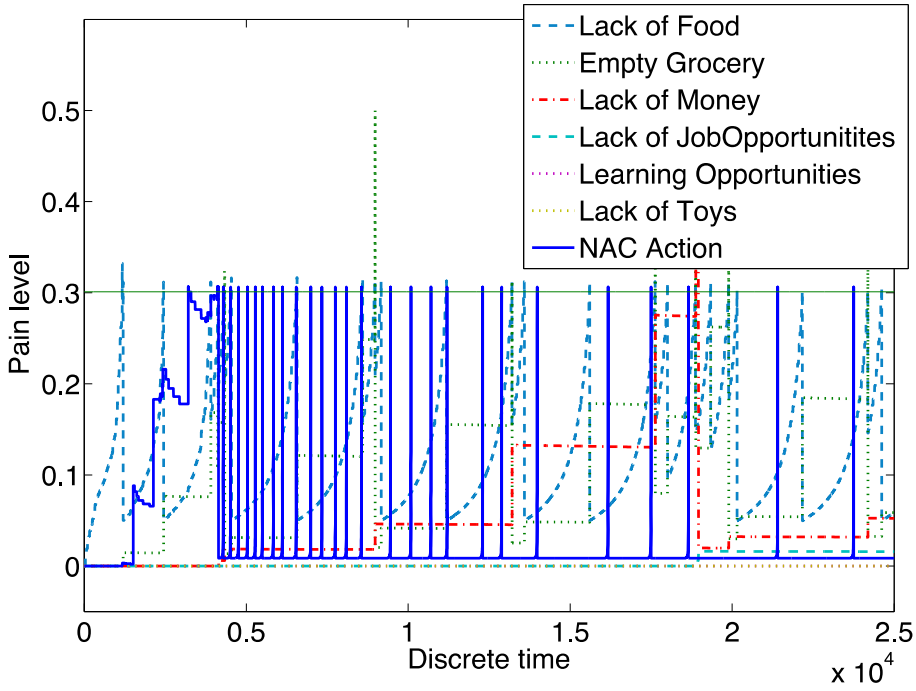


Figure 5-22. NAC pain behavior.

5.5 Conclusion

This chapter has presented the result gathered from testing the ML algorithm. Results were provided for both original and opportunistic versions of the algorithm, as well as Matlab and NeoAxis based simulations. However, so far results dealing with alternative approaches have not been discussed. Motivated Learning has many similarities to reinforcement learning based approaches, hence, the next chapter presents several comparisons between the ML algorithm and a selection of reinforcement learning based algorithms

CHAPTER 6: COMPARISON BETWEEN ML AND RL

This chapter compares the ML algorithm against other comparable algorithms based primarily around Reinforcement Learning approaches. For some review, see section 2.1 of Chapter 2, which presents a brief discussion of the some of the main differences between Motivated Learning and Reinforcement Learning. Section 6.1 and its associated subsections compare results from the initial probability based ML algorithm against a Python based TD-Falcon implementation. Following that, Section 6.2 provides a less detailed but more extensive comparison against several reinforcement learning algorithms using a "black box" environment approach that was designed to support such comparisons.

6.1 Comparing Early ML against TD-Falcon

In this section experimental results using the motivated learning approach from Chapter 3 in an environment with hierarchical dependencies and varying levels of complexity are shown and compared to RL (as a "control"). These results were originally presented in [111] and an earlier journal publication [20].

Several computational experiments to compare the effectiveness of the proposed motivated learning and reinforcement learning methods in a virtual environment were conducted. The RL algorithm was implemented through TD-Falcon [81]. TD-Falcon (Temporal Difference – Fusion Architecture for Learning, Cognition, and Navigation) or TDF is a generalization of Adaptive Resonance Theory – a class of self-organizing neural networks – that incorporates temporal difference methods (TD) for real time RL. This algorithm learns the value functions of the state-action space using temporal difference methods, and then uses them to determine the optimal action selection policy. Finally, it enables an autonomous agent to adapt and act in a dynamic environment with both immediate and delayed reinforcement signals. This algorithm was chosen because of its superior performance (in terms of learning efficiency measured by the number of trials) in comparison to other implementations of RL.

The experimental setup consists of two main components: the environment and the agent. The agent is located in a hostile environment. The term "hostile environment" means that the amount of available resources is limited, but can be renewed by the learning agent through a deliberate action. Three experiments that tested various aspects of learning in these environments are described below.

6.1.1 Setting the Environment

The environments used in the experiments in this section are based on the description given in Section 5.3 with respect to Table 5-1. The environment specified in the table has a linear hierarchy where realization of a higher order goal is a single solution to satisfy a lower order need. For the purpose of the experiments in section 6.1, environments with several different levels of hierarchy are selected, making it increasingly more difficult for both ML and RL agents to survive. In this set of tests, limited resources in the environment are represented by a gradual decline in probability that a specific resource will be available.

For instance, if the agent spends money to buy food, the money supply (probability of getting money for use by the agent) goes down, regardless of whether the machine uses all the food or not. This simulates a case when food may rot if it is not consumed. Thus, a wise use of resources is needed and the machine needs to learn this.

6.1.2 Comparison with RL Experiments (TD-Falcon)

Table 5-1, from the previous chapter, defines the initial environment used in the scenario presented here, and indicates a basic, linearly hierarchical, arrangement of sensor-motor / pain interactions. A single primitive pain (low sugar level) and a simple linear hierarchy of abstract goals are used. As described in Section 3.3 resources in the test are represented by the probability of finding the resource in the environment.

To review the efficiency of operation of the agent, Figure 6-1 shows how the probability based ML agent handles the various pains. During the course of operation, the system will learn and adjust itself until it reaches equilibrium, which occurs when the pain peaks stabilize and the average pain levels out. In Figure 6-1, only the first hundred iterations are shown with the curiosity pain on a constant level just above threshold, however, on the longer time scale, a point would be observed where the system decides to stop exploring its environment via “curiosity”. At this point, its behavior becomes more regular due to the absence of semi-random resource consumption resulting from curiosity based exploration.

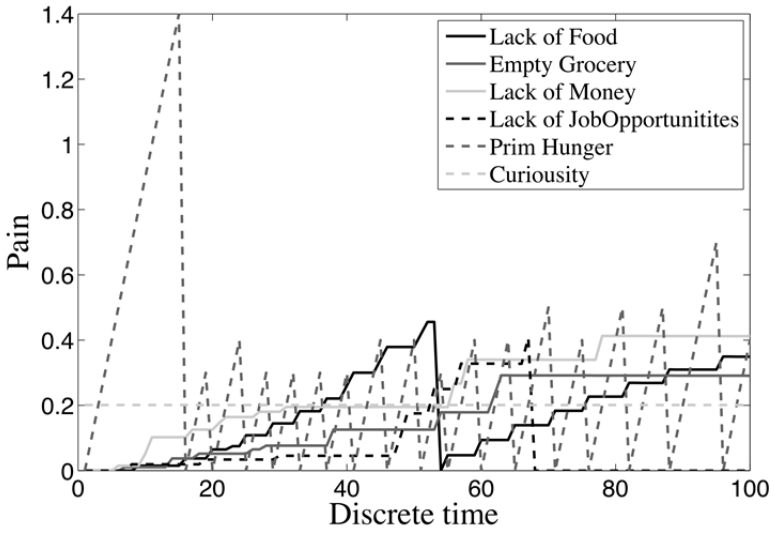


Figure 6-1. Pain signal values in the first 100 iterations.

The function which describes the probability of finding resources in the base experiment setup is as follows:

$$f_{ci}(k_{ci}) = \frac{1}{1+\frac{k_c}{\tau_c}} \tag{79}$$

where:

τ_c – scaling factor that describes a resource declining rate

k_c – number of times a resource was used

Results of simulated actions in such simple environment are shown in Figure 6-2. The moving average value of the primitive pain signal P_p is shown in Figure 6-2 *a*) for TD-Falcon (TDF - the solid line) and motivated learning (GC - the dashed line). The first observation is that ML yields much lower average pain than TDF and stabilizes sooner than TDF. Figure 6-2 *b*) shows ratio of the average pains for TD-Falcon and motivated learning (TDF/GC(ML) P_p ratio).

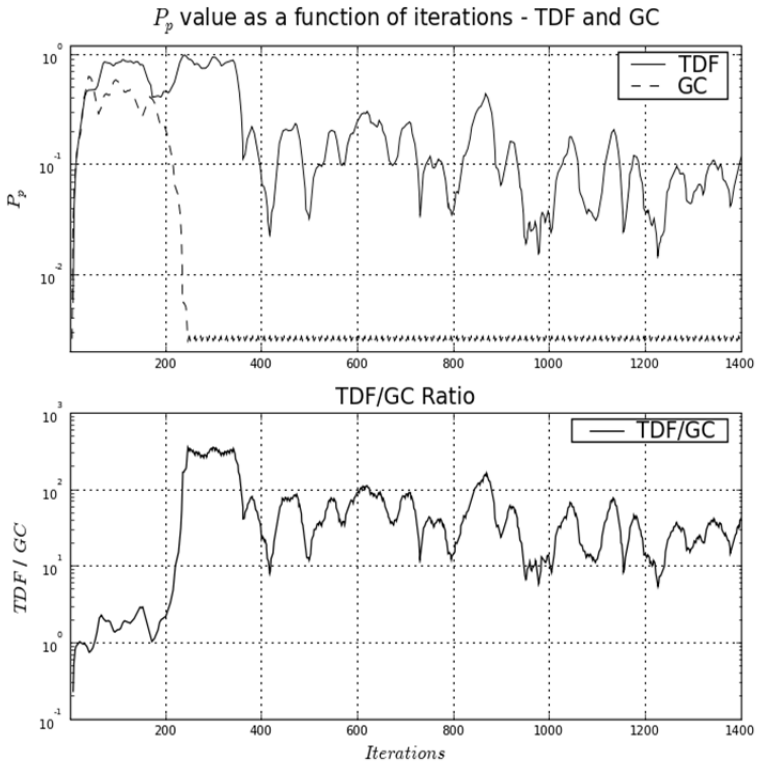


Figure 6-2. a) Moving average of P_P value as a function of number of iterations during computational experiment, b) TDF/GC(ML) P_P ratio.

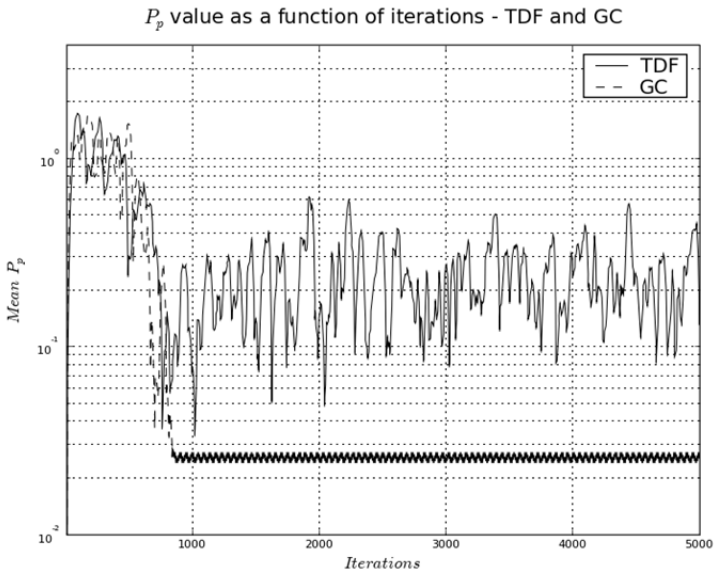
The ML based agent was able to learn how to use resources in order to minimize its pain in about half the number of iterations than the agent based on TDF could. Agents based on ML yielded from 10 to 100 times smaller average internal pain than TDF. This means that agents using the motivated learning method based on the goal creation system were able to control their environment better than those using TDF.

6.1.3 Complex Mild Environment

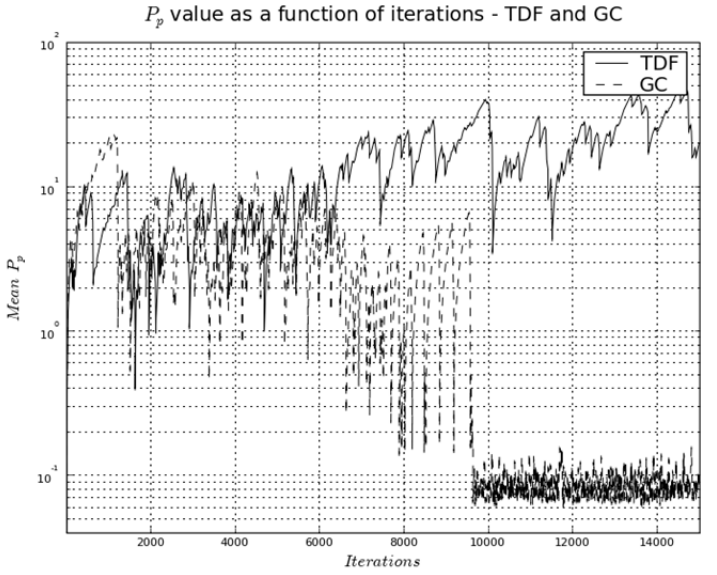
In the second experiment, instead of using an environment with only five hierarchy levels (each of which represents different resources) several environments with higher levels of hierarchy were prepared. The results obtained are illustrated in Figure 6-3 *a)* and *b)* for 8 and 18 levels of hierarchy respectively, and show the average primitive pain levels in both methods.

From these experiments it can be concluded that in some cases an agent using a TDF algorithm can control its environment quite efficiently in the early stages of simulation (it

behaves in a similar way to the ML agent). However, in later stages TDF is usually less effective at controlling its "internal pain". The reason for this initial success of TDF is that this environment was not "hostile" enough. It means that even after extensive use of resources there were still enough resources left in the environment and the TDF agent could find them through random actions. However, as the resources were depleted, an agent needs a systematic approach to restoring them. This requires introduction of higher level goals, which TDF agents don't do.



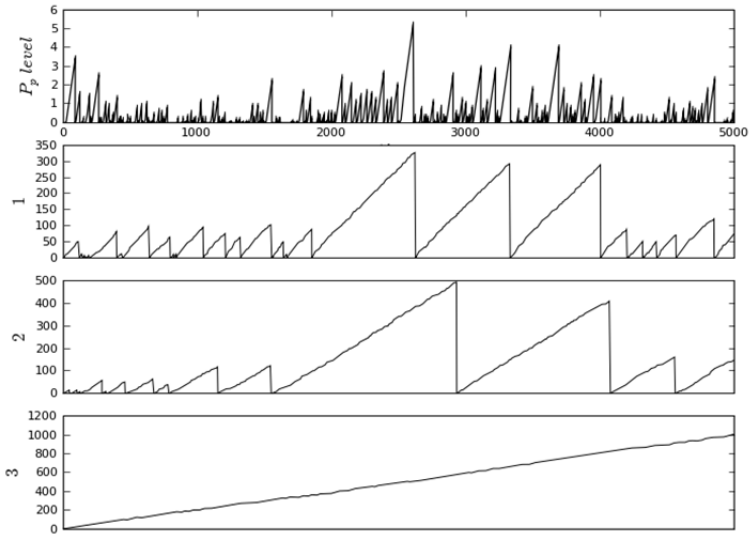
a)



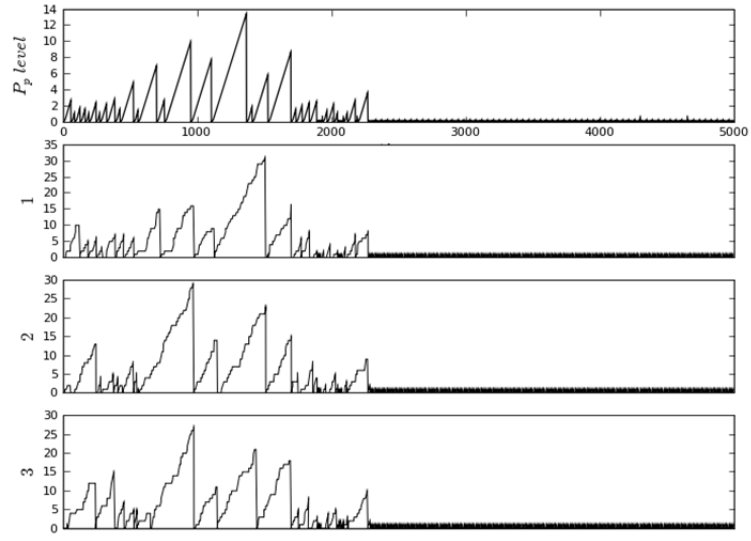
b)

Figure 6-3. Moving average of P_P value as a function of number of iterations a) 8-levels of hierarchy, b) 18 - levels of hierarchy.

More information about the efforts of both agents can be acquired by observing their use of all types of resources available in the environment. Figure 6-4 shows changes in the primitive pain signals and resource utilization on three levels of abstract hierarchy using both methods. Good resource management requires resource restoration, thus the higher the resource utilization, the more difficult is to find it in the environment and the more difficult to lower the primitive pain.



a)



b)

Figure 6-4. The difference in ability of using more abstract resources in both cases a) TDF and b) ML. TDF is not able to replenish resources on the 3 levels.

As can be observed, the ML agent is able to manage all of the needed resources, while the TDF agent learned to manage resources only at the two lowest levels (levels 1 and 2 in Figure 6-4). It uses resources from higher levels without learning how to restore them. This can be observed in the higher pain levels in Figure 6-4 *a*), versus those in *b*). By 2500 iterations, the ML agent learned how to manage all the resources, causing its primitive pain level to be low.

The only reason that the primitive pain of the TDF agent was still modest is that the probability of finding the needed resource in this environment was still relatively high. The TDF agent had no reason to learn (and explore) its environment because it was able to survive there without more significant learning effort. It had no motivation to improve its "skill level".

To expose this weakness of TDF we designed another experiment (see Section 6.1.4) for both the TDF and the ML agents in which the environment was not only complex and dynamic but also very hostile.

6.1.4 Harsh Environment

Higher hostility of the environment was achieved by changing the function that describes the probability of finding resources to the following:

$$f_{ci}(k_{ci}) = e^{\frac{k_c}{\tau_c}} \tag{80}$$

where:

τ_c – scaling factor that describes a resource declining rate

k_c – number of times a resource was used

After simulations in this more hostile environment we observed that the agent based on the RL algorithm (TDF) was not able to learn the higher dependencies between different available resources. After about 1100 iterations the TDF agent had exhausted all the base resources and was not able to replenish them. After that time its "internal pain" started to grow almost linearly (Solid line in Figure 6-5). We can also observe that the motivated learning agent (GC) (Dashed line in Figure 6-5) was still able to learn all the dependencies between the environment's resources and use this knowledge to control its internal and external pains.

It was observed that between 190 and 350 iterations the pain signal of the TDF agent was much lower than that of the GC system. However, during this time, the ML (GC) system continued to learn the complex environment neglecting its primitive pain since its abstract pains dominated. At the same time, the TDF agent used up all available resources trying to minimize its

primitive pain, while the environment conditions were worsening. This example indicated a clear failure of the TDF agent to learn behavior appropriate for this harsh environment.

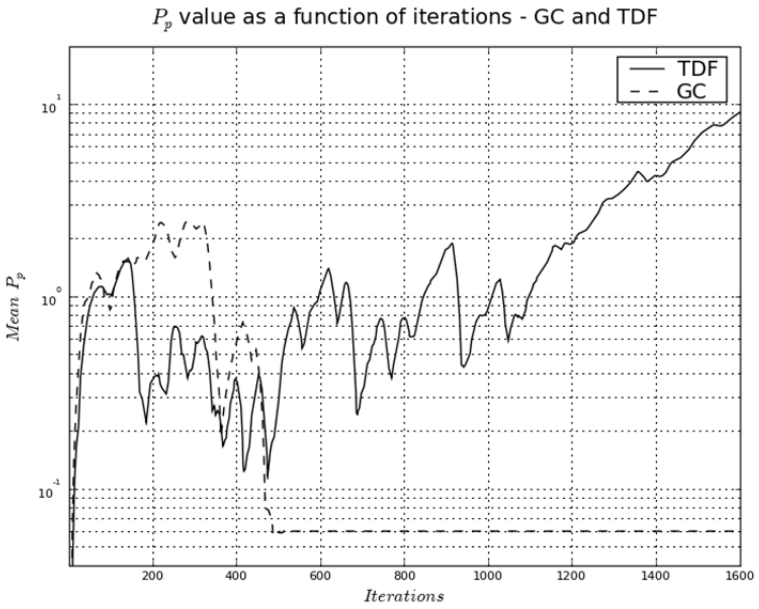


Figure 6-5. Results of experiments in a more hostile environment.

6.1.5 Summary of TDF Comparison Experiments

In these experiments some disadvantages of the reinforcement learning method compared to motivated learning were demonstrated. ML outperformed TDF in the learning task quickly converging to a stable solution, while TDF, after initial success, was unable to accommodate changes in the environment and converge to a stable solution. This was particularly obvious in the more hostile environment.

Motivated learning can be combined with reinforcement learning to search for a solution to a well-established goal (or a subgoal). The motivated learning mechanism will not only create internal abstract goals but it will also manage them. It will switch between these created goals (using internal motivations) when needed for the optimum performance. The ML agent's internal reward system will provide the "RL component" with reward information to learn an appropriate action.

We believe that this kind of hybrid system will be able to take advantage of sophisticated methods developed for RL in order to efficiently solve problems where the environment is complex and has complex relations between its factors.

Reduction of resources was used as an easy to understand and implement example of changes in the environment. The agent is also exploring its environment as it learns. If the environment changes in other significant ways (e.g. new technologies can be used to its advantage or an old one became obsolete) the agent will simply ignore older actions as "impossible" or less competitive and attempt to learn new solutions. Pain-action associations change by changing weights between them, thus new associations might be introduced and old ones may become less important.

An agent uses the goal creation approach to learn what to do and to adjust to changing environment conditions. It does so by adjusting pain biases and weights between the pain signals and actions. One of the major drawbacks of current ML systems is that significant amount of processing power and time are required to evaluate the state of a complex environment, particularly when agent needs to learn real objects and learn to perform complex actions that were not predefined for the agent. These processing requirements make the basic parallelized ML agent unsuitable for the mentioned complex situations. The MLECOG model covered in Chapter 7 and [24] addresses this issue and several others, for example, planning capabilities, complex memory, multi-step actions, etc.

6.2 Black Box Comparison against Other RL Models

TD Falcon was the first RL algorithm we compared with our ML approach, but reviewers of our work were curious if ML shows similar gains over other RL methods. Several RL agents operating in the same environment using "black box" approach were tested and the results are presented in this section. (This material was previously presented in part in [24] and in full in [112].) Since RL agents optimize the reward received, we used this as a measure of performance. The rewards received by various agents were computed and normalized to the maximum reward that could be obtained in the environment at any given time.

6.2.1 Compared Algorithms

Using the black box approach we compared our Motivated Learning algorithm with several reinforcement learning algorithms, including classical methods like Q-learning, SARSA(λ), and hierarchical reinforcement learning but also more advanced algorithms like NFQ, Explauto, and TD-FALCON. For fair comparison, we set the reinforcement learning algorithms in the same environment and with the same parameters and testing conditions. Please note that the parameters of the tested algorithms were adjusted to make a fair comparison. None of the

algorithms have any pre-encoded knowledge of the environment, and each of them receives the same information. This includes the ML algorithm, which was given no hidden parameters or information about the environment. The reinforcement learning algorithms are detailed as follows:

Q-learning: Q-learning is one of the traditional reinforcement learning algorithms and can be used to acquire optimal control strategies from delayed rewards. The agent usually has no prior knowledge of the effect of its actions on the environment (e.g. it has no model to work with). There are several variations of Q-Learning, such as Delayed Q-learning, Greedy GQ, etc. Here, we implement the Q-learning algorithm from [113] with the discount factor $\gamma = 0.9$ and learning rate $\alpha = 0.7$.

SARSA(λ): The SARSA algorithm considers the transitions from state-action pair to state-action pair, and is a standard example of a temporal difference (TD) learning algorithm. SARSA(λ) combines the Monte Carlo and dynamic programming ideas [79]. It updates reward estimates based on the other learned estimates, without waiting for a final outcome. It differs from the original SARSA with the inclusion of the trace decay (λ) parameter, which affects the distribution of reward. (Larger λ leads to a larger proportion of the reward credit being given to more distant states when there are multiple cycles between rewards.) The discount factor, learning rate, and trace decay parameters are set to $\gamma = 0.9$, $\alpha = 0.4$, and $\lambda = 0.9$.

Hierarchical RL: We used a variant of the MAXQ algorithm from [114] adopted to implement the hierarchical reinforcement learning (HRL) algorithm, where the goals are split into subgoals and subtasks. The size of the state space is reduced and learning efficiency is improved. In HRL, the agent constructs a set of policies that need to be considered during reinforcement learning. The MAXQ value function is assumed to represent the value function of any given hierarchy. The target Markov decision process (MDP) is decomposed into smaller MDPs The parameters for the algorithm were set as $\gamma = 0.9$, and $\alpha = 0.4$.

Dyna-Q+: The Dyna-Q+ algorithm includes direct reinforcement learning, model learning and planning [79]. It is believed that Dyna-Q+ can deal with changing environments better than other RL algorithms. It generates an action based on direct reinforcement learning and then updates both a Q table as well as its model. During the planning process, the Q-planning algorithm randomly chooses samples from state-action pairs that have already been visited. In this case, the model will never be queried with a pair about which it has no information. The parameters are set the same as those in Q-learning, with discount factor $\gamma = 0.9$ and learning rate $\alpha = 0.7$.

Explauto: Explauto [115] is a framework for the implementation for active and online sensorimotor learning algorithms. Explauto cognitive architecture is composed of a the sensorimotor model, which iteratively learns forward and inverse models from experience, and an interest model which makes the choices about where to explore in the environment. In our tests with Explauto we used its built-in DiscreteProgress interest model configured to match our environment with other parameters left at their default values.

TD-FALCON: This algorithm [116] incorporates a temporal difference algorithm with a family of self-organizing neural networks and has some similarities to SARSA. Based on the Adaptive Resonance Theory and using evaluative feedback from the environment, TD-FALCON works by learning the value functions of the state action space estimated using Q-Learning. The learned value functions are then used to determine the effective actions.

NFQ RL: NFQ (neural fitted Q iteration) is a batch RL learning FQI (fitted Q iteration) method [117]. FQ implements a dynamic scaling heuristic that can be seamlessly integrated into neural batch RL algorithms, which use a fixed set of a priori-known transition samples, e.g. offline learning. Fitted Q-iteration can be viewed as approximate value iteration applied to action-value functions.

6.2.2 Black Box Scenario

To compare the aforementioned algorithms, we designed a "black box" environment that would present state and reward information to the RL or ML algorithm and receive a response in the form of an action to take. This action would then be presented to the environment, which would adjust itself accordingly and respond with a reward value ranging from 0-1 depending on the action and the current state of the environment. We show the results of the comparison of these algorithms in the next section.

The environment is an 8-level hierarchy of "resources" that depend on each other for restoration, similar in structure to the basic scenario presented in [20] and the preceding ML and RL tests in Section 5.5. In this scenario, there is a single "primitive" need, which can be resolved by the correct action, which consumes a specific resource. This specific resource gets depleted over time and needs yet another action/resource combination to restore, and so on up to the "top" level of the resource hierarchy, which is not depleted.

In the "black box" scenario each potential primitive reward R_p that can be received from the environment, increases gradually after each iteration until it reaches its maximum level R_{mp}.

$$R_p(i) = min\left(i_{0p} \cdot R_{rp}, R_{mp}\right) \tag{83}$$

where i is current iterative step (time elapsed), i_{0p} is the iterative step from the last time R_p was awarded to the agent, and R_{rp} is the rate of change of this primitive reward. After the reward is received, i_{0p} is set to 0. R_p is only awarded to the agent if it performs a beneficial action to reduce the primitive need (that is well defined for all the agents). If plotted, the potential reward function would look somewhat like a sawtooth plot with peaks varying in range from 0 to R_{mp}, with the exception that once the potential reward reached R_{mp} it would remain there until it dropped due to the reward being given.

In the experiment, $R_{mp} = 1$ and $R_{rp} = S_{Rate}/S_{inp}$, where S_{inp} is the initial value of the primitive reward generating resource, and S_{Rate} is its rate of decline. We varied S_{Rate} for the tested algorithms to see how they perform under varying levels of pressure. For instance, if $S_{inp} = 40$ and $S_{Rate} = 1$, it will take 40 iterations for the resource to deplete (and make the maximum reward available). Higher values of S_{Rate} will mean it will take less time for the resource to be depleted. However, since the resources are connected in a hierarchy, it will also mean that the agent will have to learn the next resource in the chain more quickly, hence the aforementioned increase in pressure.

The received rewards are averaged and normalized using

$$Ave\ R_{Norm}(i) = \frac{\frac{1}{i}\sum_{p=1}^{n}\sum_{k=1}^{i} R_p(k)}{\sum_{p=1}^{n} R_{rp}} \tag{84}$$

where *i* indicates the current time step, *n* is the number of primitive resources, and R_{rp} is the rate at which the rewarding resource *p* is depleted.

6.2.3 Reinforcement Learning Results

In the following tests, unless otherwise stated, each plot shows the results of 25 averaged simulations, each running for 10,000 iterations. As expected, for all the algorithms, a greater S_{Rate} yields lower performance, since the algorithm has less time to explore and is under more pressure to perform. This is observed most noticeably in the Q-Learning, SARSA and HRL algorithms.

In Figure 6-6, we observe that the ML algorithm performs well (and maintains near perfect performance until it hits an S_{Rate} of about 16, at which point its performance starts to fall off.

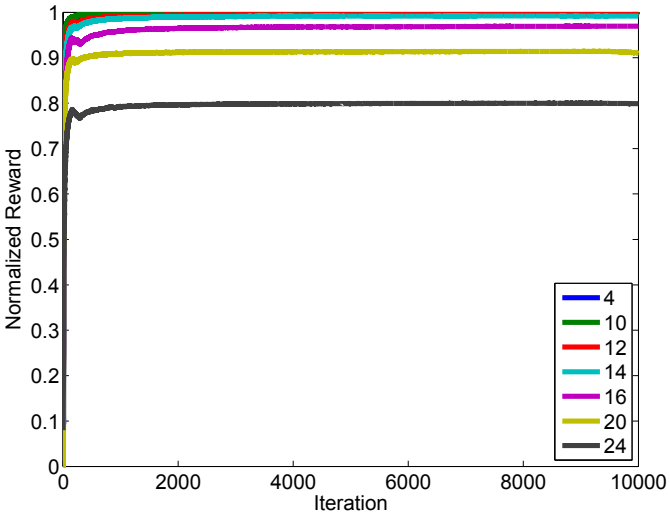


Figure 6-6. Combined results from the ML algorithm for different values of S_{Rate}.

This is because at this point it began to have difficulty maintaining the primitive resource, while also maintaining the other resources. It simply ran out of time to perform the required actions due to the high rate of decrease in the primitive resource and the associated higher demand on higher level resources, and its reward capability suffered as a result. However, ML is able to perform with higher S_{Rate} and greater overall reward than any of the other algorithms tested here.

Figure 6-7 to Figure 6-10, present the results from the Q-Learning, SARSA and HRL algorithms. Note that beyond the first few hundred iterations, the results for each algorithm are similar to each other. This is likely because all three algorithms share the same basic process and have difficulty performing properly once the known solution toward generating their reward is no longer available and they have to rely on random attempts.

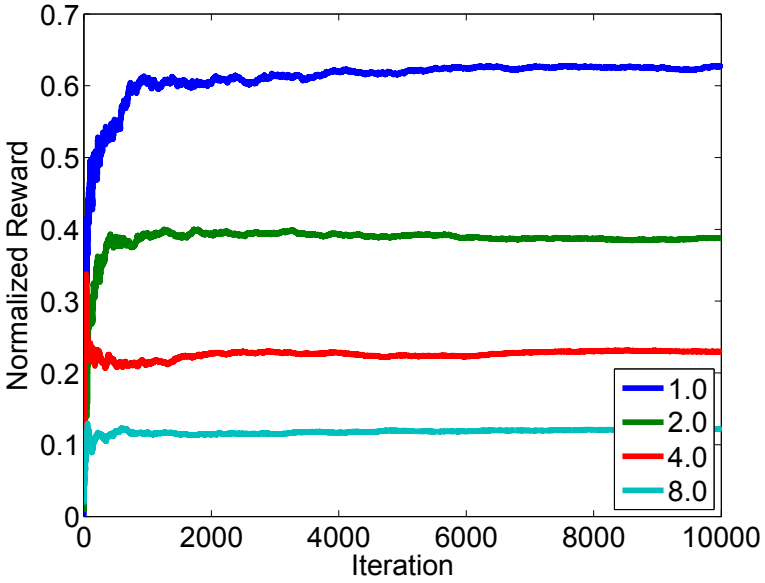


Figure 6-7. Combined results from the Q-Learning algorithm for different S_{Rate}.

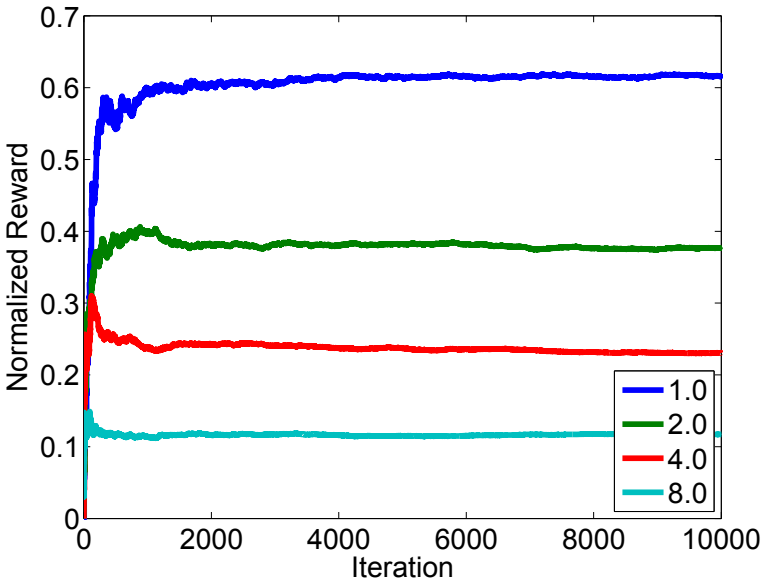


Figure 6-8. Combined results from the SARSA algorithm for different S_{Rate}.

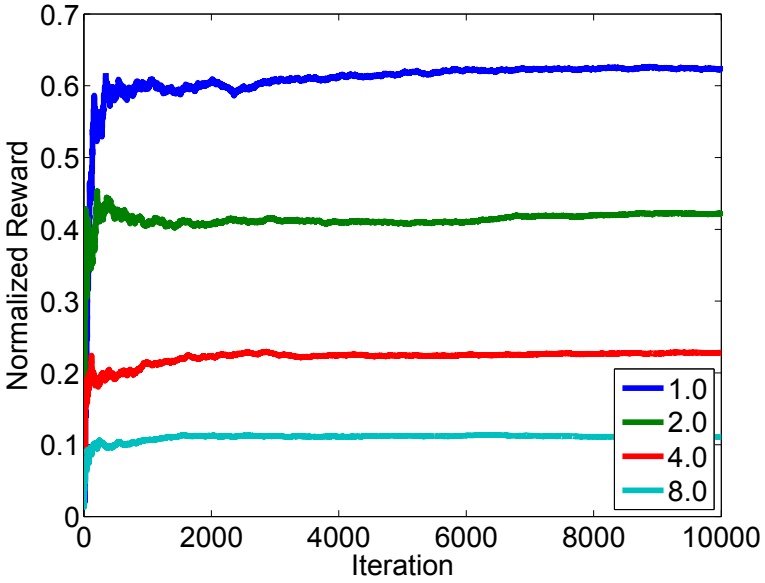


Figure 6-9. Combined results from the HRL algorithm for different values of S_{Rate}.

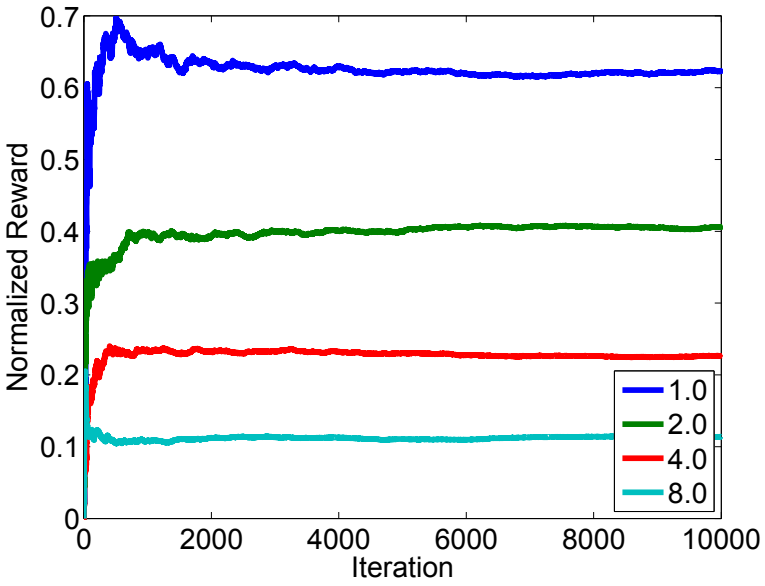


Figure 6-10. Combined results from the Dyna-Q+ algorithm for different values of S_{Rate}.

Figure 6-11 shows the results from the Explauto algorithm, which performs similarly (though slightly worse) to the Q-Learning, SARSA, HRL, and Dyna-Q+ algorithms. This is likely due to it being more oriented toward sensorimotor based RL learning rather than the type of RL hierarchy used in the Black box test.

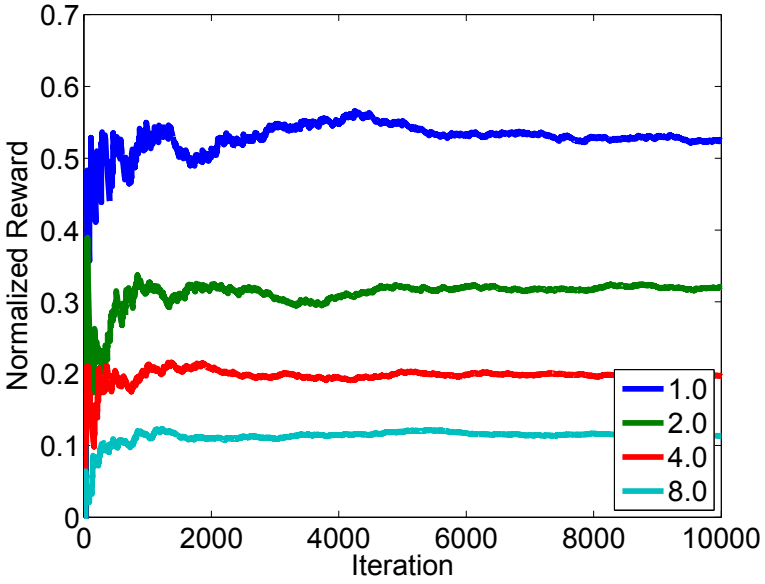


Figure 6-11. Combined results from the Explauto algorithm for different S_{Rate}.

One would think HRL would perform better since it can also derive and use subgoals. However, it is not as efficient as ML in creating abstract motivations and related goals, as demonstrated. The main difference is that in HRL the subgoals are created only when the system works on a goal that refers to the subgoal as needed for the completion of the main goal. In existing architectures either goals or motivations (or both) are given and the agent only implements these goals. In the AGI domain there are efforts to equip the agent with mechanisms for goal creation (e.g. [69]) but they are not implemented in embodied agents, or in the simple version of HRL tested in this work.

Figure 6-12 shows the results from an implementation of TD-FALCON in the chosen environment. TD-FALCON is actually one of the better performing algorithms in the tests. Like the preceding algorithms, its performance tends to decrease with the increase of S_{Rate}. Only the NFQ algorithm seems to exceed its performance, and only then at higher values of S_{Rate}.

Interestingly, the NFQ algorithm tends to perform better than (most of) the other RL methods when the S_{Rate} is increasing (see Figure 6-13). However, the algorithm's performance is not very consistent. The reason for this inconsistency is likely the oscillation observed in Figure 6-14 among the individual runs and the large confidence interval of ±0.123 as seen in Figure 6-15). The confidence intervals of Q-Learning, SARSA, HRL, Dyna-Q+, TD-FALCON,

Explauto and ML for $S_{Rate} = 8.0$ are 0.0034, 0.0035, 0.0034, 0.0031, 0.0326, 0.0026 and 0.0013, respectively.

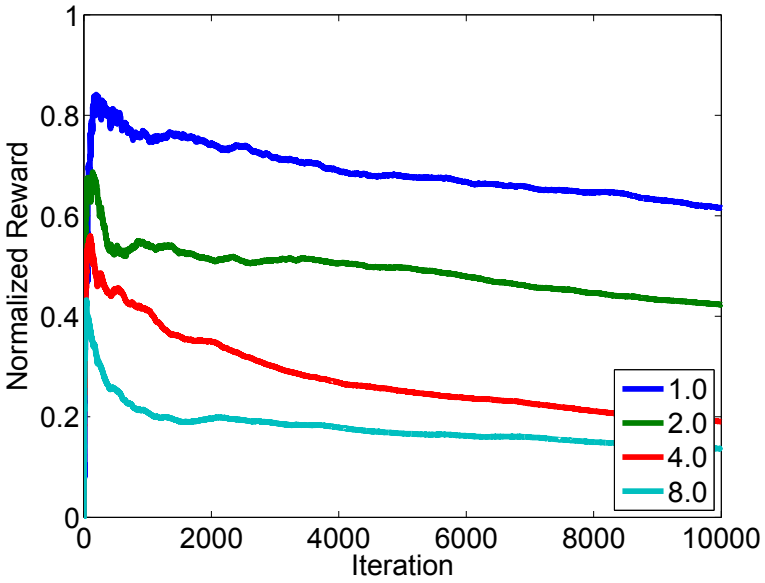


Figure 6-12. Combined results from the TD-FALCON algorithm for different values of S_{Rate}.

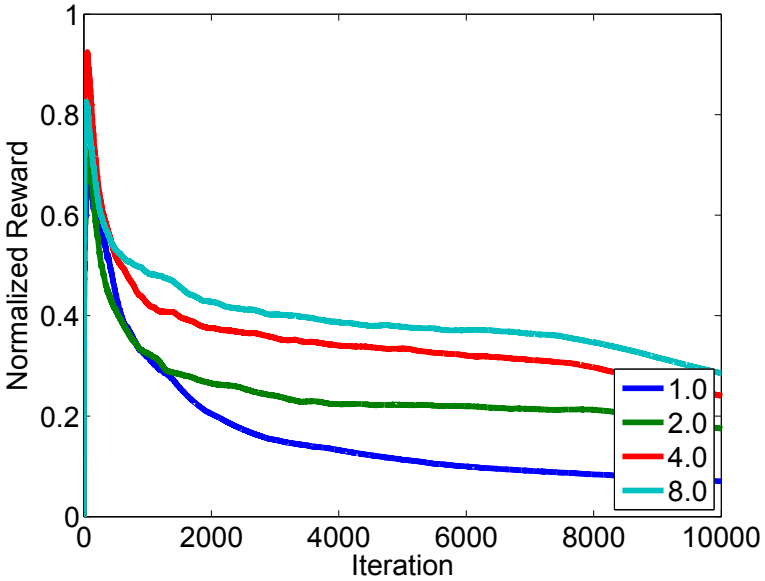


Figure 6-13. Combined results from the NFQ algorithm for different values of S_{Rate}.

Figure 6-14 gives an 'inside' view of the NFQ algorithm's results by showing performance results for all 25 runs. It can be observed that the algorithm appears to oscillate between good and bad performance (or two bands), while appearing to gradually improve overall.

For example, using the legend as a guide, run 10 is the highest of the lower performing band of runs, however, run 12 is the 2nd worst run of all of them. Run 16 has the highest average reward at the 10,000th iteration. So while time does seem to bring improvement to the performance, it is not very consistent.

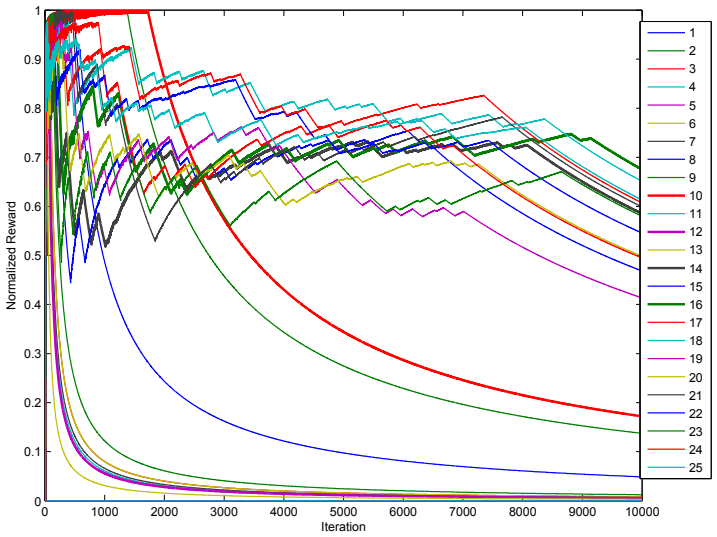


Figure 6-14. Individual results of the NFQ algorithm with $S_{Rate} = 8$.

Figure 6-15 shoes the averaged results and associated confidence interval of the NFQ algorithm at S_{Rate}=8. Note just how wide this interval is. It can also be observed that it doesn't quire represent the NFQ algorithm's operation in Figure 6-14, where results tended to be either acceptable (> 0.5) or poor (near zero).

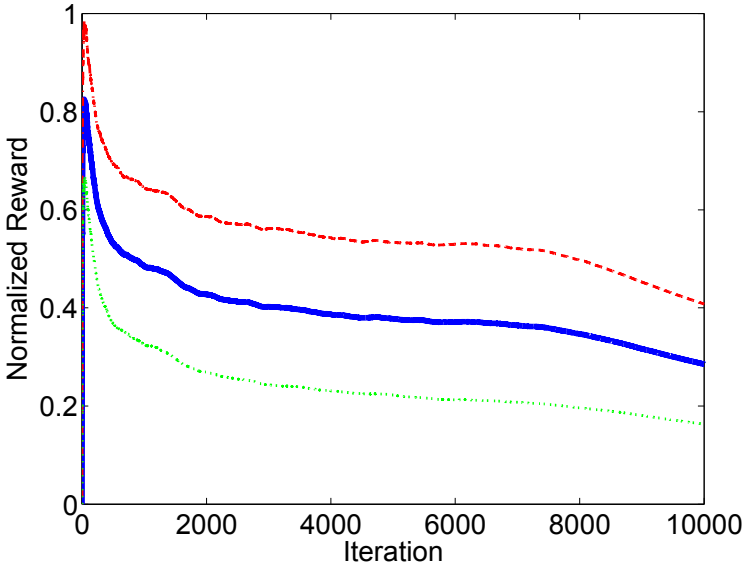


Figure 6-15. Confidence interval of averaged NFQ test with S_{Rate}=8.

In Figure 6-16 the Motivated Learning algorithm is directly compared against the reinforcement learning algorithms. TD-FALCON is better than Dyna-Q+, Dyna-Q+ performs slightly better than HRL, SARSA, or Q-learning, and Explauto with NFQ are the worst. However, ML outperforms all of them. ML appears better suited to operate in the hierarchically structured environment we provided due to the way it creates additional needs.

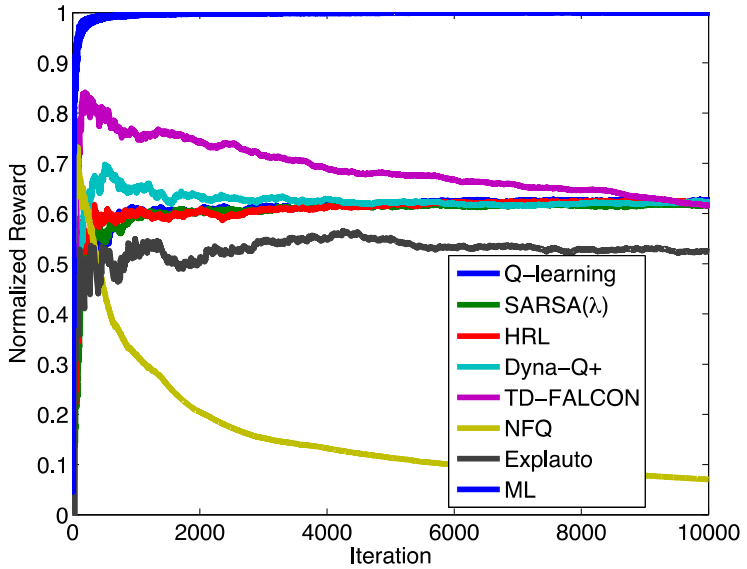


Figure 6-16. Comparison of reinforcement learning algorithms' average reward performance to motivated learning with $S_{Rate} = 1.0$.

Figure 6-17 to Figure 6-19 compare the algorithms using rate value S_{Rate} set at 2,4, and 8, respectively. It can be observed that as the rate of change of the primitive reward s_{Rate} increases, the advantage of TD-FALCON over other RL algorithms diminishes, while NFQ performance improves.

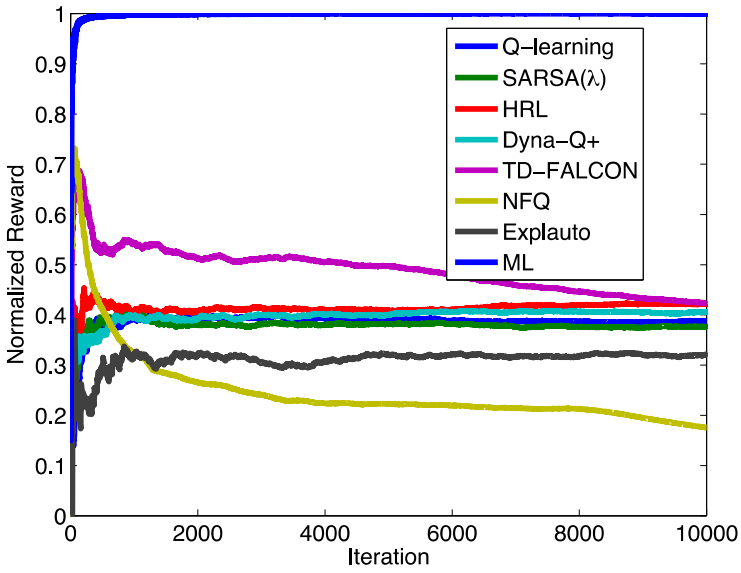


Figure 6-17. Comparison of reinforcement learning algorithms' average reward performance to motivated learning with $S_{Rate} = 2.0$.

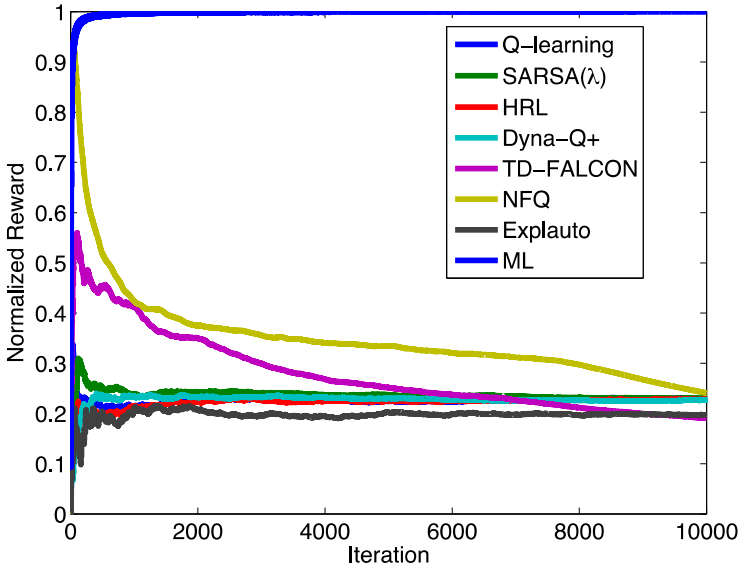


Figure 6-18. Comparison of reinforcement learning algorithms' average reward performance to motivated learning with $S_{Rate} = 4.0$.

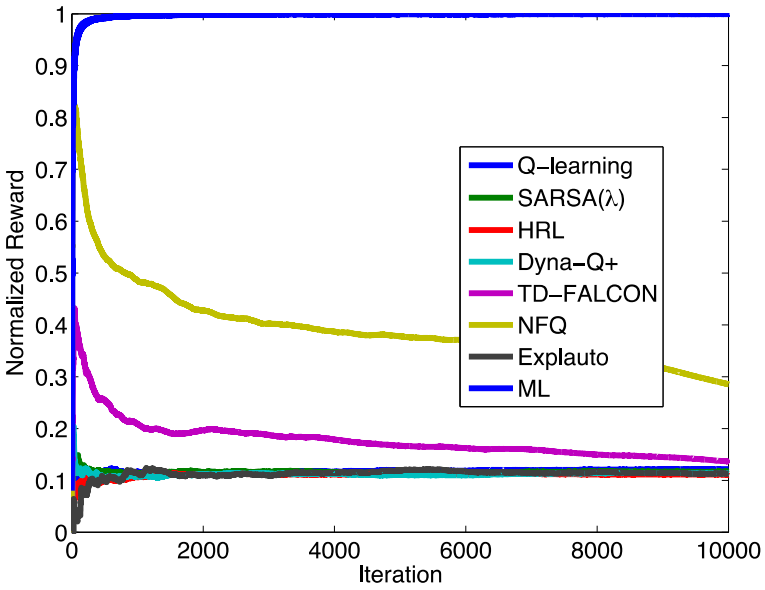


Figure 6-19. Comparison of reinforcement learning algorithms' average reward performance to motivated learning with $S_{Rate} = 8.0$.

Although NFQ seems to perform better that other RL algorithms if the resource decline rate is higher than 4, we can observe its gradual decline with the increasing number of iterations. Figure 6-20 shows the comparison between ML, NFQ, TD-FALCON, and Q-Learning methods for $S_{Rate} = 4.0$ with 20,000 iterations. We see that both advanced RL methods eventually fall below Q-learning, which indicates that in a longer run they are not capable of maintaining their advantage over other RL algorithms in this testing scenario. However, the ML algorithm was able to continue operating without declining performance.

With these results we have shown that the Motivated Learning algorithm performs favorably against several common reinforcement learning algorithms. The results support our assertion that ML outperforms RL in complex environments, particularly, when an agent needs to discover the relations between several different "resources" and is only provided feedback in terms of the environment state and a single "reward" signal.

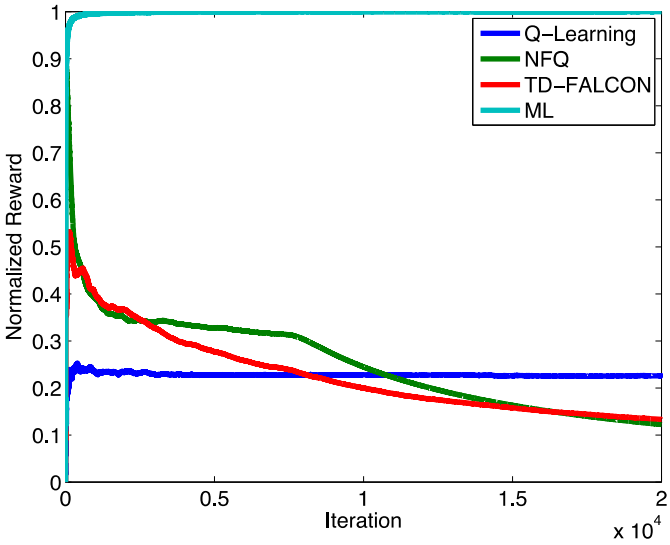


Figure 6-20. Comparison of NFQ and Q-Learning at 20,000 iterations with $S_{Rate} = 4.0$.

6.3 Discussion and Conclusion

In this dissertation it has been argued that a Motivated Learning approach is able to effectively solve resource sharing and task coordination in a dynamic environment. This is because a ML agent's internal reward system motivates it to act effectively in a dynamically changing environment with limited resources and respond to potential adversarial actions by other agents. This chapter presented a range of ML results from the different implementations of the algorithm, and in the last two sections, ML was compared against a number of RL algorithms. An earlier comparison tested ML against a TD-Falcon based RL implementation, and more recent tests used several RL algorithms in a black box scenario. In the TDF comparison, it was shown that ML's internal motivation system gave it a hand up when competing against this RL algorithm. The black box testing reinforced this observation by comparing ML against a number of other RL agents. In fact, as shown in Figure 6-6, ML was able to maximize its performance and perform relatively well even when running into time constraints. Even when the agent was running out of time to perform all necessary tasks (to avoid all its pains), its performance diminished gradually with the increased rate of change of the primitive reward. There was no catastrophic collapse of its functions as was seen in Figure 6-5 for TD Falcon or in Figure 6-20 for TD Falcon and NFQ methods.

It is planned to extend the black box scenario's complexity by including actions by other agents (that can either cooperate or compete with the agent under test). Another addition to the scenario may be the concept of movement and "space" to allow for travel time and resource

searching in the environment. While all of the more recent ML implementations handle actions by other agents as well as deal with objects distributed in the environment, none of these features were included in the black box scenario tested in this dissertation. This was to keep the black box scenario as compatible as possible with the various RL algorithms. Additionally, the black box code has been made available online in an effort to challenge owners of other algorithms and let them try their luck with the scenario, as well as to help us collect more data. More information about the scenario and the associated autonomous learning challenge can be found at this link: http://ncn.wsiz.rzeszow.pl/autonomous-learning-challenge/

CHAPTER 7: COGNITIVE MOTIVATED LEARNING

7.1 Introduction

Currently, the most advanced ML based architecture is the Motivated Learning Embodied COGnitive agent model (MLECOG) where the agent's operation becomes more complex and takes on sequential, rather than the purely parallel aspects found in the early versions of the ML model [20], [91], [97]. The basic ML model was comparatively simple and focused primarily on basic motivation, as depicted in Figure 7-1. However, the more comprehensive model discussed in this chapter shifts the focus from just motivation towards more complex interactions with memory; including attention switching, procedural and episodic memories, and emotions. A recent publication [106] provides a brief overview of the differences and some comparison between the original fully parallelized model and the "simplified" sequential model. To summarize, the MLECOG architecture builds on basic ML by adding several structural and functional components in the form of the attention switching, memory components (semantic, episodic, and procedural memories), planning, action monitoring, etc. These modules extend and enhance the capability of basic Motivated Learning as discussed in this chapter by giving the ML agent capabilities such as planning multi-step actions, developing complex associations and reducing the amount of processing it has to do to keep track of the environment. In the following sections, the MLECOG model will be described and the roles of its various components laid out. (Please note that much of sections 7.2 and 7.5 are drawn from a recent journal publication concerning this work [24].)

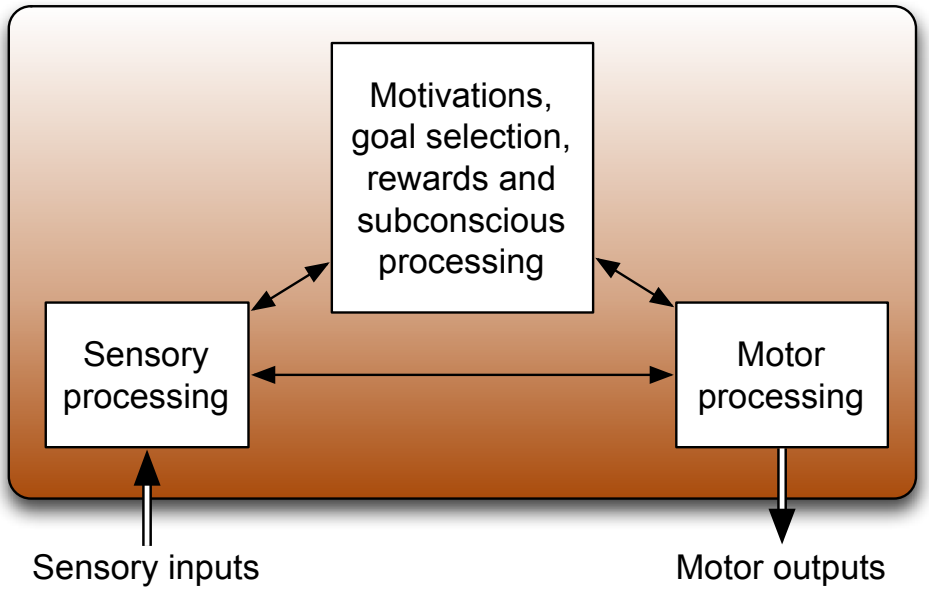


Figure 7-1. Basic ML system implementation.

7.2 The MLECOG Model

The MLECOG architecture responds to an embodied system's needs, has the overall motivation to satisfy these needs, and is similar to other need based approaches such as the MicroPsi architecture described in [58]. Using the ML approach, it can generate new needs beyond those it starts with. For instance, if it learns that it must burn fuel to keep warm, it creates a need for fuel. The difference to other approaches is that these created needs compete with all other needs for attention (and are not simply chosen via the WTA pain based and opportunistic mechanisms discussed in Sections 3.2.1 and 4.1.1). The needs may be related to resources or events in the environment and can be either positive (when the machine wants a specific resource or event) or negative (when it wants to avoid them).

The MLECOG architecture uses the same approach to ML as the previously discussed models, such that if a need is not satisfied, the machine uses the negative pain signal associated with the need, and reduction in the pain signal is equivalent to a reward. As in reinforcement learning, a reward reinforces actions that led to it. In the motivated learning (ML) approach (see Chapter 3 and/or [99]) even the signals normally associated with pleasure or reward are converted to pain signals. These signals can be reduced with proper behavior.

This is done for a specific reason; a machine that uses negative pain signals has a simpler task management system, since dominating pains would be satisfied first. Alternatively, a positive

need signal can be used, and the learning task would be to minimize this signal. In both cases, mathematically, the problem is of the minimax optimization rather than maximization, typically used in reinforcement learning. No single need will take precedence over other needs. A system that uses pleasure signals to drive its behavior may choose to maximize its dominating pleasure neglecting other needs, with devastating effects like in the case of drug abuse. Another example of pleasure maximization that leads to a disaster is the famous experiment with rats, where a rat would stimulate its pleasure center, neglecting the need for food or drink until it dies [118].

Another important aspect of the MLECOG architecture is that it uses a new mental saccades concept as an element of the attention switching mechanism [119]. Visual saccades use salient features of the observed scene to abruptly change the point of visual fixation in order to focus at the location of a salient feature. In the machine sensory system, visual saccades can be implemented by using a winner takes all (WTA) mechanism combined with inhibition. Once a salient feature at a certain location wins the competition using the WTA mechanism, the visual focus is shifted to this location in support of the visual recognition process. Once a specific location is inspected, an inhibition signal is generated to block the previous winner from winning again, so a new salient location can be selected.

Mental saccades are proposed as a concept parallel to visual saccades, their purpose being to abruptly switch from one area of memory to another based on the dominant activation of associated memory neurons (or their ensembles). If the working memory mechanism focuses on a selected concept based on the activation of neurons in the semantic memory, other concepts are partially activated through associative links. Inhibition of a previously selected concept helps to switch attention to the next most activated concept. This process of selecting the winner, followed by the winner's inhibition and selection of another winner, provides a natural search of the memory areas related to the current mental task considering the strength of associations and priming signals that arise from interactions between various memory areas.

Memory in MLECOG is organized to help it function in real time in an open environment and perform all of the cognitive and non-cognitive processing. While non-cognitive processing takes place concurrently in various parts of the memory, cognitive processing is basically sequential and relies on attention switching and mental saccades to manage several possibly concurrent cognitive tasks that the agent may be involved with.

This is not to imply that the whole process of cognitive perception and decision-making is sequential. In fact the opposite is true – before a machine places a single concept in its working memory, a number of concurrent processes take place to identify the object, activate a group of

associated concepts, and to search primed areas of memory and motivational blocks for a solution. In MLECOG, the planning and cognitive thought processes proceed from one concept to another one sequentially using mental saccades and attention switching. However, many subconscious activations are needed to support these processes. A conscious thought is the tip of the iceberg that represents a winner in the competition between many unrecognized options and possible solutions to a problem.

Figure 7-2 presents our initial rendition of the MLECOG model and shows the major building blocks of the MLECOG architecture. Before we move onto the more detailed discussion of the model, let us perform a brief overview of its functionality using this figure. The operation of the model begins with "pre-processed" data received via the sensory inputs. The input is pre-processed in the sense that it is rendered in such a way that the agent does not directly interact with raw data. Despite not being implemented in this work, feature level object recognition and sensory preprocessing are important for a robust episodic and semantic memory implementation, since the agent needs to be able to categorize and organize what it observes. Following the input being passed to the agent, the environment data is presented to the Rewards and subconscious processing, Attention switching and Semantic memory blocks.

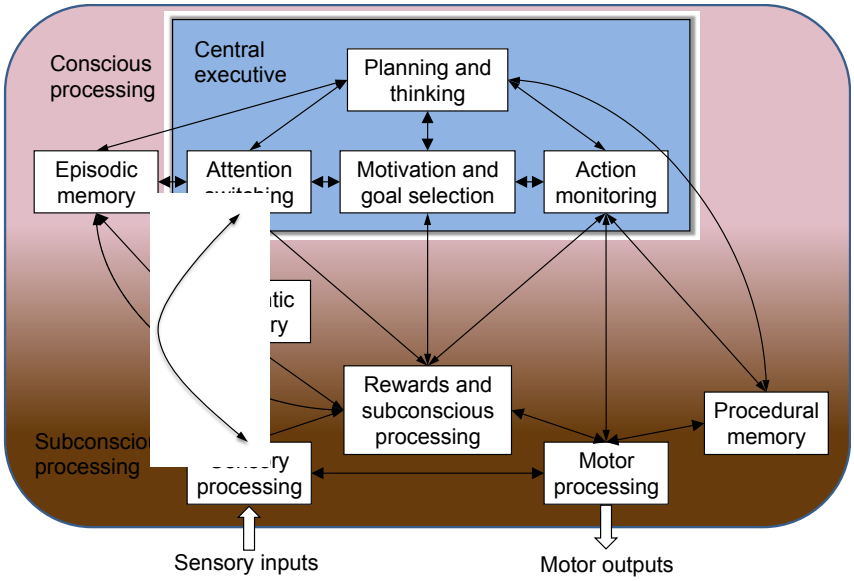


Figure 7-2. Full sequential cognitive model. (simplified diagram)

The Sensory-motor block handles sensory input and motor output. Information is fed through whichever sensory units are available and pre-processed into a form usable to the agent.

The Sensory processing block also performs other processing that may be necessary, such as object detection, motion detection, and extraction of features. At this point relevant data can be passed directly to the motor processing block to perform hard-coded "reflex" behaviors. Concurrently, the sensory data is passed onward to excite the Semantic memory and the Rewards and emotional/subconscious processing blocks. On the right side of the figure, is the Motor processing block, which contains basic motor functions that can be triggered from Procedural memory, Cognitive action control, Emotional response, or directly from the Sensory processing block.

The Semantic memory uses the environment data to "prime" its activations and trigger saccades based on the current attention focus. The semantic memory is a major component in the "flow" of information, but does not readily fit into the Central executive block. It takes in sensory input and the appropriate contextual knowledge is activated, such that it provides contextual information and "suggestions" to the system. More on how the proposed semantic memory functions in conjunction with attention switching and saccading will be covered in Section 7.3.

In addition to the semantic memory, there are also episodic and procedural memories. The episodic memory is prompted by semantic memory activation over time and tracks the generated sequences to form episodes or reads them back to activate semantic memory of events. For example, the episode of "what I did this morning" consists of a sequence of temporal events (stored in the episodic memory) with semantic knowledge of the events and any emotional context associated with them (stored in the semantic memory and emotionally marked by pain-based significance). Conceptually, episodic memories are not deleted; instead, as they decrease in importance or fade over time, they become "compressed" and lose information, to the point that they may fade into non-existence. Alternatively, they may be transferred to semantic memory and exist primarily as semantic knowledge or to procedural memory for learned actions. For instance, the memory of "what I did this morning" may initially be purely episodic in nature with good recollection of the scenes that occurred, however by the end of the day, it will likely be reduced to little more than a list of "I did this, then that, followed by the next thing." The individual may even remember this memory for several days, however, how many people have a precise recollection of what they did during the first hour of a random day of a month in their past (outside of their knowledge of common habits)? Episodic memory dynamically stores significant episodes, and episode significance affects the strength and duration of this type of memory. Recent research has found episodic memory to be crucial in supporting many cognitive capabilities, including concept formation, representation of events, and recording of progress in

goal processing [120], hence the addition of episodic memory to the model has the potential to greatly extend its capabilities.

In an effort to investigate potential avenues for implementation of semantic and episodic memories, we performed a brief study using Prof Horzyk's ANAKG [121] (active neuro-associative knowledge graph) in place of a semantic memory. Horzyk's associative memory currently operates on sequences of words. Each "sentence" is broken down into words and is presented to the network as a sequence. As each word is presented, associations between words are built. When a word is repeated, more complex associations develop. Take an example provided by Horzyk; we are given a sentence, "I have a monkey." The sentence is presented to the network sequentially and each word forms a node. Each node is linked with strength of 1.0 to the prior node; however, all latter nodes have connections to previous nodes. In Horzyk's example, there is a connection with strength 0.48 between 'I' and 'monkey'.

But what happens when a new sentence is added? When we add "My monkey is very small", we see a repeat of the term 'monkey'. However, this merely means the term 'monkey' now has two connections with strength of 1.0 leading to this word. Without going into too much detail, this approach leads to a web of interconnections and relationships that form an associative network, such that after Horzyk's example is finished receiving a set of 9 sequences regarding a monkey it can hopefully produce meaningful associations. The remaining sequences in Horzyk's example are: "It is very lovely", "It likes to sit on my head", "It can jump very quickly", "It is also very clever", "It learns quickly", "My monkey is lovely", and "I also have a small dog",. When asked the question "What is a monkey like?", it generates associated output "Monkey is very lovely small clever".

This approach to semantic memory is very useful due to the displayed sequential and associative components. In the described example, we can that the question is able to recall several properties of the monkey that were given over the course of several different sentences. The ANAKG approach seems as if it may be ideal for use in an associative semantic memory. We put it to a stronger test to see if we could justify the need for cooperation between semantic and episodic memories. Take the following test example, shown in Figure 7-3.

Previously used input sequences were appended to also include:

I have a sister.

My sister is lovely.

She is very lovely.

She likes to sit in the library.

She quickly learns languages.

I also have a brother.

When asked the question:

Question1: What is my monkey like?

ANAKG memory produces the output:

Output1: IS VERY MY LOVELY SISTER IS VERY LOVELY

In a similar way the question:

Question2: What is my sister like?

Results in the output:

Output2: IS VERY MY LOVELY SISTER IS VERY LOVELY

Figure 7-3. ANAKG example.

In the results above, the same output is generated by both questions. This is because of the associations with the other words independent of "sister" or "monkey". To clarify, we get the same answer as when we ask "What is my sister like?", because the words "What is my …. like?" are present in both question sequences and produce the same associations/activations in both questions. While the question with the word "sister" may produce slightly stronger activation to the words "is very lovely", the words 'is' and 'my' also produce activations with "very lovely" in sentences 2 and 3. In any event, this is not a desirable result. While the semantic memory is useful in building associations, it does so without the full context. Hence, we believe that without the sequential relationships of an episodic memory, the kind of confusion shown here will result.

This is where the use previously developed LTM (long term memory) cells [122], [123] as a form of episodic memory comes in. LTMs operate by storing entire sequences. The 'magic' is in the recall operation, where the similarity of the input is gauged against the existing LTMs. We can observe how closely an LTM matches the provided input, even accounting for shifts and

missing sections in either sequence. These features of LTMs make them appear as ideal for the major component of an episodic memory.

We wanted to see if we could present sequences to both semantic and episodic components and then "prompt" the semantic memory with a question. The semantic memory would then be used to excite the episodic memory to formulate a response, and we could effectively get an episodic recall. This work is currently ongoing at the time of writing, but initial results have been submitted for publication as a journal paper [124].

Section 7.3 discusses another aspect of using semantic memory in the MLECOG architecture. Associations provided by such memory help to search the observed scene for interesting artifacts and are fundamental for mental saccades, attention focus, and attention switching. Hence, Section 7.3 discusses how semantic memory interacts with, and the agent's decision making process is supported by, mental saccades, attention focus, and attention switching. Thus, proper implementation of integrated episodic and semantic memories is very important for a successful implementation of the full MLECOG architecture.

The inclusion of a procedural memory allows the agent to "cognitively offload" complex tasks once they have been learned. Take the common task of preparing cereal for breakfast. There are several steps that need to be performed, including selecting the cereal, milk, and any needed toppings. It is essentially a recipe to be followed. However, after the first few instances of making cereal for breakfast, the process will become more streamlined and not much thought will be needed to perform the task. This is because the task has been largely moved to procedural memory. The actions needed to perform the task have been internalized and defined in such a way that it is no longer necessary to explicitly think about them to do the task.

The Central executive block, or the Working Memory, is the core of the system. It is also where the "conscious" processes in the agent tend to reside. It is where actions are evaluated and behaviors are decided upon. It consists of three main components or functionalities: Attention switching (and saccading), Action monitoring, and Planning. These functionalities perform the core operations required by the Working memory to allow the agent to choose actions to perform. As can be seen from Figure 7-2, the modules are heavily interconnected and depend upon each other. However, each module plays a distinct role. The motivation and goal selection blocks are responsible for determining the machine's needs and setting goals for those needs. The Attention switching block takes in perceptual activations and focuses the agent's attention on what it (with help from the motivation and subconscious processing) decides is the most relevant. The Action monitoring block directs motor processing or procedural memory in fulfilling the actions as

specified by the planning and/or motivation modules. And finally, the Planning and thinking modules duty is to help facilitate the agent's ability to organize sequences of actions or plan for the future.

The Central executive receives perceptual information from the environment in the form of activated semantic and/or episodic memories. The Attentions switching block, in conjunction with the Motivation and Planning blocks, performs visual and mental saccades through the activations as described in [119]. To summarize, the agent visually saccades through relevant objects within the current scene. The objects are selected either via mental saccades or are brought into attention via unconscious processes. Mental saccades can trigger visual saccades and vice versa. Generally, mental saccades are triggered to resolve a need and cause the agent to saccade through its knowledge base and available objects within the environment in search of a solution, previously known or not. The winner of this saccading process becomes the focus of the agent. The agent then incorporates the focus into its current plan, or uses it as a "stepping stone" in directing the attention focus to another subject of interest. More on this process will be discussed in Section 7.3.

It is the Action monitoring module that directs the activity of the agent, while attention switching controls what the agent pays attention to, and planning and motivation affect what it is "working on". It coordinates with the other modules to actually carry out the agent's decisions and keeps track of the progress the agent is making on its set task(s). Note that there is no centralized decision making component in the model (there is no infamous homunculus known to philosophers of the mind [125]). The interplay between the Central executive modules themselves is the basis for "decisions" by the agent. The combination of various needs, the attention focus, and the planning elements combined with the execution coordinated by the action monitor all come together to allow the agent to form decisions about what to do in a particular situation.

7.3 Mental Saccades and Attention Switching

Much of the more recent work in developing the MLECOG model has been focused on designing and implementing the semantic memory and attention switching modules. When fully implemented, they will give the agent the ability to focus its attention on specifics and plan more complex solutions while still being driven by motivational signals.

Recall from Section 7.2 that the Central executive block (Working Memory) consists of several components, primarily the Attention switching, Action monitoring, and Planning

components. These components combined with the Semantic memory's ability to store contextual information will allow the agent to function effectively in its environment. To simplify the discussion, we will ignore some of the more complex functionality of the model, such as the Scene building and Episodic management for now.

The agent's internal state is updated both in parallel and in serial ways. As previously mentioned, much of the underling structures outside of the Working memory operate as parallel structures. This is especially true of the motivational block, whose outputs change when it is notified of changes in the environment that are significant to the agent. The changes include changes in the primitive needs, or when a new abstract need based on the successful reduction of a need using a previously "unknown" object/resource is created. We are making the assumption that insignificant changes would be filtered out by the subconscious attention switching mechanism, such that the agent would ignore most minor changes in the environment, which do not affect its motivations. The primitive needs are updated in parallel; however, updates to abstract needs occur only when the agent "thinks" about them by saccading to a specific abstract need and evaluating its state.

This "thinking," performed by mental saccades, depends on the semantic (and sometimes episodic) memory. As has been mentioned, the semantic memory works by accumulating knowledge that is primed by the environment data, pain levels and motivation information and makes that knowledge available to the Working Memory (via the Attention focus). When queried by the Working Memory, the Semantic memory saccades through the primed nodes (activated to various levels), allowing the agent to quickly analyze the scene and/or options, and provides a suitable action for testing via the combination of Action planning, evaluation, and monitoring.

Starzyk [119] proposes that a mental saccade process, similar to visual saccades, might exist and can be used by an intelligent agent to scan its associative memory and analyze its options. Initial saccades are random but, as the agent learns to handle its pains, the saccades are primed by the usefulness of the objects or actions towards its goals. During visual saccades, the attention focus moves from the object with most salient features to the one with less salient features. Similarly, during the mental saccades, the attention moves from the object currently in focus (either sensory or mental) to other objects or actions associated to the one in focus. As the memory saccades, the previously scanned concepts and/or actions are inhibited, using a mechanism based on [119], until associated areas of memory are searched through or a decision must be made before the search is completed (e.g. playing chess game with limited move time). This process is not predetermined by the initial state of activations of the memory neurons, since

they are influenced by the process itself. Thus, associations to neurons that were not initially activated are very likely and the whole thinking process is self-driven and responds to changes in the environment.

Note that in mental saccades, potential reduction in actual pain levels is evaluated without inhibiting real pains. The saccades can be controlled by the Planning unit to evaluate how useful the concept in focus is for its goals. In the simplified implementation (in section 7.4), the Attention Switching unit can start a saccade, but it is the focus on a specific pain-action link in the working memory that determines a potential action. Whereas, in the full MLECOG implementation (see section 7.5), there is a dedicated Mental Saccades module that works with the Action evaluation block to switch attention in the Semantic memory and adjust the Attention focus, and decide on an action via the Action planning and Action evaluation modules. In both the full and simplified implementation, motivations act as priming signals and help to direct attention to memory areas related to the most active pains. Hence, when the Planning unit requests information from the memory, it receives the most relevant information related to the current motivation.

Moreover, the knowledge stored in the Semantic memory unit is updated through the ML mechanism based on the latest pain level feedback. Thus, the agent self-directs its saccade process and learns how to fit into the environment through the sensory-motor-environment loop.

In Section 7.2, we discussed semantic memory in the context of forming associations and episodic memory to provide the context of observations. In this section we further address the semantic memory component in regards to how it relates to saccades and attention switching. In the broadest sense, semantic memory is used to recognize, associate, learn and assist in controlling an agent's operation (by providing information to other modules). Incremental knowledge is gradually gained and stored within the semantic memory via interactions with the environment. And the Semantic memory block uses the environment data to "prime" its activations and trigger saccades based on the current attention focus. This, combined with the interplay with episodic memory, allows the agent to learn a structural representation of the relations between the objects in the environment. The ability of a system to recognize objects and associate them with prior experiences is extremely useful for the learning agent and is critical for the agent's development. (Therefore, the design of mechanisms and self-organizing architectures required for the aforementioned memory components is equally critical.) The addition of semantic memory and the associated saccade scheme to the Motivated Learning approach also pushes the agent further away from the Reinforcement Learning approach. RL in addition to the

other properties discussed in Section 2.2, is primarily reactive, whereas the combination of memory, planning, and attention switching can give rise to a proactive agent.

Note that the agent's sensors can only detect the features of the objects in the environment and it is the semantic memory that uses these features to cognitively "recognize" the objects. Thus, each object in the semantic memory is represented as a set of features coming from the sensors. Such feature sets in the semantic memory provide agents with the ability to form higher-order conceptual categories. Objects with shared features can be combined to form a conceptual category. For example objects that satisfy "Hunger" pain can be grouped into "Food" category. This ability to form conceptual categories can be extended to form categories of categories. Thus each object is represented by groups of nodes representing features; this is analogous to words being represented by groups of neurons as suggested by Pulvermuller [126].

The Semantic Memory unit in the MLECOG model is designed to play a similar role, i.e., it stores information related to the facts and relationships that it has learnt. The facts would be resources an agent encountered, its sensory inputs, motors, pains, and goals, while relationships would be formed by the motor actions performed and the resulting changes in the pains and motivations. The past observations and experiences are used in formulating actions and plans, with the resulting changes in the environment being used to update the memory for future use.

As mentioned there are two types of saccades in this work: visual and mental saccades. Visual saccades occur in the model at object detection. It may be appropriate to refer to visual saccades as environment saccades since they respond to environmental stimulus of the dominant sensory input. If the agent had "ears" it could perform saccades based on auditory input signals in addition to visual and other potential sources of input. However, for the purposes of this work we concentrate on visual input alone.

Mental saccades generally occur based on the current visual saccade. Once the agent has selected a focus in the visual saccade to "pay attention to" it will proceed to saccade about the visual focus object and "think" about the various associations attached to it in the semantic memory, and maybe even choose to perform an action based on its though process. However, if at any time, something causes it to switch its attention, it can interrupt the current thought process and move on to an entirely different track. For better understanding of the saccading process we present the following example (Previously presented in an earlier journal paper our ours [99].):

7.3.1 Saccading example

Our example is taken from real life and presents a dog as an agent and his mental saccading mechanism. Figure 7-4 displays the "agent" (dog) and different objects around him outlined and labeled by letters. In this situation we can see the agent, which has entered the kitchen to find food. Looking around he saw several different things in the field of view but focused on one, a dish filled with food. During this episode a controlling mechanism for the eyes (visual saccades) is working. The brain registers one by one all of the "interesting" details in the image located in front of the agent. This process activates associations with observed objects which are currently stored in semantic (and if present episodic) memory.

Figure 7-4. Visual image saccade example with the following objects:
A) Dog(agent), B) Metal Dish with Dry Food, C) Plastic Toy, D) Cookies, E) Dish with Water, F) Carton of Milk, G) Jug of Water, H) Toy.

Some examples of the semantic associations for each of the visual saccade objects are presented in Figure 7-5. Of course these are not all the possible associations, but probably the most common.

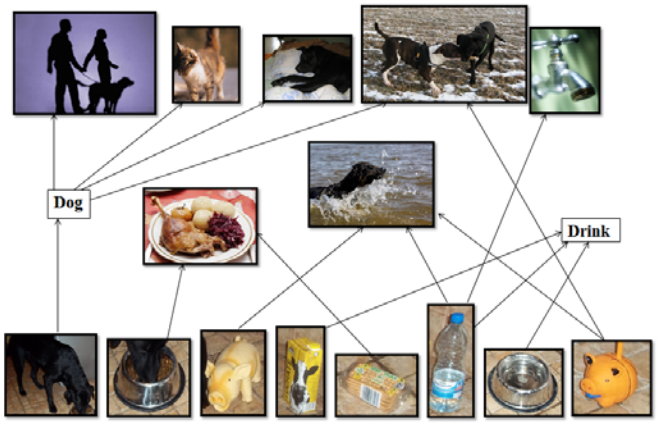


Figure 7-5. Object associations.

The most important and noticeable object in our scene is the dog. We can find many associations such as the general "dog" heading, his natural opponent (cat), dog toys, things it likes to do, certain behaviors, and even episodes recalled from its episodic memory. By these semantic associations we can find and associated needs with an agent. After the "agent" evaluation we proceed to other objects. Their evaluation and assignment of relevance to each of them will be based on the context of how useful they are to the agent with its current set of needs. When we consider objects B and E, which are strongly related to the dog's natural needs (hunger and thirst) and since they are all food or water dishes they may become part of the mental saccade for the dog. Or at least, they are likely to experience a longer visual saccade due to their relationship with the agent.

When our agent is hungry and since its dominant need is "food", it will most likely bypass the whole kitchen scene without performing any other actions. We say "Most likely" because we can't be 100% sure of the internal states of the agent. We can try to consider another need (e.g. thirst) in its current set of needs. The need for water is not likely to be as significant as the need for food. However, the sight of water may be enough to briefly bring this need to the attention of the planning module. Going further, this would cause a mental saccade to water sources. This would be the perfect opportunity for the agent to engage in opportunistic behavior and briefly interrupt its search for food to get something to drink (or perhaps simply add it to an internal queue and perform the action when done eating).

During interaction with the environment the agent often has the opportunity to change its current action and switch to another or just stop what it's doing to think. In the MLECOG model,

"opportunistic behavior" is primarily handled via the visual and mental saccades components of the model.

The Attention switching block currently acts as a subconscious attention manager in the simplified MLECOG implementation (discussed in the following section, Section 7.4) and forces the agent to pay attention to shifts in the environment and/or significant changes in motivation. In the current simplified implementation, the switching is triggered by incoming perceptual changes rather than high-level consciousness, since the memory structured need for higher level switching have yet to be implemented. By using the attention focus, the parallel processing of sensory information, internal need signals, and associations in the OML model is replaced by sequential, cognitive processing in the MLECOG model, such that an agent can focus on the things that influence its interests the most rather than everything at once. This is very important, because by utilizing attention switching, a great deal of processing power is freed up to allow the agent to "think" about the current focus of attention [119].

A parallel system presented in the basic ML model takes in the entire environment state every cycle, which, in a real-world environment is a very large amount of data that would require a significant amount of processing power/time to fully handle. On the other hand, by utilizing attention switching, the agent only has to evaluate one thing – an object, a concept, or an action - at a time. While the initial "reconstruction" of the observed scene, where the agent performs detection and evaluation of the objects within the environment, may take some processing time, more time is available for later processing activities (such as examining episodic memories or related objects in memory).

Because an MLEOG agent only updates its internal representation when it has an attention switch, and only updates it with respect to what is currently observed, abstract pains are updated only when the agent has had an attention switch to the associated resource or concept. Updating the internal environment representation this way rather than every cycle impacts nearly every facet of the agent's operation. For example, the agent does not have a true "real-time" mental representation of the environment, because its internal maps can only update a working memory feature after it has been subject to a saccade. Nor can the agent evaluate its actions until it has had the chance to saccade to the objects that its action effects. This means that an agent based on the MLECOG model has to assume that its environment does not change drastically in a short period of time. This is similar to limitations resulting from human reaction time, since we are also limited by the speed at which we can process changes in the environment. More information on

our conceptualization of how attention switching and saccading work in conjunction with an agent's memory can be found in related publications [119], [127].

7.4 Current MLECOG Implementation

The development of the ML model has been incremental. This section covers the implementation of modules that could be developed quickly enough to include in the simplified MLECOG implementation. However, it is still intended that other modules will be added in future work. The development of the current implementation is based on the simplified version of the MLECOG agent as depicted in Figure 7-6. This version, as implemented in Matlab, significantly reduces the parts played by the episodic and procedural memories, and uses a pseudo-semantic memory. Its purpose is to act as a stepping-stone from the original ML model to the full MLECOG model (discussed in Section 7.5). Doing this allows for easier refinement of the model structure and better understanding of how to implement components such as episodic memory and scene building. While Figure 7-6 omits some of the complexity in the full model, shown in Figure 7-15, it nevertheless provides a good stepping stone for the advancement of the MLECOG model. The operation of the model shown in Figure 7-6 is summarized in the following section. (This simplified implementation was previously discussed in [24], [106].)

7.4.1 Simplified MLECOG Model

In order to simplify the creation of an agent, environment complexity has been reduced to a largely symbolic form. In this simplified form of the agent, objects in the environment are perceived by the agent through their identifying ID, location, and a "quantity." As the implementation of an environment within NeoAxis [16] and the creation of the sequential model advances, elements are already in place to move the agent toward a more feature based object representation. Hence, while sensory concept grounding is not yet implemented, it is in development, and preparations have been made to allow its future incorporation into the model.

"Pre-processed" data is received via the sensory inputs. The input is pre-processed in the sense that the ML agent do not directly interact with raw data, and instead rely on a set of predetermined symbols and their features. The environment data is used to "prime" the activations of pseudo-Semantic memory block and trigger saccades based on the current attention focus. It is referred to here as a Pseudo-semantic memory block, since the semantic memory (and associated episodic and procedural memories) have yet to be implemented for use with the MLECOG model. Rewards and subconscious processing, and various other blocks work as

discussed in this chapter (see the prior two sections), although, they are implemented at reduced levels of complexity.

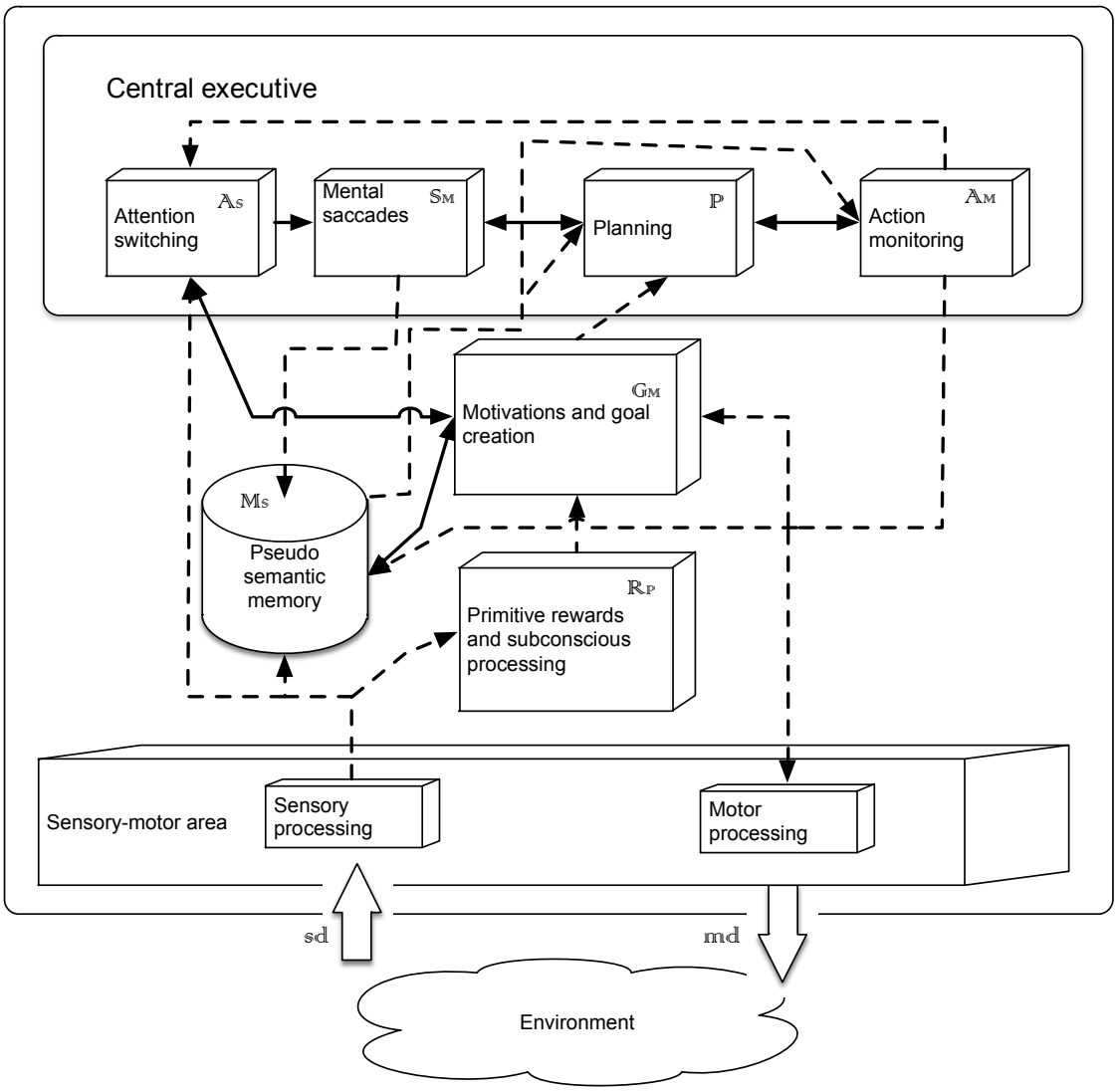


Figure 7-6. Implementation schema for the simplified sequential MLECOG model.

The Motivation and Goal Selection block in the current implementation works in much the same manner as in the basic ML model, where motivations are derived from primitive pains, and new motivations, generated as solutions to "lower-level" needs, are found. Motivations can be specific or general, i.e., specific in the sense they are directed at a single need or object, or more general in the case of a class or type of object. The current implementation does not allow

for general categories of concepts (e.g. things beyond symbolically defined "resources" and/or NACs we have included in the environment) due to limitations of the pseudo-semantic memory. For the agent to do so would require the semantic memory to be capable of categorizing and linking together objects into more general classifications, something the pseudo-semantic memory is incapable of. Because of this it cannot generalize or subdivide classification and objects.

The Planning block uses saccade data from the memory and motivation data from the Motivation block to select which options it should pursue. It currently uses the Opportunistic ML approach discussed in Chapter 4 and in [21], [99] as the core of its decision process. In general, the planning block exists to allow the agent to examine its motivations, the associations generated by semantic and episodic memory, and create a plan of action to alleviate its needs. The Action Monitoring block then tracks the course of the winning action and determines when it is complete and should be evaluated. The action monitor can also allow the agent to resume actions that may have been interrupted (e.g. when an opportunity as determined by the OML algorithm showed up).

The last significant block in Figure 7-6 is the motor processing block, which is handled by the environment interface in the current implementation. Motor processing is important because it translates the agent's planned actions into their execution in the environment. A more complex "full" MLECOG agent will possess its own learning capabilities in conjunction with procedural memory to construct complex learned motor actions. In the current implementation, "Motor" actions are handled by a simulator as it follows the action commands provided by the agent.

Additionally, it is worth noting that since the MLECOG implementation is built upon the foundation of ML from Chapters 3-4, it inherits the features of ML. This includes the more recent developments discussed in Chapter 4 and [98], such as the ability to handle NACs and improvements to bias and pain calculations, that improve the robustness of ML.

7.4.2 Simplified MLECOG Model Implementation

In this section the simplified (and symbolic) implementation of the MLECOG architecture within both Matlab and a NeoAxis based environments are presented together with some simulation results. As stated previously, a symbolic architecture works by manipulating predefined symbols without proper symbol grounding. However, this implementation allows the construction and testing of the basic modular structure of the MLECOG architecture and its

underlying motivated learning principles without having to worry about the complexities of symbol grounding and the associated memory implementation.

For the reasons mentioned in Section 5.4.2, simulated environments based on Matlab and NeoAxis were chosen to test the MLECOG agent's functionality. While an agent can be implemented via robotics, where the agent controls a robotic body, implementation via simulation, where the agent exists within a virtual environment and possesses a virtual "body," allows for less expensive testing and better control over the environment conditions.

A simplified MLECOG model (as discussed in Section 7.4.1) has been implemented in both Matlab and within the NeoAxis environment. In this implementation the agent has no separate procedural or episodic memories. The lack of procedural memory is because the simplified MLECOG implementation is designed for a purely symbolic environment, significantly limiting its choices. Actions remain in the form of sensor/motor pairs (target a sensor object and perform a motor action on it), and while actions themselves can take multiple cycles, multi-step sequences of actions have yet to be implemented. This simplified sequential implementation has been deliberately tailored to match the capabilities of the parallel OML implementation in order to make some comparison possible. The ability to compare two subsequent implementations is useful in this field of research where it can be difficult to effectively benchmark and/or compare the developed models.

To elaborate, the simplified sequential implementation has been designed with the same simple environment and the sensor/motor pair setup that the earlier implementations (ML and OML) used. However, progress was made toward increasing the system capabilities such as using additional features and more complex environments. To provide some context, let us briefly compare the earlier parallel OML implementation with sequential MLECOG.

Table 7-1 summarizes the major differences between the two motivated learning model's implementations. We shall note that in the sequential implementation the agent does not update abstract needs/motivations every cycle. This is because it only "updates" one resource at a time after attention was focused on this resource. On the other hand, primitive pains are updated every computational cycle since they are based on parallel and subconscious sensory inputs. Furthermore, the MLECOG implementation possesses more advanced memory structures. The sequential MLECOG implementation described in the table is more applicable to the "Full" MELCOG implementation rather than the simplified version has been implemented. However, the progress made in the simplified MLECOG implementation justifies continued work in the direction of the full MLECOG implementation. To clarify, since the implemented simplified

model does not contain a true semantic memory, the last two entries of the MLECOG implementation in Table 7-1 do not yet apply.

Table 7-1. Implementation Comparison

Parallel OML implementation	**Sequential MLECOG Implementation**
The agent is simultaneously aware of everything.	The agent is attention based, so it effectively "sees" only one object at a time.
All pains and motivations are updated every cycle.	Primitive pains are updated every cycle, but abstract needs are not.
No specific monitoring or evaluation blocks.	Has action monitoring and evaluation components.
Action choice based only on the state of the environment and the agent.	Associations and memories affect action choice in addition to the state of the environment and the agent.
Only capable of forming direct 1-1 associations.	Has semantic memory capable of forming hierarchical associations.

As before, the agent has certain predefined needs (primitive pains), and using the ML approach it develops higher order needs (abstract pains) that it must manage while operating in the environment. For ease of comparison, tests of the simplified MLECOG model use the same environment as the original ML implementation (as seen in Table 5-1).

The NeoAxis environment used to test the agent is essentially the same as the one used in Figure 5-21. The only difference is the agent code running beneath the NeoAxis AI code (and the simpler environment from Table 5-1). Since the development of the semantic memory discussed in this chapter is ongoing, a simplified pseudo-semantic memory was implemented to allow the implementation of the simplified MLECOG model and test its modular structure. Figure 7-7

presents a view of the interface for the MLECOG agent as is implemented in the NeoAxis environment (during the early phases of its operation).

Figure 7-7. Main simulation view of NeoAxis MLECOG implementation.

Figure 7-8 depicts the traces of various pain signals for a MLECOG simulation. Most of the pains are resolved (reduced below threshold, set here to 0.3) shortly after they pass the threshold. [24]

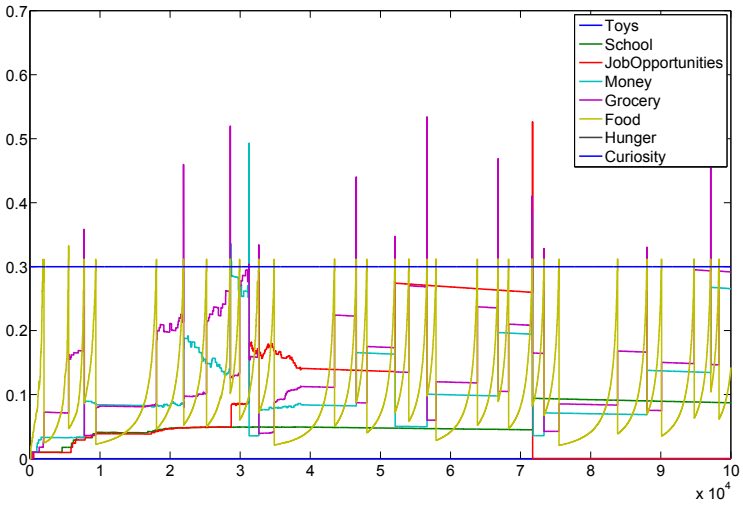


Figure 7-8. Sequential model simulation pain results.

The following list provides a simplified look at the execution of the MLECOG agent operation and a rough idea of the data flow. (Note that not all of the data links are mentioned, just the main links associated with the primary functionality and data flow of the model.) The loop is either executed indefinitely, until the agent "dies", or for some predetermined number of iterations (at which point the agent's "memory" state can be stored for later resumption).

After initialization, the algorithm performs successive iterations. Each iteration consists of the Agent Phase, where the agent observes the Environment, updates its internal state, and chooses what to do (or continue doing), and an Environment Phase where the environment performs the agent's actions and updates itself accordingly.

1. Agent phase:

   Relevant environment information is passed to the agent for sensory processing. This information consists of resource objects, their resource levels, and their distance from the agent, and information about other non-agent characters (NACs), with theirs goals, and distances to resources and the agent. The agent processes the current state and chooses what to do next (or continue doing).

   a. *Update Primitive pains* – updates primitive pains
      i. Receives pain information from the Environment (Sensory processing)
      ii. Sends results to Motivations
   b. *Update Motivations* – passes primitives and updates motivation levels
      i. Receives data from Primitive Pains, Action evaluation, and Memory (no direct link to environment/sensory)

ii. Biases are updated by taking new or updated sensory data into account.
iii. Current pain levels are compared with the pains from the previous cycle to determine what has changed.
iv. Sends data to subconscious attention switching and the sensory area of semantic memory

c. *Update attention switching* – updates subconscious attention switching to check for a visual or motivation interrupt
   i. Receives data from Sensory processing and Motivations
   ii. Determines whether data merits a switch in attention
   iii. Sends attention switch data to semantic memory

d. *Update memory* – performs memory saccade
   i. Receives data from Environment (sensory processing), Motivations, Subconscious attention switching, and Action Evaluation.
   ii. If necessary, Memory is updated to reflect the effect of a recently completed action or visual attention switch.
   iii. Based on input data and internal saccade state, it chooses an action to pass to the planning block, or in the case of a visual saccade, information on the current visual target is passed.
   iv. Sends data to Planning and Motivations (used to pass data from visual saccades for motivation updates)

e. *Update planning* – uses memory data to update plans
   i. Receives data from Semantic memory
   ii. Performs evaluation of data passed from memory and current actions being undertaken and determines the next action (or whether to continue an ongoing one).
   iii. Sends data to Action evaluation

f. *Update actions* – checks action status/monitor and tracks action evaluation
   i. Receives data from Planning
   ii. Takes decision from planning and triggers actions. Also keeps track of ongoing actions and decides when they are complete,
   iii. Sends data to the Environment (through motor control) and Semantic Memory

2. Environment Update phase
    a. Update environment state based on the agent's actions and the environment response.
    b. The current goal involving a sensory-motor pair is implemented. This requires moving the agent to the target and performing the selected motor action. (Generally a multi-cycle process unless the agent is already at the location and the motor action only takes one cycle.)
    c. The environment is changed as a result of this action (e.g. one resource is replenished, while the other, typically the one that the agent acted upon, is depleted).
    d. The sensory data to be passed to the agent in the next iteration are updated.

Above, in Agent phase 1b, *abstract* motivations are updated only when the agent has had an attention switch to the associated resource or concept. The attention switch can either be the result of normal processing, or caused by the agent noting the action is completed explicitly triggering the switch). Under normal circumstances the simplified implementation will trigger an attention switch when performing a mental or a visual saccade. It can also be triggered by a sufficiently large change in the environment (via the subconscious attention switching mechanism). This method of updating abstract motivations impacts nearly every facet of the MLECOG agent's operation. For example, the agent does not have a true "real-time" mental representation of the environment, because its internal maps can only update a feature after it has been subject to a saccade. Nor can the agent evaluate its actions until it has had the chance to saccade to the objects that its action effects.

If the agent has developed sufficient background knowledge and sophistication, it could take these delays into account in its decision making process and effectively reduce or eliminate the effect of delays in its perceptions. This would essentially be a realization of the prediction process we are looking for in the "full" MLECOG model. However, the current simplified MLECOG implementation does not yet have the ability to estimate changes in resources (outside of the primitive resources, which the agent can estimate). This means that the agent has to assume that its environment does not change drastically in a short period of time (a significant change would trigger an attention switch and update the cognitive agent's memory), and that evaluating the results of actions takes longer than in the parallel OML implementation.

The OML model implements a reactive agent, and while it is capable of learning new relationships, the relationships have to be direct (not conceptual, amorphous, or inferred). However, the sequential model can use a semantic memory capable of developing associations between related nodes and forming hierarchical relationships between concepts.

Nevertheless, both agent models currently use the same type of symbolic I/O. Hence, in order for the agent to function, information needs to be rendered into a symbolic and/or feature based format that the agent can understand. While a system that processes raw visual input and learns to dynamically control its motor functions could be designed, it would be a major project by itself (and actually is an active current research area for other researchers). However, in this work symbolic I/O is relied upon, as it is easily deliverable by virtual environments, simplifies development tasks, and accelerates progress in research on cognitive agents.

In the OML model, everything was essentially executed in parallel, followed by polling the environment. In the MLECOG implementation, there is a more complex data processing structure. Each "block" still needs to be executed in a sequence due to the nature of the programming environment; however, there are more blocks and more data flowing between them. The sequential nature of the MLECOG model comes from the multiple cycles required for choosing an action (due to attention switching and saccading). While the lower level modules (such as lower level perception, motivation and subconscious attention switching) basically operate in parallel, the more complex higher level processes of the MLECOG model give it a sequential facet.

Note again, that the simplified MLECOG implementation uses a pseudo-semantic memory that only mimics the functionality of the proposed semantic memory. It possesses basic attention switching and visual saccading, as well as the ability to form simple associations. However, it cannot form more complex hierarchical relationships and concepts as a true semantic memory would, and so it limits the capabilities of the simplified MLECOG implementation.

7.4.3 MLECOG Simulation Results

This section will present some basic results gained from the simplified implementation of the MLECOG algorithm. Due to the fact that a pseudo-semantic memory was used, the implementation remains limited in its functionality; however, because of its similarity to the OML implementation (largely due to the pseudo-semantic memory) it is not too difficult to compare plots from both implementations. Hence, we will begin by displaying resource (Figure 7-9), w_{BP} (Figure 7-10), and pain (Figure 7-11) plots from a basic OML simulation using the environment described in Table 5-1. Note that in Figure 7-11 and Figure 7-14, the primitive Hunger pain has been hidden, since it occurs often enough to occlude the other results. It is also necessary to stress, that while similar in many respects (such that they both use motivated

learning), the OML and MLECOG algorithms remain very different in how they operate, particularly in the form of attention switching.

The three figures below show some results using the OML approach. In Figure 7-9 we can see the fluctuation of resources. As we can see, once a resource is depleted (usually below a value of 0.2) it is eventually restored to a value close to 1.0, which indicates correct operation. Figure 7-10 shows the changes in w_{BP} weights, and hence when a pain/resource combination is learned. In the OML implementation the progression in the weights is fairly regular, with less used (and higher level) resource's weights growing more slowly. Figure 7-11 depicts the associated pain plot, which shows a direct correlation to the resource fluctuation in Figure 7-9.

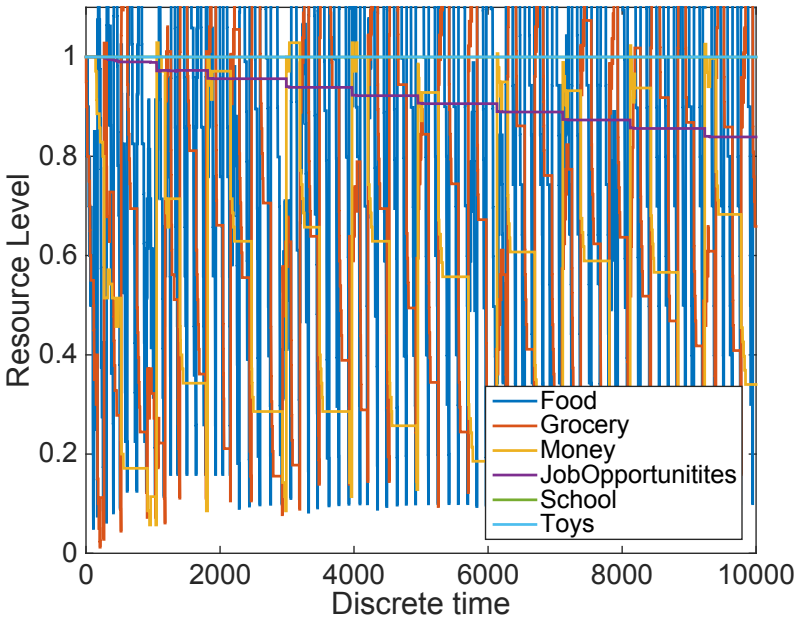


Figure 7-9. Resource plot from OML simulation.

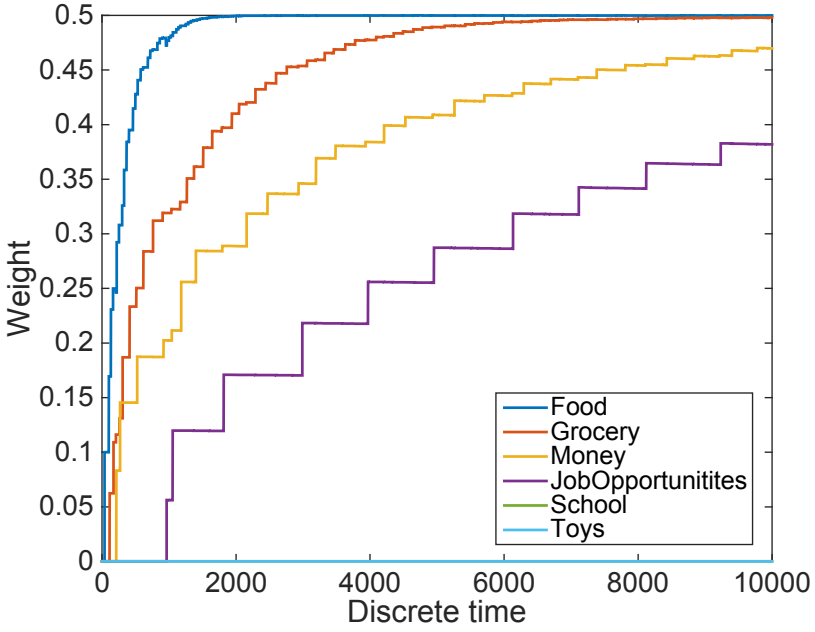


Figure 7-10. w_{BP} plot from OML simulation.

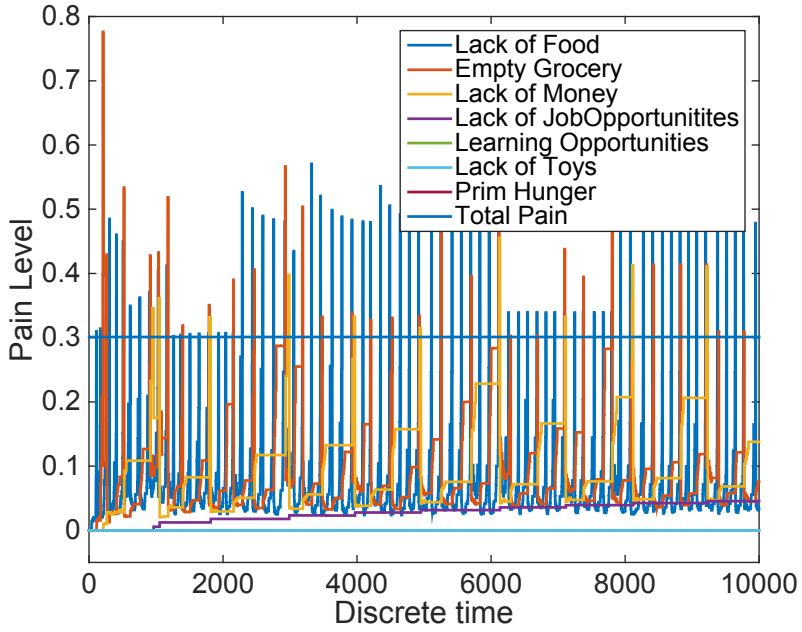


Figure 7-11. Pain plot from OML simulation.

Next, let us observe similar plots from a simplified MLECOG model simulation using the same environment setup. Note that the both simulation runs were executed with the same environment setup, in particular, the same resource ratios and motor time requirements. While

the w_{BP} results in Figure 7-10 and Figure 7-13 look similar, there are a few noticeable differences. The most noticeable differences are the order in which resources are learned, and the fact that the MLECOG algorithm actually learns the "School" resource, while the OML does not. This is likely a result of the attention switching functionality in the MLECOG algorithm, particularly in the semantic memory. Because saccading in the pseudo-semantic memory is currently implemented in a very basic way, it has a tendency to cycle through the objects found in the environment, often causing the agent to explore actions on objects more than it might in the OML implementation. In Figure 7-13, the decline of the School and Job Opportunities w_{BP} weights can also be observed after a certain amount of time. This is due to the somewhat reversed order in which they were learned. During the early stages of execution, curiosity was active, and since the agent had access to all the available resources, it was able to learn most (if not all) of the useful actions. Hence, once the algorithm learned everything properly, and curiosity stopped executing, the algorithm generally stopped using the aforementioned resources except when needed, allowing for a more uniform behavior. Since the job and school resources are not needed in early stages, this caused the gradual decline of their associated weights. Note that the Grocery and Food weights stay stable, and that the Money weight is gradually increasing. While w_{BP} results appear similar, this is not the case for the resource and pain plots in the other figures. For example, in Figure 7-12 we observe that the Food resource usually (although not always) fluctuates between 0.1 and 0.5, while in Figure 7-14 the associated pain generally doesn't go below 0.1 and in the latter half of the simulation gets as high as 0.5. This is in contrast to the OML simulation where in Figure 7-9 the Food resource level is generally fully restored and in Figure 7-11 the restored pain levels tend to be closer to zero. Much of the reason for these deviations between the two simulation approaches can be attributed to the way resource usage is estimated in the MLECOG model. It is somewhat burdened by having to correctly attribute changes in the environment to the appropriate actions, which can be difficult in cases where it has more than one action to evaluate in its evaluation queue or has to deal with multiple changes in the environment. (This is due to the fact that evaluation does not occur until the target resource is updated and hence there is time for other changes in the environment to occur.) This "issue" with resource usage estimation can be observed in Figure 7-12 with respect to the JobOpportunities resource, which ends up being restored to well over its desired state at around 20,000 iterations.

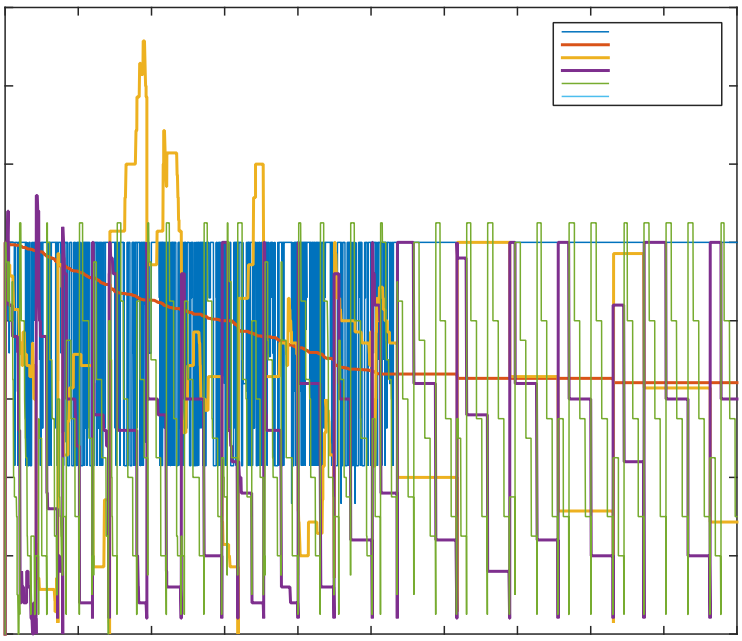


Figure 7

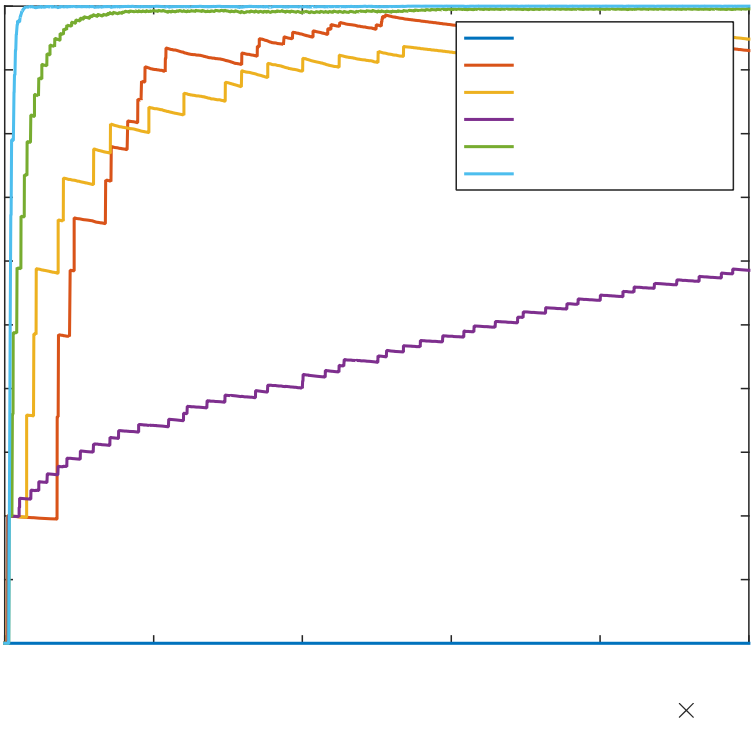


Figure 7-13. w_{BP} plot from MLECOG simulation.

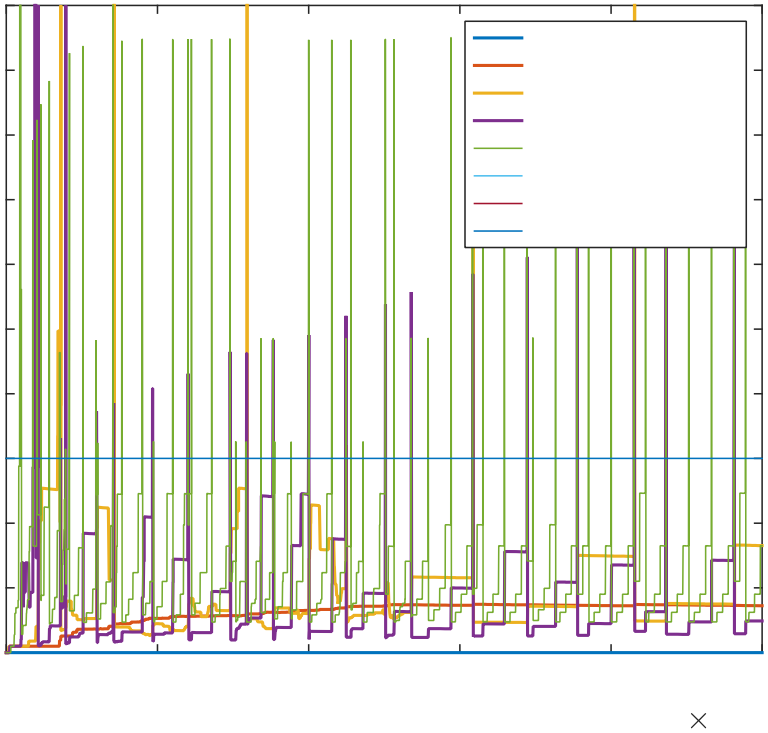


Figure 7-14. Pain plot from MLECOG simulation.

7.5 The "Full" MLECOG Model

In this section the organization of the individual blocks of the "full" MLECOG architecture is discussed. In section 7.2, the MLECOG model was discussed in fairly broad strokes in general terms regarding its operation, however, in this section, each block of the complete model is examined and a short discussion of the links between them and their mutual relationships is provided [24]. The main objective of this section is to provide an overview of the structural organization of the functional blocks rather than their detailed implementations.

The blocks in the model process their information concurrently, sending and receiving interrupt signals to redirect the focus of attention, respond to external threats and opportunities, change plans, and monitor actions. Memory blocks provide the agent with distinct memories, the accumulation of knowledge, skills, and desires, while other blocks facilitate the management of the agent's attention, memories, and needs. The complex interplay between the blocks makes the agent's satisfaction of its needs and the construction of new motivations a fully autonomous process.

The “full” cognitive model, a variation of which was initially described by Starzyk and Prasad in [100], includes memory structures, sensory and motor I/O and complex cognitive abilities. While this model is missing some functional areas (such as distinct short and long-term memories), it is more complete than our existing implementations. It is the current envisioned “destination” for our future work. The reason for this gradual approach is that it is extremely difficult to build any kind of complete cognitive model from the ground up due to the complex interplay and interdependences between the various modules.

This “full” model, shown in Figure 7-15, is followed by a brief, somewhat formalized, description of the model, and then by more comprehensive summaries of the different component blocks of the model, their proposed functions and how they link together. The reader should refer to figure while reading the explanations for various functional blocks and their interactions in order to aid with understanding how the blocks connect and work together. In the figure solid lines represent two-way connections between modules, while dashed lines represent one-way connections. There are 3 line colors represented in the image, black, blue, and green. Black lines represent connections from unconscious processes, e.g. processes that are not under conscious control, but can be affected by it. Blue lines represent conscious control signals and communications, while green lines represent a subset of conscious control; signals specifically associated with choosing and monitoring an action. A dash-dot line separates the cognitive and noncognitive areas. This separation is fuzzy and depends on the interactions between the top-down and bottom-up processing that is used to establish conscious perceptions of objects or ideas on which cognitive functions are based. For instance, conscious perception of an object may engage the association of many sensory inputs involved in object recognition. This may involve further manipulation and observation of the object, active search for characteristic features that either may confirm the expectation or may result in a different cognitive interpretation of the sensory input. Thus, which groups of activated neurons working in synchrony represent a conscious percept or a thought, may change during the observation of a single object.

Note, that Full Sequential Cognitive Model presented in Figure 7-15 represents an asynchronous architecture where information is processed concurrently in many areas, activated by sensory inputs, and sustained by internal associative memory processes. Only a single sequential process may resurface from these parallel associations and activations of different memory areas. It is a process of conscious observations, thoughts, emotions and feelings. All of them have their support in massively parallel processing, reflecting the state of the environment,

internal needs, accumulated knowledge, and past memories. These conscious observations are supported functionally by the attention switching and mental saccade mechanisms.

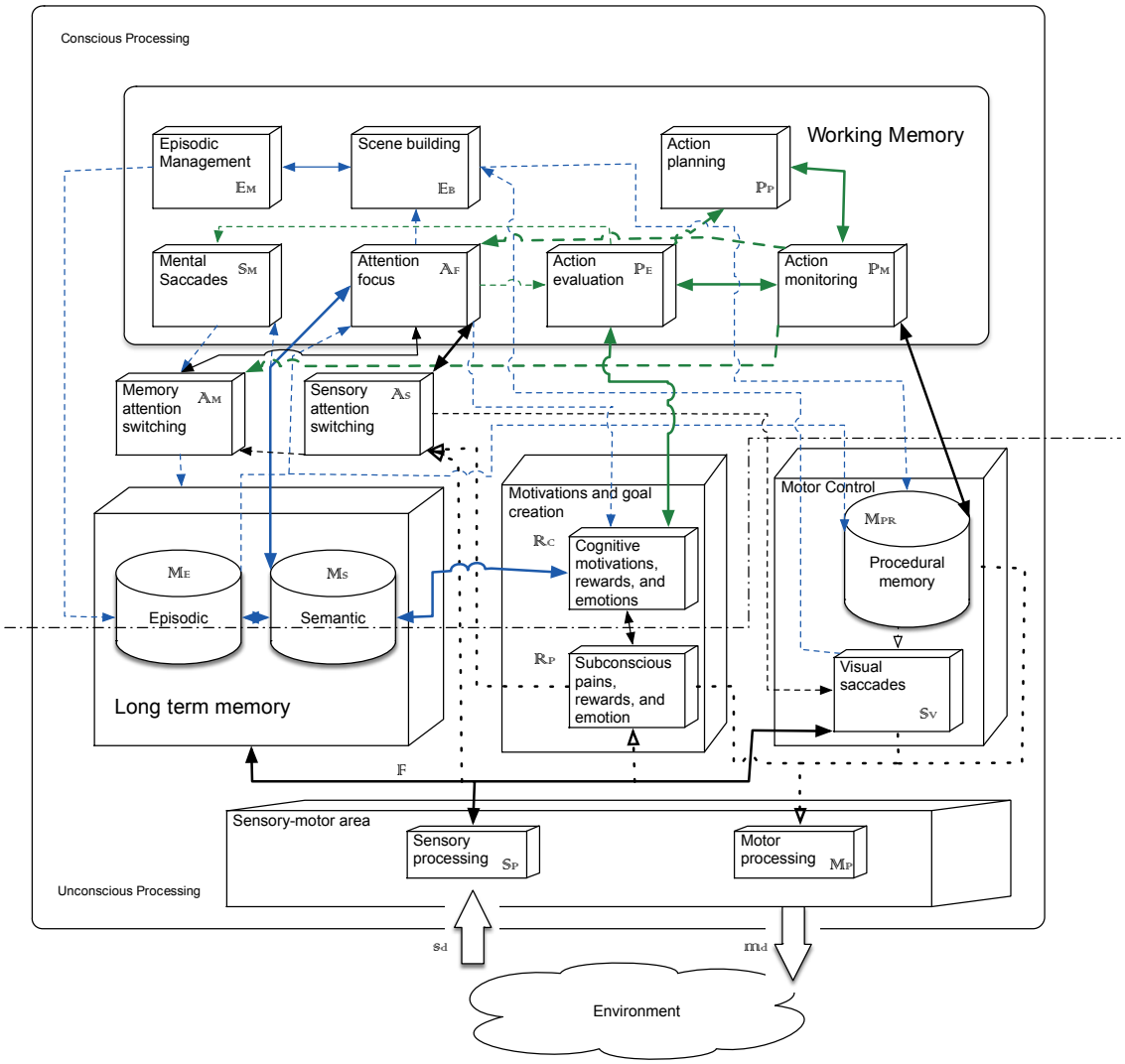


Figure 7-15. Full sequential cognitive model. (complex diagram)

Consciousness appears to have almost mystical, spiritual connotation, and is often speculated as being disassociated from a material mind. It is treated in this dissertation from the functional point of view, which reflects full agreement of observations with knowledge about the world. This agreement results from a coordinated effort of many processes reflecting the state of the memory that must resonate with each other to establish a single conceptual explanation of observations, thoughts, needs, and emotions. This agreement gives the organism the confidence

of knowing what it experiences. Any disagreement imagined or otherwise, leads to conscious illusions or in some instances to the lack of consciousness (as in epileptic seizures).

7.5.1 Formalized Model Description

Let us list the components and signals of the model in (approximate) order from bottom to top. The model can be described as consisting of the following structures:

- A body schema or embodiment that consists of the agent's set of Sensors $S = \{s_{d_1}, \dots, s_{d_n}\}$ and motor actuators $M = \{m_{d_1}, \dots, m_{d_n}\}$.
- A *Sensory processing*, $\mathbb{S}_P$, component that takes raw sensory data and preprocesses it (into useable feature information: $F = \{f_i, \cdots, f_k\}$) for the agent to use in memory, saccading, and attention switching modules. It works in conjunction with the memory to recognize objects and relevant features, which can then be used for attention switching and saccading.
- A *Motor processing*, $\mathbb{M}_P$, component that translates motor commands to provide signals $(\{m_{d_1}, \dots, m_{d_n}\})$ to the actuators.
- A *Subconscious pains, rewards, and emotions* block $\mathbb{R}_P$ that translates/generates the primitive needs/pains, and manages the agent's rewards and emotions: $R_P = \{r_{p1}, \cdots, r_{pn}\}$
- A *Cognitive motivations, rewards, and emotions* block $\mathbb{R}_C$, which generates a set of higher-level motivations:$R_C = \{r_{c1}, \cdots, r_{cn}\}$
- A *Semantic memory* module $\mathbb{M}_S$, which performs recognition tasks and retains semantic information about detected objects, $M_S = \{m_{s1}, \cdots, m_{sn}\}$. Each object m_{si} consists of a set of recognized features and linked associations to other related concepts and/or objects.
- An *Episodic memory* module $\mathbb{M}_E$ that retrains and times based memories of events relevant to the agent. $M_E = \{m_{e1}, \cdots, m_{en}\}$. Each episode m_{ei} consists of a set of sequentially listed scenes or events, $\{e_1, \cdots, e_n\}$.
- A *Procedural memory* module $\mathbb{M}_{PR}$ that learns and stores complex motor procedures for future use, such that $M_{PR} = \{m_{p1}, \cdots, m_{pn}\}$. Each motor procedure m_{pi} consists of a set of sequential primitive motor actuations, $\{m_{a1}, \cdots, m_{an}\}$.
- A *Visual saccades* mechanism $\mathbb{S}_V$, which receives control signals from procedural

memory to focus visual attention on a selected object or action as determined by the sensory attention switching block, such that $m_a = f(F, M_{PR}, A_S)$, where m_a is a motor actuation command for the motor processing block to direct vision at the target determined by the saccade function and associated control signals, F represents the feature information bus to/from Sensor processing and the Semantic Memory block, M_{PR} is control information from the Procedural memory block, and A_s represents control signal(s) from the Attention switching block.

- An *Action planning* module $\mathbb{P}_{\mathbb{P}}$ that is used to generate action plans and to cognitively check all the aspects of the planned action and its value to the system. It attempts to plan actions, $p_i = f(P_E, P_M, M_{PR})$, based on interplay with the associated action evaluation and planning modules and in conjunction with the procedural memory.
- An *Action evaluation* module $\mathbb{P}_{\mathbb{E}}$, which evaluates potential actions as the focus of attention switching, and can perform "dry runs" to determine the optimal solution for a particular situation. It evaluates actions based on a function of the attention focus, motivation state, and completed or simulated action information from action monitoring, or $p_e = f(a_f, R_C, p_{m_{sim}}, p_{m_d})$ where a_f is the current attention focus, R_C is data from Cognitive motivations, $p_{m_{sim}}$ is simulated action monitoring information, and p_{m_d} is current "real-time" information from the Action monitoring module.
- An *Action monitoring* module $\mathbb{P}_{\mathbb{M}}$, which monitors the progress of actions to determine how they are progressing, when they are done, and when they can be evaluated. The function $p_m = (p_i, p_e, m_p)$ works by taking information from the action evaluation and action planning modules in conjunction with procedural memory to monitor action progress, where p_i is the output from the action planning module, p_e is the output from the Action evaluation module, and m_p is feedback from the Procedural memory. (Note that actual progress information is transmitted and hence translated through several modules, including the sensory processing and semantic memory modules.)
- A *Scene building* module $\mathbb{E}_{\mathbb{B}}$ that constructs scene representations. A scene e_i, consists of relevant information at the time it was built, such as semantic links to objects present in the scene, and information regarding the states of the environment and the agent at the time (like object locations and quantities) such that $e_i \approx \{\{m_{sj}, \cdots m_{sk}\}, \{f_j, \cdots f_k\}, \{r_{pj}, \cdots r_{ck}\}\}$.

- An *Episodic management* module $\mathbb{E}_{\mathbb{M}}$, which initiates scene building and the formation of episodes for episodic memory. It combines a series of scenes into an episode $m_{ei} = \{e_1, \cdots, e_n\}$, and determines when an episode ends or begins.
- A *Mental saccades* block $\mathbb{S}_{\mathbb{M}}$ that a major search mechanism for thoughts and plans based on the primed signal values in the semantic memory, such that $s_m = f(M_S, p_e)$, where s_m is a command to the semantic memory to switch attention in the memory to a chosen concept as determined by input from the semantic memory itself and action evaluation.
- An *Attention focus* module $\mathbb{A}_{\mathbb{F}}$, which highlights a selected concept in the semantic memory bringing it into a short term working memory for further processing, via $a_f = f(\{m_{sj}, \cdots, m_{sk}\}, a_s, a_m)$, where a_s and a_m are information from the Memory and Sensor attention switching blocks. It works by selecting a target from a set of recent memory saccades in conjunction with input from the memory and sensory attention switching modules.
- A *Sensory attention switching* block $\mathbb{A}_{\mathbb{S}}$ that is used to switch visual attention to different features or objects in the sensory space. A sensor target a_S is selected based on $a_S = f(F, R_P, A_F)$, which evaluates information from the input space, subconscious motivations and pains, and the attention focus to select a visual target.
- A *Memory attention switching* block $\mathbb{A}_{\mathbb{M}}$, which responds to the mental saccades mechanism to process thoughts, plans, and associations based on semantic and episodic memory priming. A memory attention target is selected via $a_m = f(p_m, s_m, a_f, a_s)$.

7.5.2 Component Descriptions

Introduced in Section 7.5.1, the structural components of the sequential cognitive model can be described functionally as follows.

The *Sensory processing* block, S_p, receives information from the environment through sensory inputs, s_d, processes it and sends the result to long-term memory where it can invoke a cognitive response. While this is its main function, it also triggers primitive pain signals that regulate the agent's motivations, emotions or other noncognitive signals that are used in the Sensory attention switching block to trigger attention focus on the specific sensory input. The Sensory processing block also receives feedback signals from the long term memory to help identify the object of attention. Visual saccades help to focus on an element of the scene, but it is

the attention focus that semantic memory receives that helps to identify the object. Object identification is performed through forward-backward interactions between Semantic memory and Sensory processing similar to the general concept of adaptive resonance theory developed by Carpenter and Grossberg [128]. Other feedback signals come from the Visual saccades block that changes the focal point in the visual input. Implementation of the actual processing part of this block can vary, but its functionality is similar to invariant feature extraction and recognition.

The *Motor processing* block, $\mathbb{M}_P$, sends actuator control signals to actuators responsible for motor actions. The major input it receives comes from the Procedural memory, which controls execution of sequences of operations and also controls specific actuators in a single motor action. The input from the Visual saccades block is used to move the visual focus on the selected target. An input it receives from the Subconscious pains block triggers an involuntary reaction to remove the source of pain and to show emotions (for instance by changing facial expressions). Sensory-motor processing is performed at the bottom of the memory hierarchy.

The *Subconscious pains, rewards, and emotions* block, $\mathbb{R}_P$, receives inputs from Sensory processing and sends its outputs to Motor processing. In addition, this block sends signals to the Sensory attention switching block that switches attention to the pain source, and to the Cognitive motivations, rewards and emotions block to influence abstract goal creation, cognitive motivations, and emotions according to the theory of motivated learning [97], [100]. It receives emotional feedback from the Cognitive motivations block responding both to painful and pleasant events by triggering motor responses of fear, frustration, disgust, anger, happiness, sadness and surprise depending on the context. It also affects the significance estimation that the Cognitive motivations block assigns to Scene building.

Emotional responses build up as a result of subconscious pains and rewards without cognitive explanation or control over the emotional level. Emotions have strong influence on the cognitive process and motor activities, thereby affecting an agent's behavior. A machine will never have human emotions due to different morphology, but its emotions may have a similar meaning to human emotions. In Figure 7-15 pains and emotions are combined in the same block for convenience of description, since both are subconsciously generated and both affect system motivations and reactive motor response.

The *Cognitive motivations, rewards, and emotions* block, $\mathbb{R}_C$, receives primitive pain signals from the Subconscious pains, rewards and emotions block. Based on changes in these signals, it creates higher-level goals and sends pain information to the Action evaluation block when such information is requested. Thus, useful actions can be learned and new motivations

established, while harmful actions can be avoided. Although, the major feedback this block receives from the Action evaluation blocks result in conscious evaluation of opportunities with regard to internal needs, the level of subconscious response to a planned action may also be affected by response from subconscious pains and emotions. This may result in emotionally driven decisions without clear understanding or explanation for such decisions. The Cognitive motivations, rewards, and emotions block interacts with emotional states of the agent, by providing cognitive feedback to regulate emotional responses. Another feedback connection to this block is from the Attention focus block. If the system attention is focused on a cognitive motivation, it may trigger Action evaluation and planning. The bidirectional link to Semantic memory provides a priming signal in the semantic memory area and a specific motivation can be selected through a competitive process of mental saccades.

Semantic memory, $\mathbb{M}_S$, supports attention focus, the attention switching mechanism, formation of emerging concepts and their refinement, acquisition and storage of knowledge, novelty detection, fine-tuning of concept representations, building associations between concepts and events, goal creation, reward assessment and reinforcement in learning, as well as response prediction and planning. In addition, semantic memory aids the process of object recognition, scene building for episodic memory, cuing of past episodes, action evaluation, and action monitoring. The Semantic memory block receives its inputs from Sensory processing and learns invariant representations of the observed objects using incremental Hebbian learning. The knowledge contained in the semantic memory, activated by associations to goals and observations, and selected by the Attention focus, supports all cognitive operations.

Episodic memory, $\mathbb{M}_E$, receives scene information from the Episodic management block and writes it to long term sequential memories with significance information used to determine the duration and importance of each episode. A link from the Attention focus is used to recall episodes based on association with the focus of attention. A bidirectional link to Semantic memory is used for episodic recall and semantic memory learning. Finally, a link to Procedural memory is used to learn procedures based on recalling the past episodes where a useful action was observed.

Procedural memory, $\mathbb{M}_{PR}$, is a major block of memory responsible for performing learned actions. In this block, sequential memory cells are responsible for learning and recalling the learned sequences of operations. Initially, learned sequences involve more cognitive attention so they are supported by semantic memory, but as the system frequently performs learned operations, cognitive supervision is less intensive. Thus, we consider this memory to be a

subconscious one. Procedures can be learned directly from observations using mirror neurons and Scene building or by recalling these observations from Episodic memory. Procedural memory has bidirectional links to action monitoring. Spatial orientation is implemented as a subset of procedural memory. It receives spatial information from Scene building and uses it to create maps of familiar places. These maps are used by the agent to navigate in its everyday activities. Finally, two unidirectional links out of Procedural memory control Motor processing to execute an action, and activate Visual saccades to observe the action's result.

The *Visual saccades*, $\mathbb{S}_V$, mechanism receives control signals from procedural memory to focus visual attention on a selected object or action. This includes searches of the environment in a special procedure that is responsible for scene building. The link from the Visual saccades block to Scene building determines spatial locations of objects in the scene. Another input to Visual saccades is from Sensory attention switching to help determine the source of the interrupt signal. Outputs from Visual saccades are to Motor processing, in order to implement saccadic movement, or to the Sensory processing area to simulate such movement if an electronic equivalent of the saccadic movement is implemented. The electronic equivalent of the saccade movement is obtained when instead of saccading to a new part of the observed scene an input image is scanned in a saccading fashion to select the next focus of visual attention in the observed image.

Action planning, $\mathbb{P}_P$, is used in the planning stage to cognitively check all the aspects of the planned action and its value to the system. It initiates planned action monitoring, checking the environment conditions and its ability to perform the planned action. Thus, action planning relies on the information stored in the procedural memory to determine the sequence of operations needed to complete the planned action. Once the action plan is completed, action can be initiated by the Action evaluation block, and its status changes from a planned action to an initiated action. If the plan cannot be completed, the planned action is abandoned and is removed from the Action planning block.

The *Action evaluation*, $\mathbb{P}_E$, block responds to an opportunity for action when the Attention focus selects a new object or an idea is considered useful by the Cognitive motivation block. Action evaluation uses a bidirectional link to Cognitive motivations in order to check if the evaluated action is beneficial to the agent. If it is, the Action evaluation block activates Action planning to initiate a plan by setting an active link to the newly planned action. Action planning proceeds to plan the action according to a script in the Procedural memory. It uses Action monitoring to perform a "dry run" of the planned actions by checking conditions for action

implementation and evaluating consequences. If the “dry run” is positive Action evaluation begins a new action by initiating a real action monitoring process. Once the action is completed or abandoned the process is deactivated (removed from the working memory). It also removes a planned action process once it is determined that the action would be harmful or cannot take place due to unfavorable environment conditions. However, if the evaluated action was not beneficial, Action evaluation triggers the next mental saccade and the system switches the attention focus to the next concept.

The *Action monitoring*, $\mathbb{P}_M$, block has a bidirectional link to Procedural memory to monitor the progress of sequential actions stored in the Procedural memory. It also uses unidirectional links to the Attention focus block in order to focus attention and monitor each step of the procedure in situations where a performed action is not a routine one and requires a cognitive effort. Another unidirectional link to Memory attention switching, forces the focus of attention to switch to the planned action. Signals from Action monitoring to Memory attention switching have a higher strength than signals from mental saccades, but lower than those from Sensory attention switching. This helps to focus on the performed activities, while being alert to significant, observable changes in the external environment. The Action monitoring block also has bidirectional links to the Action planning and Action evaluation blocks. Incoming links initiate the process of action monitoring (real or planned), while outgoing links inform Action evaluation and Action planning about the progress made.

Scene building, $\mathbb{E}_B$, is important for learning, episodic memory formation, active searches, spatial orientation and map building. It includes cognitive recognition of the scene elements and their locations, activities performed, significance of the activities and the entire scene, time markers and other relative scene information. It receives unidirectional links from Attention focus and Visual saccades to obtain elements of the observed scene, their significance and location. It uses visual saccade control signals to determine object locations, and it evaluates the cognitive meaning of the object that is in the attention focus and its significance obtained through association of the selected object with cognitive motivations. Such information is then passed through Attention focus block to Scene building. Scene building is regulated by Episodic management, which replaces one scene with the next one, dynamically updating the content of the episodic memory. However, the importance of this block goes beyond support for episodic memory. Scene building is a key ingredient for spatial orientation and building a map of the observed terrain, providing links to the spatial orientation block within Procedural memory.

Episodic management, $\mathbb{E}_M$, is a part of the working memory responsible for initiating scene building and the formation of episodes for episodic memory. Episodic management writes a scene to Episodic memory when it detects a significant change in the current scene (a new event or shift in space). It also collects significance information about scene elements. Obtained scenes are arranged in episodes and are written to Episodic memory together with their significance information. Once a scene is completed and is written to Episodic memory, a new scene is initiated by Episodic management. Not all episodes are worth writing to Episodic memory, so a scene can be forgotten even before it is sent to episodic memory if its significance is close to zero and is replaced by a new scene (which may happen most often). Episodic management performs real time organization and storage of episodes. The creation of episodic memory, itself, can be read about in more detail in [127].

The *Mental saccades*, $\mathbb{S}_V$, block provides a major search mechanism for thoughts and plans based on the primed signal values in the semantic memory. It receives inputs from all signals representing primed neurons (neurons that are partially activated) in the semantic memory and subjects them to winner takes all competition. Indirectly, through activation of neurons in the semantic memory, this block is also stimulated by recalled episodes from Episodic memory and Cognitive motivations. These associative activations of semantic memory neurons are obtained through links from Episodic memory and Cognitive motivations, rewards, and emotions blocks indicated in Figure 7-15 through thick bidirectional connections. A mental saccade responds to all activated neurons in the semantic memory, including these that represent past episodes and cognitive motivations. Once the winner of competition between primed neurons is declared, the Mental saccade block sends results to the Memory attention switching block and a feedback signal from Memory attention switching is sent back to Attention focus, activating selected area of the semantic memory. Notice, that many signals from the semantic memory compete for attention together with interrupt signals from the Sensory attention switching block, where the interrupt signals have priority over other competing signals.

The *Sensory attention switching*, $\mathbb{A}_S$, block receives inputs from the sensory processing area and sends the interrupt signals to the Attention focus. In addition, it is a source of information for the visual saccades, determining the signal strength based on the color, size, motion, and other salient sensory features. It also receives signals from subconscious pains, rewards and emotions to alert the system about pains and to help focus its attention on the pain's source. Emotions change response thresholds of Sensory attention switching, resulting in a faster or slower response. Finally, it cooperates with Attention focus for object recognition and

cognitive identification. It sends a blocking signal to Memory attention switching to prevent it from selecting a cognitive focus of attention.

The *Memory attention switching*, $\mathbb{A}_M$, block responds to the mental saccades mechanism in order to process thoughts, plans, and associations based on semantic and episodic memory priming. It receives interrupt signals from Sensory attention switching and Action monitoring. Since Sensory attention switching takes precedence over Cognitive mental saccades or Action monitoring, its activation blocks Memory attention switching from selecting a cognitive winner – instead the focus of attention is shifted towards a dominating subconscious event. In a similar way, Action monitoring interrupt signal wins over a competition from semantic memory neurons.

The *Attention focus*, $\mathbb{A}_F$, block highlights a selected concept in the semantic memory, effectively bringing it into a short term working memory and it is responsible for all cognitive aspects of the system's operation. It helps to provide cognitive interpretation to all observations, actions, plans, thoughts, and memories. Its function is often associated with the similar working memory mechanism in humans and higher order animals. Attention focus has unidirectional links to Cognitive motivations and Episodic memory and a bidirectional link to Semantic memory. This is to indicate that all episodes or motivations in attention focus must be semantically interpreted by Semantic memory.

The Attention focus block receives inputs from Action monitoring as well as bidirectional links to Sensory and Memory attention switching. It also sends triggering signals to the Action evaluation block. These signals trigger a response from Cognitive motivations requested by the Action evaluation block to consider potential actions on the observed object. The main role of the Attention focus in action evaluation is to obtain information about the value of the planned action with respect to cognitive motivations. A direct link from Attention focus to Action evaluation helps to assess the feasibility of the planned action, while an indirect link through Cognitive motivations provides a value based assessment of the planned action. The link from Action monitoring provides the attention focus on the currently performed activities or those planned for the future, in order to complete or abort an action. The bidirectional links from Sensory attention switching, help to focus the system's attention on the source of the interrupt, and help to cognitively recognize this source. Finally, a bidirectional link from Memory attention switching focuses attention on a target object resulting from the mental saccades, or a planned action.

A change in the attention focus may be the result of either a subconscious interrupt from Sensory attention switching or from Memory attention switching. A memory based attention switch may come from cognitive action monitoring – what to do next – or from WTA

competition between semantic memory areas primed by episodes, concepts and motivations. Competition between primed areas of the semantic memory is the weakest source of attention switching but it is executed most often. Although, in Figure 7-15, the sensory and memory sources of attention switching are separated, they all compete for the agent's attention. Therefore, proper mechanisms must be established to adjust Sensory attention switching thresholds, as well as thresholds for inputs from Action monitoring to give them priority over semantic memory attention switching. Only when none of these inputs exceeds their corresponding thresholds does a winner of the WTA competition in the semantic memory end up in the system attention focus.

The organization of the MLECOG architecture presented in this section only indicates the dominant functionalities of the discussed functional blocks. Detailed discussion of algorithms that implement memory, saccades, motivations, needs, goals, grounded perception, motor control, etc. are discussed in other parts of this dissertation or in the provided references. Motivations, needs, and goals were discussed in Chapter 2-4. Section 7.3 provided some detail on the potential implementation of the semantic memory and how mental saccades and the attention switching process operate. For more detail on memory implementation and grounding, see the following references [26], [69], [76], [78], [102], [129]–[131]. The MLECOG architecture implementation currently contains the building blocks of the full architecture, such as the motivation, saccading, planning, and basic memory structures (see Sections 7.2 and 7.3). However, additional work needs to be done in regards to implementing the semantic and episodic memories. Progress is being made, such as with the Horzyk's ANAKG network [132] and our attempt to combine it with LTMs. But to fully implement the model we will need to also integrate the new semantic/episodic components and tie them into its functionality. This is definitely worth doing, since to the best of our knowledge nobody has a working model of semantic memory that satisfies the requirements of our MLECOG model.

7.6 Conclusion

This chapter has introduced and discussed the structural organization of the MLECOG architecture presented in this dissertation as an extension of the motivated learning paradigm. The makeup as well as the reasons behind and the basic functionality of the various components of the architecture have also been discussed. Particular attention was paid to new elements, such as the semantic and episodic memories, as well as the new attention switching and saccade mechanisms in the model. These additional modules significantly extend and enhance the capability of basic ML covered in the preceding chapters by providing the infrastructure to implement capabilities

such as planning multi-step actions, developing complex associations and adjusting the amount of processing needed to keep track of the changes in the environment in real time. The agent's planning process is dynamically controlled and its execution is managed by the cognitive functional blocks, which integrate needs and the means to accomplish them with the observed situation in the environment.

Basic simulations of the MLECOG model in both Matlab and our NeoAxis environment were also presented. Results are similar to those obtained in early ML implementations and show that MLECOG can produce, at a minimum, the same type of behavior that results in successful operation of the agent in dynamic environments. While there remains a great deal of work in terms of implementing the full MLECOG model, the current implementation both proves the viability of the approach and provides a useful stepping stone for further development. Unfortunately, it is difficult to prove the usefulness of the MLECOG model in general without the full implementation of mental saccades, associative and procedural memory, and attention switching. Such a comparison should include estimates of work efficiency in real time applications in dynamic environments using limited computational resources and adverse, often hostile environments.

CHAPTER 8: DISSERTATION CONCLUSION

"Motivated Learning" and the Motivated Learning Embodied COGnitive (MLECOG) model presented in this dissertation attempt to take a step closer toward the realization of a cognitive intelligence by creating motivations in agents and presenting an embodied cognitive model based around the ML concept. Such an agent can develop its own drives and motivations without relying extensively on designer or other external directives or rewards; it can deal with the "reality gap" by ensuring there is a form of symbol grounding; and it can use attention switching and saccading to deal with large amounts of information that might overload the agent. The described agent model uses episodic and semantic memory to store "knowledge" (data and associations). It also uses attention switching for focusing itself on specifics of the environment and switching between cognitive tasks and associations in its memory, both of which have not previously been done in conjunction with motivated learning. To the best of our knowledge, the "mental saccades" concept is a new one, not yet utilized in other approaches to cognitive agents.

The main goal of this work has been to investigate the development of cognitive agents, specifically a Motivated Learning based agent. The end goal is to develop the Motivated Learning Embodied Cognition (MLECOG) model, and the associated components it requires. Over the course of this research a great deal of effort has been spent on the development of motivated learning itself, followed by the initial implementation and/or discussion of several other components necessary for the MLECOG model. These components include various types of memory, action monitoring, visual and mental saccades, focus of attention, attention switching, planning, etc. Additionally, some elements needed for processing sensory data were examined because they are relevant to the eventual creation of a full cognitive model with proper sensory/motor I/O (see Other Work in Chapter 8). Areas examined include (in order of low-level to high-level implementation) low-level learning in the form of a modified growing neural gas (GNG) network, followed by a slightly higher level approach in the form of hierarchical "deep learning".

Several primary goals for this work were presented in Chapter 1. The first goal was the creation and implementation of a Motivated Learning scheme, along with the creation of a flexible environment and the ability of the ML agent to function in that environment. This goal has been fully realized as evidenced by the results described in Chapters 3 and 4 and the results presented in Chapter 5. Furthermore, the subgoals regarding the creation of a flexible environment and the agents ability to operate in the environment were also met, both in terms of an environmental framework created in Matlab (responsible for most of the quantitative testing in

this work) and in NeoAxis (responsible for the visual environment and interactive, real time elements of the testing).

The second goal was the design and implementation of a Motivated Learning cognitive model. The goal consisted of determining the necessary components of the model, designing and implementing them as much as possible, following by testing within the previously developed environment framework. While the design of the model was completed (see Chapter 7), some elements were not fully implemented, as was accounted for in the initial expectations for this work. A fully functional cognitive model is a large undertaking, hence the more reachable target of implementing a reduced version of the model (see Figure 7-6). Section 7.4 presented testing results for this simplified model from both Matlab and NeoAxis implementations.

The final goal was to provide quantitative comparisons between the developed ML model and various reinforcement learning algorithms. The reason for these comparisons is that Motivated Learning is similar in many respects to Reinforcement Learning and showing that ML is able to outperform RL algorithms provides weight to the claims made in regards to ML's capabilities. Some initial work had been done earlier in the progression of this research in the form of the comparison to TD-Falcon in Chapter 6.1. This idea was expanded upon with the creation of a simple "Black Box" platform from which other algorithms could be tested. The black box was designed to be simple and as compatible with other RL algorithms as possible. This allowed the testing of seven other RL algorithms (see Chapter 6.2) against ML, with hopefully more tests done in the future via the web page mentioned at the end of Chapter 6. None of the tested RL algorithms (from simple to advanced ones) could perform equally as well as ML.

While it was shown in Chapter 6 that Motivated Learning can outperform RL in certain situations, reinforcement learning is still useful as a tool for optimizing individual tasks. It seems prudent to allow reinforcement learning to direct the learning process behind a single task, but allow the Motivated Learning model to deal with the creation and selection of tasks as described in this dissertation. In many respects, this is what we have already done, except that the ML algorithms shown here are using only the most basic learning mechanisms in the form of neural network inspired learning mechanisms.

To recap, this research has provided several contributions in the form of the development of the Motivated Learning approach and the associated MLECOG model. The ML approach allows agents to develop on their own and more effectively survive in unknown environments by learning the associations within the environment and developing a motivational hierarchy, and there also remains ample room for additional work. For example, as mentioned in Chapter 7, the

current simplified implementation of the MLECOG model lacks several of the modules seen in the full version of the model, such as episodic and procedural memories, and its semantic memory implementation is only a placeholder for the real thing. The inclusion of an actual semantic memory, following by several of the other missing components, would greatly advance the model, and presumably our understanding of artificial cognitive systems. However, the inclusion of these components was both outside the scope of this work and outside of the available time limitations. That said, work is ongoing in providing components such as the semantic memory for use in the MLECOG model. We have associates in Poland that have been working on a semantic memory capable of processing visual input, and there is the work mentioned in Section 7.2 regarding the ANAKG associative memory [132] that might be useable in conjunction with LTMs [122] to form a type of semantic/episodic memory structure [124].

One thing that has not been discussed yet is the social implications of developing an intelligent agent, and the associated ethical quandaries. At some point in the future, what happens when we develop an agent that is capable of thought, feelings and self-determination? How will such a being interact with and function in our world and specifically, what might be its relation to human beings? Have we actually considered all of the major functional components our agent will need to act as a trusted member of our society? While it is likely that we will not have to be concerned with answering these questions for quite some time, we will increasingly need to consider them as we move closer to our goal of building cognitive agents.

In the ML agent, curiosity was handled as a lack of knowledge pertaining to the goals/actions available to the agent. The level of curiosity was represented via internal weights associated with actions. However, in real-life (e.g. in humans and other animals), curiosity is more complex than that. While it can be thought of as a low-level motivation, as implemented in this work, full treatment of curiosity requires more sophistication. Think about how one becomes curious about something. Curiosity usually becomes evident when an individual is faced with something unexpected or unknown and wishes to know more. Implementing this type of curiosity would require that we have a functioning semantic memory, etc. This is because the agent would have to be capable of "expectation," that it would have to be able to predict or know what it should be seeing in order to know whether it is something to be curious about. At that point, we could look at integrating curiosity as part of the attention switching/focus mechanism, where the agent could evaluate whether pursuing its curiosity about something is worthwhile.

Now, let us consider one of the more important ethical questions, the question of how do we keep an agent from harming humans? One may consider Asimov's three laws of robotics

[133], and the first law, "A robot may not injure a human being or, through inaction, allow a human being to come to harm," in particular. However, hard coding robots with the three laws is not actually feasible, since it would require significant pre-knowledge of what a human actually is, among many other things. This would be difficult to do.

That said, while it may not be possible to outright bar robots from harming humans, there are several ways to mitigate the issue. One method to curtail potential hostilities would be to implement primitive needs in such a way as to suppress activities that humans would consider undesirable. The problem with this approach, is that the agent could conceivably override its own primitive needs by creating abstract needs of higher priority, particularly if the primitive needs aren't directly required for its survival (as something like maintaining its power supply would be). In many respects, developmental robots would need to be treated as children. They would learn and develop as human children do, and eventually function as "adults". However, as with humans, not all adults always follow the social norms.

Another way is simply to teach them to distinguish right and wrong and utilizing social pressure. However, what would happen when the agent knows that something is societally wrong, yet it motivated to do it to satisfy its own needs? Depending on the level of the agent's development, we could restrict its freedom, giving us control over its movements; we could (re)train it to change its motivations; or we could destroy it. Conscious agents will presumably value their own existence and understand the implications of going against "societal norms".

Yet another thing to consider is our treatment of our creations. Will they be slaves, pets, or something else? Many writers would point out the obvious issues with using them without regard to their own wills. However, if they are only as intelligent as a simple animal it likely would not be much of an issue. Presumably, they would either enjoy what they were built for, since it would be "coded" into their very construction. On the other hand, if their development requires a learning process similar to our own (humans), with a childhood and all that it entails who knows how they might develop.

Ultimately, these are all considerations for the future, since it will be some time before we advance to the point of truly cognitive agents. We have to start at the base and work our way up. It is doubtful that the first cognitive agents will be much more complex than "simple" animals, such as crows or squirrels. While the aforementioned species display some known cognitive traits, they are still very limited.

It is likely that initial cognitive agents, for all their complexity will not be capable of much. After all they will not have the benefits (or hindrances) provided by millions of years of

DNA based pre-programmed evolution. Simply look to the various known behaviors of most animals and what they are capable of immediately after birth. Even humans, while highly dependent at birth, evidence certain common behaviors (such as motor babbling). While we can provide some level of structural organization of artificial brain or acceleration in the learning process by trying to implement some level of pre-encoded information, it will still be mostly trial and error and time consuming process.

That said, the benefits of this work should be clear. Even if it does not immediately result in actual cognitive intelligence, we will develop a greater understanding of how intelligence works as well as increase the capabilities of autonomous robots.

REFERENCES

[1] J. Mccarthy, M. L. Minsky, N. Rochester, and C. E. Shannon, "A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence," *AI Magazine*, vol. 27, no. 4, pp. 12–14, 2006.

[2] R. Pfeifer and J. C. Bongard, *How the Body Shapes the Way We Think: A New View of Intelligence (Bradford Books)*. The MIT Press, 2006.

[3] R. Sun, "The CLARION cognitive architecture: Extending cognitive modeling to social simulation," in *Cognition and MultiAgent Interaction*, 2006, pp. 79–99.

[4] R. Sun, "The importance of cognitive architectures: an analysis based on CLARION," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 19, no. 2. pp. 159–193, 2007.

[5] P. Langley, "An adaptive architecture for physical agents," in *Proceedings - 2005 IEEE/WIC/ACM InternationalConference on Web Intelligence, WI 2005*, 2005, vol. 2005, pp. 18–25.

[6] P. Langley and D. Choi, "A unified cognitive architecture for physical agents," in *Proceedings Of The National Conference On Artificial Intelligence*, 2006, vol. 21, no. 2, p. 1469.

[7] U. Ramamurthy, B. J. Baars, S. K. D'Mello, and S. Franklin, "LIDA: A Working Model of Cognition," in *7th International Conference on Cognitive Modeling*, 2006.

[8] B. J. Baars and S. Franklin, "Consciousness is Computational: The LIDA Model of Global Workspace Theory," *International Journal of Machine Consciousness*, vol. 01, no. 01. pp. 23–32, 2009.

[9] J. Laird, "SOAR: An architecture for general intelligence," *Artificial Intelligence*, vol. 33, no. 1. pp. 1–64, 1987.

[10] J. E. Laird, "Extending the Soar cognitive architecture," *Proc. 2008 Conf. Artif. Gen. Intell. 2008 Proc. First AGI Conf.*, vol. 171, pp. 224–235, 2008.

[11] J. Weng, "Developmental Robotics: Theory and Experiments," *International Journal of Humanoid Robotics*, vol. 01, no. 02. pp. 199–236, 2004.

[12] J. W. J. Weng, "A theory for mentally developing robots," *Proc. 2nd Int. Conf. Dev.*

Learn. ICDL 2002, 2002.

[13] M. Conforth and Y. Meng, "CHARISMA: A Context Hierarchy-based cognitive architecture for self-motivated social agents," in *Proceedings of the International Joint Conference on Neural Networks*, 2011, pp. 1894–1901.

[14] M. Conforth and Y. Meng, "Self-reorganizing knowledge representation for autonomous learning in social agents," in *Proceedings of the International Joint Conference on Neural Networks*, 2011, pp. 1880–1887.

[15] A. Baranes and P. Y. Oudeyer, "Active learning of inverse models with intrinsically motivated goal exploration in robots," *Rob. Auton. Syst.*, vol. 61, no. 1, pp. 49–73, 2013.

[16] "NeoAxis 3D Engine." [Online]. Available: http://www.neoaxis.com/. [Accessed: 05-May-2015].

[17] W. Singer, "The Brain, a Complex Self-organizing System," *European Review*, vol. 17, no. 02. p. 321, 2009.

[18] J. Graham and J. a. Starzyk, "A hybrid self-organizing Neural Gas based network," *2008 IEEE Int. Jt. Conf. Neural Networks (IEEE World Congr. Comput. Intell.*, 2008.

[19] J. A. Starzyk, "Motivation in embodied intelligence," in *Frontiers in Robotics, Automation and Control*, 2008, pp. 83–110.

[20] J. A. Starzyk, J. T. Graham, P. Raif, and A.-H. Tan, "Motivated learning for the development of autonomous systems," *Cogn. Syst. Res.*, vol. 14, no. 1, pp. 10–25, 2012.

[21] J. Graham, J. Starzyk, and D. Jachyra, "Opportunistic Motivated Learning Agents," in *11th Int. Conf. on Artificial Intelligence and Soft Computing*, 2012.

[22] J. Graham and D. Jachyra, "Motivated learning in computational models of consciousness," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 7403 LNCS, pp. 365–376, 2012.

[23] R. Pfeifer and C. Scheier, "The Study of Intelligence-Foundations and Issues," in *Understanding Intelligence*, 2001, pp. 1–33.

[24] J. A. Starzyk and J. T. Graham, "MLECOG – Motivated Learning Embodied Cognitive Architecture," *IEEE Syst. J. Spec. Issue Human-Like Intell. Robot.*, vol. PP, no. 99, pp. 1–12, 2015.

[25] D. Vernon, G. Metta, and G. Sandini, “A survey of artificial cognitive systems: Implications for the autonomous development of mental capabilities in computational agents,” *IEEE Trans. Evol. Comput.*, vol. 11, no. 2, pp. 151–180, 2007.

[26] F. Adams, “Embodied cognition,” *Phenomenol. Cogn. Sci.*, vol. 9, no. 4, pp. 619–628, 2010.

[27] K. Aizawa, “Understanding the Embodiment of Perception,” *J. Philos.*, vol. 104, no. 1, pp. 5–25, 2007.

[28] R. A. Brooks, “Intelligence without representation,” *Artificial Intelligence*, vol. 47, no. 1–3. pp. 139–159, 1991.

[29] R. A. Wilson, “Extended Vision,” in *Perception, Action and Consciousness*, N. Gangopadhyay, M. Madary, and F. Spicer, Eds. New York: Oxford University Press, 2010.

[30] A. Clark, “Finding the Mind: Book Symposium on Supersizing the Mind: Embodiment, Action, and Cognitive Extension (Oxford University Press, NY, 2008),” *Philos. Stud.*, vol. 152, no. 3, pp. 447–461, 2011.

[31] M. Beetz, L. Mösenlechner, and M. Tenorth, “CRAM - A Cognitive Robot Abstract Machine for everyday manipulation in human environments,” in *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings*, 2010, pp. 1012–1017.

[32] C. Balkenius, J. Morén, B. Johansson, and M. Johnsson, “Ikaros: Building cognitive models for robots,” *Adv. Eng. Informatics*, vol. 24, no. 1, pp. 40–48, 2010.

[33] R. O’Reilly, T. Hazy, and S. Herd, “The leabra cognitive architecture: how to play 20 principles with nature and win!,” *Oxford Handb. Cogn. Sci.*, pp. 1–31, 2012.

[34] J. Laird, K. Kinkade, S. Mohan, and J. Xu, “Cognitive Robotics Using the Soar Cognitive Architecture,” in *8th International Workshop on Cognitive Robotics*, 2012, pp. 46–54.

[35] G. Trafton, L. Hiatt, A. Harrison, F. Tanborello, S. Khemlani, and A. Schultz, “ACT-R/E: An Embodied Cognitive Architecture for Human-Robot Interaction,” *J. Human-Robot Interact.*, vol. 2, no. 1, pp. 30–55, 2013.

[36] J. R. Anderson, J. R. Anderson, D. Bothell, D. Bothell, M. D. Byrne, M. D. Byrne, S.

Douglass, S. Douglass, C. Lebiere, C. Lebiere, Y. Qin, and Y. Qin, “An integrated theory of the mind.,” *Psychol. Rev.*, vol. 111, no. 4, pp. 1036–60, 2004.

[37] J. R. Anderson, “Using Brain Imaging to Guide the Development of a Cognitive Architecture,” in *Integrated Models of Cognitive Systems*, 2007, pp. 49–62.

[38] M. Scheutz, G. Briggs, and R. Cantrell, “Novel mechanisms for natural human-robot interactions in the diarc architecture,” *Proc. AAAI …*, vol. 2005, 2013.

[39] F. Bellas, R. J. Duro, A. Faiña, and D. Souto, “Multilevel darwinist brain (MDB): Artificial evolution in a cognitive architecture for real robots,” *IEEE Trans. Auton. Ment. Dev.*, vol. 2, no. 4, pp. 340–354, 2010.

[40] J. Hawkins and S. Blakeslee, *On intelligence*. 2004.

[41] Z. Mathews, S. B. I Badia, and P. F. M. J. Verschure, “PASAR: An integrated model of prediction, anticipation, sensation, attention and response for artificial sensorimotor systems,” *Inf. Sci. (Ny).*, vol. 186, no. 1, pp. 1–19, 2012.

[42] D. Caligiore, A. M. Borghi, D. Parisi, R. Ellis, A. Cangelosi, and G. Baldassarre, “How affordances associated with a distractor object affect compatibility effects: A study with the computational model TRoPICALS,” *Psychol. Res.*, vol. 77, no. 1, pp. 7–19, 2013.

[43] P. Wang, “Non-Axiomatic Reasoning System: Exploring the Essence of Intelligence,” Indiana Univ., 1995.

[44] P. Wang, *Non-Axiomatic Logic: A Model of Intelligent Reasoning*. Singapore: World Scientific Publishing, 2013.

[45] R. Sun, E. Merrill, and T. Peterson, “From implicit skills to explicit knowledge: A bottom-up model of skill learning,” *Cognitive Science*, vol. 25, no. 2. pp. 203–244, 2001.

[46] R. Sun, “Motivational representations within a computational cognitive architecture,” *Cognit. Comput.*, vol. 1, no. 1, pp. 91–103, 2009.

[47] S. Franklin, S. Strain, R. McCall, and B. Baars, “Conceptual Commitments of the LIDA Model of Cognition,” *J. Artif. Gen. Intell.*, vol. 4, no. 2, pp. 1–22, 2013.

[48] S. Franklin, T. Madl, S. D’Mello, and J. Snaider, “LIDA: A systems-level architecture for cognition, emotion, and learning,” *IEEE Trans. Auton. Ment. Dev.*, vol. 6, no. 1, pp. 19–41, 2014.

[49] B. J. Baars, *A Cognitive Theory of Consciousness*. 1988.

[50] B. J. Baars, "The conscious access hypothesis: Origins and recent evidence," *Trends in Cognitive Sciences*, vol. 6, no. 1. pp. 47–52, 2002.

[51] A. S. Rao and M. P. George, "BDI Agents : From Theory to Practice," in *Proceedings of the first international conference on multiagent systems ICMAS95*, 1995, pp. 312–319.

[52] D. N. Davis, "Cognitive Architectures for Affect and Motivation," *Cognit. Comput.*, vol. 2, no. 3, pp. 199–216, 2010.

[53] M. M. Derriso, "Machine conscious architecture for state exploitation and decision making," Air Force Institute of Technology, Wright-Patterson Air Force Base, Ohio, 2013.

[54] E. Nivel, "Ikon Flux 2.0," 2007.

[55] K. R. Thórisson and H. P. Helgasson, "Cognitive Architectures and Autonomy: A Comparative Review," *Journal of Artificial General Intelligence*, vol. 3, no. 2. pp. 1–30, 2012.

[56] M. T. Cox, "Perpetual Self-Aware Cognitive Agents," *AI Magazine*, no. 2002, pp. 32–51, 2007.

[57] J. Graham, J. A. Starzyk, Z. Ni, and H. He, "Advancing Motivated Learning with Goal Creation," in *IEEE Symposium Series on Computational Intelligence*, 2014, pp. 1–8.

[58] J. Bach, *Principles of Synthetic Intelligence PSI: An Architecture of Motivated Cognition*, vol. 1, no. 1. 2009.

[59] J. Bach, "The MicroPsi Agent Architecture," in *Proceedings of ICCM-5*, 2003, pp. 15–20.

[60] J. Lin, M. Spraragen, J. Blythe, and M. Zyda, "EmoCog : Computational Integration of Emotion and Cognitive Architecture," *Artif. Intell.*, pp. 111–116, 2011.

[61] A. Haber and C. Sammut, "A Cognitive Architecture for Autonomous Robots.," *Adv. Cogn. Syst.*, vol. 2, pp. 257–276, 2013.

[62] B. Goertzel, "CogPrime: An Integrative Architecture for Embodied Artificial General Intelligence," *OpenCog Foundation*, 2012. [Online]. Available: http://wiki.opencog.org/w/CogPrime_Overview.

[63] H. Tan, “Implementation of a Framework for Imitation Learning on a Humanoid Robot Using a Cognitive Architecture,” in *The Future of Humanoid Robots - Research and Applications*, R. Zaier, Ed. InTech, 2012.

[64] N. Hawes, “A survey of motivation frameworks for intelligent systems,” *Artif. Intell.*, vol. 175, no. 5–6, pp. 1020–1036, 2011.

[65] E. Gordon and B. Logan, “Managing Goals and Resources in Dynamic Environments,” in *Visions of Mind: Architectures for Cognition and Affect. Idea Group*, 2005, pp. 225–253.

[66] F. Michaud, C. Côté, D. Létourneau, Y. Brosseau, J. M. Valin, É. Beaudry, C. Raïevsky, A. Ponchon, P. Moisan, P. Lepage, Y. Morin, F. Gagnon, P. Giguère, M. A. Roux, S. Caron, P. Frenette, and F. Kabanza, “Spartacus attending the 2005 AAAI conference,” in *Autonomous Robots*, 2007, vol. 22, no. 4, pp. 369–383.

[67] M. Scheutz and P. Schermerhorn, “Affective Goal and Task Selection for Social Robots,” *Handb. Res. Synth. Emot. Sociable Robot. New Appl. Affect. Comput. Artif. Intell.*, p. 74, 2009.

[68] M. Klenk, M. Molineaux, and D. W. Aha, “Goal-driven autonomy for responding to unexpected events in strategy simulations,” *Comput. Intell.*, vol. 29, no. 2, pp. 187–206, 2013.

[69] P. Wang, “Motivation management in AGI systems,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2012, vol. 7716 LNAI, pp. 352–361.

[70] W. Duch, R. Oentaryo, and M. Pasquier, “Cognitive Architectures: Where do we go from here?,” in *Proceedings of the 2008 conference on Artificial General Intelligence 2008: Proceedings of the First AGI Conference*, 2008, pp. 122–136.

[71] S. L. Bressler and V. Menon, “Large-scale brain networks in cognition: emerging methods and principles,” *Trends in Cognitive Sciences*, vol. 14, no. 6. pp. 277–290, 2010.

[72] G. Gong, Y. He, L. Concha, C. Lebel, D. W. Gross, A. C. Evans, and C. Beaulieu, “Mapping anatomical connectivity patterns of human cerebral cortex using in vivo diffusion tensor imaging tractography.,” *Cereb. Cortex*, vol. 19, no. 3, pp. 524–536, 2009.

[73] P. Hagmann, L. Cammoun, X. Gigandet, R. Meuli, C. J. Honey, J. Van Wedeen, and O.

Sporns, “Mapping the structural core of human cerebral cortex,” *PLoS Biol.*, vol. 6, no. 7, pp. 1479–1493, 2008.

[74] L. F. Barrett and A. B. Satpute, “Large-scale brain networks in affective and social neuroscience: Towards an integrative functional architecture of the brain,” *Current Opinion in Neurobiology*, vol. 23, no. 3. pp. 361–372, 2013.

[75] K. A. Lindquist, “The brain basis of emotion: A meta-analytic review,” 2010.

[76] J. Hawkins and D. George, “Hierarchical temporal memory: Concepts, theory and terminology,” 2006.

[77] P. A. Merolla, J. V Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, and D. S. Modha, “A million spiking-neuron integrated circuit with a scalable communication network and interface,” *Science (80-.).*, vol. 345, no. 6197, pp. 668–673, 2014.

[78] L. W. Barsalou, “Grounded Cognition: Past, Present, and Future,” *Top. Cogn. Sci.*, vol. 2, no. 4, pp. 716–724, 2010.

[79] R. S. Sutton and A. G. Barto, “Reinforcement learning: an introduction.,” *IEEE Trans. Neural Netw.*, vol. 9, no. 5, p. 1054, 1998.

[80] J. Morimoto and K. Doya, “Reinforcement learning state estimator.,” *Neural Comput.*, vol. 19, no. 3, pp. 730–756, 2007.

[81] D. Xiao and A. H. Tan, “Self-organizing neural architectures and cooperative learning in a multiagent environment,” *IEEE Trans. Syst. Man, Cybern. Part B Cybern.*, vol. 37, no. 6, pp. 1567–1580, 2007.

[82] R. C. O’Reilly, M. J. Frank, T. E. Hazy, and B. Watz, “PVLV: the primary value and learned value Pavlovian learning algorithm.,” *Behav. Neurosci.*, vol. 121, no. 1, pp. 31–49, 2007.

[83] B. Bakker and J. Schmidhuber, “Hierarchical reinforcement learning with subpolicies specializing for learned subgoals.,” *Neural Networks Comput. ...*, pp. 125–130, 2004.

[84] M. E. Harmon and L. C. B. III, “Residual advantage learning applied to a differential game,” *Proc. Int. Conf. Neural Networks*, vol. 1, 1996.

[85] R. S. Sutton and A. G. Barto, “Temporal credit assignment in reinforcement learning,” 1984.

[86] P. Y. Oudeyer, F. Kaplan, and V. V. Hafner, “Intrinsic motivation systems for autonomous mental development,” *IEEE Trans. Evol. Comput.*, vol. 11, no. 2, pp. 265–286, 2007.

[87] P. Y. Oudeyer, A. Baranes, and F. Kaplan, “Intrinsically motivated exploration for developmental and active sensorimotor learning,” in *Studies in Computational Intelligence*, 2010, vol. 264, pp. 107–146.

[88] S. Roa, G. J. M. Kruijff, and H. Jacobsson, “Curiosity-driven acquisition of sensorimotor concepts using memory-based active learning,” in *2008 IEEE International Conference on Robotics and Biomimetics, ROBIO 2008*, 2008, pp. 665–670.

[89] K. Merrick and M. Lou Maher, *Motivated reinforcement learning: Curious characters for multiuser games*. 2009.

[90] T. Teng, A. Tan, J. A. Starzyk, Y. Tan, and L. Teow, “Integrating Self-Organizing Neural Network and Motivated Learning for Coordinated Multi-Agent Reinforcement Learning in Multi-Stage Stochastic Game,” in *International Joint Conference on Neural Networks*, 2014, pp. 4229–4236.

[91] J. Graham and J. A. Starzyk, “A Goal Creation System with Curiosity,” in *International Conference on Cognitive and Neural Systems*, 2009.

[92] R. Thomason, “Desires and defaults: A framework for planning with inferred goals,” in *KR*, 2000, pp. 702–713.

[93] C. D. C. Pereira and A. G. B. Tettamanzi, “Goal Generation and Adoption from Partially Trusted Beliefs,” in *7th International Conference on Autonomous Agents and Multiagent Systems*, 2008, pp. 397–404.

[94] J. Broersen, M. Dastani, J. Hulstijn, and L. Van Der Torre, “Goal Generation in the BOID Architecture,” *Cogn. Sci. Q.*, vol. 2, no. 3–4, pp. 428–447, 2002.

[95] A. Maslow, “Motivation & Personality,” *Notes*, pp. 1–6, 1987.

[96] G. Rizzolatti, L. Fadiga, V. Gallese, and L. Fogassi, “Premotor cortex and the recognition of motor actions,” *Cogn. Brain Res.*, vol. 3, no. 2, pp. 131–141, 1996.

[97] J. A. Starzyk, “Motivated Learning for Computational Intelligence,” in *Computational Modeling and Simulation of Intellect: …*, no. Ml, 2010, pp. 265–292.

[98] J. Starzyk, J. Graham, and L. Puzio, “Needs, Pains, and Motivation in Autonomous Agents,” *IEEE Trans. Neural Networks Learn. Syst.*, 2016.

[99] J. Graham, J. A. Starzyk, and D. Jachyra, “Opportunistic Behavior in Motivated Learning Agents,” *IEEE Trans. Neural Networks Learn. Syst.*, vol. 26, no. 8, pp. 1735–1746, 2015.

[100] J. A. Starzyk and D. K. Prasad, “A Computational Model of Machine Consciousness,” *Int. J. Mach. Conscious.*, vol. 3, no. 2, pp. 255–281, 2011.

[101] J. A. Starzyk, J. Graham, and L. Puzio, “Simulation of a Motivated Learning Agent,” in *AIAI*, 2013, pp. 205–214.

[102] M. Jaszuk and J. A. Starzyk, “Building Internal Scene Representation in Cognitive Agents,” in *KICSS*, 2013, pp. 279–290.

[103] F. J. Varela, E. Thompson, and E. Rosch, “The Embodied Mind: Cognitive Science and Human Experience,” *An Int. J. Complex.*, vol. 1992, p. 328, 1991.

[104] “Second Life,” *Linden Lab*. [Online]. Available: http://secondlife.com/. [Accessed: 05-May-2015].

[105] G. Metta, G. Sandini, D. Vernon, L. Natale, and F. Nori, “The iCub humanoid robot : an open platform for research in embodied cognition.,” in *Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems*, 2008, pp. 50–56.

[106] J. Graham and J. A. Starzyk, “Transitioning From Motivated to Cognitive Agent Model,” in *IEEE SSCI Symposium on Computational Intelligence for Human-like Intelligence*, 2013.

[107] “Boost Libraries for C++.” [Online]. Available: http://www.boost.org/. [Accessed: 03-Dec-2014].

[108] N. Christofides, “Worst-case analysis of a new heuristic for the travelling salesman problem,” no. February, 1976.

[109] G. Cornuejols and G. L. Nemhauser, “Tight bounds for Christofides’ traveling salesman heuristic,” *Math. Program.*, vol. 14, pp. 116–121, 2011.

[110] J. Starzyk, J. Graham, and L. Puzio, “Managing Machine ’ s Motivations,” *Lect. Notes Comput. Sci.*, vol. 8468, pp. 278–289, 2014.

[111] J. a. Starzyk, P. Raif, and A. H. Tan, “Mental development and representation building through motivated learning,” in *Proceedings of the International Joint Conference on Neural Networks*, 2010.

[112] J. Graham and J. A. Starzyk, “A Comparative Study between Motivated Learning and Reinforcement Learning,” in *International Joint Conference on Neural Networks*, 2015.

[113] T. M. Mitchell, *Machine Learning*, vol. 4, no. 1. 1997.

[114] T. G. Dietterich, “Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition,” *J. Artif. Intell. Res.*, vol. 13, pp. 227–303, 2000.

[115] C. Moulin-Frier, P. Rouanet, and P. Oudeyer, “Explauto : an open-source Python library to study autonomous exploration in developmental robotics,” in *International Conference on Development and Learning*, 2014.

[116] A. H. Tan, N. Lu, and D. Xiao, “Integrating temporal difference methods and self-organizing neural networks for reinforcement learning with delayed evaluative feedback,” *IEEE Trans. Neural Networks*, vol. 19, no. 2, pp. 230–244, 2008.

[117] T. Gabel, C. Lutz, and M. Riedmiller, “Improved neural fitted Q iteration applied to a novel computer gaming and learning benchmark,” in *IEEE SSCI 2011: Symposium Series on Computational Intelligence - ADPRL 2011: 2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, 2011, pp. 279–286.

[118] J. OLDS and P. MILNER, “Positive reinforcement produced by electrical stimulation of septal area and other regions of rat brain.,” *J. Comp. Physiol. Psychol.*, vol. 47, no. 6, pp. 419–427, 1954.

[119] J. A. Starzyk, “Mental saccades in control of cognitive process,” in *Proceedings of the International Joint Conference on Neural Networks*, 2011, pp. 495–502.

[120] M. A. Conway, “Exploring episodic memory,” *Handb. Behav. Neurosci.*, vol. 18, no. 08, p. 19.29, 2008.

[121] A. Horzyk, “How does generalization and creativity come into being in neural associative systems and how does it form human-like knowledge?,” *Neurocomputing*, vol. 144, pp.

238–257, 2014.

[122] J. a Starzyk and H. He, “Spatio-temporal memories for machine learning: a long-term memory organization.,” *IEEE Trans. Neural Netw.*, vol. 20, no. 5, pp. 768–80, 2009.

[123] V. A. Nguyen, J. A. Starzyk, W.-B. Goh, and D. Jachyra, “Neural network structure for spatio-temporal long-term memory.,” *IEEE Trans. neural networks Learn. Syst.*, vol. 23, no. 6, pp. 971–83, 2012.

[124] A. Horzyk, J. Starzyk, and J. Graham, “Integration of Semantic and Episodic Memories,” *IEEE Trans. Neural Networks Learn. Syst.*, pp. 1–13, 2016.

[125] M. S. McGlone, “Hyperbole, Homunculi, and Hindsight Bias: An Alternative Evaluation of Conceptual Metaphor Theory,” *Discourse Process.*, vol. 48, no. August 2012, pp. 563–574, 2011.

[126] F. Pulvermüller, “A brain perspective on language mechanisms: From discrete neuronal ensembles to serial order,” *Progress in Neurobiology*, vol. 67, no. 2. pp. 85–111, 2002.

[127] Wenwen Wang, B. Subagdja, Ah-Hwee Tan, and J. a Starzyk, “Neural modeling of episodic memory: encoding, retrieval, and forgetting.,” *IEEE Trans. neural networks Learn. Syst.*, vol. 23, no. 10, pp. 1574–86, 2012.

[128] G. a Carpenter and S. Grossberg, “Adaptive resonance theory,” in *The Handbook of Brain Theory and Neural Networks*, M. Arbib, Ed. Cambridge, MA: MIT Press, 2003, pp. 87–90.

[129] T. E. Hazy, M. J. Frank, and R. C. O’Reilly, “Banishing the homunculus: Making working memory work,” *Neuroscience*, vol. 139, no. 1, pp. 105–118, 2006.

[130] L. Shastri, “Episodic memory trace formation in the hippocampal system: a model of cortico-hippocampal interaction,” 2001.

[131] L. Perlovsky, “Grounded Symbols In The Brain , Computational Foundations For Perceptual Symbol System Grounded Symbols In The Brain , Computational Foundations For Perceptual Symbol System Abstract,” *System*, vol. 1, no. 12, pp. 1–37, 2010.

[132] A. Horzyk, “Innovative types and abilities of neural networks based on associative mechanisms and a new associative model of neurons,” in *ICAISC 2015, Springer Verlag, LNAI 9119*, 2015, pp. 26–38.

[133] R. Clarke, “Asimov’s Laws of Robotics: Implications for Information Technology. 2,” *Computer (Long. Beach. Calif).*, vol. 27, no. 1, pp. 57–66, 1994.

[134] B. Fritzke, “A Growing Neural Gas Network Learns Topologies,” *Adv. Neural Inf. Process. Syst. 7*, vol. 7, no. 1, pp. 625–632, 1995.

[135] T. Kohonen, “Self-organized formation of topologically correct feature maps,” *Biol. Cybern.*, vol. 43, no. 1, pp. 59–69, 1982.

[136] J. Graham and J. Starzyk, “Self-Organizing Hierarchical Neural Network with Correlation Based Sparse Connections,” in *International Conference on Cognitive and Neural Systems*, 2008.

[137] G. E. Hinton, G. E. Hinton, S. Osindero, S. Osindero, Y. W. Teh, and Y. W. Teh, “A fast learning algorithm for deep belief nets.,” *Neural Comput.*, vol. 18, no. 7, pp. 1527–54, 2006.

[138] D. J. Watts and S. H. Strogatz, “Collective dynamics of ‘small-world’ networks.,” *Nature*, vol. 393, no. 6684, pp. 440–2, 1998.

[139] J. Graham, A. O’Connor, I. V. Ternovskiy, and R. Ilin, “The two stages hierarchical unsupervised learning system for complex dynamic scene recognition,” in *SPIE 8757, Cyber Sensing 2013*, 2013.

[140] J. Graham and I. V. Ternovskiy, “Complex scenes and situations visualization in hierarchical learning algorithm with dynamic 3D NeoAxis engine,” in *SPIE 8757, Cyber Sensing 2013, 87570J (20 June 2013)*, 2013.

[141] J. T. Graham and I. V. Ternovskiy, “Self-structuring data learning approach,” in *SPIE DSS*, 2016.

[142] J. T. Graham and I. V. Ternovskiy, “Visualizing output for a data learning algorithm,” in *SPIE DSS*, 2016.

[143] J. T. Graham and I. V. Ternovskiy, “Implementing a self-structuring data learning algorithm,” in *SPIE DSS*, 2016.

[144] M. Celenk, J. Graham, and K. J. Cheng, “Non-linear IR scene prediction for range video surveillance,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2007.

[145] M. Celenk, J. Graham, D. Venable, and M. Smearcheck, “A Kalman filtering approach to 3-D IR scene prediction using single-camera range video,” in *Proceedings - International Conference on Image Processing, ICIP*, 2006, vol. 4.

[146] M. Celenk, D. Venable, M. Smearcheck, and J. Graham, “Change detection and object tracking in IR surveillance video,” in *Proceedings - International Conference on Signal Image Technologies and Internet Based Systems, SITIS 2007*, 2007, pp. 791–797.

[147] M. Celenk, J. Graham, and S. Singh, “Traffic Surveillance Using Gabor Filter Band and Kalman Predictor,” in *International Conference on Computer Vision Theory and Applications*, 2008.

[148] M. Celenk, T. Conley, J. Graham, and J. Willis, “Anomaly prediction in network traffic using adaptive Wiener filtering and ARMA modeling,” in *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*, 2008, pp. 3548–3553.

[149] J. Graham, J. Graham, M. Celenk, M. Celenk, J. Willis, J. Willis, T. Conley, T. Conley, H. Eren, and H. Eren, “Multiple vehicle tracking using gabor filter bank predictor,” in *International Conference on Computer Vision Theory and Applications*, 2009.

[150] M. Celenk, T. Conley, J. Willis, and J. Graham, “Predictive network anomaly detection and visualization,” *IEEE Trans. Inf. Forensics Secur.*, vol. 5, no. 2, pp. 288–299, 2010.

[151] S. Carpenter, X. Yu, M. Altun, J. Graham, J. J. Zhu, and J. Starzyk, “Vision Guided Motion Control of a Biomimetic Quadruped Robot,” in *ASME 2011 International Mechanical Engineering Congress and Exposition*, 2011.

[152] X. Xu, J. Graham, J. J. Zhu, and J. A. Starzyk, “A Biopsychically Inspired Cognitive System for Intelligent Agents in Aerospace Applications,” in *Infotech@Aerospace 2012 Conference*, 2012, pp. 1–13.

APPENDIX: OTHER WORK

Over the course of completing my dissertation research, I have performed other related research in addition to the main body of material. Most of this research had related applications to the creation of the MLECOG model, and is therefore worth mentioning. The first example of this research was a foray into the development of a Hybrid Growing Neural Gas (HGNG) algorithm, which is hybrid of the standard growing neural gas (GNG) (based on Fritzke's work [134]) and a self-organizing map (SOM) network by Kohonen [135]. Neural gas networks get their name because, although they self-organize in a manner very similar to standard SOM networks, they do so in a "gaseous" manner. Meaning, that nodes in the network can break or form links at will as determined by the specific algorithm used. The GNG variant is capable of growing itself from just a few nodes to whatever maximum is specified, while the standard neural gas is static in size.

This work [18], [136] took the GNG algorithm presented by Fritzke [134] and attempted to make it more biologically plausible. Fritzke's algorithm was examined and modified to make the new hybrid algorithm with characteristics of both SOM and GNG algorithms, most specifically by imitating a SOM structure, but with moving nodes. The presence of moving nodes allows for behavior very similar to that shown by Fritzke's algorithm. The hybrid algorithm retains most of the advantages of the GNG, while adapting a parameterless design. To be specific, it retains the ability to place nodes where needed, as does the GNG algorithm, without actually having to introduce new nodes. Also, by removing the weight and error adjusting parameters, the guesswork required to determine the initial parameters' values is eliminated.

While the hybrid algorithm performed admirably in terms of the quality of results when compared to Fritzke's algorithm, it was slower, taking about 10X as long to execute for the larger network tests. It also tended to accumulate more error, although, this can be attributed to the fact that the hybrid algorithm starts with most of its nodes already present. Attempts were also made to apply the hybrid algorithm to our hierarchical invariant object recognition algorithm, which is discussed next.

Further building on the GNG research, I also looked in the creation of a self-organizing hierarchical sparsely connected recognition system. This research direction is relevant to this dissertation because it directly ties to how a cognitive agent processes its input. The implementation of such a recognition system is what will eventually allow the envisioned MLECOG agent to function within an environment by providing it with the means to process and recognize the input with which it is presented and potentially accomplish symbol grounding.

Invariant object recognition is important for any visual processing system, because any system that performs visual processing needs to reduce its inputs into a symbolic, less complex and manipulable form, if only for efficiency. In order to do this, there needs to be a mechanism for consistently recognizing these objects from within the observed scenes. Furthermore, in a dynamic, changing environment, it is important that the recognition be capable of some level of invariance. It would not be very efficient if, when slightly rotated or scaled, an object were no longer recognizable to the system.

How the invariant object recognition can be achieved is open to wide interpretation. In the case of the algorithm discussed here, the approach is performed based on a form of sparse "deep belief network" [137] or, in other words, a hierarchical, sparse, correlation based network. A sparse design was chosen since it is more efficient in terms of computation and physical realizability. Biological networks such as our brains are sparsely connected, despite being more heavily connected than we can currently fabricate. We cannot effectively mimic the connectivity of the brain due to the overwhelming number of connections; however, we can try to approximate it on a smaller scale using a "small world" network connectivity model [138].

My research on machine intelligence and what work I have done in invariant object recognition, set the stage for research during a summer internship at the AFRL dealing with hierarchical scene recognition. This work was focused on scene recognition and visualization techniques.

In the hierarchical learning model developed during the aforementioned summer research, there are effectively three layers: features, objects, and situations. Prediction of situations is described in terms of probabilities or degrees to which the situation (condition) should be believed. The first layer is the initial processed information that has been rendered into features. This information could come from any number of sources including radar tracks, visual images, infrared, etc. With the feature information provided, the system attempts to determine which objects are present via the detected features. The hierarchical scene recognition algorithm and resulting visualization method are covered in more detail in a pair of conference publications [139], [140]. More recently, I was able to continue my work in the AFRL, with more focus on developing cognitive agents, which has resulted in a few potential conference papers that are awaiting decisions [141]–[143].

In addition to the work performed directly in relation to my dissertation, I have done some research work in other closely related areas as well. For example, work with image processing and object tracking resulted in several conference papers and a journal paper [144]–

[150]. This work may have applications in the sensory input processing part of the MLECOG agent. Some work was also done, initially in relation to the "Robocat" project [151], and then later in an effort to help advance our cognitive model [152], with the idea of possibly using a Robocat like robot as an environmental platform or agent embodiment.

Thesis and Dissertation Services