Competitive intrinsically motivated agents for resource allocation

Zhao, Guopeng

2013

Zhao, G. (2013). Competitive intrinsically motivated agents for resource allocation. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/53648>

<https://doi.org/10.32657/10356/53648>

COMPETITIVE INTRINSICALLY MOTIVATED AGENTS FOR RESOURCE ALLOCATION

ZHAO GUOPENG

School of Electrical & Electronic Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirement for the degree of
Doctor of Philosophy

2013

Acknowledgments

I would like to express my deepest thanks to my supervisor Dr. Bi Guoan for his continuous support and great help during my Ph.D. study. He gave me a great amount of freedom to explore various ideas. His supervision guided me through the difficult times, and has made a profound impact on my research career.

I would like to bring my sincere thanks and gratitude to my co-supervisor Dr. Miao Chun Yan, for her enthusiastic encouragement and remarkable guidance throughout the course of my study. I benefited a lot from her advice, and her guidance has significantly shaped my way of thinking.

I also would like to thank Dr. Shen Zhiqi for his inspirations from numerous discussions, and sincerely appreciate his efforts on technical guidance. Special thanks are due to Dr. Robert Gay Kheng Leng, for providing me the research opportunity and his help during my first year of research.

I would like to take this opportunity to express my gratefulness to my wife Ailiya and my family for their love and support.

Last but not least, I also would like to acknowledge all my friends and colleagues who have helped me in one way or another throughout my Ph.D. study.

Contents

Acknowledgments	i
Summary	vi
List of Figures	viii
List of Tables	xi
1 Introduction	1
1.1 Motivation	1
1.2 Research Problem	4
1.3 Intrinsically Motivated Agent	6
1.4 Research Objectives	9
1.5 Contributions	11
1.6 Organization of the Thesis	13
2 Literature Review	15
2.1 Related Work on Motivation Theories	15
2.1.1 Time Line of Theory Development	15
2.1.2 Self-Determination Theory	20
2.2 Related Work on Motivated Agents	22
2.2.1 Evolutionary Framework on Intrinsic Motivation	22
2.2.2 Active Learning Algorithm with Developmental Robotics Approaches .	24
2.2.3 Merrick’s Self-Motivated Agent	27
2.3 Techniques Applied in Resource Allocation	31
2.3.1 Classic Approaches	31
2.3.2 Adaptive Approaches with Machine Learning	33
2.4 Summary	50

3	Modeling Competitive Intrinsically Motivated Agents	52
3.1	Theoretical Background of Agent Modeling	53
3.2	Formal Model of Competitive Intrinsically Motivated Agents	57
3.2.1	Overview of the Proposed Motivated Agent	57
3.2.2	Proposed Agent Definitions and Concepts	59
3.2.3	Computing in Motivated Agent Reasoning	62
3.2.4	Modeling Agent Main Routine Cycle	63
3.2.5	Example Modeling: Motivated Resource Selection Agent	64
3.3	Goal-Oriented Representation of Competitive Intrinsically Motivated Agents .	71
3.3.1	Motivation and Goal	72
3.3.2	A Brief on GoalNet	74
3.3.3	Goal-Oriented Modeling for Different Motivations	76
3.3.4	Use GoalNet to Represent Agent Knowledge	78
3.3.5	Use GoalNet to Represent Agent Reasoning	80
3.3.6	Use GoalNet to Represent Agent Routines	83
3.4	Summary	86
4	Need Pursuit in Competitive Intrinsically Motivated Agents	87
4.1	Introduction	87
4.2	Agent Architecture and Main Workflow	88
4.3	Competence Need Pursuit by Exploration	91
4.3.1	Goal-Oriented Representation of Novelty Pursuit	91
4.3.2	Knowledge Model of a Fast Learning Neural Network	92
4.3.3	Agent Behavior on Novelty Exploration	99
4.4	Competence Need Pursuit by Performance	101
4.4.1	Goal-Oriented Representation of Performance Pursuit	101
4.4.2	Knowledge Model of a Well-Conditioned Neural Network	105
4.4.3	Agent's Performance-Driven Behaviors	111
4.5	Relatedness Need Pursuit	116
4.5.1	Goal-Oriented Representation of Pursuit of Relatedness	117
4.5.2	Knowledge Model of Collaborative Reasoning	119
4.5.3	Agent Collaboration from Machine Learning Perspective	126
4.6	Autonomy Need Pursuit	135
4.6.1	Intrinsic Need Functions	136
4.6.2	Motivation Interpretation and Weight Adjustment	139

4.6.3	Motivation-Driven Goal and Action Selection	141
4.7	Platform Design Supporting Agent Operations	145
4.7.1	A Brief on the Agent Development Platform	146
4.7.2	Messaging, Communication and Action Implementation	148
4.8	A Summary of Developing Competitive Intrinsically Motivated Agents	149
5	Case Study I: Resource Allocation Agents for Dynamic Load Balancing in Cloud-based Large-Scale Collaborative Virtual Environments	152
5.1	Introduction	152
5.2	Problem Formulation of Load Balancing in LSCVEs	155
5.3	Related Work	157
5.4	Distributed Motivated Agents for Dynamic Workload Prediction	161
5.4.1	Motivating Example	161
5.4.2	Agent Competence Pursuit in Workload Prediction	163
5.5	Motivated Resource Allocation Agent for LSCVEs Load Balancing	165
5.5.1	Rule-based Agent Knowledge for Load Balancing	167
5.5.2	Prediction-based Agent Knowledge for Enhanced Load Balancing	170
5.6	Simulation Analysis	173
5.6.1	Simulation Settings	173
5.6.2	Analysis of Motivated Agents with Different Learning Skills	174
5.6.3	Analysis of Scheduling Performance	177
5.7	Summary	181
6	Case Study II: Applying Competitive Intrinsically Motivated Agents for Resource Provisioning in a Hybrid Cloud	183
6.1	Why Agent-Mediated Provisioning for Hybrid Cloud	183
6.2	Infrastructure Provisioning in Hybrid Cloud	186
6.2.1	Background of the Hybrid Cloud Testbed	186
6.2.2	Problem Formulation for Dynamic Resource Provisioning	189
6.3	Modeling Resource Selection Service Using Motivated Agents	191
6.3.1	Rule-based Competent of Resource Selection	192
6.3.2	Learning-based Competence of Resource Selection	193
6.3.3	Collaborative Resource Selection by Multiple Motivated Agents	195
6.4	Deploying Agent-Mediated Resource Provisioning Service in Cloud	196
6.4.1	Components for the Agent-Mediated Service	198

6.4.2	System Architecture for the Proposed Provisioning Service	199
6.5	Performance Evaluation of Resource Allocation Service	201
6.5.1	Experiment Settings	201
6.5.2	Agent Novelty Pursuit of an Accurate Prediction	205
6.5.3	Agent Performance Pursuit of High Resource Utilization	207
6.5.4	Agent Relatedness Pursuit of a Robust Decision	210
6.6	Summary	213
7	Conclusion and Recommendations	214
7.1	Conclusion	214
7.2	Recommendations for Further Research	216
7.2.1	Autonomous Knowledge Discovery for Competency Pursuit	217
7.2.2	Autonomous Knowledge Evaluation for Autonomy Pursuit	220
7.2.3	A Diversity of Agent Collaborations	222
	Author's Publications	226
	References	230

Summary

Resource Allocation maps user requirements to specific resources, and it plays an important role in enabling a ubiquitous, convenient, on-demand network access to a shared pool of computing resources. Due to the dynamic nature of resource availability and the uncertainty of runtime environment, a successful allocation service should be flexible and agile. In particular, the service should be able to answer the three questions: (1) how can the service evolve with environmental changes that cannot be anticipated during design time; (2) how can the service make reliable and comprehensive decisions from incomplete information; and (3) how can the service systematically integrate and choose from a diversity of capabilities to achieve its objectives.

To answer the above questions, **Intrinsic Motivated Agent** is viewed as a tailored approach to service adaptiveness, for an agent is considered as an autonomous problem solver, which can provide a “brain” for the resource allocation service to exhibit flexible and agile behaviors, and the agent with intrinsic motivations can further bring the service a unified sense of “self-willing” so that diverse behaviors can be integrated systematically. To this end, the thesis proposes a **Competitive Intrinsically Motivated Agent** by incorporating the psychological Self-Determination Theory (SDT) into agent modeling. The proposed approach of **Competitive Intrinsically Motivated Agents for Resource Allocation** enables an allocation service with enhanced proactiveness and improves performance in terms of the degree of adaptiveness that the service achieves.

The proposed motivated agent is driven by the pursuit of three **intrinsic needs**, namely **competence**, **relatedness** and **autonomy**. In particular, agent’s behaviors in the pursuit of needs address each of the three questions individually:

- The need of competence drives agents to adapt to the dynamically changing environment, by a constant pursuit of good performance and new skills;
- The need of relatedness enables agents to provide reliable solutions from unevenly distributed information, by advocating collaborations with other agents to collect and synthesize information;

- The need of autonomy steers agents in systematically integrating various strategies and behaviors by encouraging them to make their own decisions proactively.

The contributions of this thesis are summarized as follows. First, in light of SDT the research proposes a formal model of competitive intrinsically motivated agent, which takes account of agent's intrinsic needs. Second, in order to bridge the gap between the proposed agent theoretical model and practical implementation, a goal-oriented approach is adopted to represent the proposed agent by systematically integrating agent's psychological attributes, such as motivation and needs, with agent's design artifacts such as goals, tasks and behaviors. Third, for resource allocation, a competitive intrinsically motivated agent is realized by incorporating a set of machine learning techniques, followed by discussions on the achieved agent adaptiveness. The resultant agent-mediated service benefits both service providers and service consumers through a robust, agile and adaptive resource sharing.

Finally, two case studies are conducted to explore the practical use of the proposed agents for resource allocation in real-life applications. One is for load balancing in a Large-Scale Collaborative Virtual Environment, and the other is related to Cloud resource management for enterprise users. The evaluation results show that competitive agents with intrinsic motivations can exhibit adaptiveness in a great extend in terms of learning, performance, collaboration and decision making. Therefore, the success of the two case studies suggests that the proposed agent is not only promising but also practical.

List of Figures

2.1	Homeostatic Adaptation	17
2.2	BAS and BIS Motivation	17
2.3	Maslow’s Need Hierarchy	19
2.4	Reference Models of Evolutionary Framework of Intrinsic Motivated Agent . .	23
2.5	Information Theoretic and Predictive Approaches for Intrinsic Motivation . . .	25
2.6	Competence-based Approach for Intrinsic Motivation	26
2.7	Morphological Approach for Intrinsic Motivation	26
2.8	Reference Models of Merrick’s Motivated Agents	29
2.9	Workload Prediction Module	36
2.10	Sample Simulation Results	43
2.11	Modeling of User Requirements	45
2.12	A Sample Antibody Network	47
2.13	IDS Architecture	48
2.14	Adopted Fuzzy Neural Network Structure	48
3.1	Generic Agent Model	54
3.2	Proposed Competitive Intrinsically Motivated Agent Model	57
3.3	Sample Fragment of Motivated Agent Plan	70
3.4	GoalNet Model Components	74
3.5	Root Goal in GoalNet	76
3.6	GoalNet Modeling of Plan	79
3.7	Internal Logic of Generating Agent Motivated Behaviors	80
3.8	Unified Modeling of Goal Selection and Action Selection	82
3.9	Generic Main Task Model	84
4.1	Architecture of the Competitive Intrinsically Motivated Agent	89
4.2	GoalNet Representation of Agent’s Motivated Main Task	90
4.3	GoalNet Representation of Agent’s Pursuit of Novelty	92

4.4	ELM Neural Network	95
4.5	Pseudo-Code for Training a ELM Neural Network	99
4.6	GoalNet Representation of “To Train ELM model”	99
4.7	GoalNet Representation of “To Fulfill An External Request”	102
4.8	GoalNet Representation of “To Self-Improve”	103
4.9	GoalNet Representation of “To Search Task for Performance”	104
4.10	Algorithm for Unsupervised Input Weight Selection	110
4.11	Comparison I Between Random and Pre-conditioned Input Weight Selection . .	113
4.12	Comparison II Between Random and Pre-conditioned Input Weight Selection .	114
4.13	GoalNet Representation of “To Ask”	118
4.14	GoalNet Representation of “To Answer”	118
4.15	Comparison of Decision Boundaries from Two Independent Runs	121
4.16	Sketch of Stacking ELM Training	125
4.17	Illustration of Need Functions	138
4.18	Policy Evaluation and Improvement	145
4.19	Overview of MADE Architecture	146
4.20	MADE Main Dataflow	147
5.1	A Motivating Example for Impacts of User Movements on LSCVEs	162
5.2	GoalNet for Workload Prediction Using Neural Networks	165
5.3	GoalNet of Achieving Load Balancing	170
5.4	Proposed Load balancing Algorithms for Cloud-based LSCVEs	172
5.5	Sample User Distributions: Initial And Final	175
5.6	Testing Performances for 2 Selected Cells.	176
5.7	Performance Comparison for N_{OU}	179
5.8	Performance Comparison for N_{MU}	180
5.9	Performance Comparison for Overall Effectiveness	181
6.1	System Architecture of a Hybrid Cloud Testbed	187
6.2	Screenshot of Cloud Resource Management in Dashboard	188
6.3	GoalNet for Invoking Resource Selection	192
6.4	Learning-based Resource Allocation Workflow	195
6.5	Collaborative Decision Making for Multiple Intrinsically Motivated Agents .	197
6.6	Functional View of Agent-Mediated Resource Provisioning Service	200
6.7	Relationship between Workload and Execution Time	204

6.8	Neural Network Testing Performance for Case 1 and Case 2	206
6.9	Choices in Performance Pursuit for Resource Provisioning Agent	208
6.10	Utilization Comparison for Case 1	209
6.11	Utilization Comparison for Case2	209
6.12	Average Utility Comparison	210
6.13	Comparison of Single Agent and Multiple Agents for Task Execution Time Regression in RMSE	212
6.14	Comparison of Single Agent and Multiple Agents for Task Execution Time Regression in Pearson Correlation	212
6.15	Comparison of Single Agent and Multiple Agents for Task Classification	213
7.1	Pseudocode for Discovering Knowledge in the Form of GoalNet	218
7.2	Illustration of GoalNet Abstraction	219

List of Tables

2.1	Three Types of Motivated Agents	28
3.1	Conceptual Relationship Between Motivation and Goal	73
3.2	Mapping among Needs, Motivations and Root Goals	78
4.1	Comparison Between Random Input Weight Selection and Proposed Input Weight Selection	115
4.2	Specifications of Regression Problems	128
4.3	Comparison of Single ELM and Stacking ELM of Same Structures For Regression Problems	129
4.4	Comparison of Single ELM and Stacking ELM of Different Structures For Regression Problems	130
4.5	Specifications of Classification Problems	131
4.6	Comparison of Single ELM and Stacking ELM of Same Structures For Classification Problems	132
4.7	Comparison of Single ELM and Stacking ELM of Different Structures For Classification Problems	132
4.8	Weights Assigned to Five Level-0 Models	133
4.9	Comparison of Least Squares Generalizer with and without Non-Negative Constraint, and Single Best	134
4.10	Rules for Selecting a “Root Goal”	140
6.1	Attributes Of Computer Hardware Dataset	202
6.2	Comparison Between BP and ELM	207

Chapter 1

Introduction

1.1 Motivation

Resource consumption takes up an essential portion of IT cost on industries. Modern industries are constantly looking for the conversion of IT from capital cost to operational cost. The most effective approach to achieve cost conversion is decoupling infrastructure resources from the differentiating factors of their business. To achieve this, industry is increasingly adopting principles of Service-Oriented Architecture (SOA) in their design of infrastructure resource provisioning. The thesis has its setting of story in this industrial promotion of Infrastructure as a Service (IaaS).

Advocated by SOA, outsourcing part of an industry's IT operations emerges as a matter of course. **Enterprise consumers** gain more agility in the cost reduction and has better focus in their business innovation. In the meanwhile, **service providers** see their business growth and continuously attract new enterprise customers. Because of this win-win situation in the industry sector, research efforts in Grid and Cloud computing are rapidly transferred to business, which are advocated as popular examples for service use in provisioning infrastructure resources.

Recent advance in Cloud computing offers an unparalleled opportunity to enterprise customers. Cloud computing provides a flexible infrastructure such that it becomes possible to scale business operations exponentially and flex resources rapidly in response to changing demands. In contrast with multi-server architecture, one of the most immediate benefits of

Cloud-based infrastructure is the ability to add or drop servers quickly. The characteristics of incremental scalability and infrastructure flexibility allow business to respond quickly and flexibly to variable demands. Meanwhile, the on-demand provisioning of resources eliminates the cost to procure hardware and the cost to manage. The utility-based economics drive more and more enterprises to adopt Cloud-based infrastructure, due to the largely reduced capital expenditures.

Most enterprises have expressed their readiness for Cloud computing. However, they have **many concerns** over embracing this emerging technology in spite of its vast potential. Outages, e.g., April 2010 when Amazon EC2 went down for four days, reveal part of the rationale behind it. A report by Cloutage indicates that about 350 outages have happened over last 3 years. The problems are related to many aspects, such as the aspects of availability, governance, compliance, security, etc. However, current development of IaaS focuses on protocols and middleware that facilitate network access to describe, discover, aggregate and harness infrastructure resources. Existing effort merely produces specifications and technologies for realizing SOA in resource sharing across distributed domains. There are less concerns on control mechanisms that are reliable to deal with failures and adaptive to cope with changing environment. As a result, the technology is largely left behind and less mature than the industry expectation.

One of the main challenge in Cloud computing is the efficient allocation of resources to the tasks required by users. **Resource allocation** refers to the process that maps user resource-consuming activities to specific resources. it plays an important role in enabling a convenient, ubiquitous, on-demand access through network to a shared pool of computational resources. Allocation service is usually provided in the scheduling components of a system. It monitors resource performance, implements allocation and priority policies, and optimizes the execution performance for a given task. The challenges posted by resource allocation comprise the optimization of the execution of a task (demand) in response to various service level requirements

and a variety of constraints. The above challenges becomes more appealing in an open, dynamic distributed environment, where infrastructures are shared and multiple client processes are executed concurrently. The uncertainty and dynamics within the environment demand extensive study and careful attentions.

Due to the **dynamic** nature of resource availability and the **uncertainty** of runtime environment, a successful allocation service should be flexible and agile. To achieve this, the thesis emphasize on **how to achieve the adaptiveness of resource allocation in a dynamic and uncertain environment**.

In order to achieve the adaptiveness of allocation service in a dynamic and uncertain environment, the service needs to be **intelligent**: it needs to estimate resource availability, predicate trace congestion, model workload resource requirements, satisfy individual Quality of Service (QoS) requirement, and meet job deadlines; the service needs to be **robust**: allocation should have its capability in identifying and selecting a reliable resource based on the information that is incomplete and noisy. As a summary, an adaptive resource allocation service should exhibit sophisticated behaviors, that are not only driven by client requests, but also thorough reasoning, concerning on reliability and performance,utilization optimality, etc. In short, such service exhibit **agent**-like behaviors.

By definition, an agent can be described in the following quote [1],

An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future.

Above definition entails several features of an agent. An agent inhabits a certain environment, which is either fully or partially observable and either static or dynamic. In this environment, the agent requires to be capable to perceive and act, and sometimes to be of some domain knowledge. Embodying autonomous behaviors into agent systems is the ultimate goal of agent

research. In the thesis, the agent autonomy refers to allowing agents to decide whether or not to perform a certain action by themselves. To achieve autonomy, an agent has to be stimulus-independent, and inputs from outside only modulate agent's interior ongoing activity [2]. An agent should determinedly acts in pursuit of its own agenda. After each agent acts to the environment, its action may have an effect on its later sensing.

It is also well known that agents are considered as **autonomous problem solvers** that can flexibly act in an uncertain and dynamic environment. In the work “Why Grid and Agents Need Each Other” by I. Foster and N. R. Jennings [3], the authors suggested that a service in a distributed, administratively heterogeneous, and failure-prone environment should exhibit flexible autonomous actions. In short, adaptive infrastructure services will exploit agent-like behaviors.

The integration of agent and resource allocation will unavoidably create new challenges. Nevertheless, the resultant allocation service that is agent-mediated, could exhibit flexible autonomous behaviors, and thus benefit both service providers and service consumers through a robust, agile and adaptive resource sharing.

1.2 Research Problem

The statement of our research problem in one sentence is: how to achieve the adaptiveness of agent-mediated service for resource allocation in a dynamic and uncertain environment. The thesis tackles resource allocation problem particularly for IaaS, and the stated problem comprises three sub-problems.

- **Dynamically Changing Environment**

Agent by definition is acting to environment, but if we want to deploy our agent to a real-world scenario, we must face environment which is usually uncertain and dynamic, noisy and error-prone. Examples of a complex, open environment for real-life applications include game-industry's Large-Scale Collaborative Virtual Environment (LSCVE)

or a multi-tenant platform for managing Cloud resources. To adapt to a changing environment, an agent that is designed for certain conditions should be able to possess sufficient set of skills. The competence of an agent should evolve together with the changing environment. However, for a long term, it is impossible for service/system designers to prepare for all the possibilities, nor anticipate all future environmental changes. The information is simply not available at the design time. Therefore, adaptiveness requires that an agent should have the capability to intelligently self-adapt to a wide range of derived and emerging situations over time.

- **Imbalanced Information Distribution**

The “living scope” of a real-life application usually spreads across an entire system, network or domain. The characteristic of the practical scenario causes an imbalanced distribution of information: (1) many problems are geographically distributed in the real world; (2) many complex problems in the real world are functionally distributed, and diverse expertise is required for problem solving.

Due to the limitation of knowledge, information, resources and ability of the problem solving, most single agents can only solve parts of the problem. Since a single agent is usually within a particular limited scope, there is a problem that a single agent usually lacks of a global picture to decide whether its “local” solution valid for the “global” problem. Usually noise is usually across the entire system, and agents who make local decisions based on noisy data produce unreliable solutions. In order to solve such problems, different agents should work cooperatively. Collaborating agents assemble and synthesize information collectively, so that they can identify the noise, and make right decision for right problem. When different agents cooperate with each other, the problem solving ability can be increased significantly. For example, agents used in a distributed sensor network system, collects and interprets data from a specific location. These agents

then cooperate to make a global map across the area. Therefore, collaboration of multiple agents plays an important role in the real-world applications.

- **Lack of Systematic Integration for a Diversity of Behaviors**

Real-life applications usually comprise multiple tasks. Each task has a specific set of operations. The operations vary a lot from sensing and acting to planning and training. The situation becomes more complicated if allowing substitutions and task hierarchy.

In the design of agent adaptiveness in a complex environment, the efficiency of trade-offs between alternatives and organization of multi-leveled tasks rely on the wisdom of integration of a diversity of behaviors. In view of the fitness of agent's high-level objectives, an agent needs to balance effectively between being too committed to its present course of actions and too responsive to external stimuli. Too much commitment causes the agent to neglect more important tasks that could not be dealt with in a timely manner; too much responsiveness makes the agent changing its targets too frequently such that no task could be completed ever. Unfortunately, there is a lack of support in unifying different behaviors within one well-defined integrality.

1.3 Intrinsically Motivated Agent

As mentioned in the problem statement, for a complex environment not all relevant information is available at design time. Therefore, agents should possess the capability of adaptiveness in a great scale of flexibility.

In the agent literature, the adaptiveness of agent has been addressed by the recent advance in **Intrinsically Motivated Agent** (IMA), which is inspired by the psychological development on intrinsic motivation [4]. The reason that IMA is attracting agent researchers is central to the its capability of **learning new knowledge and skills** in the course of solving a variety of problems which might arise after agent deployment.

To acquire new skills, **exploration** of the environment is paramount, and **self-experimentation** is an essential way to conduct the exploring process. When it comes to the **learning capability**, machine learning algorithms such as reinforcement learning and supervised learning will be considered quite naturally. However, these algorithms require some specific forms of learning stimuli, such as a reward signal or a defining of correct patterns. Consequently, these algorithms have typically been used for **single, simple and isolated tasks** for which it is easy to set a task-oriented reward signal or task-oriented behaviors [5]. In short, machine learning is designed for solving single tasks [6]. Nevertheless, a large-scale system is likely to be asked to learn more than one task quickly and effectively. Therefore, we can consider any machine learning algorithm as a “building block” for the “low-level” task-driven situations [7]. In contrast with low-level learning algorithms, what is the “high-level” agent modeling that allows **building a house with the building blocks**?

To answer this question, IMA emerged recently, which specifically is designed to engage in play, exploration, and other behaviors driven by internal stimuli in the absence of explicit reward [8]. To understand the new type of agent, it is necessary to begin with the definition of motivation. The term “**motivation**” comes from the Latin word *movere*, meaning “to move”. Psychologists use this concept to help explain how internal factors move animals and human toward certain goals. We therefore regard motivation as an influential process related to persistence, direction, and drive of goal-oriented behaviors.

Recently there has been a surge of interest in modeling motivations in the agent literature. Motivation consists of intrinsic and extrinsic motivations. They are distinguished by characteristics of behaviors (actions taken to achieve the goal) and activities (goals). By definition, **extrinsic motivation** is to move an agent for external rewards, money, grade, threat of punishment, competition to beat others. On the contrary, **intrinsic motivation** drives to seek out challenges, new opportunity, enjoyment, activity for its own benefit (rather than external rewards coming with the activity). Evidences report that intrinsically motivated entities will be

in good mental state and be effective in deal with tasks [4]. Modern psychology finds that intrinsic motivation is engaged/initialized by interest and maintained by satisfying basic needs. Only if both behavior is self-determined and is inspired by the inherent pleasure of the activity, an **agent** is considered **intrinsically motivated**, and the agent is called **Intrinsically Motivated Agent**.

The difference between “intrinsic” and “extrinsic” motivation deserves a further clarification. For autonomous agents, the key difference is not whether a stimulus that drives an agent is an external signal or not. In fact, the motivations are not exclusive of each other, and they are essentially two different levels of signaling. Extrinsic is primary while intrinsic is secondary. Primary means that an agent receives a stimulus as a direct, task-specific signal, while secondary means that the agent evaluates and processes the primary signal within its own value systems that is task-independent and life-long, and the generated secondary signal guides the agent to develop a plan. In this sense, the deliberated plan is out of agent’s own interest, and “for its own sake”. Meanwhile, secondary signal shall not be a direct translation of task-specific objective functions [9, 10].

No matter whether a learning algorithm is supervised, unsupervised or reinforcement-based, this thesis treats them equally as a detailed implementation of learning behavior. Although supervised learning algorithm requires environmental feedbacks to tune the learning performance, the external signal is only used for determining primary signals (i.e. event detection). It is still up to the agent’s secondary signals (i.e. intrinsic motivations) to decide if it should continue improving learning, or turn to other activities (such as performing tasks). In short, intrinsically motivated agents still interact with and need feedbacks from external environment, but the behaviors are not solely dependant on that.

Different from agents in the state of the art, many of which are designed with a specific algorithm and typically used for single, simple and isolated tasks. These tasks are easy to set a task-oriented reward signal or task-oriented behaviors. On the contrary, motivated agents are

designed for dealing with a large variety of tasks, especially when not all relevant information is available at design time. In order to cope with a uncertain and dynamic environment, motivated agents are expected to be able to continuously evolve by learning and acquiring new skills. To guide this exploration behavior, a high-level model that is independent of specific environment stimuli is needed, so that it allows achieving new goals from low-level primitives. This high-level model is central to agent autonomy, and it is modeled as intrinsic motivation in motivated agent research. Inspired by the psychological phenomena of animals and human, intrinsic motivation will bring an inclination toward integrating different capabilities into a unified sense of “self-willing”, and this unified approach will synthesize isolated behaviors into a full play and thus offers a great flexibility to the agent adaptiveness.

1.4 Research Objectives

Although intrinsically motivated agent could be conceptually contribute to the adaptiveness of real-life applications, there is a gap between contemporary agent development and practical usage, especially for industries. Existing research has done several work on motivated agents [5–29]. Some of them only focused on the difference between internal and external stimuli; some simply concentrated on aspects related to “interest”; and some merely modified certain learning algorithms. As forerunners, they have made significant contributions, but concerning the ultimate goal of intrinsically motivated agent, their work is still weak in the scope and scale. A detailed literature review is discussed in Chapter 2.

In order to equip agents with the capability to develop and evolve by acquiring, developing and exploiting skills in an open-ended environment, this thesis aims to investigate and propose a new type of intrinsically motivated agent, called **Competitive Intrinsically Motivated Agent**, considering and integrating multiple perspectives of motivations in a systematical manner.

Our research is based on **Self-Determination Theory** (SDT), which is an approach for modeling human motivation and personality that focuses on three fundamental psychologi-

cal needs –competence, autonomy, and relatedness– and on how they relate to intrinsic and extrinsic motivation [30]. The origin of SDT comes from the postulate that “humans are active, growth-oriented organisms who are naturally inclined toward integration of their psychic elements into a unified sense of self and integration of themselves into larger social structures” [30]. In particular, SDT discusses intrinsic motivation with the level of satisfaction of **three psychological needs**, namely Competence, Relatedness and Autonomy.

Competence is related to the tendency of mastering skills, exploring new opportunities and challenges, achieve progress in performance. Specifically, the need of competence can be divided to two aspects.

- **Need of Novelty**, which reflects the need to master new challenges. This need will drive the agent to explore the environment, doing self-experiment, and acquire new knowledge and skills.
- **Need of Performance**, which refers to the need to experience oneself as capable, and to perfect skills. This need will drive agents to complete their tasks, sharpen their related skills and even look for tasks when they are free.

Relatedness is related to our desire to form meaningful bonds with others –to care and to be cared for. This need will drive agents to interact with each other and *exchange their information*, such as asking questions and answering the queries.

Autonomy is related to the sense of self-determination versus control. The selection of behaviors and activities is from the decision of agent itself. Agent feels a great level of freedom to choose according to the knowledge of itself. What to do and the way to achieve a goal are not told by external, but from its own knowledge. Autonomy here is very similar to autonomy talked about in the computer literature, where agent is said to be autonomous meaning agent behavior is independent of designer, and agent is capable of learning, self-sufficient, situation-aware when dealing with environment and able to evolve by itself after deployment.

Considering the relationship between the **research problem** of resource allocation and the needs system, the proposed **Competitive Intrinsically Motivated Agents for Resource Allocation** have following niche properties. The **Need of Performance** is always important, as a good performance is the fundamental of any intelligent system. Moreover, the **Need of Novelty** will help agents to adapt to the dynamically changing environment (Problem 1) since the need will push agents to continuously obtain new skills. Meanwhile, the Need of Relatedness will help agents to solve the problem of imbalanced information distribution (Problem 2) since the collaboration between agents will enhance the information collection. Finally, the **Need of Autonomy** facilitates agents to systematically integrate their strategies (Problem 3), fully control their behaviors, and proactively make decisions according to their needs intrinsically.

To sum up, our research objectives includes three aspects:

- (i) To investigate multiple perspectives of motivation based on three human intrinsic needs represented in Self-Determination Theory.
- (ii) To integrate them into a systematically organized agent modeling with a goal-oriented approach.
- (iii) To implement the proposed Competitive Intrinsically Motivated Agents for resource allocation with practical case studies to access whether the agent-mediated service can help resource allocation adapt to the uncertain environment and achieve good performance.

1.5 Contributions

In line with above research objectives, the main contributions of this thesis include:

- (i) A novel type of agent modeling for competitive intrinsically motivated agent is defined in light of a sound and well-known psychological theory, named Self-Determination Theory (SDT). The research formulates the proposed agent model following SDT by

incorporating agent's trinity of autonomy, competence and relatedness. The proposed formulation is the first attempt in making a motivation theory in cognitive psychology computational for implementing an artificial agent. Agent can benefit from SDT by systematically managing various behaviors.

- (ii) A goal-oriented approach is adopted to implement the proposed intrinsically motivated agents for complex problems. The goal-oriented approach allows agents to manage various tasks as goals, decompose a complicated goal into simpler sub-goals, and achieve each goal through a sequence of actions. Scaffolded with goals, high-level motivations make sure deliberation and low-level behaviors coherent with agent's objectives. This research further designs an agent routine as its main task, through which the agent decides the actions to take in pursuit of its intrinsic needs. Specifically, a variety of implementations of agent knowledge and plans are represented with the goal-oriented approach in fulfillment of intrinsic needs. The proposed design promotes motivated agents to learn, to perform, to collaborate and to make decisions. A diversity of machine learning algorithms, including supervised, unsupervised and reinforcement learning, are leveraged and evaluated showing that the proposed intrinsically motivated agent exhibits a great level of adaptiveness.
- (iii) The proposed agents of competitive intrinsic motivations are evaluated in two case studies of real-world resource allocation applications. The case settings are completely different. One scenario is related to a Large-Scale Collaborative Virtual Environment, leveraging the proposed agents for distributed data collection and analysis, and one central agent that synthesizes information and make decisions for load balancing across the entire virtual environment. The other, targeting the Cloud resource management for enterprise users, employs the proposed agents for provisioning computing resources in two scenarios: resource utilization optimization and resource feasibility classification. For

both cases, the same agent design methodology have been exploited, showing that the proposed approach is generic and has potential to be applied in other cases. Moreover, the evaluation results show that competitive intrinsically motivated agents can exhibit adaptiveness in a great extend, and proved that this type of agent is not only promising but also practical.

1.6 Organization of the Thesis

The rest of the thesis is organized as follows.

Chapter 2 first introduces the psychological background of motivation theories, and focuses on the Self-Determination Theory (SDT), which is the fundamental of this thesis. After that, the use of motivation in agent technology is reviewed thoroughly, and we discuss limitations of existing approaches by comparing with SDT. The latter part of this chapter presents a brief literature of applying both classic and adaptive techniques in infrastructure provisioning.

In Chapter 3, the emphasis lies in the modeling of Competitive Intrinsically Motivated Agent from formalization to a goal-oriented representation. Before that, a theoretical background of agent modeling is given with discussion on a variety of agent models. The formal model of the proposed agent follows the Self-Determination Theory (SDT) by considering the trinity of autonomy, competence and relatedness. Finally, to realize the proposed motivated agent, a goal-oriented representation is then introduced to manage agent behaviors in an integrated and systematic manner.

In Chapter 4, an overview of agent architecture is first introduced with the design of agent's main task. Several sub-goals as the pursuit of agent needs are introduced with a GoalNet representation, a detailed system-wise design, and a detailed component-wise implementation. To gain a better understanding, each sub-goal is instantiated with a simulation example, followed by a discussion on the advantage of each proposed agent behavior. Finally, an agent running platform is then explained to support the execution of the proposed motivated agents.

Chapter 5 and Chapter 6 explore two case studies in the domain of resource allocation. The proposed agents are applied for the both sides of a business transaction, from the view of a service provider and the view of enterprise consumers. In Chapter 5 service providers of Large-Scale Collaborative Virtual Environment adopts motivated agents to achieve load balancing across their entire business; and in Chapter 6, enterprise consumers leverages motivated agents to manage a hybrid Cloud infrastructure with heterogeneous resources. In both real-life cases, the proposed competitive intrinsically motivated agents have been successfully employed for resource allocation, and demonstrate great adaptiveness in dealing with the uncertain and dynamic environment.

Finally, Chapter 7 concludes the thesis with a summary of thesis contributions and a discussion of future work.

Chapter 2

Literature Review

2.1 Related Work on Motivation Theories

The origin of the word motivation is from “movere” in Latin which means “to move”. It is used to describe how internal factors move people toward certain goals. In psychological theories, motivation is defined as “a process that influences the direction, persistence, and vigor of goal-directed behavior [31]”. It is the reason, the purpose of behaviors. Motivation answers the question that why people move, or in other words, people do activities in order to satisfy their motivation.

Motivation pertains to biological, cognitive, and social norms. In practice, people value motivation very much due to its impacts: “motivation produces” [32]. There are many trends of theories to explain motivation. Like many other basic concepts in psychology, the explanation of motivation has large numbers of versions. Different theories depict different facets of motivation, and we can utilize a certain type of theory to examine motivation in a specific purpose even though there is no universal agreement on motivation theories.

2.1.1 Time Line of Theory Development

At the beginning of the 20th century, psychologists utilized *instincts* to describe motivation under Darwin’s theory influence. An instinct [33] is “an inherited characteristic, common to all members of a species that automatically produces a particular response when the organism

is exposed to a particular stimulus”. With the millions of years of natural selection, the advanced instincts are remained and form our human beings. These types of work are known as *evolutionary theories* which view motives as a selection that has adaptive value in terms of fostering reproductive fitness [34]. However, researches on instincts failed as it is easy to fall into circular reasoning such as defining selfishness as a human instinct since we can find this character from human beings; but why we have this character since it is an instinct, an intrinsic part of human nature. This logical problem cannot build up a solid research, and consequently cause this theory to step down from the stage of history.

In 1932, a new concept *homeostasis* is proposed by Walter Cannon, which means “a state of internal physiological equilibrium that the body strives to maintain” [35]. A homeostasis system involves four parts, an internal state to provide information of body changes, sensors to detect these changes, a controller to generate responses from sensed information, and a response system to restore bodily equilibrium. In 1943, Clark Hull developed homeostasis theory and introduced *drives* [36] which are generated when homeostasis is physiologically disrupted, and this internal tension will prompt the operation of an organism to reduce this tension. Drives are widely used in explaining hunger, thirst, and sex, but it cannot explain situations that people try to increase states of arousal rather than reducing. For instance, people choose to eat less in order to lose weight, or people give up sleep time to accomplish a task. From these cases, we can find that drive theory only fits for explaining the deficiency of physical needs, but it cannot explain the so called high level psychological needs. So far, we find that the discussion of motivation needs to be concerned from both “*physical*” and “*psychological*” aspects. Meanwhile, as homeostasis focuses on “*internal*” physiological equilibrium, we also need to specify the “*external*” factors from environment. All these aspects will be discussed in the following paragraphs.

First, we will specify the internal factors versus external factors. Unlike drives theory focused on internal factors, *incentive theories* emphasize on external stimuli that push people

Figure 2.1: Homeostatic Adaptation. Original picture is from [31].

Figure 2.2: BAS and BIS Motivation. Original picture is from [31].

toward a goal. This time the reason of movements is not from the internal physiological disequilibrium; it is out of human body and is from the outside environment, such as we tend to pursue the things we like, and to avoid the things we dislike. This theory strengthened by Jeffrey Gray's arguments in 1991, that he claimed a discovery of two neural systems in the brain, the *Behavioral Activation System* (BAS) and the *Behavioral Inhibition System* (BIS). BAS produces actions driven by signals of positive need satisfaction and potential reward; In contrast, BIS reacts to stimulus that signals "potential pain, non-reinforcement, and punishment" [37]. This research is currently at the forefront among motivation theories since it has great influence

on behavioral, physiological and cognitive processes. It uses the physical evidence in brain to elaborate the interaction between outside environment and internal reactions.

Besides the internal factors versus external factors, there is another pair –physical factors versus psychological factors. As we have mentioned, the drive theory only considers the physical or biological perspective, but with the development of understanding on human cognition, more and more researchers realized that the psychological needs are essential for human beings, and these types of high level motivations distinguish human from animals. White in 1959 [38] proposed an intrinsic, *psychological motivation* for helping people effectively interact with the living environment. In cognitive theories, psychological motivation is further specified from two dimensions the extrinsic motivation, behaving for external reward, and the intrinsic motivation, behaving for internal interest. In detail, *intrinsic motivation* [4] is defined as “the doing of an activity for its inherent satisfaction”, such as for the fun and enjoyment of the activity, or the challenge entailed in the activity per se. Intrinsic motivation is related to the interaction between a person and an activity. We say a person is intrinsically motivated when he/she does something because he/she is interested in it. On the contrary, *extrinsic motivation* [4] pertains to activity that is done “for attaining some separable consequence and instrumental results”, such as the activities to avoid punishment, earn one’s living, or obey social regulations. Comparing these two types of motivations, researchers found that intrinsic motivation could lead to better performance and more flexibility in creativity, learning, well-being, and cognition. Deci and Ryan subsequently developed a theory, *Self-Determination Theory* (SDT) [32], and stated that the psychological needs for competence, autonomy, and relatedness underlie intrinsic motivation, and if doing an activity can satisfy these needs, people will be highly intrinsically motivated. This theory further examined how people’s needs are related to intrinsic and extrinsic motivations. The details will be discussed in Section 2.1.2.

In the mid-20th century, researchers began to consider not only the biological aspects but also motives in social experiences. Abraham Maslow proposed a broad motivational model

need hierarchy in 1954. The needs related to physical and social survival are arranged at the bottom, and the needs uniquely concerned with human growth are at the top. In detail the five levels of the hierarchy from low level to high level respectively are [39]:

- Physiological needs: the requirements for survival, such as the need for air, water, food, etc.
- Safety needs: the requirements to ensure personal and financial security, to keep health, and to maintain the safety net against unexpected events.
- Belongingness and love needs: the feeling of being involved in intimacy, family, and friendship.
- Esteem needs: the requirement to be respected and to have self-esteem and self-respect.
- Self-actualization needs: the desire to become more and more what one is, “to become everything that one is capable of becoming”.

Figure 2.3: Maslow's Need Hierarchy. Original picture is from [31].

This theory involves people in pursuit of a meaningful life mentally and socially, which is based on a belief that people strive for personal growth and the ultimate human motive is self-actualization.

2.1.2 Self-Determination Theory

Self-Determination Theory (SDT), by definition, “is an approach to human motivation and personality that focuses on three fundamental psychological needs –competence, autonomy, and relatedness –and on how they relate to intrinsic and extrinsic motivation” [40].

In SDT, needs refers to innate psychological nutriments that are crucial for sustainable psychological growth. The satisfaction of people’s fundamental needs will highly flourish their intrinsic motivation towards the related activities. The three fundamental needs defined in SDT are competence, autonomy, and relatedness. *Competence Need* reflects a basic human need to experience oneself as capable in mastering new challenges and perfecting skills. This need encourages exploratory and self-growth behaviors. *Autonomy Need* is satisfied when people experience their actions as a result of free choice without external interference. This need is related to the feeling of ownership, self-control, and self actualization. *Relatedness Need* refers to our desire to form meaningful bonds with others –to care and to be cared for. This need helps people feel freer to be themselves, and further improve their relationships among others.

We have mentioned that intrinsic motivation and extrinsic motivation are two branches of psychological motivation. Many studies have confirmed that comparing with extrinsic motivation, intrinsic motivation entails better conceptual learning, greater creativity, more cognitive flexibility, and enhanced well-being [32]. Therefore, to effectively motivate a person, the intrinsic ways are better than extrinsic ways.

Although, intrinsic motivation is from the inside of human nature, we can adapt social conditions to maintain and enhance it. Deci and Ryan have studied many different external factors related to intrinsic motivation, and they received an interesting outcome, that monetary

rewards or prizes tend to diminish people’s innate interest on an activity. However, factors from external parties, such as “acknowledging one’s feelings, offering choices, and giving positive performance feedbacks”, can foster people’s intrinsic motivation and result better performance. These findings inspired them to find the “fundamental needs” of human beings and eventually enhance one’s intrinsic motivation via satisfying these needs.

Taking the need of autonomy as an example, the external factors –rewards, directives, deadlines, and threats –are easy to be interpreted as controllers of one’s behavior; in contrast, the external factors such as choice, acknowledgement, informational suggestions, and positive feedbacks are viewed as supports of one’s autonomy. Of course, different people have different interpretation and reflection of the external factors. There is no absolute boundary of supportive or destructive factors; it is the separated satisfaction level of distinct persons’ fundamental needs.

As a summary, the reason of choosing SDT as the foundation of the proposed agent model is stated as follows. Drive theory can only explain the deficiency of physical needs (e.g. avoid hunger, thirsty), but cannot explain high-level psychological need (e.g. give up sleep time for a task). Incentive theories emphasizes on external stimuli that push people towards a goals, pursuing things people like and avoid things people dislike. It merely emphasizes external impacts, so when the environment is still, agent will stay idle. Therefore, it will provide limited support to achieve agent’s autonomy, where agent performs “for its own sake”. In cognitive theories, psychological motivation is divided into extrinsic and intrinsic motivations. It provides conceptual support for modeling agent’s behaviors in terms of both responding and interacting with environment, and acting for its own interest. SDT is one widely accepted cognitive theory, while other cognitive theories like Maslow’s Need Hierarchy either do not explain how to deliberate actions from needs or are too complicated to be implemented.

For this thesis, it is a niche fit between SDT theory and the problem statement of resource allocation: the need of competence drives agents to adapt to the dynamically changing environment, by a constant pursuit of good performance and new skills, while keeping balance

between exploration and exploitation; the need of relatedness enables agents to provide reliable solutions from unevenly distributed information, by advocating collaborations with other agents to collect and synthesize information; and the need of autonomy steers agents in systematically integrating various strategies and behaviors by encouraging them to make their own decisions proactively.

2.2 Related Work on Motivated Agents

Motivation has been widely applied in many domains, such as economics [41], education and sports [4]. For the domain of artificial systems, research on designing intelligent systems with motivation has a substantial history, including Lenat’s AM system [16] focusing on heuristic definitions of “interestingness”; Schmidhuber [17–20] using reinforcement learning to implement curiosity; and Sloman and Croucher [21] discussing domain specific motives. In the epigenetic robotics community, related work concerns the role of curiosity and novelty in intelligent behavior [22, 23]. From the field of architecture and design, research [24] based on Berlyne’s theories proposed a system that considered novelty and surprise as perspectives of intrinsic motivations which contributed on searching design spaces.

Besides above literatures, three main groups of researchers analyze the theories of motivation from the cognitive perspective and concentrate on “intrinsic motivation” distinguishing “extrinsic motivation” to build on their artificial systems; these are the Satinder Singh, Andrew G. Barto, and Michael Littman’s group [7, 8, 10], the FLOWERS research team at INRIA [9, 14, 25, 26], and Merrick and Maher in Sydney university [6, 27–29]. Details are given in the following sections.

2.2.1 Evolutionary Framework on Intrinsic Motivation

Satinder Singh, Andrew G. Barto, and Michael Littman’s work can be summarized from two high-level viewpoints. One is that they deeply discussed the notions of extrinsic and intrinsic

Figure 2.4: Reference Models of Evolutionary Framework of Intrinsic Motivated Agent

rewards which try to answer the question: “what does it mean in the computational reinforcement learning framework to do something because of the inherently intrinsic motivations?” Their answer is from the evolutionary perspective that the years of evolution history of animals and human beings urge subjects to generate motivations for pursuing a *reproductive* success.

In light of this argument, they established their second view of introducing an evolutionary framework to computationally simulate intrinsic motivation and distinguishing it from extrinsic motivation. Traditional reinforcement learning requires predefined reward functions to reward states, and it is considered as “extrinsic”, because the agent has no idea on how it is going to award itself after perceiving the environment. To cope with intrinsic motivation, researchers formulated a fitness function and added it to the notion of an optimal reward function. The fitness function is analogous to the degree of an animal’s reproductive success in nature [8]. In this way, the agent is simulated to play in the environment for a number of “life-long” rounds, and each round generates a history of episodes. The agent is evaluated by the expected fitness of episodic history, and progressively finds the optimal reward functions that maximize its expected fitness.

Figure 2.4 shows the idea of categorizing environment into internal and external. The reward signal of intrinsic motivation comes from the internal environment instead of the external environmental stimuli. Note that for the 1st figure, primary reward is fed to the agent directly

from the environment, while the 2nd figure shows a refinement where the environment is refined into an internal and external part, with all reward coming from the internal [10].

In their work, intrinsic motivation explicitly referred to agent's need in searching for the optimal reward function. In their another work [7], intrinsic motivation referred to agent's curiosity on salient events in searching for new skills. These approaches were within hierarchical reinforcement learning framework. Designed for multi-task learning in reinforcement learning settings, they lacked of a domain independent model for defining agent's intrinsic motivations. The need of exploring was expressed as novelty pursuit in SDT. The limitation lies in the facts that solely driven by novelty, an agent was difficult to keep up with one task consistently, resulting in switching among multiple tasks. This indicates that the agent was less likely to accomplish a task, especially when the task was long-run and required a focus.

2.2.2 Active Learning Algorithm with Developmental Robotics Approaches

Oudeyer's work is based on the developmental robotics approaches. The fellows in FLOWERS research team at INRIA Bordeaux Sud-Ouest tried to equip robots with capabilities to explore and develop various kind of skills as well as to be able to reuse the skills later on. This type of robots have mechanisms to drive themselves in the exploration of novel skills. Researchers believed that intrinsic motivations were crucial for self-organizing developmental trajectories and directing the learning of a general and reusable skill [25]. Till this end, they proposed a "robust intelligent adaptive curiosity algorithm (R-IAC)" to improve traditional "intelligent adaptive curiosity algorithm (IAC)", and argued that IAC can be viewed as an active learning algorithm, and the R-IAC can learn both forward and inverse work in a complex sensorimotor space with subspaces that are traditionally not easy to learn. The intrinsic motivations were used as heuristics to push a robot to explore the environment while regulate the growth of complexity in sensorimotor. Their work also focused on artificial curiosity, social learning, morphological computation, and maturational constraints.

Another significant contribution of Oudeyer's group is from their review papers on computational models based on intrinsic motivations. They divided the measures of intrinsic motivation or interestingness into three broad types [26].

- Knowledge-based models comparing the perceived results of agent's actions to its expectations and knowledge. (as Figure 2.2.2).
- Competence-based models measuring agent's competence that can achieve self-determined results or goals (as Figure 2.6).
- Morphological models analyzing the information obtained simultaneously from multiple sources of stimuli by comparing different parts of the perceptive world (as Figure 2.7).

This typology focuses on the computational reinforcement learning framework (CRL), and highlighted the definition of rewards related to intrinsic motivation.

Figure 2.5: Information Theoretic and Predictive Approaches for Intrinsic Motivation

As a summary, Oudeyer proposed a range of computational models of motivations, and suggested an IAC system which pushed an agent towards the situations in which it maximized learning outcome. The drive made the agent concentrate on the situations which are “neither

Figure 2.6: Competence-based Approach for Intrinsic Motivation

Figure 2.7: Morphological Approach for Intrinsic Motivation

too predictable nor too unpredictable”. As claimed in SDT, however, curiosity is referred as novelty pursuit belonging to the competence need, and optimal challenges or arousal theory was mentioned as one of the drive of learning in SDT. Therefore, the proposed model can be

considered as a subset of SDT, and the model was not design for accomplishing any task after learning. In addition, there was no discussion on how the agent could consistently achieve the learning goal of a task (i.e. obtain the expected learning feedback) without switching to learn other tasks too early.

2.2.3 Merrick’s Self-Motivated Agent

According to Merrick and Maher’s arguments, in traditional artificial intelligence the reasoning process focuses on the rules and the learning. Planning approaches are designed to meet fixed goals. But for motivated agents, the approaches focus on the combination of problem-finding and problem-solving. In other words, the agent needs to identify highly motivating goals and then solve them. They have proposed a set of motivated agents, and the details are summarized in Table 2.1, and the corresponding agent models are illustrated in Figure 2.8.

Agent models listed in the figures include a generic agent; motivated reflex agents that respond to highly motivating stimuli with a predefined action; motivated reinforcement learning agents that use trial-and-error to learn behaviors which cause highly motivating stimuli; motivated supervised learning agents that learn to mimic highly motivating behavior [27].

Merrick’s work was more related to learning task selection. In order to synthesize motivation and learning in designing artificial agents, they on one side proposed computational models to depict motivation, and on the other side extended traditional learning (supervised and reinforcement learning) to their “motivated” counterpart to adapt multi-task learning to dynamic environments. As the most active research group on motivation agents, they explored a motivation agent model for novelty-seeking, interest-seeking, and competence-seeking explorations. Novelty was expressed in the same way as in curiosity-driven behaviors; interest preferred the choice of learning that was never too difficult nor too easy, similar as in optimal challenge; and competence was maintained active until the agent could repeat the behavior with small enough error, showing the agent’s competence with this task, similar as in the error minimization. Their recent research [29] indicated that “existing computational models

Table 2.1: Three Types of Motivated Agents [12]

	Motivated Agent	Reflect	Motivated Reinforcement Learning Agent	Motivated Supervised Learning Agent
Agent Definition	It extends traditional reflect agents with a computation model of motivation such that recognized states is known in advance		It uses systematic trial-and-error to learn behaviors that maximize the long term value of a reward signal	It learns behaviors by mimicking examples of correct behavior from other agents, either human or artificial
Traditional Approach	Use mechanisms such as rule-bases and fuzzy logic to define how the agent should respond to certain, recognized states		The reward signal tends to define a specific task or goal to be achieved	Standard examples tend to be hand-picked by the system designer and oriented towards a specific task or goal
Motivated Approach	Use recognized motivational states produce a response		The reward signal is modeled as a motivation value defining task-independent concepts such as interest or curiosity	Use a computational model of motivation to focus attention on a subset of examples from which to learn
The Goal	Perform some pre-defined actions in response to the motivational state of the agent		Develop new, structured behaviors adapted to their physical form and/or their environment	Minimize the error between the actions they choose to perform and the example actions they observe

of motivation cover only a small subset of psychological motivation theories”, and proposed a goal-selecting behavior driven by motivation, similar to risky avoidance and achievement pursuit.

From the view of SDT, it was a subset of the theory, belonging to the category of competence need fulfillment. The competence need was satisfied by exploring a novel or a challenging skill, or mastering the learning tasks with positive feedbacks (e.g. small learning errors). Similar to the approach of the thesis, they modeled agent’s value system by finding evidence

Figure 2.8: Reference Models of Merrick's Motivated Agents

from psychology. However, their research was limited to discussing the impacts of agent's value system on behaviors. Their recent research showed the ambition to consider intrinsic motivation from a wider spectrum, including striving for standards of excellence, building social relationships with others, and gaining their influence to control resources. This was closely related to the thesis, which aims to cover the most influential set of motivations. However, their approach was restricted to investigating various agent behaviors (goal-selection) as a result of disparate agent personality (value system). They did not discuss how a heterogeneous action-

able plan can be derived from agent’s abstract motivations.

As a summary, several researchers discussed their opinions over defining intrinsic motivation: (1) Barto *et al.*, from an evolutionary perspective, considered “reproductive success” as the drive for agents to behave proactively; (2) Oudeyer *et al.* designed robots to pursue activities for which “learning progress is maximal”; (3) Merrick *et al.* considered agent’s self-motivated exploration as seeking for “novelty, interest, and competence”. Although the focus of exploration is paramount to motivated agents, they did not discuss how to trade off exploration and exploitation. For the proposed competitive intrinsically motivated agent, learning is consider as part of agent behavior, and a lot of behaviors, such as performing a task and collaborating, are also considered. It is insufficient to merely enlist many tasks. The proposed agent synthesizes the various behaviors into a unified sense of “self-willing”, and models how tasks are in pursuit with the support of a psychological framework, i.e. SDT.

2.3 Techniques Applied in Resource Allocation

2.3.1 Classic Approaches

Efforts have been seen in recent years to develop schedulers for grid computing systems. It is a convenient strategy that extending an existing low level scheduler to adapt to an high level scenario. Condor and Condor-G [42] are one example. Condor is originally an automatic scheduling software for a cluster of local resources. It makes fully utilization of CPU resources, stores checkpoints for interrupted process, and is able to resume the execution by migrating it to another idle resource. ClassAd mechanism is employed for publishing resource and job requirement specifications at a place where is known a Condor pool. Condor has its management divided into job management and resource management, controlling and monitoring the status of job queue and execution machines accordingly. Condor-G is an enhancement on Condor's job management part, which includes an interface to Globus Toolkit. It achieves transparency such that external resources can be viewed in its local Condor pool. Furthermore, external users can submit their jobs to the local pool through Condor-G. All in all, Condor-G harnesses global resources as if they are local. However, job and resource matching is more than merely a minimum requirement filtering. It involves more consideration about resource's reliability and expected performance, which further requires an intelligent approach. Moreover, when there are many jobs waiting for scheduling, one matching engine to search in the pool is not enough. This would be an bottleneck for the entire system. It is the scalability that needs to be carefully studied.

Local Sharing Faculty (LSF) [43] is a commercial job scheduler which dispatches a job to a server that has enough memory bandwidth to meet the requirement, and the memory information is collected host-by-host through a standard benchmark. Such determination of available memory bandwidth is prior to the configuration of the platform using LSF, and therefore it can not be adaptive to the dynamics of global resources, restricting itself to applications of

local resources. In addition, the job requirement is simplified to take memory bandwidth into account only, which is not suitable for Grid's diversity of QoS requirement cases. Similarly, commercialized PSB Pro [44] requires manual operation to specify the job requirement or the destination of execution. A user must acquire complete resource information in order to obtain a high performance for job execution. In this case, too much user effort is involved. Both LSF and PSB Pro are not satisfying for a truly dynamic Grid as they assume a perfect and accurate information, and are not able to provide intelligent resource selection services.

AppLeS [45] specializes on parameter sweep application, and leverages Network Weather Service (NWS) for forecasting and filtering a list of candidate resources. GrADS [46] implementing application-specific execution puts NWS as a component of its Resource Selector to obtain dynamic resource. Both attempt to adapt to the dynamic environment by constantly forecasting the network condition and resource performances. However, there are limitations with NWS, and machine learning approach is expected to have better predications [47]. Nimrod/G [48] is similar to AppLeS except that it employs an economy-based model to make schedules according to budgets as well as deadlines. Nimrod/G requires users to manually add resource and resource information to a SQL-based database. Then resource selection utilizes these information for scheduling, which are static and may be out of date at the time of execution.

We can conclude that current off-the-shelf schedulers like LSF and PSB Pro are mainly local resource schedulers, and usually ignore the fact that complete resource information and control are always out of hand. Some on-going projects like Condor and Nimrod/G focus on general architecture and bypass details of intelligent modeling for resource selection. AppLeS and GrADS do consider adaptiveness issue in Grid computing, but still lack of a sound model to achieve both intelligence and robustness. Additionally, all of existing schedulers somehow neglect the requirement of adaptability to dynamic behavior of to-be-scheduled jobs. That is, in a reasonable time span, Grid scheduler should provide reasonable response to a certain number

of jobs, from quite a few to numerous. This requires the high level scheduling instance have the ability to scale up and down according to the number of requests.

2.3.2 Adaptive Approaches with Machine Learning

This section presents numerous case studies of soft computing techniques adopted in Grid/Cloud middleware infrastructure to facilitate and improve resource selection service. Techniques are reviewed from artificial neural networks, case-based reasoning, fuzzy logic, genetic algorithms to autonomous agents, such as negotiation and trust. This section is organized in various categories of techniques applied in the resource allocation, including load modeling, resource management, security and trust enhancement, etc.

2.3.2.1 Artificial Neural Networks

Artificial Neural Network (ANN) is one of the most commonly applied technology for a wide range of resource allocation cases. This section reviews a few classic examples of how ANN is applied for applications of workload prediction.

Workload Predication Workload or resource usage usually refers to CPU time in the context of computational Grid environment. Soft computing techniques provide a novel approach for such nonlinear function approximation, which have already gained its fame in parallel & distributed systems and high performance computing. In this section, we are going to present some current employments on global and localized optimization methods.

A successful Grid is capable to provide service to meet client's diverse Quality of Service (QoS) requirements. Typical QoS requirement involves completion of submitted tasks before specific deadlines. Related scheduling schemes are Earliest Deadline First (EDF) and Max-Min approach. One strictly assigns the resource to the task with most urgent deadline, whereas the other ensures fair access among all users, enough computational power to finish all within deadlines or during congestion, proportional reduction of resource assigned to each

task according to individual's contribution (e.g. price) [49]. Current Grid infrastructure desires further implementations to guarantee particular user's demands. However, Grid, as a collective platform, often faces contradictory demands, since many users compete for the usage of resource. Thus, meeting the requirements of some users may results in scarification of others. In this scenario how to fulfill different users' requirements is essential to the success of scheduling. An appropriate scheduling scheme is driven for the needs to meet as many QoS requirements as possible. The heart of solution is to estimate the possible task execution time before running any scheduling scheme.

Doulamis *et al.* [50] propose a workload prediction mechanism in the Grid architecture for realistic computer graphic generation, where large computational complexity of simulation is solved more feasibly and efficiently than that of a single resource. A Multilayer Perceptron (MLP) network with one hidden layer approximates the nonlinear mapping from task attributes to rendering process time. The predictor reads task descriptions from RIB format file, extracts significant attributes and forwards the obtained feature vector to neural network. [51] further applies PCA analysis to selected features to improve performance. The adopted neural network is trained using N sample set beforehand, which contains pairs of the form $(\mathbf{x}_i, y_i)$, where vector $\mathbf{x}_i$ includes the parameters of a specific rendering environment and y_i is the corresponding workload obtained from experiments. The network is particular trained using Marquardt-Levenberg algorithm for its known efficiency and fast convergence. However, the proposed approach is limited to a specific type of rendering environment, where training set is generated. Due to the nature of Grid resource, one cannot restrict the source of computational power, either from supercomputer or from PC or mobile devices like PDA. Panagakis *et al.* [49] further builds a more concerned model, incorporating resource performance parameters in addition to workload parameters. Thus, the execution time on various resources can be calculated as the quotient of workload over resource parameters. The proposed resource modeling utilizes a benchmark process to measure every heterogeneous resource platform and

obtains a resource performance vector $\mathbf{r} = [r_1, r_2, r_3]^T$ where r_i , $i = 1, 2, 3$ refers to CPU speed (in MFLOPS/sec), average memory I/O (in MB/sec), and average disk I/O (in KB/sec). Then to estimate workload several platforms with known resource parameters invoke the task and records every runtime performance as training data. The model assumes x_i , $i = 1, 2, 3$ is the task workload parameter, whose elements are CPU instruction (in MFLO), average memory I/O amount (in MB) and average disk I/O (in KB) respectively, and t_i has the relation with x_i and r_i as follows

$$t_i = \frac{x_i}{r_i} \quad (\text{Eq. 2.1})$$

Then the total task runtime t_{run} is further assumed as

$$t_{run} = \sum_{i=1}^3 t_i \quad \text{or} \quad t_{run} = \sum_{i=1}^3 x_i \cdot r_i^{-1} \quad (\text{Eq. 2.2})$$

Workload vector $\mathbf{x}$ is determined by minimizing

$$E = \sum_{j=1}^N (t_{run_j} - \sum_{i=1}^3 x_{i,j} \cdot r_{i,j}^{-1})^2 \quad (\text{Eq. 2.3})$$

where subscript j denotes the j th sample platform measurement. In the sequel, scheduler makes an estimation on a new platform with its known resource performance parameters using the calculated workload parameter in accord to (Eq. 2.1) and (Eq. 2.2). More sophisticated method may invoke neural network to learn from pairs $(\mathbf{r}, t_{run})$ directly.

In addition, [49] and [52] advocate an advanced Grid architecture, consisting of workload predictor, capacity estimator, runtime estimator, and scheduler for commercial application, shown in Fig 2.9. The workload predictor can be also independently estimated using task descriptor parameter vector. Neural network is recommended due to its approximation capability for “any continuous and any non-linear function with any degree of accuracy”.

A straightforward MLP may include task parameters together with resource capability description to forecast workload. Yuan *et. al* [53] suggest a simple model with 4 inputs, namely host load, processor speed, memory availability and task scale mapping to task runtime. Then

Figure 2.9: Workload Prediction Module

a Min-min scheduling algorithm determines the task executing order and host assignment to meet QoS as much as possible.

ANN for Available Resource Predication Artificial Neural Network (ANN) is engaged in the predicative mechanism for network bandwidth predication [47]. Bandwidth has a direct impact on application performance, even for Grid computing. Grid computing relies on service-oriented architecture, whose resources are in the form of web service. The web service performance greatly depends on bandwidth for packet delivery, or to say, the traffic condition between two terminals. Bandwidth has complex behavior, and only systems able to provide good approximation for nonlinear function are suitable for the choice.

A Multilayer Perceptron (MLP) is adopted for its well known characteristic of approximating any nonlinear function with any degree of accuracy. The devoted predictor utilizes the network trace data from AUCKLAND [54] for learning and testing. AUCKLAND IV traces are from real-life passive measurement, collected of 1-day trace between February and April

2000, in total 45 days, at the University of Auckland with both uplink and downlink. Bandwidth is assumed to be relative to number of packet in each second and number of bits in each second. The original trace data is parsed and converted to a three column data file, second, corresponding packet and bit number. User first decides the predicated duration, for example 5 mins. Then a set of bins of 300-second size separate the entire dataset non-overlappingly, followed by calculating bin parameters, such as, average packet rate, average bit rate. The newly created bin file with bin parameters is used as training data. A MLP with 2 hidden layers of 3 neuron each is trained, using backpropagation algorithm. The output of neural net can be one-step or multi-step predication, i.e. the predication duration is a multiple of bin size.

The performance indicates that neural net provides acceptable predication if the traces cover a long enough period. This is because long trace data takes account of diversified bandwidth behavior and ANN will only make good approximation when it has similar “experience” before. Experiments show traffic data in 7-10 days is enough for AUCKLAND scenario. Unfortunately, huge amount of training data inevitably causes learning process costly for both time and storage. The tradeoff remains an issue for consideration when different types of application are considered. Moreover, 95% training data in AUCKLAND is TCP data, and thus UCP and ICMP are treated as noise, since they are used for different applications. By noise reduction, the performance is improved. Therefore, it implies that for each type of data, applying a dedicated predictor is appropriated. The proposed ANN has been compared with the Network Weather Service (NWS). The results exhibit ANN has a better predication capability than NWS, and ANN would be an alternative to future network forecasting. Grid Harvest Service (GHS) [55] is constructed based on above predication mechanism. Primary components like task allocation and task scheduling partition parallel workload into subtasks and assign subtasks to a resource based on prediction of execution time. The posed GHS scheduling decreases the task completion time and reduces the number of machines involved. However, researcher has to bear in mind that unlike the weather forecasting, for which utilization of predication will not affect the climate considerably, the usage of network bandwidth estimation would drive resource

consumption concurrently due to the predication that “next time the resource will be plenty”. That would fail user’s expectation, and more seriously would cause network congestion. As a consequent, further research should take collaborative operation into account.

For sharing huge amount data, replication in Grid can reduce access latency and bandwidth consumption. It can also improve the reliability and availability of the system by creating data redundancy. Replica can be thus regarded as resource. However, it is the question how to select the best replica if several copies are available for an application site. By definition of best, we simply mean the fastest transmission speed, most secure, and least cost. Neural network based predication is quite active in such scenario. The method posed in [56] employs 3 layer feedforward back-prorogated neural network for mapping file size and available bandwidth to file transfer time predication. The most simultaneous response replica site will be chosen. The results showed that neural networks predicted the correct location for replica selection in most cases, even in changing network conditions.

Other work [57] not only considers next step resource availability but also takes reliability into account. Resource reliability is modeled as variance of future resource capability. The posed approach utilizes NARX (a feedforward neural network) based load predication to define allocation of resources, with the information of predicated CPU load and its variance. It achieves good performance if the dynamical behavior of resource is considered.

As Grid/Cloud provides flexible access to global resource for heterogeneous platforms, it is vulnerable to various threads from flood attacks(e.g. Distributed Denial-of-Service) to logical attacks. Different from ordinary situation, attacks to Grid may be massive and frequent. Then a solid detection system with high accuracy and consideration of load balancing as well as fast response deserves researcher’s attention.

MLP-based Intrusion Detection [58,59] invite a distributed intrusion detection mechanism for a set of network units residing in an organization. Traffic flows in every ten seconds are

stored in a Flow File, which is dispatched to a selected Detection Node (DN) according to DN's performance score that a center scheduler maintains. The scheduler achieves load balancing and effectiveness via assigning jobs to those competent. As a consequence, the total detection processing time is shortened. Inside each DN, back-propagation neural networks are applied to recognize logical attack and other types of attacks. Inputs are packet protocol, packet length, time to live, destination port etc. With one hidden layer of 9 neurons with transig and linear activation function neural networks are trained using 9 learning algorithm. Levenberg-Marquardt is adopted due to its fastest performance. It comes out in the experiment that neural network can detect all attacks and make 7 false alarms out of 2000 runs. The posed architecture provides more capable solution to tradition intrusion detection systems, especially when the organization faces a high volume attack.

SOM-based Intrusion Detection Self-Organizing Map (SOM) also facilitates detection of abnormal user behavior. SOM possesses of useful features, such as input space approximation, density matching, topology preserving, and feature selection. The learning of SOM is unsupervised, and thus preparation of training data becomes easy. SOM has been applied in intrusion detection for quite a few years [60, 61], though recent publications do not specially focus on Grid scenario. However, without losing generality, proposed SOM methodologies will be still able to apply to the newly emerging heterogeneous platform. Two level SOMs are suitable for network traffic detection: the first level selects the most significant features out of numerous network parameters, whereas the second level classifies the reduced reference vectors (SOM neurons) into normal and abnormal regions. Its online version Growing Grid Neural Network (GGN) has particular usage to handle new input patterns. GGN starts with small number of neurons and then is grown by adding a row or column to the structure until a maximum neuron number is reached or any stopping criteria. Able to learn new patterns continuously from previous network, GGN has nearly constant training time for increasing number of training patterns [61]. Experiments show that SOM possesses of good performance to detect intrusion.

Resource Classification Grid resource refers to any accessible entities in the collaborative environment, including personal computers, high performance mainframes, storages, instruments, computational services, bandwidth, so on and so forth. Resource discovery, which also implies resource selection, is the process of retrieving resource states in the context of the distributed environment of the Grid, and identifying resources whose properties and states match user requirements [62]. After discovery, resource manager figures out how to allocate the resource to clients efficiently and satisfyingly.

Metacomputing Directory Service (MDS) provides clients a means to query diversified resources in the Grid environment. MDS usually contains information to describe static host computers information (such as type and version of OS, CPU specification and memory capacity), dynamics host information (such as mean load and number of current processes), storage system, etc. However, users cannot always specify the basic information of required resources, and sometimes application-orientated query will simplify user's work especially for novice. Therefore, a design of a Grid resource discovery service based on application-oriented classification is presented in [63]. The novel query service is built on the top of existing Grid Index Information Service (GIIS), with an ANN classifier, known as Grid Resource Classifier (GRC), categorizing different resources into application catalogue. A single hidden layer feedforward neural net gets basic information pair from GIIS and trains itself to produce result vector $\mathbf{t} = (t_1, \dots, t_i, \dots, t_k)$ where $t_i \in [0, 1]$ presenting the suitability of a resource to the i th out of k applications. A modified back-propagation with gradually increasing step size optimizes the learning process towards the target value that implies the effort from experts. Despite of lack in experiment results in [63], this proposal urges the modification of resource querying from technical terminology to more user friendly application-oriented. However, because of supervised learning the posed approach definitely requires large amount of costly data, which may not be ready for collection in real life. Moreover a predefined application domain would limit the dynamical and heterogeneous usage of Grid. A unsupervised and self-organized approach like SOM will help to group complicated resources and leaving the choice

to a particular application will be enough for application managers to make further decision and even negotiation.

2.3.2.2 Case-based Reasoning

Some researchers [64] prefer localized method to global one in workload prediction, for the reason that Grid performance is greatly subject to the dynamic nature of runtime environment, and predication works meaningfully only for a bounded duration. The resource usage estimator exploits Instance-Based Learning (IBL) algorithms, including Weighted Average, Nearest-Neighborhood, and Locally Weighted Polynomial Regression, for simulation on *PUNCH*, a network-computing system at Purdue University. The estimator forecasts CPU time of executing on semiconductor simulation tools, where a large group of students are likely to access tools for assignments concurrently, exhibiting temporal locality of running tasks.

Localized methods avoid trying to reach a global approximation of CPU runtime which includes far earlier records. The feature space is divided into several partitions, and then the problem becomes selecting a nearest feature partition according to some distance metrics. The conducted experiment firstly selects features by domain experts, and compares performances between the three IBL algorithms. Further modifications like incorporating a cache to mimic short-term memory and instance editing exhibit comparative advantage over basic IBL algorithms. Short-term memory of the system speeds up the lookup process by a factor of two, which exposes the validity of the assumption of temporal locality. The knowledge base size is tremulously reduced by employing instance editing; they selectively considered the features that are incorrectly predicted and discarded the features that consistently produced incorrect results.

This localized approach displays the diversified characteristics of different cases in Grid infrastructure that will have an effect on the choice of soft computing techniques. As a consequence, workload predication has to bear on individual application nature, and decides whether global or localized method is more suitable.

2.3.2.3 Fuzzy Logics

Network-based computing often requires estimation of resource statues, including CPU utilization, available bandwidth, etc. A sound scheduler employs it to foresee network congestion and emptiness, and thus achieves better resource utilization.

Rule Mining on Availability of Resource Aforementioned scenarios focus on resource usage predication of a specific running task, whereas there is another circumstance especially when available resource is of repetitive time series patterns. The operation addresses the future state of a Grid resource which is without load and behaves periodically on its own, for example self-checking or backup. Having previously developed a MLP for time series predication, Yaik *et al.* [65] declare that association rules outperform the “black-box” in convincing users by provided explicit semantics in the knowledge base, though neural network produces accurate results.

The proposed work flow consists of patten mining, realtime predication and adaptive reinforcement. Patten mining module computes the gradient of every two consecutive points on the time series, classifies the set of gradients into subsets, and assigns labels to each subset. Thus, the original series of points convert to a chain code. The classification is analogue to fuzzification in accord to fuzzy rules. The paper leaves the choice of implementing rules as an open question, not limited to either external experts or adaptive learning phrase. Pattern mining then proceeds with the chain code to identify repetitive patterns. Each pattern scores a support value that shows how frequently it occurs. The predication module recognizes the current chain code, for example, ABC, followed by searching the pattern base to analyze the possibility of next pattern’s occurrence. This is based on the assumption that the possibility of the remaining segments of a repeatable pattern appear is high, if a segment of that pattern has occurred. For exmaple, the support of ABCD turns up 10 and that of ABCE 15 with $support(ABC) = 25$, then the confidence of D to be the next pattern is $support(ABCD)/support(ABC) = 40\%$

Figure 2.10: Sample Simulation Results

and confidence of E is $support(ABCE)/support(ABC) = 60\%$. Note that the system identifies the current chain code as 3 code length. In a more complex situation, the code length N is determined dynamically, and more patterns may be possible for the module to generate. Reinforcement learning makes the system self-evolving and to be lifelong process, by either rewarding a support value to a pattern which makes correct predication, or punishing to one wrong decision. Moreover, the proposed mechanism also allows new patterns to emerge in the database.

The paper has demonstrated some simple repeatable time series test cases (with one example described in Fig 2.10) with acceptable performance. Using association rules, this approach can successfully identify random data, since random data do not have repetitive behaviors. Compared with neural networks, the system outputs explicit rules and confidence level instead of numerical presentations in the black-box, which makes the process easier to be justified. However, current model only supports univariate time series, and pattern matching process consumes large computational power, as well as defuzzification used at the output causes degree of inaccuracy when converting to original time series.

Fuzzy logics are also used in user requirement analysis. In order to yield efficient and satisfying resource allocation, scheduler should sufficiently analyze user requirements. Departed from workload estimation, human operators may instead stipulate the constraints that a

resource scheduler is to take consideration of. By the definition given in [66], there are hard, soft and prioritized constraints (or originally hard, preference and prioritized requirements). Hard constraints ought to be satisfied without compromise, whereas soft ones with tolerance to non-exactly matching. Prioritized requirements are high level constraints expressing the preference of one constraint over another. [66] further provides a fuzzy constraint modeling for each category. Fuzzy constraints assert the needs to naturally modeling semantics that are fuzzy and uncertain, especially applicable when involving human operators. Following gives an example in [66] to describe roughly how modeling works.

R1 A human operator requires 7 CPUs in total for a given task, x for network I/O, y for computation and z for visualization purpose.

R2 There should be around 3 CPUs for visualization, ideally 3, possibly 2 or 4.

R3 Computation requires 2 or 3 CPUs.

R4 I/O performs the best if using 4 CPUs.

R5 R4 is more important than R3.

Firstly we define a five level scale $L = (\alpha_1 = 0, \alpha_2, \alpha_3, \alpha_4, 1)$, where $\alpha_3 = 0.5$ and $\alpha_2 = 1 - \alpha_4$.

Let priorities R3 and R4 be α_3 and α_4 , $\alpha_3 < \alpha_4$ thus satisfying R5. Continuously, we have

- R1: hard constraint

$$\mu_{R1}(x, y, z) = 1 \text{ if } x + y + z = 7$$

$$\mu_{R1}(x, y, z) = 0 \text{ otherwise}$$

- R2: soft constraint

$$\mu_{R2}(z) = 1 \text{ if } z = 3$$

$$\mu_{R2}(z) = \alpha_4 \text{ if } z = 2 \text{ or } 4$$

$$\mu_{R2}(z) = 0 \text{ otherwise}$$

Figure 2.11: Modeling of User Requirements: R2(a) and R3(b)

- R3: prioritized hard constraint

$$\mu_{R3}(y) = 1 \text{ if } y = 2 \text{ or } 3$$

$$\mu_{R3}(y) = 1 - \alpha_3 = \alpha_3 \text{ otherwise}$$

- R4: prioritized soft constraint

$$\mu_{R4}(x) = 1 \text{ if } x = 4$$

$$\mu_{R4}(x) = \max(\alpha_4, 1 - \alpha_4) = \alpha_4 \text{ if } x = 2 \text{ or } 3$$

$$\mu_{R4}(x) = 1 - \alpha_4 = \alpha_2 \text{ otherwise}$$

R2 and R3 are illustrated in Fig 2.11. After modeling, set of algorithms, like Branch and Bound algorithm, neural network and genetic algorithms can solve the fuzzy constraint problem. Although there is a lack of detailed experiment simulation in [66] and cases would be far more complex than the example illustrated above, it calls for our attention to possible events when human operators are involved in the direct interaction with Grid environment. In this scenario, fuzzy logic plays an important role to model human's implicit and uncertain linguistics.

Before resource allocation, scheduler may also take user reputation into account to against selfish nodes or malicious nodes who are unlikely to contribute or may intentionally damage resources. Li *et al.* [67] formulate an allocation scheme to consider a combined metric, composed of bid price and trust value. This is actually a hybridized approach from negotiation and trust evaluation. Resource providers allocate resources according to the weighted combined metric of a user. Q-learning technique is applied to predict the optimal policy of the weighted adjustment.

2.3.2.4 Genetic Algorithms

Grid infrastructure demands powerful and intelligent schedulers to accomplish a wide range of applications in response of adaptive and changing environments. In another way, dynamics and adaptiveness can also be achieved through creating evolvable Grid services (applications). The first attempt to apply an artificial immune system into the adaptive computing manifested itself in Lee and Suzuki's work [68]. This approach referred to a kind of complex adaptive system, where a complex system or pattern emerged out of a multiplicity of relatively simple interactions. An application (a bee colony) consists of multiple bees (agents). Each agent implements a function as part of the system, such as reply of html content to http request. Each agent has an attribute (ID), body (an implementation to service), and behaviors that are non-service related actions, including migration, energy exchange and storage, communication, replication, death and swapping (swapping geographical location with other agents). Analogical to immune system, each agent, inside has a network of antibodies (policies) to cope with virus (environment changes). If there is an antigen invasion, agents will firstly detect the antigen as either self or non-self. A non-self signal triggers the antibody network. An antibody connects with one or more, via links through which its concentration value is either driven in from other antibodies or absorbed (as shown in Fig 2.12). Concentration indicates the activeness of a particular antibody. Concentration flow is calculated at each time step in the therapy process. Typically, after 200 iterations, the policy that an antibody with the highest concentration represents will be selected. The adaptive system-wise behavior emerged out of many simple interactions among antibodies.

Furthermore, [69] and [70] enhance previous version in providing a fitness calculation mechanism so as to define a evolutionary process (using genetic algorithm) for reproducing more competent offsprings. Agent's genes are coded as relationships among antibodies and their affinity values (link weight). In light of fitness, agent either mates or replicates itself. Subsequently, agent operates standard genetic algorithm approaches, such as crossover, and

Figure 2.12: A Sample Antibody Network

mutation. Experiment results [68] show that the proposed adaptive Grid service scheme allows agents to dynamically adapt to changing environments for improving their fitness in terms of throughput, workload balancing, and response time. Currently agent design and behavior policies require expert intervention, which is usually unavailable for unpredictable situation.

2.3.2.5 Hybrid Soft Computing Approach

Research by [71] candidates a novel hybrid approach to detect malicious actions by modeling user's behavior. The suggested approach involves a SOM for feature deduction, Fuzzy Neural Network (FNN) for user behavior modeling, and Genetic Algorithm (GA) for tuning FNN. The Intrusion Detection System (IDS) is presented in Fig 2.13, comprising several agents, some of which are responsible for sensing data packet, user identification, intrusion recording, whereas some for feature reduction, intrusion detection and parameter adjusting. Database are used for user behavior model retrieval and intrusion evidence access, etc. The core of the proposed structure is a fuzzy neural network employing TSK defuzzification shown in Fig 2.14. Reduced operation description of a particular user is as input to the FNN, followed by fuzzification, and identification of winning rule (with largest firing strength). A set of TSK coefficient associated with the winning rule maps the user operation inputs into a scaler. The FNN is trained on user's normal behavior, and the output should be greater than 0.9 in order to stop learning. If a predicative measure of user behavior is greater than 0.8 then it is considered as normal, otherwise abnormal and alarms occur. Parameter is subject to adjust if greater than 0.8 but less

Figure 2.13: IDS Architecture

than 0.9, where an online learning is demanded. FNN does not only provides classification results but also introduces a set of fuzzy rules at the end of learning. Due to the complexity of FNN, GA tunes the parameters of FNN, which consists of fuzzy membership functions, rule weights and TSK coefficients. It turns out that proposed architecture is applicable for most types of attacks, and its explicit knowledge base can be examined by external experts, a typical advantage of FNN. This approach has its novelty in modeling user behavior instead of network packet monitoring, useful for massive traffic environment. However, requirement on learning every user behavior model may also result in difficulties in obtaining learning data as well as longer training time span.

Figure 2.14: Adopted Fuzzy Neural Network Structure

2.3.2.6 Autonomous Agent and Multiagent Systems

Typical goals in Grid/Cloud service selection concerns requirements in performance metrics in bandwidth or response speed, etc. To guarantee satisfactory of such QoS, detailed configuration parameter is expected from administrator. For agent-mediated service, policy-based management simplifies and automates the administration of Grid/Cloud environment, since users only need to specify goals instead of detailed and complicated settings. Policy transformation facilitates administrators to map goal-specific to configuration-specific. Agents may leverage a case-based reasoning transformation module [72], which contains a rich set of cases for relating a new set of configurations to goal measurements. Principle Component Analysis (PCA) for feature reduction and k-nearest neighbor clustering for noise removing and case base simplification can be adopted. To enhance online policy transformation, a supervising module adaptively can adjust the case base for adding a relevant case history.

A simplified version of scheduling is to assign m jobs to n resources with constraints that only one jobs can be allocated to exactly one resource, known as job assignment problem or Traveling-Salesman problem. However, as the number of resources and concurrent jobs increases usually in the environment, the scheduling becomes N-P hard. Agents can utilize genetic algorithm, neural networks (e.g. Hopfield) [73], simulated annealing search [74] and chaotic neural network [75] in facilitating the search of optimal solutions in a reasonable time. Multi-Agent System (MAS) is naturally fit for distributed Grid platform. Agent acts as an autonomous entity equipped with intelligent and adaptive behavior will competently handle dynamic requirements and response to uncertain yet possible Grid stat changes. Grids with highly computational power and robust infrastructure requires intelligent agents to be effective in pursuing their goals and to be equipped with flexible and intelligent decision making skills [3]. A hot spot in this realm is negotiation where MAS has a batch of sound theorems. Grid requires effective negotiation between resource owner (seller) and resource user (buyer) to achieve maximum utilization and load balancing. A model of adaptive negotiation framework

[76] summarizes possible approaches in the agent literature that would help to realize Grid-oriented objectives. In this context, soft computing will surely have its values in determining the right time to change negotiation strategy as well as the correct strategy to switch to. For an instance [77], Q-learning is selected to dynamically adapt agent's negotiation strategy to make time dependent decisions.

Besides, trust and reputation, as one critical problem stated in [3], are some other essential factors in the distributed systems involving interactions between participants. Grid concepts treat services as goods on the shelf, and engages in establishing an open "market" (platform) for "customers" (end-users). In this context, trust and reputation for a resource provider or Virtual Organization have strong influence on its benefit. Durad and Cao [78] suggest a set of parameters to quantitatively measure Grid trust. Likewise, neural net and fuzzy logic among several other methods are candidates to construct such modeling.

2.4 Summary

This chapter first presented the development time line of motivation theories from psychology point of view. Self-Determination Theory was identified as the best candidate in terms of our research objectives. Three groups of motivated agent research have been reviewed, and were compared with SDT. Existing research was limited in the categories of intrinsic needs they considered, and most of them merely exhibited the behaviors of novelty-seeking, without understanding of how they trade off between various behaviors. Besides the absence of agent deliberation, current research is merely for toy experiments and lacks of practical usage for real-life problems.

In the latter part of this chapter, the techniques applied in infrastructure resource provisioning were discussed. To handle uncertainties in practical problems, adaptive approaches outperformed classic ones. Examples of adaptive approaches applied machine learning algorithms to handle uncertainty. The techniques varied from neural networks, fuzzy logic to Genetic

algorithms. Soft computing techniques were active in load modeling, resource management, security, etc. The success of deploying adaptive algorithms demonstrated the feasibility of embedding machine learning in industrial applications. However, compared with learning algorithms, agent is more promising due to agent's capability in problem solving in addition to reasoning.

Related work brought real cases for incorporating intelligent solutions into infrastructure services. This chapter offers our research relevant experiments, and numerous ways to setup a simulation. The reviewed approaches focused on the accuracy for predication of network condition, resource availability, and user requirements, multiple organization modeling. On the contrary, agent-based approach emphasizes the usage of intelligent problem solvers and the emergent adaptiveness out of multiplicity of agent interactions, which would provide a border view of employing agents in real-life applications.

Chapter 3

Modeling Competitive Intrinsically Motivated Agents

This chapter emphasizes the modeling of a *Competitive Intrinsically Motivated Agent* from formalization to a goal-oriented approach. It starts with a general agent model, drawing forth essential components in an agent model. The definition of a competitive intrinsically motivated agent is proposed in this chapter. The formal model of the proposed agent follows the Self-Determination Theory (SDT) by considering agent's motivations, the trinity of autonomy, competence and relatedness and the relationship blended among motivations, intrinsic needs, internal states, external events and agent behaviors, including actions and reasoning.

To bridge the gaps between agent theoretical model and practical use, a goal-oriented representation, GoalNet, is applied to describe the agent in an integrated and systematic fashion. The design of a motivated agent is simplified, and becomes manageable by adopting the GoalNet representation; the proposed approach decomposes a complex task in a top-down manner, where a hierarchy of sub-tasks is managed conveniently and comprehensively. The concepts of a competitive agent are mapped to elements of the GoalNet methodology; and agent motivated reasoning is mapped to the goal selection and action selection of GoalNet.

This chapter explains, from the conceptual level, how intrinsic motivations allow agent adaptiveness to be clearly defined and designed as motivated goal selection and action selection. Three categories of motivations are identified, and corresponding agent plans are defined

in various GoalNets. In particular, agent's learning behavior displays its competency over mastering new skills; agent's accomplishment of routine jobs shows its striving for excellence over performance; agent's social interactions, such as cooperation and coordination, expose its endeavor for relatedness with other agents; and finally agent's decisiveness on goal/action selection exhibits its pursuit of autonomy.

3.1 Theoretical Background of Agent Modeling

The section gives a definition of a generic agent models firstly, continues explaining the pattern of interactions among various modules, and highlights the importance of agent's reasoning mechanism. The reasoning differentiates a diversity of agent research, and it is the focus of this thesis as well.

Intelligent agent is a buzz word, which has different definitions among a variety of researchers and organizations. Some definitions emphasize a particular aspect of agent, which serves their own interests in a defined domain. We adopt the definition from Franklin and Graesser [1], which rules out a agent from a piece of program, and the description of an agent is given in Chapter 1.

The agent definition entails several aspects of an intelligent agent. An agent inhabits in a certain environment, which is either fully or partially observable, episodic or sequential, deterministic or stochastic, discrete or continuous, and static or dynamic. In this environment, the agent requires to be capable to perceive and act, and sometimes to be of some domain knowledge. In this context, environment refers to the problem domain, and an agent would not work if the domain switches. This thesis is not attempting to rigidly offer one solution to all problems, but agents should have the ability to adapt to the environment in an acceptable extend, in the similar application domain.

Moreover, the agent does not requires other involving party to provide its input or use its output. To achieve agent's autonomy (different from psychological autonomy needs), an agent

Figure 3.1: Generic Agent Model. Original picture is from [79]

has to be stimulus-independent, and inputs from outside only modulate agent's interior ongoing activity [2]. An agent should determinedly acts in pursuit of its own agenda (goals and sub-goals). Each agent acts to the environment, so that its current action may have an effect on its later sensing. This characteristic often distinguishes an intelligent agent from a program, for the reason that a program, for example a payroll program, would not normally change the state of environment and what it perceives later [1].

In addition, an agent has the property of temporal continuity, and acts continually over a period of time, whereas a program, once executed, would wait to be called again. The mentioned agent definition is not quite differentiable from other definitions; however, in our opinion, it spells out some most important characteristics of an agent. Instead, other agent properties such as communication, learning, flexibility, mobility, etc, are useful to classify agents into diversified categories, but they are excluded in our main considerations of defining an agent.

There are many agent models that exist in the agent literature. However they are sharing one common characteristic, for which an agent is modeled as an active entity that continuously percepts environment, makes decisions, and acts on environment. The generic agent model is shown in Figure 3.1. The decision making process is marked by the box with question mark and it presents internally there is a concrete implementation of agent function. The agent function maps any perception or any sequence of perceptions to an action or an action plan, which constitutes the agent's behaviors.

Environment is one prominent constituent for agent design. It specifies the problem domain which the agent is designed for. In real-life scenario, the states of environment is not practical to be perceived to infinite level of details, either due to the constraints of sensory devices or limitation of computational resources. Environment usually has a finite set of states, that are discrete and instantaneous. A continuous environment can be discretized to any desired degree of accuracy. Therefore, we have $\mathbf{E} = \{e_1, e_2, \dots, e_i, \dots\}$ where environment is the set of discrete states e_i . The agent that perceives the environment and acts accordingly can be modeled as follows.

Definition 3.1 *An agent is formally specified as a tuple $Ag = (\mathbf{P}, \mathbf{A}, \mathbf{R}, \mathbf{E})$, where*

- $\mathbf{P}$ *is the set of percepts that the agent senses about the environment,*
- $\mathbf{A}$ *is the set of primitive actions that the agent can take,*
- $\mathbf{R}$ *is the set of reasoning mechanisms that the agent follows to make decisions,*
- $\mathbf{E}$ *is the set of environment states that an agent interact with.*

Percepts are not simply duplicates of environmental states, but they are agent's projections from external objective facts into internal subjective beliefs. Therefore, $\mathbf{P} = \{p_1, p_2, \dots, p_i, \dots\}$ where $p_i = \text{perceive}(e_i)$. The perception function *perceive* maps environment states to percepts $\text{perceive} : \mathbf{E} \rightarrow \mathbf{P}$. The mapping is not static and may change over different agent internal states. For example, resource selection agent may see the arrival of job request differently when doing allocation task or learning scheduling strategy. In the other hand, the arrivals of two different jobs may be perceived as one job for the agent, given the agent is interested in the common attributes of the jobs only.

An agent acquires a set of possible actions available to it, $\mathbf{A} = \{a_1, a_2, \dots, a_i, \dots\}$ where a_i is the primitive action, and it is considered as atomic since no further decomposition is possible. The action set is finite to any desired accuracy and changeable while the agent evolves. An agent can execute atomic actions directly. It is possible for the agent to have meta-actions composed by several atomic actions to facilitate hierarchy and easy management. But in this

modeling we do not consider decomposable actions in the action set $\mathbf{A}$. Instead, meta-actions are usually modeled as plans in some reasoning mechanisms $\mathbf{R}$.

The reasoning mechanisms, $\mathbf{R} \subseteq \mathbb{R}$, where $\mathbb{R}$ denotes a universal function space of agent reasonings, are central to agent intelligence and autonomy. A mechanism $r \in \mathbf{R}$ defines the agent behaviors in the course of interacting with environment. There are various mechanisms available in the literature, and according to Russell and Norvig [79], they classified agents into four categories, from the least to the most complex.

Simple reflex agents react to immediate stimuli, and have no memory at all whose reactive behavior is produced by set of IF-THEN rules. An action is carried out if the antecedent of a rule is satisfied. Therefore, the principal benefit is the fast response of an agent to determine the next action. Therefore this rule-based r contains a rule base $\mathbf{Rule}$ and a rule searching function $f_{search} : \mathbf{P} \rightarrow \mathbf{A}$, i.e. $r = (\mathbf{Rule}, f_{search})$.

For a model-based reflex agent, the reasoning mechanism r has a memory of states $\mathbf{Memory}$, which reminds them of earlier experiences. This more sophisticated version keeps track of how the world evolves, and acts in the light of existing knowledge of the states.

Goal-based agents can plan ahead before taking any action. They are usually associated with a set of plans $\mathbf{Plan}$, which define the pre-conditions $\mathbf{P}$ and post-conditions $\mathbf{Goal}$ and corresponding sequence of actions $\mathbf{A}$ to achieve such state transition. Therefore, r involves a evaluation function $f_{eval} : \mathbf{Plan} \rightarrow \mathbb{R}$, where $\mathbb{R}$ denotes real numbers, to understand whether a candidate plan $plan \in \mathbf{Plan}$ helps to achieve the goal $goal \in \mathbf{Goal}$.

Last but not least, utility-based agents are modeled at a much higher level to meet the criteria $f_{utility} : \mathbf{P} \rightarrow \mathbb{R}$, where $\mathbb{R}$ denotes real numbers, measured in happiness and preference. As a consequence, r adopts a utility measure which guides an agent to make rational decisions when multiply goals are conflicting or the likelihood of success has to be weighted against their importance [79].

3.2 Formal Model of Competitive Intrinsically Motivated Agents

This section first introduces the models of intrinsically motivated agents. Components and relationships are explained. Based on the proposed model, a set of concepts related to the agent are defined. In addition, typical task cycles are formulated and illustrated.

Figure 3.2: Proposed Competitive Intrinsically Motivated Agent Model

3.2.1 Overview of the Proposed Motivated Agent

In this section, we propose a conceptual model for a motivated agent and the proposed agent model is shown in Figure 3.2.

The conceptual model is based on SDT theory. The proposed model constitutes essential components for describing the relationships among agent's motivation, needs and behaviors. Detailed explanations and definitions are given in the rest of the section.

The environment (states of the world) is sensed by *sensors*, and then the sensory information is perceived by the agent as a set of percepts **Percept**, which contains agent beliefs of the environment. **Motivation Interpreter** computes the intensity of each agent's motivation

Motivation, followed by generating a motivation with the maximum intensity. The computation of motivation intensity takes inputs from **Percept** and the agent's intrinsic needs **Need**. According to SDT, the three intrinsic needs include *competence*, *relatedness* and *autonomy*, whereby competence needs are further categorized as *novelty* driven and *performance* driven. Correspondingly, an agent has motivations as the pursuit of novelty, performance, relatedness and autonomy. Since **Need** is an intrinsic characteristic of an agent, the computation of needs is independent of external stimuli. Driven by an explicit motivation, an agent will behave and act accordingly, and go through a deliberative process for planning. An agent's motivated behaviors comprise goal setting, goal selection and goal realization by identifying a course of actions. Agent's motivated behaviors are considered at different levels: high-level goals are managed in *Goal Elicitor* and low-level actions are handled in *Action Elicitor*. The functionality of each components are elaborated as follows,

Goals serve as high level plans of an agent, which are considered as part of its knowledge **Knowledge**. The agent driven by motivation actively decides which goal to select, so that the fulfillment of the goal will satisfy the pursuit of needs. In *Goal Elicitor*, the decision process requires input from agent's derived motivation, percepts of the environment and plan knowledge. A high-level goal may contain many sub-goals in its hierarchy, and then a recursive decomposition process is carried out to identify more fine-grained goals. Decomposition terminates until atom actions exist for all refined goals. Once a fine-grained goal is intended, it is implemented in *Action Elicitor*, which selects executable motor skills from agent's **Knowledge**. The selection involves a similar reasoning process as goal selection, taking input from motivations, goals and percepts.

At the end of reasoning, a detailed and executable action sequence is forwarded to *Actuator* module. The course of actions specified by the *Action Elicitor* is stacked, and finally agents act to environments via actuators.

Similar to other agent modeling, the reasoning module is the center component in the agent modeling. The selection of plans to be executed refers to a *deliberation* process, while planning on how the goals are to be achieved refers to *means-ends reasoning* [80]. In this model, we do not differentiate the two types of planning, and for both motivation analysis is critical for motivated reasoning process. In the following section, we give a formal definition of the competitive intrinsically motivated agent as well as the motivated reasoning process.

3.2.2 Proposed Agent Definitions and Concepts

Definition 3.2 A *Competitive Intrinsically Motivated Agent* is formally defined based on the generic agent definition as in Definition 3.1. $Ag^M = (P, A, r^M, E)$, where the motivated reasoning mechanism r^M is a tuple of (M, N, K, MI, GE, AE) , and

- M** is the set of motivations that potentially drives an agent,
- N** is the set of intrinsic needs that an agent intrinsically has,
- K** is the knowledge of an agent that holds for directing its behaviors,
- MI** is the motivation interpreter for deriving agent's motivations,
- GE** is the goal elicitor for goal selection and decomposition,
- AE** is the action elicitor for action selection in implementation realization.

The motivated reasoning mechanism r^M is the differentiating factor from other agents. The rest of this section explains the essential components of the reasoning mechanism in light of SDT concepts.

Intrinsic Needs

Definition 3.3 According to SDT, intrinsic needs of a motivated agent can be expressed as $N = \{Competence, Relatedness, Autonomy\}$.

- $Competence \in N$ denotes agent's competence need which relates to skills mastered (Novelty) and performance-based achievements (Performance),
- $Relatedness \in N$ denotes agent's relatedness need which relates to contacting and collaborating with other agents,
- $Autonomy \in N$ denotes agent's autonomy need which relates to carrying out self-determined behaviors and having self-generated plans.

The level of needs determines the extend to how an agent is intrinsically motivated. Every motivated agent has intrinsic needs, but they may have their own ways to compute the need levels, and have customized motivation computing by mapping need values to motivation intensity (discussed in Section 3.2.3).

Motivations An motivation in $\mathbf{M}$ of an agent is innate, instead of designed by agent designer. It describes the intrinsic inspiration that drives an agent to act during its deployment in an environment.

Definition 3.4 *A competitive intrinsically motivated agent inherently has a explicit set of motivations $\mathbf{M}$. A motivation is defined as the pursuit of the underlying intrinsic need. Motivations have following properties,*

1. *By definition, M_i is defined as the pursuit of N_i , where $N_i \in \mathbf{N}$,*
2. *The set of an agent's motivations is equal to the number of intrinsic needs that agent has, i.e., $|\mathbf{M}| = |\mathbf{N}|$,*
3. *A complete set of $\mathbf{M}$ is $\{\text{Pursuit of Novelty}, \text{Pursuit of Performance}, \text{Pursuit of Relatedness}, \text{Pursuit of Autonomy}\}$.*

Agent Knowledge Knowledge $\mathbf{K}$ of a competitive intrinsically motivated agent serves as a knowledge repository of the agent. It acts an important component for agent's deliberation process and behavior generation. Types of agents have their own definition of knowledge; particularly, the knowledge of an motivated agent is defined as follows,

Definition 3.5 *A competitive intrinsically motivated agent has a variety of knowledge in knowledge base $\mathbf{K}$. There exist three subsets of knowledge, namely **Plan**, **Domain Knowledge** and **Runtime Data**, where*

Plan	<i>consists of a set of plans that are executed by the agent to reach the intended goals,</i>
Domain Knowledge	<i>contains tools and algorithms for solving a specific problem,</i>
Runtime Data	<i>stores temporary status or data, keeps track of deliberative process, and historical experience.</i>

Plan may contain a set of plans by executing which the underlying goals are to be achieved.

A plan definition can be shown as,

Definition 3.6 *A plan $Plan_i$ has following properties, identifier, goal, context, body and options; specifically,*

<i>Identifier</i>	<i>is the unique identification of the plan,</i>
<i>Goal</i>	<i>is the identification of the goal to be achieved by the plan,</i>
<i>Context</i>	<i>is the set of conditions that are to be necessarily satisfied for the plan to be valid,</i>
<i>Body</i>	<i>is the content of the plan,</i>
<i>Options</i>	<i>is the field for specifying supporting information, configuration, exception, performance metrics, etc.</i>

A plan body may contain only ground atoms, which are immediately executable by agent implementations. In this case body contains a sequence of actions $A_1, \dots, A_n$, where $A_i \in \mathbf{A}$ denoting a set of admissible actions, and n is the number of steps of the plan. On the other hand, a plan body may involve other plans, in which another round of plan invocation is required to determine the immediately admissible actions. Plans are searched by underlying goal specification, indicating a plan is only required if the goal of the plan is intended by some other plans. Plans are activated only the context of the plan are necessarily fulfilled. The body of the plan is adopted either by evaluating action impacts in agent's action elicitor **AE** or by analyzing goal impacts or searching for sub-goals in agent's goal elicitor **GE**.

Domain Knowledge can be invoked by special types of actions. This kind of knowledge represent the skills and a tool set that can be directly exploited by an agent. For example, a neural network model is a domain knowledge for forecasting. Agents can immediately apply a set of special actions, such as “feed input data to the neural network” and “execute this network model”. The agent then continues to expect the response from the neural network in the next sensing cycle. However, to adopt this knowledge requires agents to have specific involvement to build or acquire such domain knowledge. For example, the agent may have to

“train” the neural network with an explicitly-defined input format, which is specified by the domain knowledge.

Runtime Data can be thought of a memory for maintaining agent intermediate status, historical experience and progress of deliberation process, and it is used for agent self-regulation, monitoring of deliberation and behaviors.

3.2.3 Computing in Motivated Agent Reasoning

The motivated reasoning mechanism $r^{\mathcal{M}}$ is crucial for agents to have intrinsically motivated behaviors. It has the unique reasoning mechanism that reflects environmental stimuli $\mathbf{P}$ and the agent’s internal states (as $\mathbf{M}$ and $\mathbf{N}$). In other words, behaviors of this type of agents are decided by extrinsic events in conjunction with agent’s intrinsic status.

Definition 3.7 *Computes in $r^{\mathcal{M}}$ involve both values and functions. Their definitions are given as follows.*

$\mathcal{R}$	The set of real numbers,
$\mathbb{F}$	The universal of functional spaces,
$\mathcal{V}^{N_i} \in \mathcal{R}$	Level of N_i ,
$f^{N_i} \in \mathbb{F}$	Characteristic function for $N_i \in \mathbf{N}$, where $\mathcal{V}^{N_i} = f^{N_i}(t)$, meaning the level of N_i is sampled at time t ,
$\mathbf{w} \in \mathcal{R}^{ \mathbf{N} }$	Weight vector of each intrinsic needs, where $ \mathbf{N} $ is the size of $\mathbf{N}$,
$f^{\mathbf{MI}} \in \mathbb{F}$	Characteristic function of motivation interpreter, used to determine the weight vector,
$\mathcal{V}^{M_i} \in \mathcal{R}$	Intensity of a motivation ($M_i \in \mathbf{M}$),
$\mathcal{V}^{G_i} \in \mathcal{R}$	Impact of a goal ($G_i \in \mathbf{G}$) on a motivation,
$\mathcal{V}^{A_i} \in \mathcal{R}$	Impact of an action ($A_i \in \mathbf{A}$) on a motivation,
$f^{\mathbf{G} \sim M_j} \in \mathbb{F}$	Function for impact evaluation over a set of goals $\mathbf{G}$ on a motivation M_j ,
$f^{\mathbf{A} \sim M_j} \in \mathbb{F}$	Function for impact evaluation over a set of actions $\mathbf{A}$ on a motivation M_j .

From a functional perspective, an agent with reasoning mechanism driven by motivation, $r^{\mathcal{M}}$, has following characteristics,

- The agent evaluates the values of its intrinsic needs, which are determined by agent characteristic functions f^{N_i} . The characteristic function is assumed independent of external and takes no inputs from environment.
- The agent has an explicit set of motivations, and has weights associated with each motivation. The values of weights is controlled by the compute module within **MI**, which considers perceived external inputs **P** and the levels of intrinsic needs **N**.
- The compute of **MI** calculates weights as $\mathbf{w} = f^{\mathbf{MI}}(\mathbf{P}, \mathbf{N})$. **MI** chooses the motivation with the maximum intensity, M_i , such that $M_i = \arg \max(w_j \times \mathcal{V}^{M_j})$, where $\forall M_j \in \mathbf{M}$.
- The compute of **GE** and **AE** involves a set of strategies for goal and action selection. Goals and actions are ranked by their impact on the fulfillment of underlying motivation. The impact of goals $\mathcal{V}^{G_i}, \forall G_i \in \mathbf{G}$ and actions $\mathcal{V}^{A_i}, \forall A_i \in \mathbf{A}$ are assigned according to the variety of motivations.

3.2.4 Modeling Agent Main Routine Cycle

Generally, an agent, after deployment, runs its tasks repeatedly in a cycle. Such execution forms a Perceive-Reason-Act (PRA) cycle:

1. Perceive: The agent perceives its environment continuously to get sensory information.
2. Reason: The agent infers actions bases on its knowledge and sensed input.
3. Act: The agent executes the intended actions.

More specific than a general agent, a competitive intrinsically motivated agent has its main routine executed in a cycle, which is of a PM(N)R²A style.

1. Perceive: The agent perceives its environment continuously to get sensory information.

2. Being Motivated: The agent interprets its motivations based on sensed input and intrinsic need levels.
3. Reason for Goal Elicitation: The agent infers the next goals and decompose goals based on its motivation and sensed input.
4. Reason for Action Elicitation: The agent infers actions bases on its motivation, goals, knowledge and sensed input.
5. Act: The agent executes the intended actions.

Digging into the chain of R^2A , it encapsulates a variety of sub-tasks, including learning, seeking for performance, and social interacting, etc. The details of sub-tasks are explained in next chapter as agent's plan for need pursuit.

3.2.5 Example Modeling: Motivated Resource Selection Agent

To make formal model more comprehensive, an illustrative example is given in this section. The example is based on a simplified scenario in the context of resource allocation.

3.2.5.1 Background

Distributed resource provisioning becomes popular soon after its emergence in the late of last century. One example is the adoption of Grid and Cloud computing. By combining heterogeneous computational resources all over the world, Grid/Cloud will have a similar transforming effect to power grid, in favor of new classes of applications to emerge [62]. A scheduler is an essential component which provides job and resource management, and however it is operating on a global level with incomplete and sometime inaccurate resource information [81]. Therefore, in the dynamic and uncertain environment, how to select a resource for a particular job is a challenging issue. In this working example, we focus on resource selection for Grid computing, which involves decision making for task-and-resource matching, (different from Cloud computing, for requests are relatively long-term, compared with short lifespan Grid jobs).

In a Grid environment, there are two levels of resource allocation, one for Grid schedulers and the other for local resource schedulers [82]. The two schedulers are different in functionalities, for which a Grid scheduler searches resources over multiple remote sites, and allocates the job to one of them, while a local resource scheduler actually manages the local resources and does actual execution according to its local policy. Since in general local resources are unlikely to surrender ownership or full control to other parties, Grid scheduler can only make its best effort to select a resource. Therefore, it is difficult to provide any assurance that a job is given a guaranteed quality of service level.

Modeling a Grid module for resource selection as an agent offers a great level of autonomy and flexibility. As its name suggests, agents can be easily presented as a delegate of human operators, and scheduling behavior can be completely hidden from Grid architecture. Therefore, designers only need to focus on communication between Grid modules. The key idea of directly programming agents in terms of mentalistic notions (such as motivations, needs) that agent researchers have developed to represent characteristics of agents is to reach an abstraction of properties of complex system in the way to mimic the behaviors of humans. Decision making is a desired feature of resource selection agents, since task-and-resource matching usually involves complexities, especially in the dynamical Grid environment. Some existing systems have implemented a resource selection component in Grid scheduling [83], [84], [46], [45], but they lacked an intelligent agent model which supports adaptation in the Grid dynamics.

Later part of working example will illustrate the proposed modeling of motivated agents. Definitions and proposed models will be applied in the context of the demonstrated scenario for easy understanding.

3.2.5.2 Generic Modeling for Resource Selection Agent

The purpose of resource selection agent is to automatically provide suitable resource for a given job. After deployment, the agent continues sensing job request from the environment. A

detailed problem formulation (i.e. the environment) for resource selection in Grid is presented in this section.

The primary environment $\mathbf{E}$ contains an unknown but finite set of job arrival events, so $e_i \in \mathbf{E}$ and e_i is the arrival of job J from the set of possible jobs.

The agent's percepts $\mathbf{P}$ towards a submitted job J in a predefined format. J is defined by job property $property_J^{job}$ and job requirement req_J^{job} .

Consequently, we have $J = \langle property_J^{job}, req_J^{job} \rangle$, where

$$property_J^{job} = \{ \langle p_J^1, expr_J^1 \rangle, \dots, \langle p_J^{m_3}, expr_J^{m_3} \rangle \}$$

and $req_J^{job} = \{ \langle r_J^1, expr_J^1 \rangle, \dots, \langle r_J^{m_4}, expr_J^{m_4} \rangle \}$. For example, a job submitted by a user from *Organization A*, which consists of one thousand lines of code, and requires more than two CPUs to be utilized. According to above formulation, this job can be represented as,

$$J = \langle \{ \langle user, A \rangle, \langle scale, 1000 \rangle \}, \{ \langle cpu_number, \geq 2 \rangle \} \rangle$$

The action set $\mathbf{A}$ available to the agent is the set of possible choices of resources, meaning that the agent can allocate any one resource from the resource set for given a job. Let $\mathcal{R} = \{R_1, \dots, R_n\}$ be the entire set of Grid resources. Each of them is characterized as resource properties and resource requirements $R_i = \langle property_i^{res}, req_i^{res} \rangle$ for $1 \leq i \leq n$ and $n = |\mathcal{R}|$. Resource property $property_i^{res}$ contains multiple pairs of property identifiers and corresponding expressions. Therefore, it can be expressed as

$$property_i^{res} = \{ \langle p_i^1, expr_i^1 \rangle, \dots, \langle p_i^{m_1}, expr_i^{m_1} \rangle \}$$

where $m_1 \in \mathbb{N}_1$ and $\mathbb{N}_1 = \{1, 2, \dots\}$ denotes the set of natural numbers. Similarly, resource requirement req_i^{res} is a collection of pairs of requirement identifiers and corresponding expressions. We have $req_i^{res} = \{ \langle r_i^1, expr_i^1 \rangle, \dots, \langle r_i^{m_2}, expr_i^{m_2} \rangle \}$, where $m_2 \in \mathbb{N}_1$. For example, a

candidate resource with 1.0GHz *CPU* and 256Mb *Memory* and with restricted access within *Organization A* can be described as,

$$R_1 = \langle \{ \langle \text{cpu}, 1.0\text{GHz} \rangle, \langle \text{memory}, 256\text{Mb} \rangle \}, \{ \langle \text{access_within}, A \rangle \} \rangle$$

The agent reasoning mechanism r is simply adopted as minimum requirement filtering followed by resource priority assignment. Before selecting a resource, Grid scheduler does a minimum requirement filtering, which shortlists available resources by comparing resource requirement req_i^{res} and job property $property_J^{job}$ for $1 \leq i \leq n$, and $n = |\mathcal{R}|$ and vice versa. In previous examples, the access requirement $\langle \text{access_within}, A \rangle$ is fulfilled by the property $\langle \text{user}, A \rangle$ of job J . However, the requirement $\langle \text{cpu_number}, \geq 2 \rangle$ is not satisfied by the property of R_1 , which only has one available CPU, so R_1 is filtered out.

Given above formulations, a resource allocation policy for Grid scheduler can be defined as

$$\Phi(\text{preference}) : \mathcal{R} \times \mathcal{J} \rightarrow \mathbb{N}_1 \quad (\text{Eq. 3.1})$$

where $\mathcal{J} = \{J\}$ referring to the set consisting of the job to be scheduled. The preference *preference* can be measured in a single criterion such as job execution time and financial cost, or a combination of multiple criteria. Preference may change over different applications. Therefore, it is application-dependent, and is often implied in the context or explicitly specified in the job description. For simplicity we assume all jobs are sharing one common preference, which is known from the context. Note that $|\mathcal{J}| = 1$, since resource allocation in this example considers only one job at a time. The set of positive integers $\mathbb{N}_1$ denotes the index of a selected resource. For the underlying preference, a policy is defined as a mapping from a set of candidate resources to a selected resource, by considering the job submitted. The resource allocation policy should require minimum user intervention in order to comply with Grid's transparency. Furthermore, the scheduling policy should be able make fast and intelligent decisions especially in a dynamic and uncertain Grid environment. As a consequence, how to design such a policy becomes vital for the success of Grid scheduler.

Existing schedulers [83–86] either merely consider minimum requirement filtering, or require users to specify preference of a resource. For example, Condor [83] provides an option for users to specify a rank expression of resources, such as $Rank = CPU_speed + Memory_size$. Using above definitions, the policy of using ranking heuristics can be modeled as,

$$\Phi : \hat{k} = \arg \max Ranking(R_k), \text{ for } 1 \leq k \leq |\mathcal{R}| \quad (\text{Eq. 3.2})$$

The problem with current mechanism is that this policy requires users to have sufficient prior knowledge for a particular problem in order to determine the ranking function. Meanwhile, the defined rank expression cannot directly reflect how a job will be executed. For example, if a resource with higher computing power is preferred, however it is not necessary that a machine with bigger sum of CPU_speed and $Memory_size$ is more powerful. In fact, a simple addition cannot capture the complex nonlinear relationship between resource properties and computing power. To solve this problem, learning can be adopted to approximate the underlying mapping. Meanwhile, learning can be used to directly predict how a job will be executed on a particular machine. Therefore, a prediction-based scheduling policy is a better candidate. Prediction-based reasoning mechanism is beyond the scope of this modeling example, and the details will be given as an instantiation of competence pursuit in the next chapter.

3.2.5.3 Proposed Motivated Modeling for Resource Selection Agent

In this scenario, the resource selection agent has intrinsic needs in,

- Competence for getting positive feedback, meaning the agent would like to meet job requirements by scheduling a resource to carry out the job execution, and
- Competence for mastering new skills, meaning the agent would like to learn new methods which potentially facilitate its decision making, and

- Autonomy for having controls over its own plans, meaning the agent would like to develop its own plans and solutions based on the knowledge it acquires during the course of scheduling, and
- Relatedness for collaborating with other agents, meaning the agent would like to communicate with other Resource selection agents and work together to share knowledge and decisions.

Assume for a simplified resource selection agent, there is a single objective of performance to achieve, which is "dispatch a job to a resource". Therefore we have $goal^{RS} : < act : dispatch; obj : job; option : [act.target : resource] >$. In general, we have $\mathcal{V}^{\mathcal{M}_C} = \alpha \times \mathcal{V}^{\mathcal{N}_C}$, $\mathcal{V}^{\mathcal{M}_R} = \beta \times \mathcal{V}^{\mathcal{N}_R}$, $\mathcal{V}^{\mathcal{M}_A} = \gamma \times \mathcal{V}^{\mathcal{N}_A}$, where $\alpha, \beta, \gamma \geq 0$ and $\alpha + \beta + \gamma = 1$. For the simplified case, we design $\alpha = 1$ and thus $\beta, \gamma = 0$ by the function of motivation interpreter ($f^{\mathbf{MI}}$), so that we have the resource selection agent concentrated on competence while ignoring other intrinsic needs. The agent designed in this manner is performance-centric and keen to learn new skills.

In this example, we want to demonstrate the modeling of adaptive behavior in terms of learning used in the resource selection plan. Learning behavior extends agent's capability in adapting changing environments; it is important for meeting intrinsic needs especially for autonomy, since learning allows the agent to acquire its own knowledge without external interference. Learning behaviors, just as other agent mental activities, define part of agent behaviors, instead of all of them. Although learning and applying learning results are important aspects of agent reasoning, a learning module should complement, rather than complete, agent mental activities. Learning-based intelligence should be synthesized seamlessly into agent modeling, just as agent's sensor model and actuator model. Meanwhile, learning must consider agent's mental states, so that agent's learning behavior follows the pursuit of its own goals, instead of a simple reflection of external stimuli. Agents must have the ability to decide when to apply

Goal :
ACHIEVE resource_selection(job_id);

Sense:
FACT job_pending(job_id);
FACT job_attributes(job_id);
FACT resource_nature(resource_set);

Act :
EXECUTE allocate(job_id,selected_resource_id);

Plan : {
Id: 001;
Goal: ACHIEVE resource_selection(job_id);
Context:
 FACT job_pending(job_id);
 FACT job_attributes(job_id);
 FACT resource_nature(resource_set);
Body:
 selected_resource_id = knowledge_query(FACT job_attributes(job_id), FACT
resource_nature(resource_set)) ;
 EXECUTE allocate(job_id,selected_resource_id);
}

Figure 3.3: Sample Fragment of Motivated Agent Plan

learning to achieve its goals, and must be able to handle possible goal conflicts. Therefore, a lack of a learning module within a well-defined agent architecture will inhibit the competence of machine learning approach to be applied in agent applications. Therefore, we incorporate a learning subtask in a plan to enable the plan to take account of learning outcome.

Figure 3.3 illustrates a fragment of proposed agent model which is inspired by the work in [87]. The goal of the plan is to do resource selection for a particular job indicated by *job_id* with expected status “Achieved”. Therefore, it is the root plan that would be executed once the agent is deployed. A set of environment interpretation results are spelled out in the interpretation section, including whether there is a job pending for scheduling, what is job requirement and specification of the pending job, and what are the resource status in a resource pool. The sample plan is identified by the plan identification number 001. Plan 001 is searched from

the plan library by querying the agent's knowledge base by asking "which plan can achieve the goal of selecting a resource for *job_id*". After identify the plan from the plan library, the context of plan is checked against the interpreter module, for the validity of execution. For Plan 001 to be executable, the agent must received the information listed in the context section. A job must be waiting for scheduling with known requirements and specification. A set of resource information must be acquired from the resource pool. The sample plan body allocates the pending job to a selected resource, by querying a learning-enabled decision module. This module makes resource planning based on received job requirements and resource status. The actions specified in the plan body are to be stacked in the Execution module. Once the actuator successfully accomplishes the list of actions required, The "Achieved" status of this scheduling task can be assigned and recoded by the agent.

The section is meant for introducing motivated agent modeling for resource selection scenario. The plan illustrated in the working example demonstrates how agent adaptiveness can be incorporated into agent implementation. However, as the complexity of application increases, such conceptual modelings will be not effective, for it lacks a support in modeling sophisticated plans and selections, for which decisions for resource selection are made on the fly. To bridge this gap between theoretical models and practical use, a goal-oriented representation for self-determination agent is given in the rest of this chapter.

3.3 Goal-Oriented Representation of Competitive Intrinsically Motivated Agents

The modeling proposed formalizes the concepts and identifies the necessary components for constituting a motivated learning agent. However, there is a gap between motivation modeling and a solution for a practical problem. This section proposes to adopt a goal-oriented methodology to aid the designers to model and specify motivated agents. The proposal reduces the complex task of building an automatic system into one generic main task and several sub-tasks.

We first discuss relationship between motivation and goal from the psychological point of view, and then bring the discussion to computer science field by linking motivated agents and a goal-oriented approach GoalNet. After that, a generic main task model is proposed for motivated agents. The main task model serves as the agent routine, and is complemented by the design of sub-tasks for achieving adaptive behaviors..

3.3.1 Motivation and Goal

There is a close relationship between motivation and goals according to psychology findings. A goal or objective described by [88] is “a desired result a person or a system envisions, plans and commits to achieve, which is a personal or organizational desired end-point in some sort of assumed development”. We can infer that a goal is a system state an agent is expected to reach. In a complex system, goals are analogous to mile stones along a road map. Motivation is valued as a process that contributes to the creation of action intentions (plans), or equivalently, “motivation refers to goal setting, and volition refers to goal pursuit that is seen as a process that leads from intentions to actual behaviors [89]”.

Goals help a person or a system to identify the path toward achieving them by prescribing the behavior that is required to achieve them, while motivation sustains people’s efforts to stay on the path [90]. Driven by motivations, goals are being set. Motivation can be seen as a driving force to form the goal path, while a sequence of goals defines a means to achieve that motivation.

As explained in [31], motivation involves striving for definable goals, and it provides the impetus for goal-directed behavior. Goals can be set to satisfy virtually any need that we value. When such striving is organized, systematic, and based on sound psychological principles, success is maximized. Of all the approaches advanced by psychologists to enhance motivation and improve performance, none approaches systematic goal setting in its demonstrated effectiveness in a wide range of performance settings, from business to education, from science and medicine to sports. Stated simply, goal setting is a motivational technique that really works.

Table 3.1: Conceptual Relationship Between Motivation and Goal

Criteria	Motivation	Goal
taxonomy	tendency	system state
objective	sustaining the consistency of efforts	identifying steps to achieve the desired state
functionality	guiding goal setting	guiding behaviors
style	abstracted high level	abstracted & concrete
hierarchy	flat	flat & vertical
accessed by	meta-plans	meta-plans & decomposed plans
design role	agent designer	agent designer & agent self
updatable	statistic	dynamic

One of the mechanisms underlying the effectiveness of goal setting is self-efficacy, the belief that one is capable of carrying out the specific behaviors to attain one's goals. As positive self-efficacy beliefs are consistently related to success in behaving effectively and achieving goals, social cognitive theorists have been strongly interested in practical measures of enhancing self-efficacy. Psychologist found that when people are successful and when they attribute their success to their own competencies, their self-efficacy increases and assists them in subsequent goal-directed efforts. Moreover, successful people incline to master the skills by setting challenging but realistic goals, reflecting on what they need in order to achieve the goals, and making commitment to proceed. By realizing the goals one has set, one becomes more knowledgeable and increases one's sense of personal efficacy. Goal setting also helps meet the needs for competency and autonomy emphasized by SDT, because on one side it improves one's capabilities, and on the other side, enhances self-directed and personal control.

As a summary, the difference and similarity between motivation and goal from a psychological point of view can be conceptualized as in Table 3.1. Based on the relations between the

Figure 3.4: GoalNet Model Components

two concepts, it is possible to link their variances in computer science in the next section.

3.3.2 A Brief on GoalNet

In order to realize the transformation from models to design and implementation in a practical usage, GoalNet is adopted as a modeling tool by motivated agent designers. The definitions and characteristics of GoalNet are introduced first, followed by the proposed mapping from agent model to GoalNet based agent design.

GoalNet [91] is a goal-oriented modeling approach. A typical GoalNet is illustrated in Figure 3.4.

Definition 3.8 *The modeling approach of GoalNet is a graphical representation of agent goals and actions. The GoalNet can be denoted as a tuple $(S, T, Arc, Br, token)$, where*

- S is the set of goal states, that are intended for agents to achieve, (as rounded corner rectangles, whereas special states including start and terminated states are denoted as solid cycles)
- T is the set of transitions, that are used for representing tasks, by performing which agents can transit the system states from one to another, (as vertical bars)
- Arc is the set of arcs, that connect states to transitions, specifying the logical relationship between states and transitions, (as arrows)
- Br is the set of branches, that denote the boundary of sub GoalNets of a composite goal state, (as dashed lines)
- $token$ present dynamic behaviors of a process and the goal pursuing status. (as a black dot)

GoalNet modeling has following properties,

- **Hierarchy**— Since a state can either be atomic or composite, GoalNet allows hierarchical structure representation in the GoalNet model. A higher level of goal can be decomposed into lower-level sub goals which are related to agent's concrete behaviors.
- **Temporal**— To facilitate agents in achieving their goals in time, the temporal relationships are also modeled, namely sequence, concurrency, choice, synchronization.
- **Polymorphism**— The transitions have three types of action selection strategies, namely direct, conditional, and probabilistic.

An agent is usually required to achieve a set of goals in an environment that is dynamic and unpredictable. The agent needs to decide on its own which goal to select next and what actions to take such that goals could be attended successfully. The reasoning involves goal selection and action selection, respectively. Goal selection is to identify a plan path, highlighting the next goal for agents to attempt, while action selection focuses on the choices of transitions from one goal to another. Goal selection takes path value into account and optimization takes place to select paths with optimal index value. For action selection, three types of strategies were proposed:

- **Sequential execution**: the simplest method, for which agents move from one state to another in a sequential manner, by executing a series of actions.
- **Rule-based inference**: agents make decisions according to rules and values of current state, useful when the premises for action selection are straightforward to be evaluated.
- **Probabilistic inference**: agents make decisions based on a Bayesian network which encodes the relations between states and actions, useful for information that is incomplete and uncertain.

Figure 3.5: Root Goal in GoalNet

3.3.3 Goal-Oriented Modeling for Different Motivations

Following sections attempt to map components of a competitive intrinsically motivated agent defined in Definition 3.2 to the GoalNet modeling approach introduced in Definition 3.8. The purpose is to use GoalNet as a practical designing and implementing tool for real agent-mediated software system. As we use a goal-oriented approach, this section will begin with the identification of agent goals, and the entire modeling process includes four stages.

As we have mentioned in Section 3.3.1, motivation is the driving force to form a goal path, and goals provide the means to achieve a certain motivation. Therefore, for our proposed motivated agent, all the agent goals are derived from the agent's current motivation. In Definition 3.2, we have defined four types of motivations, which are the *Pursuit of Novelty*, the *Pursuit of Performance*, the *Pursuit of Relatedness*, and the *Pursuit of Autonomy*. Here, we are trying to use the modeling of agent goals to realize the motivations.

These goals are represented as the root goal in GoalNet as in Figure 3.5.

By definition, one GoalNet is characterized with one root goal, which can be hierarchically decomposed into several sub-goals and transitions. The sub-goals and transitions in the lower hierarchy support the achievement of the goal in the higher hierarchy. Agent's interest is designed as the root goal which is of high-level semantics and abstract. Motivated learning agent is always pursuing action plans to fulfill its root goal, and therefore action plans serves the ultimate purpose which is derived from its motivations. Apparently the design of motivated

agent GoalNet is application-dependent. For different cases, an agent may have different ways to represent the four perspectives of its needs. In the case of multiple independent specific needs within the same perspective of the dividing of needs, we can design multiple root goals, and these root goals can be specified to these four types.

Agent modeled by GoalNet has to make decisions when facing choices. Choices may be given to agent on how to make a plan (how to create/select GoalNet) and what concrete means to attain for a subtask. Intrinsic motivation is the key drive for agent decision making. For our proposed goal oriented motivated agent, all the decision makings are derived from the agent's intrinsic needs. Then what are the crucial factors to generate agents' need? As we mentioned in the problem description of this thesis, our objective is to achieve the **adaptiveness** of agents in a dynamic changing environment. Thus, to discuss the needs of the proposed motivated agent, adaptiveness is one of the main concerns.

Generally speaking, the adaptiveness of agent is contributed from two aspects- knowledge exploration and knowledge exploitation. Exploring knowledge includes creating brand new knowledge and abstracting common features from a category of knowledge, namely plan formation and abstraction. While exploiting knowledge in a specific application scenario includes selecting an acquired plan category and binding available actions to form an executable plan, targeting at coarse and fine selections. These two aspects are naturally related to our previously defined motivations and needs. The exploring of knowledge is related to the *need of novelty*. We just define the discovery of all the new and unknown things as a knowledge exploring process. While the exploiting of knowledge is related to the needs of performance and relatedness, as the *need of performance* is the fundamental of exploiting a task, and the interaction with others (*need of relatedness*) may improve one's knowledge and skill to efficiently realize the exploitation. Finally, the *need of autonomy* is an integral factor to dominate all the needs working together to achieve a smooth functioning. As a consequence, the motivation *Pursuit of Novelty* is modeled through a GoalNet with a root goal To be novel; the motivation *Pursuit of*

Performance is modeled through three different GoalNets specifically with root goal **To fulfill external request**, **To self-Improve**, and **To search for task**; similarly, the motivation Pursuit of Relatedness is modeled through two GoalNets with the root goal **To ask** and **To answer** (as Table 3.2). For the motivation *Pursuit of Autonomy*, it is realized via the coordination of the entire model to achieve the autonomic selection and fulfillment of the other three motivations.

Table 3.2: Mapping among Needs, Motivations and Root Goals

Needs	Motivations	Root Goals
Novelty	Pursuit of Novelty	To achieve novelty
Performance	Pursuit of Performance	To fulfill external request
		To self-improve
		To search for task
Relatedness	Pursuit of Relatedness	To ask
		To answer

3.3.4 Use GoalNet to Represent Agent Knowledge

Begin with a root goal, then subsequently connecting all the sub-goals and states identified for achieving the root goal, an instantiated “net” structure can be created. These GoalNet structures will be stored in the agent’s knowledge base as the *Plan* knowledge.

a plan is a tuple of $(Plan_name, Goal, Context, Body, Options)$, and a plan body is composed of a set of sub-plans and atomic actions. For example, main task is actually a meta plan which can be further decomposed to a combination of sub plans and atomic actions. GoalNet methodology is built in nature to support agent planning, and we adopt GoalNet to model agent plans.

We look into the component of a GoalNet first. A GoalNet G and has a set of goal states S , in which $S = \{root\} \cup C \cup Atom$. G has exactly one root goal $root$, a set of composite goals C and a set of atomic goals $Atom$. In GoalNet, a root goal goes through a sequence

Figure 3.6: GoalNet Modeling of Plan

of executions of its states s , either $s \in C$ or $s \in Atom$. States are interlaced by various transitions $Tr = (P_{Tr}, F_{Tr}, S_{pre}, S_{post})$ in between. Transition Tr specify a list of parameters P_{Tr} required to fire and a function F_{Tr} that defines the behavior. The successful firing of transition Tr transfers the GoalNet G from beginning state S_{pre} to S_{post} . In the case of the occurrence of a composite goal C , an iterative execution of the sub-routines is required for individual sub-goals in the subsequential hierarchies.

Plans defined in Definition 3.6 are graphically designed and represented by GoalNet. The mapping between agent plan and GoalNet is illustrated in Figure 3.6.

Plan goal $Goal$ is mapped to the *root* of G , and a plan is identified by searching the *root* of G . Plan's condition $Context$ is kept being checked at S_{start} until the plan's $Context$ are satisfied. Goal states S and transitions Tr in between compose the plan body $Body$. Plan's subroutines and atomic actions are modeled as composite goals C and atomic sub-goals $Atom$ respectively in GoalNet. Hierarchically, a composite goal C is the root goal for the parent plan's children. In turn, the children plan contains its goal states and transitions in its hierarchy.

Parent GoalNets and children GoalNets are nature to model routines in plans and routines in sub plans. A special case of plans constitutes a set of reusable templates for achieving goals.

A template defines a skeleton allowing agents to substitute sub template components or do concrete routine binding in real time. Dynamic binding enables agent to reuse and dynamically adjust plans according to information on-the-fly. GoalNet supports the features through

Figure 3.7: Internal Logic of Generating Agent Motivated Behaviors

querying knowledge base (GoalNet pool) and gluing detailed implementations by functional invocation. The dynamic composition and realization of plans are a compound of formation and selection in light of intrinsic criteria, which are discussed in details as in the rest of the section.

3.3.5 Use GoalNet to Represent Agent Reasoning

The execution from motivations to real actions includes two levels of decision making, namely coarse selection and fine selection. Coarse selection aims at *choosing the right category* of plans given a problem to solve, for which the output is a subset of plans tentative to be able to fulfill the sub-problem; fine selection is more focused on *choosing an executable course of actions* from candidate plans, for which the output is a chain of executable atomic actions for agent to carry out. Both are proposed to be modeled using GoalNets. Agent may acquire a set of knowledge in the form of GoalNets, and the GoalNets are either from expert knowledge at design time or from autonomous plan formation/abstraction after deployment. In this context, *coarse selection* involves deciding the sub-goal to achieve, allowing searching GoalNets that are relevant to the problem, while *fine selection* includes finding optimal action binding given agent's intrinsic needs. The two levels, coarse and fine-grained, are naturally dependent on each other, and thereby in our GoalNet modeling, we integrate them a systematic process, which is the GoalNet reasoning.

To sum up, the generation of agent behaviors originates from agent reasoning, and the logics of agent dynamic actions can be described as four different stages shown in Figure 3.7.

In GoalNet, agent may have choices from one state to another, and only one sub-sequential state can be reached. Agent needs to decide which goal state to attempt in the next move. This

process called **Goal Selection**. Various methods of goal selection can be defined in GoalNet, for example, conditional rules and maximizing goal index in terms of time and cost constraints and expected returns. Besides goal selection, going from one goal state to another must complete several certain tasks, and these tasks are embedded in the transitions. Going through each transition, we need to select a proper action from the available functions, and this process is called **Action Selection**.

Goal selection in GoalNet is modeling graphically as Figure 3.8.a. From State S_0 , an agent chooses either S_1 via Transition t_1 , or S_2 via t_2 . Goals S_1 and S_2 are exclusively selected.

However, S_0 requires a decision making process (an action) and this modeling violates our goal definition where a goal is essentially a system state, rather than tailed with any behavior or action. To solve this problem, we introduce a *choice* transition and intermediate goal states to extend current goal selection modeling. The extension is illustrated in Figure 3.8.b, where the choice transition is graphically represented as a diamond, connected with two intermediate goal states. The intermediate states serve as an instantaneous state, characterized by the changes of agent's mental status, yet environmental state. The choice transition is constrained to be instantaneous as well, such that environment state is consistent with previous state S_0 . From the introduced states, agent fires transitions t_1 and t_2 to reach S_1 and S_2 respectively.

Action selection is modeled in GoalNet as Figure 3.8.c, where transition t is fired by an action $a_i, 1 \leq i \leq k$ from a list of actions. However, this modeling suffers separation problems when dealing with states and behaviors. Meanwhile, treating goal and action selection separately introduces decision complexity for system designers. In the proposed modeling, we unify goal and action selection as part of plan selection so that one consistent mechanism can be universally applied within the system. Similarly, we re-apply the concept of choice transition, and in this case we convert the previous GoalNet to the one in Figure 3.8.d. Note that in the proposed modeling we prefer one action applicable to exactly one transition. Comparing the ending states, an agent reaches different states by acting differently. The details of plan selection are explained as goal selection and action selection in Section 4.6.

Figure 3.8: Unified Modeling of Goal Selection and Action Selection

For motivated agents, decision making is based on consideration of intrinsic needs. Therefore, plan selection is based on the evaluation of the degree of satisfaction of intrinsic needs. In general, agent always prefers the plan that maximizes its need satisfaction in terms of competence, autonomy and relatedness. In Table 3.2, we corresponded intrinsic needs with root goal states. The agent modeling process can be conducted via the sequence of decision making on goal selections and the realizing of action selections. The detailed analysis of the entire agent architecture and the specific design of GoalNet with root goals which provide means to achieve agent's intrinsic motivation will be discussed in the next chapter.

Knowledge and reasoning are related to the successful completion of agent's routine task. The following section discusses the use of the goal-oriented approach to represent agent routines with mentioned agent knowledge and reasoning.

3.3.6 Use GoalNet to Represent Agent Routines

GoalNet is used for modeling agent main tasks, which are abstracted from agent's daily workflows. Explanation of modeling agents using GoalNet is in the order of hierarchy, as follows.

The root goal of a motivated agent is the running of agent's routines. Agent's daily operations can be regarded as a series of processes, in which the agent interacts with its environment to reach its own objectives. Viewing from the outside of the box, an agent always repeats the cycle that starts with receiving environmental stimuli and ends with sending responses back to the environment. Generally after an agent is deployed and initiated, the agent must carefully choose its responses or actions upon the arrival of a message or decision of acting to its own plan; its behaviors reflect whether the agent achieves the ultimate goal, that is to exhibit adaptiveness and competence for the purpose of its creation. Therefore, during each routine cycle an agent faces a decision problem: how to respond a critical environment event, and how to proactively act in the absence of environment stimuli. The general main procedure to solve a problem requires: analyze the situation to categorize it into one of the predefined sub-problems; make a decision of choosing a sub-problem plan; and act accordingly.

The main task model of an agent can be regarded as a continual and interactive decision making procedure, each cycle of which consists of four phases. In the sequential order, they are event receiving, situation analysis, sub-task decision making and behavior execution. The main task is infinitely looped through an agent's life cycle, and terminates only when the agent is dismissed. The generic main task model is shown in Figure 3.9.

Event receiving is triggered either by an informative signal intercepted by an agent from the environment or by a time out of the agent's internal clock. The type of signal can be an environment stimulus observed by the agent, or a delegated message explicitly sent to the agent. During event receiving phase, agents should be designed not to miss any critical event, and neglect events that are trivial and non-important. The distinction between critical and non-critical information depends on the nature of sub-tasks, and as a result, the design of event

Figure 3.9: Generic Main Task Model

receiving follows the design of sub-tasks. It is also possible that there is no stimuli perceived for a predefined period. An motivated agent is required to proactively act according to its own needs, instead of merely react to stimuli. Therefore, situation analysis becomes the next important step.

Situation analysis amounts to the understanding of the semantics of an event and intrinsic needs, facilitating the classification of an situation into one of different subtask categories. One requirement of situation analysis is to abstract information from received events. Abstraction is modeled as an inference function that translates arbitrary input into predefined event format. For example, an overloading event when a server of capacity 100 is hosting 120 concurrent users is translated as $Server_{id} < Users : 120, Capacity : 100 >$, which is understood by its sub-task solver. The other requirement of situation analysis is to recognize agent's internal needs. The synthesis of extrinsic event and intrinsic needs contributes to the sub-task decision making.

The category of subtasks that the agent should look into given a situation is firstly identified. There will be multiple sub-tasks an agent faces during its life cycle. It is critical to allow the agent to know which sub-task to look for, so that the allocation of a suitable plan becomes possible. The abstraction and the classification are interlaced with each other. Abstraction firstly formats event coarsely so that classification can determine the sub-task category. After category is found, abstraction continues with a fine conversion to meet the format requirements of a specified sub-task.

Afterwards, subtask decision making continues to handle specific problems encountered during the agent's pursuit of its own needs. It is the most important phase that determines the agent's goal-pursuing effectiveness and overall performances. Various agent types differentiate with each other in the way of handling problems. The design of this phase determines agent behavioral characteristics, such as autonomy, adaptiveness, and flexibility. For example, reactive agents have a set of condition-action rules as its decision making mechanism, and an action is selected only if the corresponding condition is true. Instead of simply checking present conditions against rule base, deliberative agents attempt to find a sequence of actions that will achieve the goal by matching the post-conditions of actions against the desired goal. The final plan is inferred from knowledge base through symbolic AI deductive methods.

It is typical for an agent to have multiple sub-tasks in practice. For example, a scheduling agent must handle tasks such as shedding jobs to machines, recovering from machine failures, monitoring job status, etc. A capable agent needs mechanism to deal with each predefined subtask, and has potential to come out with its own solution for any new sub-task appearing after agent deployment. Hierarchically, a sub-task may consist of a sequence of smaller tasks. Therefore, it is convenient to model subtasks as subgoals for an agent to achieve its ultimate goal, and sub-tasks have lower-level goals in their hierarchies. Iteratively, we can break down sub-tasks into atom actions in the top-down manner, and a detailed plan become available for the agent to follow.

Behavior execution is meant for carrying out either agent's response towards the perceived environmental stimuli, or agent's plan according to its own needs. It is meant to have an executable plan beforehand which comprises a sequence of atomic actions. These atomic actions should be the smallest elements which cannot be decomposed further or requires further planning; they should be readily implemented in the agent's action library. Note that in a non-deterministic environment, there exist uncertainties about the resultant system state after performing an action. As a result, agents require to continually observe the environment for feedbacks, which starts another round of agent operation cycle.

3.4 Summary

The key focus of this chapter is the formal model of a new type of agents based on the essence of SDT. It begun with a generic agent modeling, drawing attentions that reasoning is the differentiating factor across different agent models. The proposed motivated agents is unique in the motivated deliberating process. This chapter first gave the agent definition, followed by a computational modeling of components in SDT from motivation, intrinsic needs to cognitive functions. A working example was discussed later, showing the suitability of adapting the proposed agent for a real-life problem.

The implementation of competitive agent was proposed by adopting GoalNet as a goal-oriented representation. The employment of GoalNet made it possible for the development of motivated agents in a real-life problem. The complexity of real world was well handled by the GoalNet methodology. Afterwards, an in-depth representation of agent knowledge, reasoning process and main routine was described. The proposed representation allows a straightforward modeling of agent's competency over mastering new skills, performance-seeking for task fulfillment, collaborative intent for improvement and reliability, and finally agent's autonomy for determination over its own behaviors.

Chapter 4

Need Pursuit in Competitive Intrinsically Motivated Agents

4.1 Introduction

The previous chapter explains the model of a competitive intrinsically motivated agent and how the model can be represented by the goal oriented framework in a complex environment. However, there are gaps on how to make an agent pursue their needs from a design and implementation perspective.

For an agent, the knowledge base is of high importance. Each of the intrinsic needs has a particular set of tasks to be designed and implemented. Since agent's main task is designed at the high level, agent's detailed sub-tasks are to be considered carefully for practical use. The three important sub-tasks are for the pursuit of intrinsic needs, namely novelty, performance, relatedness and autonomy. The sub-tasks are putting together by agent's sense of "self-willing", by which competitive intrinsically motivated agents exhibits autonomous behaviors through motivated and proactive reasoning for motivation, goal and action selection.

In this chapter, an overview of agent architecture is first introduced with the design of agent's main task. The components of the proposed competitive agent are the building blocks for agent need pursuit. In the main part of this chapter, several sub-tasks as the pursuit of agent needs are introduced with: (1) a GoalNet representation; (2) a detailed system-wise design;

and (3) a detailed implementation with typical instantiation. Finally, a running platform is explained to support the execution of the proposed motivated agent.

4.2 Agent Architecture and Main Workflow

An overview of the agent architecture is presented in Figure 4.1. This figure extends agent modeling as shown in Figure 3.2, revealing details of some internal structures.

In the “Intrinsic Needs” module, four types of needs are outlined and three of them associate an internal function for generating need values. Note that autonomy always drives an agent to contemplate, and it represents the unique sense of “self-willing”, so the agent always considers autonomy, regardless of the value of autonomy need. Correspondingly, there are four types of intrinsic motivations on pursuit of each needs. The motivation steers the agent in choosing its next plan.

Goal and action elicitors have similar components, including a compute and a policy. In light of the policy of goal/action selection, compute calculates the value of each alternatives and then allows the agent to choose the most appropriate behaviors. Agent makes decisions depending on its knowledge base.

The categories of knowledge comprise plan, domain knowledge and runtime data. Agent plan in the format of GoalNet follows the goal-oriented strategy closely; domain knowledge is implemented as functional modules that can be installed and plug-in with minimum efforts, while agent should be allowed to recognize its usage and exploit easily; and for runtime data, its implementation should allow information storing, retrieving and updating conveniently, and the supported format of information is open to a wide range of types of format.

Speaking of the functionality, there is a control thread concurrently running with other threads, such as compute, data, etc, and it implements the thread for agent deliberation. The main routine of a motivated agent requires proper operations of all the components. Thus,

Figure 4.1: Architecture of the Competitive Intrinsically Motivated Agent

following discussions of agent's behaviors motivated by need pursuit will refer back to this architecture frequently.

Refereing to the $PM(N)R^2A$ main task cycle introduced in Section 3.2.4, the main task cycle is represented in the GoalNet shown as Figure 4.2.

The root goal of this GoalNet is “execute motivated main task”, and it is the routine of an intrinsically motivated agent. The root goal is repeatedly accessed at a pre-defined frequency. Thus, external event is sampled at this frequency by agent's action of checking event from the environment. Different from other types of agents, agents must be self-motivated intrinsically. Extrinsic environment stimuli do have impact on agent's behaviors, yet external input does not necessarily imply any behavior from the agent. It is shown in the figure that if an event

Figure 4.2: GoalNet Representation of Agent's Motivated Main Task

is absent, the agent determines its motivation based on need levels; whereas upon an even is received, the agent's motivation is related to both need levels and event. It is necessary that the need measurement is made available at the pre-defined sampling frequency, and thus no matter whether an event is presented or not, agent will deliberately proceed with its main task. **MI** fires one of the motivations according to the computed intensity, taking into account **P** and **N**. Agent leverages a decision module to implement this selecting process, which returns the reference of selected GoalNet, presenting one of agent's sub-tasks. The control module then initiates the execution of the selection GoalNet in **GE**. GoalNet information is retrievable from **K**; decomposition and selection are carried out subsequential in **GE**; After the resultant leaf sub-goals are all atomic goals, the action selection in **AE** is conducted; Actions are carried out immediately if it is internal or pushed onto actuators for execution. Agent's control is returned to control module, once the end goal of the sub-task is reached. The weights of intrinsic needs are updated in **MI** by taking into account how the sub-task has been executed by retrieving and analyzing runtime data stored in **K**.

4.3 Competence Need Pursuit by Exploration

Agent's competence exhibits in two ways: mastering a novel skill/knowledge or receiving recognitions from performance feedback. This section focuses the former approach, and knowledge acquiring is considered as a learning behavior in the context of agent modeling. Agent learning is a buzz word, for which the hype exaggerates the actual capability of the agent as a software programm. This thesis treats agent learning as the capability of discovery, semantics-understanding and invocation of an available function module. The implementation behind learning includes the dynamic invocation and ontology matching, that are popular in modern IT industry. Intrinsically motivated agent is given the access to the set of technologies, so that their adaptiveness is largely extended in an open environment, where both challenges and opportunities exist through the dynamic interaction. A GoalNet model of agent's pursuit of novelty is introduced first, followed by the knowledge model of a fast learning algorithm. The realization of the knowledge model is shown in the latter part of this section, together with the learning details specified as a sub-GoalNet.

4.3.1 Goal-Oriented Representation of Novelty Pursuit

Referring to the main task in Figure 4.2, **MI** may decide it is time to learn a new knowledge, and then initializes a motivation of pursuit of novelty. The GoalNet representing agent's learning sub-task is shown in Figure 4.3.

Initially, the root goal "*To achieve novelty*" is initiated by **MI**. The control module checks the external knowledge base and compares with its internal knowledge base. Domain knowledge in **K** maintains a currently mastered knowledge/skill set, and a return message states whether a new type of knowledge is externally available. A positive reply will be with a piggy-back that contains candidates to be learnt. An absent of a new skill set results in the termination of this GoalNet. Note a early termination would result in **MI** increases the weight of the novelty need, so that next round the agent still keeps a high level of enthusiasm to the novelty

Figure 4.3: GoalNet Representation of Agent's Pursuit of Novelty

pursuit. Upon detecting a new set of domain knowledge, the control chooses a knowledge to learn randomly or based on available information such as popularity among other agents and relevance to agent's past experience as runtime data. For simplicity, the agent randomly choose a knowledge. The control module sends out a query to retrieve the meta data of the knowledge and initiates the learning process accordingly. The details of a learning process will be introduced later, after knowledge modeling. Once the learning process is finished successfully, the control updates $\mathbf{K}$ by inserting the recode of the learnt knowledge as a new domain knowledge.

4.3.2 Knowledge Model of a Fast Learning Neural Network

Neural Network handles non-linearity efficiently. Agent task require data-driven tool to cope with uncertainty in the environment, where hard-coded knowledge implementation at design time is inappropriate. *Speed and Accuracy of Learning* is critical to real problem. It has essential implications to the adaptiveness of an agent. *Representing ELM as a Data Structures*

Neural Network is well-known for its capability in learning unknown functions, and is widely applied in regression and classification. Multi-Layer Perception (MLP) is a typical class of neural networks, which is able to approximate nonlinear mappings from input space to output space. However, the real-time learning capability is always a question to most MLPs

due to the iterative parameter tuning process of back-propagation algorithm (BP). The training usually requires thousands of iterations and thus it is very time consuming. This makes the real-time application of MLPs difficult to comply with the fast-responding requirement.

To overcome such difficulty, Huang [92] demonstrated the success of Extreme Learning Machine (ELM) by using less training time, yet obtaining acceptable performance. ELM is a single-hidden layer feedforward neural network (SLFN) with arbitrarily selected input weights (the synaptic weights connecting the input layer to the hidden layer) and hidden layer biases. The output weights (the synaptic weights connecting the hidden layer to the output layer) are analytically determined by Moore-Penrose generalized inverse. Without iteratively tuning parameters as in BP, the training speed can be thousands of time faster than traditional feed-forward training algorithms. Meanwhile, ELM solution has declared to be unique, minimum training error and smallest norm of weights [92].

The novel idea of ELM is that explicit training of input weights and biases is replaced by random selection of those values, and simple least squares solution takes place of traditional time-consuming optimization algorithms.

4.3.2.1 ELM Model

ELM [92] is a fast learning algorithm for single-hidden layer feedforward neural networks (SLFNs) with arbitrarily selected input weights and hidden neuron biases. The output weights of ELM are analytically calculated by Moore-Penrose generalized inverse. The details of ELM training are explained below.

Given a training dataset $L = \{(\mathbf{x}(n), \mathbf{t}(n)), n = 1, \dots, N\}$, where $\mathbf{x}(n) = (x_1(n), \dots, x_d(n))^T \in \mathbb{R}^d$ and $\mathbf{t}(n) = (t_1(n), \dots, t_m(n))^T \in \mathbb{R}^m$, an ELM with activation function $g(\cdot)$ and $\tilde{N}$ hidden neurons can be analytically modelled as,

$$\sum_{j=1}^{\tilde{N}} \beta_j g(\mathbf{w}_j^T \mathbf{x}(n) + b_j) = \mathbf{t}(n), n = 1, \dots, N \quad (\text{Eq. 4.1})$$

where $\mathbf{w}_j = (w_{j1}, \dots, w_{jd})^T \in \mathbb{R}^d$ is the weight vector connecting the input layer to the j th hidden neuron, b_j is the bias of the j th hidden neuron, and $\beta_j = (\beta_{j1}, \dots, \beta_{jm})^T$ is the weight vector connecting the j th hidden neuron to the output layer. The output neurons of ELM are linear. A sketch of an ELM neural network is presented in Figure 4.4. Note that in the figure $z_j(n) = \sum_{i=1}^d w_{ji}x_i(n) + b_j$. Eq. 4.2 can be written in the matrix form as:

$$\mathbf{H}\boldsymbol{\beta} = \mathbf{T} \quad (\text{Eq. 4.2})$$

where

$$\mathbf{H} = \begin{bmatrix} g(\mathbf{w}_1^T \mathbf{x}(1) + b_1) & \dots & g(\mathbf{w}_{\tilde{N}}^T \mathbf{x}(1) + b_{\tilde{N}}) \\ \dots & \dots & \dots \\ g(\mathbf{w}_1^T \mathbf{x}(N) + b_1) & \dots & g(\mathbf{w}_{\tilde{N}}^T \mathbf{x}(N) + b_{\tilde{N}}) \end{bmatrix} \in \mathbb{R}^{N \times \tilde{N}}$$

$$\boldsymbol{\beta} = \begin{bmatrix} \beta_1^T \\ \vdots \\ \beta_{\tilde{N}}^T \end{bmatrix} \in \mathbb{R}^{\tilde{N} \times m} \text{ and } \mathbf{T} = \begin{bmatrix} \mathbf{t}_1^T \\ \vdots \\ \mathbf{t}_N^T \end{bmatrix} \in \mathbb{R}^{N \times m}$$

Traditional SLFN like MLP utilizes BP algorithms to optimize parameters such as synaptic weight $\mathbf{w}_i$, β_i and bias b_i . Gradient-decent learning algorithms update parameter vector $\mathbf{w}$ iteratively according to:

$$\mathbf{w}_k = \mathbf{w}_{k-1} - \eta \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} \quad (\text{Eq. 4.3})$$

where $E(\mathbf{w})$ is the cost function defined as:

$$E = \sum_{j=1}^N \left(\sum_{i=1}^{\tilde{N}} \beta_i g(\mathbf{w}_i \cdot \mathbf{x}_j + b_i) - \mathbf{t}_j \right)^2 \quad (\text{Eq. 4.4})$$

However, due to user specified constants and large number of iterations, as Huang pointed out in [92], BP algorithms suffer from a set of problems: it has a slow convergence rate; networks may be trapped at local minima; they may be easily over trained and it is very time-consuming, which causes difficulty in real applications.

ELM learning algorithm first chooses input weight $\mathbf{w}_i$ and hidden neuron bias b_i randomly,

after which matrix $\mathbf{H}$ defined in Eq. 4.2 can be calculated immediately. Then the problem of minimizing cost function in Eq. 4.4 is equivalent to finding such that:

$$\left\| \mathbf{H}\hat{\boldsymbol{\beta}} - \mathbf{T} \right\|_2 = \min_{\boldsymbol{\beta}} \left\| \mathbf{H}\boldsymbol{\beta} - \mathbf{T} \right\|_2 \quad (\text{Eq. 4.5})$$

where $\|\cdot\|_2$ denotes the 2-norm.

Since $\tilde{N} \leq N$ for most cases, the output weights can be calculated by solving the least squares of the linear equation in Eq. 4.2, as follows,

$$\hat{\boldsymbol{\beta}} = \mathbf{H}^\dagger \mathbf{T} \quad (\text{Eq. 4.6})$$

Figure 4.4: ELM Neural Network

Due to the simplicity of training process, with respect to learning speed ELM was reported to outperform traditional SLFNs which adopt back-propagation (BP) learning algorithms [93]. Meanwhile, ELM can also provide acceptable performance for a wide range of applications [92, 94].

4.3.2.2 A Close Look at ELM

Comparing ELM with neural networks like RBF, we see some relationship in these feedforward structures. In this section, a close look at ELM is expressed in the language of higher

dimensional space mapping and interpolation in the two-staged learning process. In the first learning stage of ELM, *Cover's theorem on the separability of patterns* is applied. Given a set of input data $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$, where $\mathbf{x}_i \in \mathcal{R}^{m_0}$, each of them belongs to one of k classes. The classification problem is equal to finding a set of surfaces such that all the k categories are separable from one another. Cover's theorem [95] states that if it is hard to find such surfaces in the original space $\mathcal{R}^{m_0}$, the probability of existence of solution will be closer to one if original data is mapped from to a higher dimensional space $\mathcal{R}^{m_1}$ ($m_0 < m_1$) by a vector $\varphi(\mathbf{x})$, shown as:

$$\varphi(\mathbf{x}) = [\varphi_1(\mathbf{x}), \varphi_2(\mathbf{x}), \dots, \varphi_{m_1}(\mathbf{x})]^T \quad (\text{Eq. 4.7})$$

where $\varphi_i(\mathbf{x})$ is a nonlinear function that provides a mapping from $\mathbf{x} \in \mathcal{R}^{m_0}$ to a scalar. The original space $\mathcal{R}^{m_0}$ is called input space, whereas the higher dimensional space $\mathcal{R}^{m_1}$ is referred to hidden space (since the space dimension is determined by the number of hidden neurons) or feature space. Moreover, $\varphi_i(\mathbf{x})$ performs as a “basis” of $\mathcal{R}^{m_1}$ when input patterns are transformed into the feature space.

A typical ELM network multiplies an input vector $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{in})^T \in \mathcal{R}^n$ by the input weights $\mathbf{w}_i = (w_{i1}, w_{i2}, \dots, w_{in})^T$ and offsets the vector product by the hidden neuron bias b_i . Then a nonlinear activation function $g(x)$ in hidden neurons maps the original input patterns into a new space of high enough dimension. In light of Cover's theorem, it is usually beneficial to transform a nonlinear separable classification problem into a linearly separable one or an easier one that involves linear mapping. Similar approach can be found in RBF networks, for which $\varphi_i(\mathbf{x})$ is a radial-basis function. In despite of Cover's theorem's specialization in pattern classification, it is however still feasible in regression, since classification can be regarded as a regression for discrete class values. Evidences can be seen from the nonlinear regression capability in the RBF case.

In the second stage of training, ELM provides a map s from $\tilde{N}$ -dimensional hidden space to m -dimensional output space. Purposely a single output is chosen to simplify the exposition

without loss of generality i.e. $s : \mathbb{R}^{\tilde{N}} \rightarrow \mathbb{R}^1$. Consequently, the map s can be viewed as a hypersurface in the space $\mathbb{R}^{\tilde{N}+1}$. As a result training phase is the same as constructing and optimizing a hypersurface to fit the data points in the feature space, whereas testing is synonymous with interpolation between the known data points along the constrained surface.

In an extreme case, the hypersurface is constructed such that it passes through all the training samples. It is only achievable when $\mathbf{H}$ in Eq. 4.2 is a square matrix or equivalently the number of hidden neurons is exactly the same as the number of training samples, i.e. $\tilde{N} = N$. It is called strict interpolation and Eq. 4.2 can be rewritten as:

$$\hat{\mathbf{H}}\boldsymbol{\beta} = \mathbf{t} \quad (\text{Eq. 4.8})$$

The N -by-1 vectors $\boldsymbol{\beta}$ and $\mathbf{t}$ denote the output weight and desired response vector respectively for a single output case. $\hat{\mathbf{H}}$ is an N -by- N matrix, which is also called interpolation matrix [95]. To solve Eq. 4.8 for the weight vector $\boldsymbol{\beta}$, assuming $\hat{\mathbf{H}}$ is nonsingular and thus it has its inverse matrix $\hat{\mathbf{H}}^{-1}$, then:

$$\boldsymbol{\beta} = \hat{\mathbf{H}}^{-1}\mathbf{t} \quad (\text{Eq. 4.9})$$

The problem now becomes how to guarantee the non-singularity of matrix $\hat{\mathbf{H}}$. Huang in his paper [96] has proved that matrix $\mathbf{H}$ defined in Eq. 4.6 with random weights and biases and infinitely differentiable activation functions g in any interval, is full rank with probability one. In light of his theorem, in theory matrix $\hat{\mathbf{H}}$ in our case can be guaranteed nonsingular, and thus its inverse exists. A similar approach of interpolation can be found in RBF networks, which also search for matrix inverse. However, the non-singularity for RBF network cases is guaranteed by selection of a class of radial-basis functions, for example, Gaussian functions and Multiquadrics [95].

In practice, the size of training samples will often exceed the number of degrees of freedom of the underlying physical process [95]. Moreover, input data is more likely to be corrupted by noise, so overfitting training data may result in poor generalization performance. Therefore,

fewer hidden neurons than the data size are always chosen, i.e. $\tilde{N} \leq N$. As illustrated in Eq. 4.6, the overdetermined linear equation is solved using Moore-Penrose general inverse and the resulted hypersurface approaches the data as closely as possible to minimize the cost function, which is defined in Eq. 4.4. This kind of approximation is also known as curve fitting. After the surface is constructed, testing is done by interpolating between the known data points along the constrained hypersurface.

4.3.2.3 Knowledge Representation of ELM

The representation of an ELM network, *Network* class, simply aggregates two classes, namely *Neuron* and *Synapse*.

Network class has properties in describing the structures of neural networks, including the number of layers that is 3 as a constant, the number of nodes in each layer and activation functions. A network also has a list of nodes as neurons and synapse between neurons. *Neuron* class are defined with properties as the type of neuron (input, output, hidden), bias for hidden neurons and the positions (i.e. i^{th} neuron on j^{th} layer). Randomized input weights and calculated output weights are instances of class *synapse*; a instance of *synapse* include the weight and the connection in terms of the source neuron and the destination neuron.

Training The training of an ELM network is invoked by the function

train(**P**, **T**, #Hidden, Activation Function) which takes inputs from input data **P**, target data **T**, the number of hidden neurons #Hidden and the type of activation function. The output is a trained neural network structure defined in the network class. The network is trained according to the algorithm specified in Section 4.3.2.1, and a pseudo-code is illustrated in Figure 4.5.

Reasoning The reasoning of a trained knowledge model is to obtain the output of the received values of inputs. It carries out in a forward manner: the input is firstly multiplied with input weights; the result is then processed inside each neurons by taking activation function and

input : input P , target T , #Hidden and activation function

output: a trained ELM network **ELM_NET**

- 1 **initialize:ELM_NET** by parse P , T and #Hidden;
- 2 randomly generate in $[0, 1]$ input weights w_i and biases b_i of hidden neurons;
- 3 calculate hidden neuron output matrix H ;
- 4 calculate output weights β_i ;
- 5 store trained values in **ELM_NET**.

Figure 4.5: Pseudo-Code for Training a ELM Neural Network

Figure 4.6: GoalNet Representation of “To Train ELM model”

offset by bias; the temporary hidden layer output is then feed-forwarded to multiple the output weight; and finally the output of the reasoning is obtained. The process is invoked by function $reason(ELM_NET, input)$, where ELM_NET is a trained ELM network.

4.3.3 Agent Behavior on Novelty Exploration

The GoalNet representing agent behaviors for “To train ELM model” is shown in Figure 4.6.

The training of an ELM shows general steps that could be applied to a wide range of domain knowledge learning process. A source of data is firstly selected, and the selection of data source is related to agent’s prior choice on which skill to master. It is usually the case that the agent selects the data that could be useful to its routine performance. After the data is collected, the agent needs to format its data compliant to the requirement of the domain skill. For ELM, the

requirement is specified as in Section 4.3.2.1. Besides to training data, configurations such as number of hidden neurons and the type of activation function are to be decided.

As discussed before, the learning of domain knowledge is implemented as a remote service call. Above mentioned configurations of parameters and format are explicitly defined as service interface explicitly defined in Web Service Description Language (WSDL). However, the agent should be designed with a minimum set of pre-defined semantics, that any other WSDL-described service is using, such as "input", "parameter". In addition, the semantics could be simplified significantly by default settings and a limited space of parameter space. For example, the number of hidden neurons are integers from $[5, 20]$, and the activation function is limited to either *linear* or *sigmoid*.

The selected configuration and data are uploaded to the host of the remote learning service via HTTP or FTP protocol, etc. Since the agent knows the existence of a new learning skill after its deployment, service binding at design time or compile time is not possible. Thanks to the Web Services Invocation Framework (WSIF), invoking external services becomes possible through simple API invocation. Regardless where or how a service is provided, the framework enables great flexibility for invoking any service that is described by WSDL. Within this framework, the agent does not need to know how to invoke a service until its motivated novelty pursuit fires at runtime; the agent can retrieve a new domain knowledge service location and description in WSDL from the external knowledge base (the service "registry"); dynamic invocation and late binding enable the agent to dynamically generate a client partner to access the remote service; as a summary, agents do not require to recompile its code. It ensures the minimal human intervention, and thus the maximum agent autonomy. However, dynamic service invocation is discussed as a solution to the possible implementation issue, and the detailed service and client implementation is out of the thesis scope.

Speaking of competence over mastering a new skill, it is important to allow the agent to know its "competency". It means that learning is not a one-time stop process; it involves

a continuous refining and error correction instead. A test procedure is executed to access agent's competence. A good result signals the termination of the learning process, otherwise the parameters are refined and the same training is looped. After a successful training, the network information is stored for the usage in reasoning.

Reasoning is close related to performance and it is discussed in more details in the next section.

4.4 Competence Need Pursuit by Performance

Referring to the main task in Figure 4.2, **MI** may decide to solve problems it is designed for, and then initializes a motivation of pursuit of performance. **MI** has three GoalNets to follow in different situations, including “To fulfill external request”, “To self-Improve” and “To search for task”. The GoalNet representations of the three approaches are discussed individually. To show how performance pursuit works given two alternative choices, a new knowledge model is presented. After that, experiments are given to show the variation of performance as well as the impact of feedback on autonomous goal selection and action selection.

4.4.1 Goal-Oriented Representation of Performance Pursuit

In this section, three sub-goals of how to pursuit of performance are explained.

4.4.1.1 Root Goal of “To Fulfill an External Request”

The GoalNet representing agent's sub-goal of “To fulfill an external request” is shown in Figure 4.7.

Before initiating the GoalNet, the control module interpret external events from environment. **MI** identifies the event as a request to be fulfilled, and then starts the running of the GoalNet whose root goal is “To fulfill an external request”. The control module parses the request and understands the semantics. For example, a request is “To give a prediction of

Figure 4.7: GoalNet Representation of “To Fulfill An External Request”

input x in the context of c ”, and according to pre-defined syntax, the request is express as $\{ \text{“type”} : \text{“request”}, \text{“request_type”} : \text{“predict”}, \text{“context”} : c, \text{“input”} : x \}$. The control module queries agent’s domain knowledge in **K** and gets a list of candidates. By following the candidates to the compute in **AE**, a domain knowledge is selected. The control initiates the sub-goal processing at **GE** to prepare the invocation parameters as well as input format. After notified by the readiness of settings, the control thread invokes the domain knowledge with the pre-specified format. Note a remote execution is carried out, and it is the **A**’s role to deliver the external invocation to the right destination. Agent proceeds with the return of result (e.g. the predicted value), and sends out the result to the requester. The way of how to invoking knowledge to fulfill a request depends on the job nature. It is usually to have one more layer of sub-goals in a complex situation. There are cases that a feedback arrives at **P** and agents store the feedback at the runtime of **K** for further improvement or referencing.

4.4.1.2 Root Goal of “To Self-Improve”

Feedback is particularly useful for domain knowledge refining. Refining knowledge for agents is shown as agent’s pursuit of self-improvement. The GoalNet representing agent’s sub-goal of

Figure 4.8: GoalNet Representation of “To Self-Improve”

“To self-improve” is shown in Figure 4.8.

Agent’s **MI** may decide to do self-improvement as the pursuit of performance. It is usually done independent of environment stimuli. The control module starts with getting a collection of previous performance feedback, which are stored in agent’s runtime in **K**. The main thread asks compute in **GE** for decision on self-improvements. **GE** carries out an analysis on feedback data. For example, it uses a window of a specific period to determine whether environment has been changed significantly over the past, and whether previous domain knowledge is still suitable for this windowed period. A negative response leads to the termination of this GoalNet, implying there is no need to update current domain knowledge. Otherwise, the control is proceeded by selecting and retrieving meta information of a domain knowledge. **GE** is consecutively asked to prepare the input format and running parameters. Afterwards, the agent invokes the updating or simply a learning process to the remote host. Finally, an acknowledgement of the completion will trigger the control to update its own domain knowledge in **K**. It is apparent that the latter part of this GoalNet is similar to the GoalNet defined in Figure 4.3, because self-improvement can be realized by updating a previous knowledge model, or by acquiring a new skill asset. Both are driven by the motivation that the agent strives for excellent performance, though approaches to achieve that may vary. An example of improving by learning a new skill is shown later in Section 4.4.2

It is worth noting that self-improvement is an agent’s autonomous behavior driven by competent need. The improvement process is not driven by external events or external requests.

Figure 4.9: GoalNet Representation of “To Search Task for Performance”

To complete the discussion of performance pursuit, similar self-motivated behavior also occurs when the agent proactively requests a task to work on.

4.4.1.3 Root Goal of “To Search Task for Performance”

The GoalNet representing agent’s sub-goal of “To search task for performance” is shown in Figure 4.9.

Agent’s **MI** may sometimes chooses this GoalNet as the pursuit of performance, particularly if the agent has not been received any request for a long period. The control firstly starts preparing the request message using a specified format; this would be handled by **GE** to decide which kind of task to request; this involves rounds of queries with agent’s plan base in **K**. After the message is ready, the main thread passes the control back to **GE** to instantiate a communication channel. The control module gets the notification when the channel is established, and then asks actuators of **A** to send the message to the request listener in the environment. The hand-shake ends after the agent receives the acknowledgement from the **P**.

Above sections complete the discussion of goal-oriented modeling of the pursuit of competence. The following sections aim to explain how different behaviors would result in different performance feedback. Before that an alternative of ELM learning is introduced as a comparative study.

4.4.2 Knowledge Model of a Well-Conditioned Neural Network

Following the discussion of the knowledge model of Extreme Learning Machine (ELM) in Section 4.3.2.1, Without iteratively tuning parameters as in Back-Propagation (BP), the learning speed of ELM can be thousands of times faster than gradient-decent learning algorithms.

Many efforts have been emphasized on the accuracy of solutions obtained by ELM, whereas numerical stability is generally ignored [94]. Numerical stability is an important aspect of a system. An ill-conditioned system may have its solutions very sensitive to perturbation in data. In particular, for ill-conditioned system even though the change in the problem data is small, the change in the solution may be large. This implies that the estimated solution from data may be nowhere near the true solution, and thus becomes useless. Unfortunately training of ELM with large hidden neurons usually constitutes an ill-posed problem. Therefore, the solutions obtained by ELM may be sensitive to data perturbation and become a poor estimation to the truth.

In order to improve the conditioning of ELM, this section proposes an input weight selection algorithm for an ELM with linear hidden neurons. In the previous section, ELM has been introduced, and it can be written in the matrix form as:

$$\mathbf{H}\boldsymbol{\beta} = \mathbf{T} \quad (\text{Eq. 4.10})$$

where

$$\mathbf{H} = \begin{bmatrix} g(\mathbf{w}_1^T \mathbf{x}(1) + b_1) & \cdots & g(\mathbf{w}_{\tilde{N}}^T \mathbf{x}(1) + b_{\tilde{N}}) \\ \vdots & \ddots & \vdots \\ g(\mathbf{w}_1^T \mathbf{x}(N) + b_1) & \cdots & g(\mathbf{w}_{\tilde{N}}^T \mathbf{x}(N) + b_{\tilde{N}}) \end{bmatrix} \in \mathbb{R}^{N \times \tilde{N}} \quad (\text{Eq. 4.11})$$

Since the number of hidden neurons is usually smaller than the sample size, the output weights of ELM can be calculated as the least squares solution of linear equation defined in Eq. 4.10, as follows:

$$\hat{\boldsymbol{\beta}} = \mathbf{H}^\dagger \mathbf{T} \quad (\text{Eq. 4.12})$$

We argue that the calculation of finding least squares solutions of linear system Eq. 4.10 may constitute an ill-conditioned problem. Specifically, random input weights may lead to an ill-conditioned $\mathbf{H}$, especially when a large number of hidden neurons are used. The conditioning of a matrix can be qualitatively characterized by condition number. Condition number is defined from the analysis of error bounds for linear system $\mathbf{Ax} = \mathbf{b}$, and formulated as the product of two matrix norms:

$$\kappa(\mathbf{A}) = \|\mathbf{A}\| \cdot \|\mathbf{A}^{-1}\| \quad (\text{Eq. 4.13})$$

for any consistent norm. Throughout the paper 2-norm condition number and consistent matrix norms are adapted. In particular for 2-norm we have,

$$\kappa_2(\mathbf{A}) = \frac{\sigma_{\max}(\mathbf{A})}{\sigma_{\min}(\mathbf{A})} \quad (\text{Eq. 4.14})$$

By definition $\kappa(\mathbf{A}) = \|\mathbf{A}\| \cdot \|\mathbf{A}^{-1}\| \geq \|\mathbf{AA}^{-1}\| = \|\mathbf{I}\| = 1$, and $\kappa(\mathbf{A}) = \infty$ for singular matrix. Given $\mathbf{x}$ as the least squares solution for $\mathbf{Ax} = \mathbf{b}$ and $\mathbf{x}(\varepsilon)$ for $(\mathbf{A} + \varepsilon\mathbf{F})\mathbf{x}(\varepsilon) = (\mathbf{b} + \varepsilon\mathbf{f})$, we would like to see how solution changes according to the perturbations. As shown in [97], for nonsingular square matrix the relative error for $\mathbf{x}$ is proportional to $\kappa(\mathbf{A})$,

$$\frac{\|\mathbf{x}(\varepsilon) - \mathbf{x}\|}{\|\mathbf{x}\|} \leq \kappa(\mathbf{A}) \left(|\varepsilon| \frac{\|\mathbf{F}\|}{\|\mathbf{A}\|} + |\varepsilon| \frac{\|\mathbf{f}\|}{\|\mathbf{b}\|} \right) + O(\varepsilon^2) \quad (\text{Eq. 4.15})$$

For least squares cases, the error bound is more complicated, and using consistent 2-norm, the relative error is proportional to $\kappa_2(\mathbf{A}) + \|\mathbf{r}\|_2 \kappa_2(\mathbf{A})^2$ [97], where $\mathbf{r} = \mathbf{A}\hat{\mathbf{x}} - \mathbf{b}$ is the residual for estimated solution $\hat{\mathbf{x}}$. An ill-conditioned system has large condition number while a well-conditioned system has small condition number. For most problems, there is a prior that if ε is small, the change in the solution $\frac{\|\mathbf{x}(\varepsilon) - \mathbf{x}\|}{\|\mathbf{x}\|}$ should be small as well. However, it is apparent that for linear system $\mathbf{Ax} = \mathbf{b}$, if $\kappa(\mathbf{A})$ is large the upper bound of relative error will be large, which means relative small perturbation $\mathbf{A}$ in $\mathbf{b}$ and may lead to large change in the solution. In practical perturbation due to measurements in the input or approximation during computation is difficult to void. As a consequence, for an ill-conditioned system the calculated least squares

$\hat{\mathbf{x}}$ is usually a poor estimate to the truth and thus becomes useless. Note that since the error upper bound is affected by the condition number, it does not necessarily mean that the relative error is large for a particular case. Nevertheless, condition number is a good indicator of a system conditioning which shows how close a system is to be ill-conditioned. In this paper we propose an input weight selection algorithm to minimize the condition number of coefficient matrix $\mathbf{H}$, and the problem formulation is shown in the next section.

4.4.2.1 Problem Formulation

One essential learning step of ELM is the least squares calculation for linear system Eq. 4.11. Because from Eq. 4.2 the elements of coefficient matrix $\mathbf{H}$ are functions of input weights $\mathbf{w}_i$ and input data $\mathbf{x}_j$, the condition number of $\mathbf{H}$ is determined by input weights and input data. Therefore, it is apparent that arbitrary input weights consider no robustness for the solution, and only an elaborate selection of input weights with respect to a particular set of input data may achieve a well-conditioned system. As mentioned before, we adopt condition number of coefficient matrix $\mathbf{H}$ as a qualitative metric of robustness. Combined with ordinary least squares, the problem of network learning is formulated as,

$$\min_{\mathbf{w}, \boldsymbol{\beta}} (\|\mathbf{H}\boldsymbol{\beta} - \mathbf{T}\| + \lambda\kappa(\mathbf{H})) \quad (\text{Eq. 4.16})$$

where the first term is the least squares error and the second term indicates the condition number of the coefficient matrix. λ controls the trade-off between residuals of fit and robustness of the solution. When $\lambda \rightarrow 0$, the network finds solutions with more concerns on accuracy, and when $\lambda \rightarrow \infty$ more on robustness instead. The formulated learning can be regarded as a multi-objective optimization problem. For simplicity this paper applies a greedy approach which consists of two consecutive steps of optimization. In light of the layered network structure, the first step optimizes condition number of coefficient matrix $\kappa(\mathbf{H})$, by selecting input weights according to input data $\mathbf{x}_j$, and then second step calculates least squares solution $\boldsymbol{\beta}$ as original

network does. However, minimizing the condition number is not trivial since currently we lack of a closed-form expression of $\kappa(\mathbf{H})$ on variable $\mathbf{w}$. In this paper, we focus on a simplified SLFNN with only linear hidden neurons. Note that for this case we have $g(z_j(n)) = z_j(n), \forall j$ as in Figure 4.4. Therefore, we have

$$\mathbf{H} = \begin{bmatrix} \mathbf{w}_1^T \mathbf{x}(1) + b_1 & \cdots & \mathbf{w}_{\tilde{N}}^T \mathbf{x}(1) + b_{\tilde{N}} \\ \vdots & \ddots & \vdots \\ \mathbf{w}_1^T \mathbf{x}(N) + b_1 & \cdots & \mathbf{w}_{\tilde{N}}^T \mathbf{x}(N) + b_{\tilde{N}} \end{bmatrix} \in \mathbb{R}^{N \times \tilde{N}} \quad (\text{Eq. 4.17})$$

Although neural networks with linear activation function are only able to express linear input-output mapping, by limiting our scope on linear neural networks, the study can be analyzed in mathematical detail with linear algebra directly applicable.

4.4.2.2 Knowledge Model of a Conditioned ELM

In this section, an unsupervised learning algorithm is proposed to improve the conditioning of ELM. It is unsupervised because labeled data is not required for the proposed conditioning; the conditioning is done on input weight selection directly with no access of labeled data. The knowledge model is explained as follows.

Given a matrix $\mathbf{H} = [\mathbf{q}_1, \dots, \mathbf{q}_{\tilde{N}}] \in \mathbb{R}^{N \times \tilde{N}}$ where $\mathbf{q}_i$ is the i^{th} column vector of $\mathbf{H}$, we can define a correlation matrix $\mathbf{A}$ such that $\mathbf{A} = \mathbf{H}^T \mathbf{H} \in \mathbb{R}^{\tilde{N} \times \tilde{N}}$. By definition we have $\sigma_i(\mathbf{H}) = \sqrt{\lambda_i(\mathbf{H}^T \mathbf{H})} = \sqrt{\lambda_i(\mathbf{A})}$ where $\sigma_i(\mathbf{H})$ denotes the i^{th} singular value of $\mathbf{H}$, and $\lambda_i(\mathbf{A})$ denotes the i^{th} eigenvalue of $\mathbf{A}$. Therefore, according to Eq. 4.14 the spectrum condition number can be rewritten as,

$$\kappa_2(\mathbf{H}) = \sqrt{\frac{\lambda_{\max}(\mathbf{A})}{\lambda_{\min}(\mathbf{A})}} \quad (\text{Eq. 4.18})$$

where $\lambda_{\max}(\mathbf{A})$ and $\lambda_{\min}(\mathbf{A})$ are matrix $\mathbf{A}$'s largest and smallest eigenvalues respectively.

Let $\mathbf{q}_i \perp \mathbf{q}_j$ and $\|\mathbf{q}_i\|_2 = \|\mathbf{q}_j\|_2 = k \geq 0, \forall i \neq j$, and then we have $A_{ij} = \mathbf{q}_i^T \mathbf{q}_j = \begin{cases} k^2 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$. It can be seen that $\mathbf{A}$ is a diagonal matrix of identifiable diagonal elements.

Therefore, it follows that $\lambda_{\max}(\mathbf{A}) = \lambda_{\min}(\mathbf{A})$. From Eq. 4.14, the 2-norm condition number $\kappa_2(\mathbf{H})$ is one, which is the global minimum.

Remark 1: By constructing a coefficient matrix $\mathbf{H}$ of which columns satisfy $\mathbf{q}_i \perp \mathbf{q}_j$ and $\|\mathbf{q}_i\|_2 = \|\mathbf{q}_j\|_2 = k \geq 0, \forall i \neq j$, we can guarantee the perfect conditioning of matrix $\mathbf{H}$. As a consequence, the calculated solution of Eq. 4.2 is robust with respect to changes of $\mathbf{H}$ and $\mathbf{T}$.

If linear hidden neurons are considered, coefficient matrix $\mathbf{H}$ can be expressed in the form of $\mathbf{H} = [\tilde{\mathbf{X}}^T \tilde{\mathbf{w}}_1, \dots, \tilde{\mathbf{X}}^T \tilde{\mathbf{w}}_{\tilde{N}}] \in \mathbb{R}^{N \times \tilde{N}}$, where $\tilde{\mathbf{w}}_i = [\mathbf{w}_i^T b_i]^T \in \mathbb{R}^{(d+1)}$ and $\tilde{\mathbf{X}} = [\tilde{\mathbf{x}}(1), \dots, \tilde{\mathbf{x}}(N)] \in \mathbb{R}^{(d+1) \times N}$, by defining $\tilde{\mathbf{x}}(n) = [\mathbf{x}(n)^T 1]^T \in \mathbb{R}^{(d+1)}$.

Remark 2: According to *Remark 1*, we want to adjust $\tilde{\mathbf{w}}_i$ for $i = 1, \dots, \tilde{N}$, such that for $\forall i \neq j$, $\tilde{\mathbf{X}}^T \tilde{\mathbf{w}}_i$ is orthogonal to $\tilde{\mathbf{X}}^T \tilde{\mathbf{w}}_j$, i.e. $(\tilde{\mathbf{X}}^T \tilde{\mathbf{w}}_i)^T (\tilde{\mathbf{X}}^T \tilde{\mathbf{w}}_j) = 0$ and $\|\tilde{\mathbf{X}}^T \tilde{\mathbf{w}}_i\|_2 = \|\tilde{\mathbf{X}}^T \tilde{\mathbf{w}}_j\|_2$.

In order to find input weights for all columns, we consider columns one by one. Given a randomly selected $\tilde{\mathbf{w}}_1$, we have $\tilde{\mathbf{X}}^T \tilde{\mathbf{w}}_1$ as the first column of $\mathbf{H}$. For the second column it is necessary to have $(\tilde{\mathbf{X}}^T \tilde{\mathbf{w}}_2)^T (\tilde{\mathbf{X}}^T \tilde{\mathbf{w}}_1) = 0$ according to *Remark 2*. Note that $(\tilde{\mathbf{X}}^T \tilde{\mathbf{w}}_2)^T (\tilde{\mathbf{X}}^T \tilde{\mathbf{w}}_1) = \tilde{\mathbf{w}}_2^T \tilde{\mathbf{X}} \tilde{\mathbf{X}}^T \tilde{\mathbf{w}}_1$, and since $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{w}}_1$ are known, we can find a vector space $\mathbb{S}$ such that $\forall \mathbf{a} \in \mathbb{S}$, $\mathbf{a}$ is orthogonal to $\tilde{\mathbf{X}} \tilde{\mathbf{X}}^T \tilde{\mathbf{w}}_1 \in \mathbb{R}^{d+1}$. This vector space $\mathbb{S}$ is also known as the *null space* of matrix $[(\tilde{\mathbf{X}} \tilde{\mathbf{X}}^T \tilde{\mathbf{w}}_1)^T] \in \mathbb{R}^{1 \times (d+1)}$. Therefore, we can simply choose φ which is one normalized basis of the null space, and then calculate $\tilde{\mathbf{w}}_2 = \frac{\|\tilde{\mathbf{X}}^T \tilde{\mathbf{w}}_1\|}{\|\tilde{\mathbf{X}}^T \varphi\|} \varphi$ to allow the two columns to have identical norms. For the third column, using obtained $\tilde{\mathbf{w}}_1$ and $\tilde{\mathbf{w}}_2$ we continue to find $\tilde{\mathbf{w}}_3$ such that $\tilde{\mathbf{w}}_3$ is orthogonal to both $\tilde{\mathbf{X}} \tilde{\mathbf{X}}^T \tilde{\mathbf{w}}_1$ and $\tilde{\mathbf{X}} \tilde{\mathbf{X}}^T \tilde{\mathbf{w}}_2$. Similarly, this can be solved by finding the null space of matrix $\begin{bmatrix} (\tilde{\mathbf{X}} \tilde{\mathbf{X}}^T \tilde{\mathbf{w}}_1)^T \\ (\tilde{\mathbf{X}} \tilde{\mathbf{X}}^T \tilde{\mathbf{w}}_2)^T \end{bmatrix} \in \mathbb{R}^{2 \times (d+1)}$.

In this manner, iteratively we can analytically calculate $\tilde{\mathbf{w}}_i$ for $i = 1, \dots, \tilde{N}$. However, the existence of $\tilde{\mathbf{w}}_i$ should be discussed. Assume that we want to calculate $\tilde{\mathbf{w}}_{\tilde{N}}$, and from previous iterations we have $\tilde{\mathbf{w}}_i$ for $i = 1, \dots, \tilde{N} - 1$. Then $\tilde{\mathbf{w}}_{\tilde{N}}$ is supposed to be calculated by solving the null space of matrix $\mathbf{\Gamma} = \begin{bmatrix} (\tilde{\mathbf{X}} \tilde{\mathbf{X}}^T \tilde{\mathbf{w}}_1)^T \\ \vdots \\ (\tilde{\mathbf{X}} \tilde{\mathbf{X}}^T \tilde{\mathbf{w}}_{\tilde{N}-1})^T \end{bmatrix} \in \mathbb{R}^{(\tilde{N}-1) \times (d+1)}$. Assume this matrix is not rank deficient, i.e. $\text{rank}(\mathbf{\Gamma}) = \min(\tilde{N} - 1, d + 1)$. For the dimension of null space of $\mathbf{\Gamma}$, we have [97],

$$\dim(\text{null}(\mathbf{\Gamma})) + \text{rank}(\mathbf{\Gamma}) = d + 1 \quad (\text{Eq. 4.19})$$

```

1 Uniformly randomly select  $\tilde{\mathbf{w}}_1 \in \mathbb{R}^{d+1}$ ;
2 for  $i = 2, \dots, \tilde{N}$  do
3   Calculate  $\tilde{\mathbf{X}}\tilde{\mathbf{X}}^T \tilde{\mathbf{w}}_1, \dots, \tilde{\mathbf{X}}\tilde{\mathbf{X}}^T \tilde{\mathbf{w}}_{i-1}$ , where  $\tilde{\mathbf{X}} = [\tilde{\mathbf{x}}_1, \dots, \tilde{\mathbf{x}}_N] \in \mathbb{R}^{(d+1) \times N}$ ;
4   Find the null space  $\mathbb{S}$  of matrix  $\begin{bmatrix} (\tilde{\mathbf{X}}\tilde{\mathbf{X}}^T \tilde{\mathbf{w}}_1)^T \\ \vdots \\ (\tilde{\mathbf{X}}\tilde{\mathbf{X}}^T \tilde{\mathbf{w}}_{i-1})^T \end{bmatrix} \in \mathbb{R}^{(i-1) \times (d+1)}$ ;
5   Randomly select a vector  $\varphi$ , where  $\varphi$  is one normalized basis of  $\mathbb{S}$ ;
6   Calculate  $\tilde{\mathbf{w}}_i$  by  $\tilde{\mathbf{w}}_i = \frac{\|\tilde{\mathbf{x}}^T \tilde{\mathbf{w}}_{i-1}\|}{\|\tilde{\mathbf{x}}^T \varphi\|} \varphi$ ;
7   Loop until either  $\tilde{N}$  or the upper bound  $d + 1$  is reached
8 end

```

Figure 4.10: Algorithm for Unsupervised Input Weight Selection

In order to guarantee the existence of non-empty null space, i.e. $\dim(\text{null}(\Gamma)) > 0$, from Eq. 4.19 we have $d+1 - \text{rank}(\Gamma) > 0$. Therefore, $d+2 > \tilde{N}$ and the upper bound of applicable hidden neuron is $d + 1$.

As a summary, the algorithm of finding input weights up to $d + 1$ is shown in Figure 4.10. After calculating the input weights, output weight is calculated as the same as original ELM algorithm.

Computational complexity has been investigated to analyze efficiency of the proposed algorithm. According to Figure 4.10, Step 3 requires the computation of matrix $\tilde{\mathbf{X}}\tilde{\mathbf{X}}^T$. Although it takes $(d + 1)^2 N$ operations [97], only one time calculation is required by saving the result. In addition, Step 3 takes $\tilde{N}^2(d + 1)^2$ operations to obtain $\tilde{N}$ columns. If assuming that both output weights and null space are computed using Singular Value Decomposition (SVD), then number of operations for calculating output weights is $14N\tilde{N} + 8\tilde{N}^3$ and $4N\tilde{N} + 8\tilde{N}^3$ for null space [97]. Let $\tilde{N} = d + 1$, then the upper bound of operations is $11\tilde{N}^4 - 2\tilde{N}^3 + 15\tilde{N}^2N$ which is $O(\tilde{N}^4 + \tilde{N}^2N)$. Comparing the complexity of original ELM which is $O(\tilde{N}^3 + \tilde{N}^2N)$, additional complexity is introduced due to the iterations used for input weight selection. However, the number of iteration is bounded by input dimension d , and the additional complexity can

be ignored for most practical problems $N \gg d$. Exceptions are problems for biological data which usually incline to have more input dimensions than number of data.

4.4.3 Agent's Performance-Driven Behaviors

After the introduction of two algorithms, supervised ELM and its unsupervised preconditioning -Conditioned ELM, a behavior analysis is to be discussed in this section by examining how agent's need of the pursuit of performance impact an agent's competence.

Once the availability of the two knowledge from external knowledge base is perceived by the agent, the intrinsically motivated agent will follow the GoalNet with the root goal "To Achieve Novelty" to master the skills. The learning process is identical to the GoalNet in Figure 4.3 and 4.6, and the details are thus ignored in this section.

Given the two algorithms as the domain knowledge of the intrinsically motivated agent, numerical experiments are to be conducted to compare the behaviors of agent driven by performance. The agent is proactively to use the GoalNet in Figure 4.7 in fulfilling external request. The knowledge selection is made between the original ELM with random input weights and the conditioned ELM with robust input weight selection. The performance feedback is accessed for both domain knowledge. The experiment is explained as follows.

4.4.3.1 External Request: Prediction of Breiman's Linear Regression

Breiman's linear regression was used for comparisons between original and proposed algorithm. The regression data was synthesized according to Breiman's work [98] which is commonly used by statisticians as a mouse experiment of testing algorithms. The input vector is a random vector $X = [X_1, \dots, X_{30}]^T$ which follows a multivariate normal distribution with zero means. The covariance matrix Σ was designed as,

$$\Sigma = R^{|i-j|}, i, j = 1, \dots, 30$$

R was set to be 0.7 for this experiment. The response was generated according to the linear model,

$$Y = \beta_1 X_1 + \dots + \beta_{30} X_{30} + \epsilon$$

where the noise $\epsilon \in N(0, 1)$ and the coefficients $\beta_m = \gamma \alpha_m$. Given h is a positive integer, $\{\alpha_m\}$ can be defined in three groups,

$$\text{if } |m - 10| < h, \alpha_m = (h - |m - 10|)^2;$$

$$\text{if } |m - 20| < h, \alpha_m = (h - |m - 20|)^2;$$

$$\text{if } |m - 30| < h, \alpha_m = (h - |m - 30|)^2;$$

The value of γ is determined such that the signal to noise ratio was approximately equal to one. Two values 1 and 5 were used for h . For $h = 1$, there were 3 strong nonzero coefficients, and for $h = 5$ there are 23 weak nonzero coefficients. For this experiment, 60 observations were generated.

Agents require a normalization of all the input attributes (except expected outputs) into the range of $[0, 1]$. Random variables for both original ELM and the conditioned ELM were uniformly randomly distributed on the interval $[0, 1]$. Calculations of least squares and null space of a matrix were implemented by SVD. Results and analysis are presented in the rest of this section.

4.4.3.2 Comparison of Agent Performance Feedbacks

Both accuracy and robustness of the solutions were considered. Before each trail of experiment, we randomly split the entire data into training data and testing data according to the ratio of 1 : 1. Totally 80 trials of each dataset were repeated to obtain reliable results and the average over all the trails was used for comparison.

The performance was evaluated in terms of Root Mean Square Error (RMSE) and size of condition number for accuracy and robustness respectively. Condition number of coefficient matrix $\mathbf{H}$, $\kappa_2(\mathbf{H})$, was calculated using Matlab function *cond()*.

(a)

(b)

Figure 4.11: Comparison Between Random Input Weight Selection (solid line) and Pre-conditioned Input Weight Selection (dashed line) in Terms of Condition Number (a) and Testing RMSE (b) for Breiman's Linear Regression Data with $h = 1$. 10 Hidden Neurons Were Used In This Case

(a)

(b)

Figure 4.12: Comparison Between Random Input Weight Selection (solid line) and Pre-conditioned Input Weight Selection (dashed line) in Terms of Condition Number (a) and Testing RMSE (b) for Breiman's Linear Regression Data with $h = 1$. 25 Hidden Neurons Were Used In This Case.

Figure 4.4.3.2 and Figure 4.4.3.2 present the comparison results between random and pre-conditioned input weight selection for ELM networks of 10 and 25 hidden neurons. $h = 1$ was set for Breiman's linear regression for the two figures. In Figure 4.4.3.2 the robust algorithm

achieved perfect conditioning with $\kappa_2(\mathbf{H}) = 1$ for all trials, while the ELM with random input weight selection had much larger $\kappa_2(\mathbf{H})$ around 260. In addition, for the robust weight selection algorithm accuracy in terms of RMSE was still comparable with original ELM.

If 25 hidden neurons were used for the same data, Figure 4.4.3.2 indicates that ELM of random input weights increased its condition number to 4000, and the conditioned ELM maintained its perfect robustness with comparable accuracy.

Table 4.1: Comparison Between Random Input Weight Selection and Proposed Input Weight Selection

Setting	#Neurons ¹	Testing RMSE		Condition Number		Training Time ²	
		Random	Proposed	Random	Proposed	Random	Proposed
$h = 1$	10	1.5168	1.4612	264.8923	1	3.91E-04	2.73E-03
$h = 1$	25	2.4491	2.2361	4381.9110	1	1.95E-03	1.84E-02
$h = 5$	10	1.8002	1.7311	275.2170	1	1.56E-03	3.91E-03

1. The Number of Hidden Neurons Used in Corresponding Neural Networks; 2. in Seconds.

The average results for 80 trials are presented in Table 4.1. Similar to previous results shown in the figures, accuracy in terms of RMSE was not downgraded by the conditioned learning algorithm, while stability in terms of condition number was greatly improved. However, the conditioned learning algorithm required more training time than random selection. Table 4.1 shows that training time for conditioned ELM was around ten times more expensive than that for original ELM. It is because for this experiment N was not much larger than d . Therefore, additional complexity posed by the conditioned ELM cannot be neglected. Nevertheless, the actual cost of time was still acceptable which was just about a few milliseconds.

The feedbacks of agents with the two domain knowledge are different. Agent with original ELM domain knowledge achieves better speeding, while agent with conditioned ELM outperforms the former with better robustness. The feedbacks of the two domain knowledge are stored in agent's knowledge base; an intrinsically motivated agent will use the scores of

feedbacks in the goal/action selection, which will be discussed in Section 4.6. It will be the case that an agent choose the conditioned ELM if it considers the numerical stability more than the responsiveness, or it concerns more on the performance given perturbations in the data. However, the calculation of null space take a great amount of time and computational resources; further more, it is only suitable for linear problems, though stability is significantly improved without downgrading accuracy. Given a non-linear and time-limited request, the feedback score of original ELM will be better than the one with input weight conditioning. In either case, agent will proactively turn to the better solution, once driven by the performance.

4.5 Relatedness Need Pursuit

Social interactions among multiple agents take several forms, including consulting with others, learning from experienced veterans, collaboration and coordination in dealing with one problem. There also exist other forms of interaction that are more adverse or hostile, such as competition against or negotiation with others. According to SDT theory, the pursuit of relatedness is relevant to being cared by others or helpful to peers. Therefore, in this thesis we consider only cooperative interactions, either requesting a suggestion, or responding to a request.

Cooperation has itself proved in the fact that it is one nature way to conquer the limitation posed by unbalanced distribution of information and noisy input from an open environment. The rational is that a single agent is unavoidably constrained to limited resources, while multiple agents are likely to cover a larger scope and have a better understanding of a problem; sharing knowledge and exchange opinions are beneficial for a more generalized and thus accurate problem solving.

However, sharing with other agents can happen at various levels. Agent can simply share opinions of a problem with others, or share one's entire knowledge base with other agents. At the finest level, relatedness occurs as an alternative of a skill, meaning instead of acting out

of its own effort, an agent may start a round of pooling requests from others and synthesize individual responses as its own decision. This process is initiated by the urge of relatedness need, and evaluated in the same way as the performance-related competence.

At the highest level, relatedness occurs at the formalization of knowledge. Information of states and sub-goals is passed from one agent to the other as the building block of agent's own knowledge. Relatedness provides the need of knowledge sharing and promotes the reusability of essential GoalNet building blocks for agent autonomous knowledge formalization.

The purpose of this section is to explain how to allow an motivated agent to act according to its related need, as an appreciation of agent collaboration in the context of a multiagent system. The sophisticated latter approach is out of the scope. Particularly, the thesis focuses on how to exchange decisions and how such collaboration can benefit the entire ecosystem.

4.5.1 Goal-Oriented Representation of Pursuit of Relatedness

This section discusses two types of agent GoalNet for the relatedness need, namely “To Ask” and “To answer”.

4.5.1.1 Root Goal of “To Ask”

The GoalNet representing agent's sub-goal of “To ask” a suggestion from peers is shown in Figure 4.13.

The root goal is initialized by **MI**, when the relatedness need is high; it is usually associated with a certain performance task, in which the agent is uncertain about its performance; a sample case is that the agent's performance is fluctuated a lot over a recent period. Once the GoalNet is selected, the control module starts to prepare request according pre-define message format for agent communication. The message body specifies {“type” : “collaboration”, “collaboration_type” : “ask”, “ask_type” : “prediction”, “context” : *context_value*, “input” : *question_input*}. After the message is ready, the main thread passes

Figure 4.13: GoalNet Representation of "To Ask"

the control back to **GE** to instantiate a communication channel. The control module gets the notification when the channel is established, and then asks actuators of **A** to send the message to the request listener in the environment. The hand-shake ends after the agent receives the acknowledgement from the **P**.

4.5.1.2 Root Goal of "To Answer"

Similarly, the GoalNet representing agent's sub-goal of "To answer" a peer's question is shown in Figure 4.14.

Figure 4.14: GoalNet Representation of "To Answer"

The root goal is initialized by **MI**, when the relatedness need is high and there is an ex-

tern event requesting this collaboration. Once the root goal has been initiated, the control module proceeds to prepare the reply message compliant to pre-define format for agent communication. The message body specifies {“type” : “collaboration”, “collaboration_type” : “answer”, “answer_type” : “prediction”, “context” : *context_value*, “output” : *answer_output*}. After the readiness of the message, the main thread passes the control back to **GE** to establish a communication channel. Similarly, the control module waits until the channel is established, and then asks actuators of **A** to send the message to the request listener through communication modules. The hand-shake ends after the agent receives the acknowledgement from the **P**.

It is worth mentioning that above example is almost identical to the GoalNet shown in Figure 4.9 which are seeking for tasks. This gives a hint that the design of GoalNet library could be simplified if the building block is implemented in modularity. A modular design also facilitates knowledge formation, which will be discussed as the pursuit of autonomy in Section 4.6.

The GoalNets built for cooperation requires the support of distributed reasoning, where the agent knowledge model should be able to take the advantage of agent collaboration. A knowledge model for distributed reasoning is illustrated as an example. The details are given as follows.

4.5.2 Knowledge Model of Collaborative Reasoning

Multi-agent based collaboration is a distributed artificial intelligence approach with machine learning capability. Two terms are essential to agent’s collaboration for real-life applications, namely response speed and accuracy. At hardware level, it is well known that distributed and paralleled approach outperforms single processor in terms of scalability, speed of response, etc. We need a pilot project to evaluate whether multiple learning instances can achieve better performance than a single learning entity in terms of the correctness of knowledge acquired.

Extreme Learning Machine (ELM) is a good starting point, due to the fact that it has learning ability with low algorithm complexity, can produce fast learning rate which is essential to practical usage. Although ELM currently only provides implicit knowledge which is not the proposed for knowledge sharing and collaboration, a simple collaboration approach will be proposed in this section based on each ELM learning. Despite the novelty of a single ELM, the random assignment creates inconsistency of network decisions, and particularly the decision boundary for a classification problem is changing from each run. Such problem affects the generalization ability for an ELM network. Due to the fast learning speed of ELM, it is possible to apply meta-learning to form a committee of networks instead of a single one. In this section, we propose a method to assemble a set of ELM networks using stacked generalization.

We first point out the problem that an ELM may have, and propose an ensemble of ELM using stacked generalization as a representation of collaboration.

4.5.2.1 Limitation of a Single ELM

From Section 4.3.2.1, we understand that ELM randomly assigns input weights and biases instead of adaptively tuning them. The idea is to replace interactive training process into a single strike. It has advantage in gaining dramatic fast learning speed, yet pays its price. Due to the simple structure of ELM, the output weights analytically calculated have a close relationship to the initialized input weights and biases, which are in randomness. This characteristic of ELM inevitably results in a stochastic behavior of its performance.

To illustrate this idea, experiments have been conducted on two selected features (petal length and petal width) of UCI Iris data [99]. The two features were used to classify 150 Iris data to three classes (setosa, virginica and versicolor). A sample decision boundary of ELM is plotted in Figure 4.5.2.1.

ELM can correctly classify most data points, though some classification errors occur around the boundary between Class 2 and Class 3. Figures 4.15.a and 4.15.b come from two independent runs of ELM, and it is clear that decision boundaries for the two runs are quite different,

(a)Run One

(b)Run Two

Figure 4.15: Comparison of Decision Boundaries from Two Independent Runs. This figure gives the 2D decision boundary obtained by ELM for Iris data. Square, circle and star denote 3 classes of data points, Class 1 (sentosa), Class 2 (virginica) and Class 3 (versicolor) respectively. The shaded areas present the regions belonging to Class 2. X-axis and Y-axis denote the normalized petal length and petal width respectively.

exhibiting a degree of randomness. Attentions should be paid around point (0.7, 0.4) of the figure, where inconsistent decisions are made, (i.e. for one figure it belongs to Class 2, yet for the other not). This inconsistency comes from the random assignment at the initial stage. The claim can be justified for the fact that there are only two stochastic processes in the training of ELM, one for permutation of input data and the other for the weights and biases assignment. It is obvious that switching rows in $\mathbf{H}$ and $\mathbf{T}$ will not affect the solution β in Eq. 4.6. In consequence, it is the arbitrary assignment for input weights and biases that results in the inconsistent decisions. However, it is worth mentioning that other feedforward networks, e.g. RBF, employing stochastic approach in the initial stages more or less experience inconsistency as well. In addition, such stochastic behaviors often occur in the areas where no training samples have been presented, or equivalently the regions which require generalization.

It is well known in statistics that combining estimators outperforms selecting the single best [98]. To improve the generalization for which a single ELM incorporating random assignment may not do quite well, we adopt meta-learning approaches, in the field of which numbers of studies have found that a collection of models, i.e. a committee, are less biased, and have improved generalization performance. We particularly select stacked generalization which offers an effective way to combine decisions from the committee. The following sections introduce the integration of multiple ELMs by stacked generalization.

4.5.2.2 A Knowledge Model of Collaboration with Stacking ELM

From previous sections, we understand that a single ELM network simply randomly assigns input weights and biases, instead of explicit training as in gradient-decent based network. The idea of random assignment releases the feedforward neural net from demanding training for input weights or biases. It also ineluctably leads to certain stochastic behaviors of ELM, which bring about uncertainty and susceptibility to errors. This section first gives the basic idea of stacked generalization, followed by introducing the proposed stacking ELM.

Background of Stacked Generalization Stacked generalization [100] comprises two different stages, namely level-0 and level-1. Level-0 estimators refer to a set of models in the committee, and original data, called level-0 data, is applied to train level-0 estimators. There is no difference between level-0 training and normal individual neural network training. After training level-0 estimators, a validation set of inputs is applied to the trained models, and the collection of their individual predicated outputs together with the desired output of the validation dataset form the training data, called level-1 data, for a higher level learning. A higher level model, i.e. level-1 generalizer, uses the outputs of individual level-0 estimators to predicate expected outputs, and then learns the mapping between the two. However, the design of level-1 generalizer is an open art, and may have various choices from decision trees to neural networks.

Stacked generalization has its novel idea in adopting a higher level generalizer to make group decision based on individual members, instead of just selecting a single best model to represent the group idea, which is usually used in cross-validation model selection. The decision power is distributed from a single winner to the ensemble in order to achieve less biased performance. After stacked generalization opened the door to new thinking about model selection, many researchers shift thinking from selecting of the best single model to forming combination of models using level-1 data.

However, the limitation of stacked generalization is the loss of explanatory power. Combining weighted decisions is difficult to comprehend, though assigning larger weight is usually an analogue to giving more influence or more decision power to an individual, and combined decision can be understood as assembled confidence among the committee. However, the results of stacked generalization still cannot be produced by a single model, and it usually results in a complex decision border for a classification problem, making resulted boundaries difficult to interpret.

Proposed Stacking ELM This section describes the procedure of constructing the proposed stacking ELM as follows:

Firstly define K members of the committee network, M_k for $k = 1, \dots, K$, which are so called level-0 estimators. Given a dataset $L = \{(\mathbf{x}_i, \mathbf{t}_i), i = 1, \dots, N\}$, where $\mathbf{x}_i \in \mathbb{R}^n$ and $\mathbf{t}_i \in \mathbb{R}^m$ represent input vectors and output vectors respectively, randomly split dataset L into J equal parts $L_1, L_2, \dots, L_J$, where J is an integer. For the j^{th} fold of the J-fold cross-validation, define $L^{(-j)} = L - L_j$ as the level-0 training data and L_j as the testing data. Separately invoke the K level-0 estimators in the committee using $L^{(-j)}$ data to produce a set of trained level-0 models, $M_k^{(-j)}$, for $k = 1, \dots, K$.

Apply the testing data L_j of the j^{th} fold to each $M_k^{(-j)}$ model and obtain their corresponding outputs. Consequently, we have predictions $\{\mathbf{z}_{i1}, \mathbf{z}_{i2}, \dots, \mathbf{z}_{iK}\}$ for each input $\mathbf{x}_i$ in L_j , where $\mathbf{z}_{ik} \in \mathbb{R}^m$ denotes the k^{th} model prediction on $\mathbf{x}_i$.

Then keep a copy of the new pairs $\{(\mathbf{z}_{i1}, \mathbf{z}_{i2}, \dots, \mathbf{z}_{iK}, \mathbf{t}_i), i = 1, \dots, |L_j|\}$, where $|L_j|$ indicates the number of data pairs in L_j .

After the entire J-fold cross-validation, obtain a new dataset $L_{CV} = \{(\mathbf{z}_{i1}, \mathbf{z}_{i2}, \dots, \mathbf{z}_{iK}, \mathbf{t}_i), i = 1, \dots, N\}$. Compare it with original dataset L . Note that original $\mathbf{x}_i$ is replaced with corresponding level-0 output vectors $\{\mathbf{z}_{i1}, \dots, \mathbf{z}_{iK}\}$. L_{CV} is so called level-1 data. At the stage of level-1 learning, we derive a level-1 model $\tilde{M}$ from L_{CV} . Figure 4.16 summarizes the stacking ELM training process.

We adopted a linear combination of level-0 estimator predications as the level-1 generalizer $\tilde{M}$ which has been exploited in [98], [101], for its simplicity and restricting attention to stacked generalization itself. Specifically, the committee M makes predication on input $\mathbf{x}$ of the form:

$$M(\mathbf{x}) = \sum_k \alpha_k M_k(\mathbf{x}) \quad (\text{Eq. 4.20})$$

where $M_k(\mathbf{x})$ is the k th level-0 predication, and α_k is its corresponding weight. Since a multi-response problem can be always decomposed into multiple single-response problems [101],

Figure 4.16: Sketch of Stacking ELM Training. This figure summarizes the procedure of training a stacking ELM. The rounded corner rectangular denotes the ensemble of individual ELM networks (shadowed circles). J-fold cross-validation is applied to produce level-1 data. Level-1 generalizer is trained on the generated level-1 data.

we try to find the solution for minimizing the m th single-response:

$$\sum_j \sum_{(\mathbf{x}_i, t_{im}) \in L_j} \left(t_{im} - \sum_k \alpha_{km} M_{km}^{(-j)}(\mathbf{x}_i) \right)^2 \quad (\text{Eq. 4.21})$$

where a scalar t_{im} presents the expected output of the m th single response, and subscript km denotes the m th single response of the k th model. We also simply implement the level-1 generalizer as averaging the predictions of all level-0 models, i.e. $\alpha_k = 1/K$ for all k . According to [101], the averaging level-1 generalizer produces an error rate almost identical to majority vote. We evaluated both operators in our experiments.

Besides, several regulation techniques are specified in [102], which replace the unrestricted least squares estimators with restricted ones. Breiman [98] suggests that non-negative constraint $\alpha_k \geq 0$ provides consistently good results.

After training the level-1 model, we discard all the intermediate $M_k^{(-j)}$ models, and complete the stacking ELM training by deriving final level-0 estimators M_k from all the data in L . The generalization process is straightforward: Simply pass a new instance to M_k and acquire a set of output vectors from level-0 estimators, $\{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_K\}$. This is the input to the level-1 generalizer $\tilde{M}$, whose output is the final predication for that instance.

4.5.3 Agent Collaboration from Machine Learning Perspective

In this section, we conduct a series of experiments, and provides detailed result analysis from the view of machine learning. The purpose of this experiment is to justify collaborative reasoning of multiple agents outperforms single agent for a noisy environment with limited access of resources.

GoalNets under motivation ‘Pursuit of Relatedness’ are involved in agent’s collaborative reasoning. A collaboration initializer starts with the GoalNet with root goal “To ask” shown in Figure 4.13. To respond the request, a collaboration agent replies in the way defined in the GoalNet with root goal “To answer” shown in Figure 4.14. Zooming in the responding GoalNet, a respondent prepares the message by making its own prediction on the request, which is

similar to the “To fulfill external request” in Figure 4.7. Correspondingly, the requestor receives the replies and stores them in its runtime data repository. The experiment is conducted from the perspective of the collaboration starter agent. After the starter agent collects all the feedbacks, a collaborative reasoning is carried out followed by the knowledge model of stacking ELM. The collaborative reasoning is preceded by collaborative learning explained in previous section. The performance of the proposed stacking ELM is compared with single ELM, and evaluation has been done on different stacking formalizations. We employed several benchmarking data, either artificial or real-world, for both regression and classification problems.

For all experiments, we first randomly split the entire data into L and L_{TS} according to the ratio of 9:1 for training and testing unless otherwise specified. Recall that L is used in the exact way as stated in Section 4.5.2.2 for training stacking ELM. L_{TS} was applied to test and compare structures. The entire procedure was repeated for 50 trials to obtain reliable results, and the average over all the trials was used for comparison.

To complete the analysis of stacking ELM network, comparison between stacking ELM and its single best estimator was designed. Single best estimator represents an individual network in the committee, which has the best performance in the J-fold cross-validation on data L . That is, during J-fold cross-validation we can always identify a single network from the ensemble with the highest classification rate or the smallest mean squared error among all the level-0 models. We extracted this single best model from the committee and compared it with its ensemble network it belongs to on data L_{TS} .

Because of the initial random assignment in ELM network, number of hidden neurons is the only free parameter. We also investigated the performance of combining ELM networks of identical and different numbers of hidden neurons.

We normalized all the input attributes (except expected outputs) into the range $[0, 1]$. Input weights of ELM were uniformly randomly distributed on the interval $[-1, 1]$, and biases on $[0, 1]$. Sigmoid was selected as the activation function, and Moore-Penrose general inverse was

calculated using Single Value Decomposition (SVD). The results and analysis are presented in the rest of this section.

Regression Problems In the experiments, 7 regression benchmarking datasets [99], [103], [96] were applied to compare single ELM and stacking ELM. Table I gives the specifications of the regression problems.

Table 4.2: Specifications of Regression Problems

Data	# Cases	#Attributes	# Neurons
Auto Price	159	15 (1)	15
Machine CPU	209	6 (1)	10
Auto-MPG	392	7 (1)	25
Boston Housing	506	13 (1)	30
Stock Domain	950	9 (1)	110
Delta Ailerons	7129	5 (1)	45
SinC	10000	1 (1)	20

Datasets are shown in the order of increasing size. Missing values in Auto-MPG are removed from the dataset. SinC is artificial data, whereas others are real-world data. SinC is halved for training and testing. The column of “# Attributes” contains input attributes, and output attributes (shown in parentheses). “# Neurons” indicates the average number of hidden neurons used in single ELM and stacking ELM.

First, 5 identical ELM networks were used to assemble the proposed stacking ELM. Every ELM, either a single one or a member of the committee, shares the same structures. The numbers of hidden neurons for different regression problems are presented in Table 4.2. Table 4.3 compares the least squares (defined in Eq. 4.21) and the averaging level-1 generalizers. From the results, it is clear that stacking ELM providing smaller RMSE outperforms single ELM in generalization for both small and large regression problems. For most cases stacking ELM also obtains a smaller standard error. A smaller standard error indicates the underlying network has more consistent performance through all the runs. Additionally, the averaging level-1 generalizer has a slight improvement over the least squares one. This implies that constructing

Table 4.3: Comparison of Single ELM and Stacking ELM of Same Structures For Regression Problems

Data	Single	Stacking (least squares)	Stacking (averaging)
Auto Price	2810.9 (690.84)	2589.2 (696.56)	2540.9 (683.11)
Machine CPU	56.945 (31.644)	51.058 (28.748)	48.418 (30.027)
Auto-MPG	2.7469 (0.4970)	2.6959 (0.4826)	2.6679 (0.4701)
Boston Housing	4.3714 (0.9350)	3.7738 (0.6271)	3.6819 (0.5996)
Stock Domain	0.9205 (0.0800)	0.8700 (0.1191)	0.8581 (0.1129)
Delta Ailerons	1.6648e-4 (8.91e-6)	1.6259e-4 (6.19e-6)	1.6255e-4 (6.24e-6)
SinC	0.0477 (0.0185)	0.0452 (0.0039)	0.0446 (0.0021)

Results for regression problems are measured using RMSE. Mean and standard error (shown in parentheses) for 50 trials are presented. The best performance is in boldface. Note that there is no normalization for expected outputs.

stacking ELM with far cheaper averaging operator other than the least squares is efficient for regression.

We also compared single ELM with stacking ELM of different hidden neuron numbers. The stacking ensemble was made of 5 ELM networks with the number of hidden neurons in the ratio of 5:8:10:12:15. For example, if the candidate single ELM has 20 hidden neurons, then the comparing ensemble is made of networks with 10, 16, 20, 24 and 30 hidden neurons. Thus both candidates have the same average number of hidden neurons (numbers are shown in Table 4.2). Experiments were conducted on the same set of regression problems, of which the results are given in Table 4.4 along with single ELM. Similar to the one of the same structure, stacking ELM of different number of neurons performs better than single ELM, and the

Table 4.4: Comparison of Single ELM and Stacking ELM of Different Structures For Regression Problems

Data	Single	Stacking (least squares)	Stacking (averaging)
Auto Price	2810.9 (690.84)	2662.8 (935.83)	2553.5 (912.29)
Machine CPU	56.945 (31.644)	53.983 (26.134)	49.892 (21.762)
Auto-MPG	2.7469 (0.4970)	2.7307 (0.5614)	2.7230 (0.5761)
Boston Housing	4.3714 (0.9350)	3.7390 (0.7329)	3.5744 (0.7984)
Stock Domain	0.9205 (0.0800)	0.8793 (0.2966)	0.8623 (0.2401)
Delta Ailerons	1.6648e-4 (8.91e-6)	1.6522e-4 (8.71e-6)	1.6511e-4 (8.55e-6)
SinC	0.0477 (0.0185)	0.0439 (0.0014)	0.0436 (0.0004)

Results for regression problems are measured using RMSE. Mean and standard error (shown in parentheses) for 50 trials are presented. The best performance is in boldface. Note that there is no normalization for expected outputs.

averaging operator still slightly improves the generalization results over the least squares one. Although it may not be meaningful to compare different stacking ELM networks in Table 4.3 and Table 4.4, there is no clue that committees of the same or different structures have obvious advantage over another, so the choice of the ensemble topology depends on a specific problem.

Classification Problems Capability of stacking ELM for classification was also examined. Our experiments applied 6 sets of benchmarking data of various sizes from UCI [99]. A detailed description is listed in Table 4.5. In a classification problem, the output is not continuous, but falls into one of the I classes, where I is an integer. Thus we need to preprocess a class label into a vector of length I , which has a one in the i th position and zeros otherwise.

Table 4.5: Specifications of Classification Problems

Data	# Cases	#Attributes	# Neurons
Iris	150	4 (3)	10
Ionosphere	350	33 (2)	20
Balance Scale	625	4 (3)	10
Pima	768	8 (2)	20
Vehicle	846	18 (4)	60
Image Segment	2310	18 (7)	200

Datasets are shown in the order of increasing size. The column of “# Attributes” contains input attributes, and number of classes (shown in parentheses). “# Neurons” indicates the average number of hidden neurons used in single ELM and stacking ELM.

Similarly to what has been done for regression, we set up stacking ELM of the same and different structures, and compared them with single ELM. The successful classification rates are specified in Table 4.6 and Table 4.7. The standard errors are left out for clarity. Stacking ELM consistently outperforms single ELM, except for the “Pima” case where one ensemble out of two is worse than the individual network. For classification, the least squares and the averaging level-1 generalizers have comparative scores, whereas the averaging operator is always superior to the least squares in regression cases.

From above comparisons, we can conclude that stacking ELM has better generalization performance than a single ELM network for both regression and classification. It validates the statement in statistics that combining estimators provides less biased results, and stacked generalization offers ELM based networks a possible approach to achieve improved generalization. In addition, we found that the averaging operator was always comparative to the least squares. This is due to the fact that ELM has random parameter assignment for every trial, and in consequence two ELM networks of identical structure will not act differently from each other in the long run, so it is not useful to assign different weights to members of the same structures in a stacking ELM.

Table 4.6: Comparison of Single ELM and Stacking ELM of Same Structures For Classification Problems

Data	Single	Stacking (least squares)	Stacking (averaging)
Iris	94.67	94.93	95.07
Ionosphere	87.26	88.91	88.80
Balance Scale	89.06	89.81	89.81
Pima	76.79	76.82	76.61
Vehicle	79.76	81.38	81.40
Image Segment	95.26	96.05	96.09

Results are shown in success rate (100%). The mean value for 50 trials is presented. The best performance is in boldface

Table 4.7: Comparison of Single ELM and Stacking ELM of Different Structures For Classification Problems

Data	Single	Stacking (least squares)	Stacking (averaging)
Iris	94.67	98.13	97.87
Ionosphere	87.26	88.97	88.80
Balance Scale	89.06	89.94	90.23
Pima	76.79	76.71	76.92
Vehicle	79.76	83.38	83.60
Image Segment	95.26	96.14	96.36

Results are shown in success rate (100%). The mean value for 50 trials is presented. The best performance is in boldface.

Miscellaneous Aspects of Stacking ELM Through the entire evaluation, we kept a track of the weights assigned to each level-0 model. Table 4.8 selects some cases from previous experiments. The first two rows of data are from regression, and the rest are from classification. The table shows that Row 1 and 4 have weights equally distributed among 5 models, for which the least squares operator has identical effect as the averaging. Row 5 is quite close to the averaging assignment. Furthermore in Row 2 and 3, some models contribute more to the committee whereas some are negligible.

Table 4.8: Weights Assigned to Five Level-0 Models

Data	M_1	M_2	M_3	M_4	M_5
Machine CPU	0.2000	0.2000	0.2000	0.2000	0.2000
SinC	0.0047	0.0453	0.1243	0.3469	0.4791
Iris	0.0215	0.0857	0.1750	0.2467	0.4719
Balance Scale	0.2000	0.2000	0.2000	0.2000	0.2000
Image Segment	0.2333	0.2584	0.2207	0.1659	0.1143

Weights of the 1st class single response are shown above. The average weight for 50 trials is presented. Weights are obtained using least squares level-1 generalizer under non-negative constraint. $M_{1...5}$ denotes 5 level-0 models with increasing number of hidden neurons.

We have analyzed the least squares level-1 generalizer with and without constraints. Recall that we may regulate the optimal solution in (Eq. 4.21) with a non-negative constraint, i.e. $\alpha_k \geq 0$. In our experiments, we sometimes noticed a portion of α_k being less than zero, indicating a negative weight assigned to the corresponding level-0 model. It often causes difficulties in interpreting the negative weights. Therefore, a non-negative constraint was added to ensure semantics integrity. Selected results are listed in Table 4.9. It is clear that constrained stacking ELM always scores higher than the non-constrained stacking network, or at least equal (as shown in Row 3), for both regression and classification problems. This finding is consistent with Breiman’s proof [98]. As a result, stacking ELM with non-negative constrained least squares achieves both higher accuracy and better semantics. Table 4.9 also compares stacking ELM with single best estimators. The result shows stacking ELM is always better than single best estimator, which follows the statement of Breiman that stacking never performs worse than single best. It is worth emphasizing that single best estimator is not necessarily always the same level-0 model. For example, in 50 runs 5 level-0 models, $M_1, \dots, M_5$ (in order of increasing number of neurons) have won as the single best for 0, 11, 7, 16 and 16 times, for “Auto-MPG” regression. Therefore single best performance is always better or at least equal to that of all individual models.

Table 4.9: Comparison of Least Squares Generalizer with and without Non-Negative Constraint, and Single Best

Data	Single Best	Stacking	Stacking (constraint)
Machine CPU	54.9196	53.9827	49.8917
Auto-MPG	2.81297	2.73065	2.72471
Iris	97.0667	98.1333	98.1333
Ionosphere	87.2000	88.9714	89.0286
Vehicle	82.3333	83.381	83.4524

Mean values for 50 trials are shown in the table. The first 2 rows are measured in RMSE, and the rest are in success rate (100%). The best performance is in boldface.

In this section, we view ELM in light of Cover’s theorem and interpolation, which delivers a better understanding and a historic consistency to show the rational of ELM. Despite the advantage of simplicity in training, ELM associates with problems for making inconsistent decisions for generalization. To illustrate the dynamic behavior of ELM, a plot of ELM decision boundary is provided.

To improve the generalization performance, we have introduced a meta-learning method, stacked generalization, which learns from the output of member models. Instead of just selecting a single winner, a group of decision power is collected. In Section IV, we have proposed the stacking ELM. And in the following section, we have demonstrated stacking ELM performance through experiment results. It is noticeable that stacking ELM networks generally outperforms single ELM for both regression and classification. Furthermore, different choices of level-1 generalizer were implemented and compared.

The training time of an ensemble is usually the aggregate of all the individual candidates, thus requiring large computational cost and long time for training. However, ELM is a fast learning algorithm, which is less time consuming compared with traditional back-propagation algorithms. Due to this nature, it is possible to incorporate stacked generalization with several

ELM networks even for time critical applications.

However, the experiment results show that the improvement of using stacking ELM is limited. As pointed out in [100], there is no difference between level-0 and level-1 generalization, and they are both seeking for a decision hypersurface which minimizes the error/cost related to the known points in the hyperspace. In this sense, “dumb” generalizers ought to be replaced by some sophisticated generalizers. By “sophisticated”, we mean those are capable to approximate much nonlinear relations, since in most case level-1 data are nonlinear. Therefore, future works will be emphasized on adopting sophisticated level-1 generalizers, e.g. decision trees or neural networks, in order to make a more significant improvement.

We can conclude that inspired by the promising results, applying multiple simple intelligent models can achieve a better performance than any single one. As a pilot project, it has successfully assessed the idea of using multiple learning instances to acquire improved domain knowledge. The later stage related to ELM will focus on explicit knowledge extraction, and incorporating ELM learning methodology into agents particularly for resource selection in Grid environment. Agents enhanced with learning capability such ELM, are competent in discovering allocation strategies and assisting other agents in resource selections via the communication links between them.

4.6 Autonomy Need Pursuit

The autonomy of a competitive intrinsically motivated agent is its decisive control over its own behaviors. Two aspects of such decisiveness: the agent makes choices considering its internal state (i.e. needs and motivation) coupled with external stimuli; the agent follows a plan (i.e. goal and action selection) which is originated from its sovereignty. In this thesis, we assume that an intrinsically motivated agent is firstly an autonomous agent, for which the need of autonomy is always being pursued by the subject. It implies when dealing with environment the agent has its own plans from the knowledge it has internally, instead of being instructed

externally. Comparing with reflex agent, the behaviors are hard-coded by designers and agents simply reacts to the environment without any flexible control over itself.

Therefore, it is important for an autonomous agent to select a right plan at the right time. The capability is reflected as agent's pursuit of autonomy. Throughout the pursuit of agent's need, it may encounter different situations. Each situation may require agent's series of actions. This research considers the series of actions to solve a problem as agent's knowledge to deal with this situation. The key problem a design of motivated agents faces is how to transform the design purpose to the pursuit of agent's intrinsic needs. The selection of knowledge is through an analysis of the problem and retrieving suitable GoalNet from the knowledge base.

Specifically, in this thesis the need of autonomy is satisfied by the computing in the need function, the motivation interpretation, and goal/action selection. This section discuss how autonomy is achieved by the interpretation and goal/action selection.

4.6.1 Intrinsic Need Functions

Agents have their innate functions for different needs. At any given time, an agent may be challenged from the combination of different needs with different levels of activation. Different needs have various relationship between need strength and time. The dominant need has the strongest signal strength and sets the pursuit of this particular need as its current motivation.

This research assumes that as long as a selection behavior is from agent's own decision and choices are given from agent's own knowledge, agent's autonomy is satisfied regardless of which choice is eventually selected. Therefore, competence and relatedness are evaluated for choice selection.

In this thesis, we simply discuss the possible choices for implementing agent's need functions. The proposed functions do not intent to exhaustively cover all possibility in any way. Designers may have a customized design of their agents. There are three types of functions as candidates for implementing needs.

An impulse signaling is used for generating needs across times. The impulse represents the internal tendency towards the pursuit of a needs. Three impulse frequencies are used, $freq_{lo}$, $freq_{me}$, $freq_{hi}$. The assignment of frequencies is up to the agent designer. In the thesis, pursuit of performance is assigned with the highest frequency to stimulate agent to accomplish its performance goals; pursuit of relatedness is with lowest to encourage independent work and reduce non-necessary communication cost.

However, impulse cannot model agent's intrinsic need well over time. The thesis further proposes a set of smooth functions (soft-windowing) to simulate the need changes over time.

The soft-windowing functions are defined using equations depicted as follows,

$$\begin{aligned}
 Duration_i &= 1/freq_i \\
 Win_i &= (t, t + Duration_i) \\
 \mu_i &= \frac{1}{2} Duration_i \\
 \sigma_i &= \frac{1}{factor_i} Duration_i \\
 WinFun_i(t) &= \exp\left(-\frac{(t - \mu_i)^2}{2\sigma_i^2}\right)
 \end{aligned} \tag{Eq. 4.22}$$

where t is the current time; $i \in \{\text{"low"}, \text{"mediate"}, \text{"high"}\}$ denotes each of the three different need frequencies; Win_i refers to window i spanning from t (impulse firing time) to $t + Duration_i$; $Duration_i$ is the time span of window i ; μ_i and σ_i are the center and spread of the i^{th} windowing function respectively; $factor_i$ is a predefined constant; and $WinFun_i$ is the Gaussian membership function defining the i^{th} window.

The functions of Novelty and Performance can be determined using Eq. 4.22 which comprises a gaussian function. However, the smooth tails at both sides may not well reflect the relationship between need and its emergence. For example, Relatedness is less frequently needed, but once it is intended the need function should excite itself immediately. To model

Figure 4.17: Illustration of Need Functions

this relationship, a **Rayleigh** distribution function is used as follows.

$$\begin{aligned}
 Duration_i &= 1/freq_i \\
 Win_i &= (t, t + Duration_i) \\
 \mu_i &= 1/2 \cdot Duration_i \\
 WinFun_i(t; \varsigma) &= \frac{t}{\varsigma^2} e^{-t^2/2\varsigma^2}
 \end{aligned} \tag{Eq. 4.23}$$

From the definition of Rayleigh distribution, we have

$$\begin{aligned}
 \mu_i &= \varsigma \sqrt{\frac{\pi}{2}} \approx 1.253\varsigma \\
 \sigma &= \frac{4 - \pi}{2} \varsigma^2 \approx 0.429\varsigma^2
 \end{aligned}$$

Therefore, we can express the ς in terms of $Duration_i$.

An illustration of need functions is shown in Figure 4.17. The horizontal axis is the time line, and the vertical axis is the need level. Three need functions are outlined in the figure. The green and blue present the need functions with gaussian membership. They are different

in terms of mean and variance. The red function is with Rayleigh type membership. From the figure, it is easy to observe that Rayleigh type function has a sharper head and a longer tail comparing with gaussian types.

4.6.2 Motivation Interpretation and Weight Adjustment

The motivation interpreter **MI** deals with tasks, including calculating motivation intensities $\mathcal{V}^{M_i}$; eliciting motivated root goals based on external stimuli; and adjusting weights of intrinsic needs according to internal need updates.

Calculating Motivation Intensities An intrinsically motivated agent has three motivations as listed in Table 3.2. Agent interprets its motivations based on the calculation of motivation intensity, and proactively selects the motivation with the highest intensity. The intensity is dependent on intrinsic need levels and need weights.

$$\mathcal{V}^{M_i} = w_i \times \mathcal{V}^{N_i}, \quad \forall i \quad (\text{Eq. 4.24})$$

where w_i denotes the need weight of need N_i , and $\mathcal{V}$ denotes the intensity of motivation and levels of need.

With the calculated motivation intensity, **MI** adapts the *winner-takes-all* selection to find the most desirable motivations.

$$M_i = \arg \max \mathcal{V}^{M_i}, \quad \forall i \quad (\text{Eq. 4.25})$$

Eliciting Root Goals With the knowledge of the desired motivation, the agent finds the plan to fulfill its need pursuit. The plan is identified as the selection of a proper “root goal”. The set of valid root goals are listed in Table 3.2. A rule-based root goal selection is proposed in the thesis by considering motivations and external events.

Table 4.10: Rules for Selecting a “Root Goal”

1	Motivation is “Pursuit of Novelty”	Root Goal “To Achieve Novelty” is selected
2	Motivation is “Pursuit of Performance”	
2.1	Event contains a external job request	Root Goal “To Fulfill External Request ” is selected
2.2	Event is absent and improvement is needed	Root Goal “To Self-Improve” is selected
2.3	Event is absent and improvement is not needed	Root Goal “To Search for a Task” is selected
3	Motivation is “Pursuit of Relatedness”	
3.1	Event is absent	Root Goal “To Ask ” is selected
3.2	Event contains an interaction request	Root Goal “To Answer” is selected

Adjusting Need Weights After the execution of agent plans specified as GoalNets, a need update signal is sent to **MI** to reinforce or weaken the potential of need firing. The potential of firing a need is controlled by the values of need weights. Generally speaking, if a GoalNet in the need pursuit is successfully fulfilled, the weight of the need is weakened, signaling **MI** of the need having been satisfied; On the contrary, if a GoalNet inhibits the need satisfaction, the weight of the need is strengthened, making **MI** tending to excite the need firing again.

In this thesis, a simple weight adjusting is proposed,

$$w_i = w_i + \beta^n - \beta^p \quad (\text{Eq. 4.26})$$

where w_i denotes the need weight of need N_i , β denotes an updating rate in range $(0, 1]$, and n or p denotes the number of times that the need is satisfied negatively or positively. For example, a negative satisfaction exponentially increases the need weight, making the need to be fired more likely in the near future.

A normalization is further enforced on the updated weights to ensure weights bounded within range $[0, 1]$ by,

$$w_i = \frac{w_i}{\sum w_j} \quad (\text{Eq. 4.27})$$

4.6.3 Motivation-Driven Goal and Action Selection

Once a root goal is activated, agent allows different bindings of sub-goals and actions to achieve the root goal. The basic mechanism of selecting a goal/action follows the goal/action selection algorithms as in GoalNet, except following,

- Each goal/action associates with a set of motivation indices, which records the scores of a GoalNet for different motivations;
- *Winner-Take-All* is carried out through the process of forward selection, which selects the subject with the maximum index of the underlying motivation;
- Corresponding index is updated once the feedback of the execution of a plan including goals and actions in a backward *credit assignment* fashion.

Motivation Index Dating back to the formalization of agent knowledge plans in Section 3.6, a plan has a property “options” to store vital information, such as utility. This optional field is used for a motivated agent to keep a track of goal/action performance. The motivation index is defined as $\text{index}_i^{M_j} \in \mathbb{R} \text{ and } \geq 0$, for i^{th} GoalNet or action with Motivation $_j$. Therefore, each entity in the pool of GoalNets and actions has the option filed as a tuple, Motivation Indices $(M_j : \text{index}_i^{M_j}), \forall M_j \in \mathbf{M}$.

The index is accessed by the computes in **GE**, **AE** for goal/action selection. One plan may be utilized for different motivations, and different motivation indices should be considered. For instance, the GoalNet of “To Fulfill External Request” has a zero index for motivation “Pursuit of Novelty”, and positive index for “Pursuit of Performance”. In other cases, for example,

the sub-GoalNet of “Establishing Communications” has positive values for motivations both “Pursuit of Performance” and “Pursuit of Relatedness”.

Forward Goal/Action Selection Given the indices of goals and actions, the **GE** and **AE** are able to evaluate the choices by comparing the candidate index. The forward calculation is straightforward; when the agent encounters a choice state as shown in Figure 3.3.5, it simply picks up the candidate with the maximum index. The policy of selection is to maximize the motivation of the agent, in a winner-take-all manner,

$$\text{entity}_{\text{selected}} = \arg \max(\text{index}_i^{M_j}), \text{ for } M_j, \forall \text{ candidate}_i \quad (\text{Eq. 4.28})$$

This policy is similar to the policy used in reinforcement learning, where a greedy approach is taken. For reinforcement learning, the optimality of the policy is determined by the value assignment. Similarly, a backward index assignment is proposed in the rest of this discussion.

Backward Index Reinforcement In previous work related to GoalNet goal selection, a recursive algorithm based on the sequential execution in the GoalNet model was developed. The algorithm tries to find all possible paths to the ultimate goal. Each path is similarly evaluated by an index value. A path with the optimal index value is chosen following Dijkstra’s shortest path algorithm. However, the global optimal is not easily attempted, since the value of a goal state is hardly known for practical environment. However, the agents do receive a feedback concerning the overall performance of a goal/action plan. The thesis proposes to adopt a simple way to update index values for each goal/action.

Upon receiving the feedback, it is the overall performance of the entire plan which comprises multiple choices and branches. How to update each component within a plan is a difficult problem. This thesis proposes a hierarchical reinforcement learning approach to iteratively update indices. Reinforcement learning incrementally update its reward function in a trial-and-error manner, interacting with the environmental dynamics. The policy based on the value of

states simply selects an action with the maximum expected reward. Theoretically reinforcement learning is only workable for Markov Decision Process (MDP), but recent research find this Markovian approach also provides effective solutions for a more generic and open environment.

Due to the nature of GoalNet, a kind of hierarchical reinforcement learning is proposed to access the GoalNet and its sub-GoalNets. The idea is to propagate the feedback to each goal/action preceding the completion of the GoalNet, and to provide a simply way to attempt an optimal selection strategy which is flexible enough in adapting the changing environment. A GoalNet under question is firstly converted to the unified modeling of goal/action selection in Figure 3.3.5. The choice points are then identified, for which the index value of a goal/action is the value of the node after the choice point. The choice point itself take no value, since it is a dummy state for identifying goal/action selection only.

Choice points remind the agent that there may exist candidates that could be used as an alternative. Therefore, evaluating the candidate states becomes more critical than the other sequential states. For simplicity and computing efficiency, only the candidate states are indexed. In addition, since the candidate states could be a parent goal of a sub-GoalNet as well, a hierarchical decomposition is required to assign the index values for those states.

The details of the algorithm are shown in Figure 4.18. The agent initially follows an arbitrary policy, which randomly selects a goal or action at each choice point. The agent observes the feedback from the environment and stores as runtime data in **K**. One trace from the beginning to the end of the GoalNet of root goal “To fulfill external request” in Figure 4.7 is called one episode. The feedback of one episode is to be prorogated back to precedent goal/action state. The feedback is initialized assigned as the index of the end goal, and the rest of states are assigned with arbitrary values or zeros. An update of index is carried out for each visited states, either goal or action state, by appending a discounted index of the following first occurrence of this state. Therefore, the proposed algorithm is base on *First-Visit Monte Carlo* method.

The first-visit MC method averages just the returns following first visits to s . Each episode provides a step towards the true evaluation of visited states. However, special attentions are paid for parent goals with sub-GoalNet. Sub-GoalNets are to be evaluated in the same way. Thus, a recursive call for policy evaluation continues for its states in the sub-GoalNet. After propagating to all states, a policy improvement updates policy in a greedy approach, where the policy is to select the goal/action that can receive the maximum expected values from its successors. This algorithm repeats in an infinity loop, due to the mathematical fundamental, which is elucidated in the proof.

Proof: The first-visit MC converges to the true index value of a state s , $\mathcal{V}^s$ given a policy π , as the number of visits to s goes to infinity. The method assumes that each return is an independent, identically distributed estimate of $\mathcal{V}^s$. By the law of large numbers the sequence of averages of these estimates converges to their expected value. Each average is itself an unbiased estimate, and the standard deviation of its error falls as $1/\sqrt{n}$, where n is the number of returns averaged. After the true index value is obtained for each state s , it is possible to proceed with classic policy iteration method, which comprises both policy evaluation and policy improvement as illustrated in Figure 4.18.

The use of Monte Carlo is important for real cases, since it is difficult to estimate the complete probability distributions of all possible transitions that are required by a dynamic programming methods. The algorithm comes with its cost: firstly the optimality is hard to guaranteed due to the infinity sampling of the environment; secondly an early exploring must be implemented to explore the choices sufficiently enough, otherwise some candidate goals and actions may not be fully covered. Therefore, a self-improvement by the agent is largely required to achieve its performance pursuit. However, the optimality or the complexity to achieve optimal autonomy is out of the thesis scope, and there are many approaches in the literature which provide improvement over the proposed approach for some cases. Nevertheless, the algorithm proposed in this section gives a better understanding of how goal/action selection is implemented to achieve agent's pursuit of autonomy.

a platform for running intrinsically motivated agents is discussed in this section. The intended design answers the challenges from environment configurations, communication protocols to the agent internal architecture.

4.7.1 A Brief on the Agent Development Platform

This thesis makes use of Multi-Agent Development Environment (MADE), which provides a development and running platform for the proposed competitive agents. Since it uses GoalNet model as agent design model, the framework is the best candidate for running motivated agent. This platform emphasizes two issues, namely how to assist developer to easily design a goal-oriented agent and how to automatically translate this design into agent implementation. The overall system architecture is illustrated in Figure 4.19.

Figure 4.19: Overview of MADE Architecture. Original picture is from [91]

In this architecture, there are three major modules: GoalNet Designer, GoalNet Storage and Agent Framework. GoalNet designer is the module to assist agent developers during a design of a GoalNet. After a GoalNet is designed, the design data are stored persistently into data storage. In the next step, the underlying GoalNet is converted to an agent system according to the agent-goal binding policies. Thus, the overall goal of the system can be pursued by a group of agents and each of them are pursuing one or more sub-goals. Once these agents are

Figure 4.20: MADE Main Dataflow. Original picture is from [91]

identified, the stored GoalNet data can be loaded into agents' knowledge base, and then the agents start running by pursuing their own goals.

Agent Management module defines the framework where agents reside and operate. The module establishes the logical reference model in a FIPA-compliant system, which enables creating, registering, locating, communicating with, migrating, and terminating an agent. The functionalities that the agent management provides include naming service, directory facilitator, directory enquiry, and agent life cycle management.

Data flow shown in Figure 4.20 can be summarized as follows:

- Control Unit requires Process Unit about the next goals (there are more than one next goal available when there is a choice transition).
- Process Unit informs Control Unit about all the possible next goals.
- Control Unit sends these goals to Compute Unit
- Compute Unit does a computation on these goals according to the goal selection algorithms, and sends the selected goal back to Control Unit.
- Based on the selected next goal, Control Unit ask Process Unit about the transition that can reach the selected goal.

- Process Unit sends Control Unit the corresponding transition.
- Control Unit sends the transition to Compute Unit for action selection.
- Compute Unit finds the best action to execute, and sends this action back to Control Unit.
- Control Unit sends the selected task to Action Unit for execution.
- Action Unit executes the given task by dynamically invoke the task class name, and sends acknowledge to Control Unit once the task is done.
- Control Unit transits from its current goal, and asks Process Unit about the next goals.

4.7.2 Messaging, Communication and Action Implementation

A communication channel is the way that different agent can interact with each other. In the MADE framework, a communication system is used for message passing between agents, as shown in Figure 4.19. Communication system is shared by all the agents. When an agent needs to send a message to another agent, it just pushes the message into the communication system. Similarly, an agent regularly checks its channel. If there is a message dedicated to it, the agent will take the message and delete it from a message queue. Within the agent platform, the messages are handled by Communication Unit. Communication Unit has three major components: Message Writer, Message Receiver and Message Handler. Message Writer tries to format the sending message properly, and posts it onto message queue. Message Receiver is responsible for checking the incoming message from message queue. Message Handler, which is just an interface in a dummy agent, is to deal with an incoming message. For example, it's the Message Handler that should answer whether the agent send an acknowledgement of an incoming message. The communication model of the agent framework is implemented using windows socket, with messages that are written in XML format.

It is essential to implement event receiver as a queueing system. All important messages are buffered in the queue, so that by sophisticated mechanisms, messages can be processed according to priorities, preemptively interrupted and even synthesized collectively for analysis. For environment there are challenges to define semantics of event, so that agent understands and allows to grep necessary content for the event message. The ontology of event

includes event category and messages. For communications between agents and environment and among agents, the challenges exist for definition of messaging format, the establishment of communication channel and protocol used for the communication.

Actions that are executed by the actuators, are considered as an agent's physical activity. It implies the agent's effect on the environment, which is influential to the external entities. Agent actuators are designed as a set of modules that are separated from its mental activities. The implementation and actual running environment may be isolated from agent running environment, but the invocation method or APIs must be available to agents. For example, an agent could have the access to an installed motor, which is operating separately from the physical running environment of that agent. However, the agent can invoke the API of running as "run()", which will start the motor remotely.

4.8 A Summary of Developing Competitive Intrinsically Motivated Agents

Basically, the development of an application system mediated by the proposed competitive intrinsically motivated agents follows the conventional software engineering approach: requirement analysis, design, implementation, test and deployment. The development summary specifies the common procedures with following details.

Requirement analysis is carried out by firstly understanding the objective of an application, and then analyzing the *problem domain* of the underlying scope. We must have a specific problem set for our agents to deal with. A problem is characterized and identified, together with requirements of intended solutions. A problem must be specific, so that solving the problem contributes to the achievement of agent's need pursuit. It is common for an agent to deal with multiple problems along the way. However, if two problems are independent of each other, they can always be considered as two isolated domains. If two problems have sequential or causal relationships, they can always be considered as two sub-problems of a more general

one. For problem analysis, it is essential to identify those problems that are independent and able to be solved separately.

Identifying problems are similar to identifying goals for a scenario. One goal is considered as the achievement of solving one problem. Goals have hierarchy, so do problems. For problem analysis, top level goals/problems are to be identified, and then decomposed before designing solutions in the next phase.

Design phase focuses on the solution of one problem defined in analysis phase. Designers need to give an abstracted plan in the form of a general main task. An abstracted GoalNet is expected to solve one particular type problem. During this phase, sub-tasks are to be specified. Sub-tasks deal with activities commonly required by agents, even for different applications. For some cases, even multiple agents can be fit into one high-level main workflow, but agent behaviors can vary a lot for the subtasks. One sub-task defines a category of common activities which fulfills the same goal by different means. For example, predicting workload is one sub-task, for which different machine learning algorithms may apply. Several sub-tasks that are commonly used for motivated agents are explained in Chapter 4 for intrinsic need pursuit.

Implementation requires coding on agent components, such as motivation, needs, and agent functions, such as compute, goal/action selection. Implementing operational logics that are coherent to agent internal needs is expected in this phase. In additional, computational algorithms and libraries are to be created or loaded into system.

Test phase allows designers to have a close observation over the behaviors of the created agents. It is usually done in a constrained local environment, featuring most events that would happen in a more open scenario. This phase gives the designer a feeling of whether agents are acting as expected and raises their confidence level for deploying in the final phase.

Deployment becomes the final stage and makes itself no difference between other software engineering approaches. However, working with motivated agents requires a more regular supervision over agent behaviors. Re-analysis and re-design are hard to ignore, and designers should be also actively involved in the continuity of software development life cycle.

Since common practice of software engineering still applies for motivated agents, from requirement analysis to test and deployment, in this research we focus mainly on the parts that are unique for competitive intrinsically motivated agents. The summary of the chapter is given as follows,

An overview of agent architecture is first introduced with the design of agent's main task. Several sub-goals as the pursuit of agent needs are introduced with a GoalNet representation, a detailed system-wise design, and a detailed functional implementation. To gain a better understanding, each sub-goal is instantiated with a simulation example, followed by a discussion on the advantage of each proposed agent behavior. A set of reasoning mechanisms and agent learning techniques (supervised, unsupervised and reinforcement learning) are discussed within this intrinsic motivation context; the proposed agent modeling shows the strength in supporting a wide range of agent planning; and this assures a potential transformation from other agent types to motivated agents by reusing the legacy artifacts. Finally, an agent running platform is then explained to support the execution of the proposed motivated agents. The following chapters of this thesis will present two case studies for practical use of competitive agents.

Chapter 5

Case Study I: Resource Allocation Agents for Dynamic Load Balancing in Cloud-based Large-Scale Collaborative Virtual Environments

5.1 Introduction

With the explosive growth of the Internet technology, Large-Scale Collaborative Virtual Environment (LSCVE), such as online 3D virtual world becomes a new paradigm for human-to-machine and even human-to-human interaction. It has a broad spectrum of applications for military and industrial applications, for example, medical team training, war simulation, collaborative scientific visualization, collaborative design and engineering, etc. It also has been used for commercial and entertainment applications with the power of Internet, like virtual shopping malls and showrooms, virtual libraries, serious games for eLearning and Massively Multiplayer Online Games (MMOG), etc. Virtual world will continue to enable more engaging online applications to shape the future Internet.

In LSCVE, a typical requirement of servers for data dissemination is to ensure the persistence of virtual world, for which servers must continuously send related updates of scenery objects and avatars to clients through network. At the same time, servers must deal with a large amount of data processing, including physical simulations such as gravity and collision

detections, animation simulations and scripting routines [104]. Both data dissemination and processing are computationally demanding, and a highly scalable system is needed to fulfill the operational requirements on servers. As a consequence, the most challenging problem faced by current research is the scalability of virtual world, over which a huge number of players simultaneously and geographically scatter over the Internet.

The non-uniform distribution of users over virtual world incurs workload imbalances within the system. Some parts of the system may be crowded and busy, while some idle and under-utilization. This skewed distribution is caused by user dynamic behaviors, such as clustering at interesting places. As a result, user dynamics may lead to an inconsistent provisioning of quality of service (QoS) across the system. In this case study, we consider communication delay as the parameter of QoS in LSCVEs. Both overloading and migrating a server contribute to such delay. Users would experience long communication delay due to the slow response of overloaded servers and overheads occurred during server migration.

To solve the load balancing issue, peer-to-peer and multi-server architectures are adopted in the literature. However, both of them have their own limitations. Although peer-to-peer architecture can improve the scalability by avoiding performance bottlenecks, strong consistency and peer cheating-prevention are not easy to achieve [105]. For multi-server architecture, all the workloads are shared by multiple servers. Overloaded servers dynamically transfer excessive workloads to light-loaded ones. However it has two disadvantages,

- The limited and fixed numbers of servers are easy to become bottleneck, and in the presence of non-deterministic fluctuations of workload in real time, the multi-server architecture is not agile enough to utilize new resources in a cost-effective manner.
- Upon deciding how to do a migration, a system scheduler needs to determine how much workload to be transferred from an overloaded server (e.g. Server A) and which server to take the migrated workload (e.g. Server B). After scheduling, it is difficult for existing

schedulers to know in the future whether A will be overloaded again and whether B will exceed its capacity because of this migration. It is possible that the system may require another round of scheduling in the immediate future.

The goal of our research is to provide an improved load balancing scheme to ensure a consistent delivery of QoS among distributed servers for a large crowd of simultaneous participants in virtual world. This case study attempts to achieve the goal in two folds.

- This case study adopts Cloud computing architecture to dynamically and cost-effectively provision new servers in accord with real-time computational demands.
- In order to avoid re-scheduling in the immediate future, this case study proposes to predict workload of virtual world, allowing schedulers to consider future system state during decision making.

Recent advance in Cloud computing offers an unparalleled opportunity to virtual environment providers. Cloud computing provides the infrastructure that enables the exponential scaling of servers and rapidly flex resources according to changing demands. In contrast to multi-server architecture, one of the most tangible advantage of Cloud-based architecture is the capability to acquire new servers quickly. The characteristics of incremental scalability and infrastructure flexibility allow LSCVEs to respond more quickly and flexibly to variable demands. Meanwhile, on-demand resource provisioning capability of Cloud computing eliminates the cost to procure hardware and the cost to manage. The utility-based economics drive more and more virtual service providers to adopt Cloud-based architectures, due to the largely reduced capital expenditures. Therefore, we adopt Cloud computing architecture to improve the scalability of LSCVE dynamically and cost-effectively.

With a motivating example, this case study argues that if a load balancer only considers current system state, the system may encounter overloaded servers and needs re-scheduling

again in the immediate future. However, this situation can be improved by incorporating future information through workload prediction. Nevertheless, existing load balancing schemes for LSCVEs largely rely on current state only.

This case study proposes the use of proposed multiple motivated agents to actively collect system states and constantly update predictions of workload to central scheduler for system-wide load balancing. Learning capability is mastered by individual agents as the result of competent pursuit, and local information is timely submitted to central scheduler as the result of relatedness pursuit.

For the central resource allocation agent, it is driven by the need of competence in mastering the way of analyzing system state collectively, and by the need of relatedness in pulling predicted distributed system state from other agents. By analyzing the predicted system state collectively, the resource selection agent can effectively distribute workload among existing servers or adding new servers, as well as avoid re-scheduling in the immediate future. This case study is organized as follows. Section 5.2 introduces the load balancing problem of virtual worlds, followed by reviews of related work in Section 5.3. By showing an example, this case study argues that without prediction, the virtual world system may affect the end user experience by introducing re-scheduling at immediate future steps. Section 5.4 proposes to adopt a neural network with fast learning speed to predict workload as the competence pursuit of motivated agents in LSCVE. Consequently, Section 5.5 proposes a rule-based and a prediction-based load balancing algorithm for Cloud-based LSCVEs. Section 5.6 presents simulation experiments and discussions of the results. The case study ends with conclusion and future work in Section 5.7.

5.2 Problem Formulation of Load Balancing in LSCVEs

For load balancing, system initiates a scheduling in two circumstances. One is triggered by workload imbalances detected in the system, and consecutively a load balancing scheme is ex-

ecuted to minimize the imbalances so as to achieve the fairness of load sharing across multiple machines. The other concerns required service provisioning, which is commonly specified in Service Level Agreement (SLA) with expected fulfillment of QoS. In this case, a load balancing is activated only when the underlying SLA is or will be likely violated.

Virtual environment applications foster end user experience. Therefore workload imbalance among servers should not catch much attention, as long as it can fulfill the requirements specified in SLA. In addition, trying to achieve strict fairness of server loads may be problematic. Since load distribution inevitably introduces overheads and communication delay, scheduling itself may affect end user experience for those reallocated due to load distribution. As a consequence, LSCVE applications usually schedule a load distribution according to the delivered QoS status.

The parameter of QoS concerned in this case study is the communication delay, which directly reflects the end user experience. With a longer communication delay, users will have to endure longer response time. Communication delay is composed of the time of data processing and network latency, but we ignore the network latency, because it is an orthogonal issue, beyond the scope of this work. In this case, a violation of SLA is caused either by overloaded servers or by the overheads occurred in load distribution. Overloaded servers respond to client requests at the rate slower than the norm rate, and thus affect end user experience by delayed response time. However, load distribution itself unavoidably introduces overheads and additional workload to the system, which in return adds more delays to the response time for those migrated players. To determine whether a server is overloaded, it is impractical to rigidly assume one upper bound of workload for all cases. Instead, this case study exploits the number of concurrent users as a generic indicator, assuming the workload generated by each user is more or less the same. This approach follows the findings in [106, 107], which were based on a closely observed gaming scenario in a virtual world. Researchers found that network stacking processing time and game related processing time increased linearly with the number of

players, and servers was able to keep up with player requests and able to respond at the normal rate until servers were saturated. As a result, it is safe to assume that a server can host up to a certain number of players without overloading itself or compromising QoS. For different types of machines and applications, a pre-analysis is to done to determine the upper bound of concurrent users for a particular type of machines. In most cases, the upper bound is defined below the maximum capability of a server, so that the server will not be downgraded immediately once the upper bound is exceeded. Nevertheless, we consider the predefined upper bound as a threshold between the overloading and normal status of a server.

Eq. 5.1 is applied to quantitatively measure the overall QoS delivered by a given system, which is hosting N users simultaneously during its life time T .

$$total_SLA_violation = \sum_t^T \sum_n^N SLA_violation_t^n \quad (\text{Eq. 5.1})$$

where $SLA_violation_t^n$ denotes the violation of SLA for $User_n$ at time T , and it equals to 1 if $User_n$ is on an overloaded server or $User_n$ is reallocated to another server due to load distribution, otherwise it equals to 0. The sum of SLA violations reflects the total occurrences of events when an end user receives a compromised QoS. Eq. 5.1 indicates that a good load balancer must reduce the occurrence of overloaded servers as well as limit side effects on users as a result of scheduling. However finding the optimum solution is nontrivial. In research [108], the problem of optimal task selection by distributing load is proved to be NP-complete, and in order to obtain solutions in polynomial times, merely approximate algorithms are feasible.

5.3 Related Work

In LSCVE, both peer-to-peer and multi-server architectures are extensively studied. Related work is reviewed in this section, and advantages and limitations are explained.

For a peer-to-peer (P2P) architecture, data of virtual environment is duplicated by each peer node before a virtual environment session can begin. Each peer straightforwardly delivers his

state updates to a group of related users. LSCVE systems using P2P, such as SPLINE [109], NPSNET [110], SCORE [111], and DIVE [112], can scale up effectively using an IP-multicast data filtering mechanism. However, P2P architecture lacks of the support of peers with heterogeneous underlying capabilities, for such architecture assumes each peer has similar network bandwidth and computing power. Moreover, the consistency control is hard to achieve, because the P2P environment is highly distributed and consistency is difficult to guarantee. Well-designed synchronization mechanisms, such as bucket synchronization [113], time-warp [114], local-lag [115], is needed to execute at each peer, which has $O(n^2)$ computational complexity and special latency requirement to guarantee the consistency among the peers. As a consequence, most LSCVE systems using P2P can only have a approximate control in consistency, and P2P architecture is usually applicable, when dedicated resources, e.g. computers and networks, are available. Examples of a P2P LSCVE include experimental environment in academic and war training system in military.

In multi-server architecture, the existing researches can be grouped into two methods. The first method is to have multiple replicas of the virtual world on different servers while ensuring the state consistency across the servers in real time. It is also called mirrored server architecture. As multiple servers maintain the same shared world, the state synchronization mechanisms in peer-to-peer architecture are required to guarantee the consistent state among the servers. Special designed synchronization mechanisms for the mirrored servers, such as effective synchronization by obsolete event dropping [115], are also proposed to improve the synchronization responsiveness. Dynamic load balancing is also adopted to distribute the workloads among servers. However, the introduction of multiple servers does incur some overhead. The output bandwidth required by each server to support the virtual world application is greater than the resources required in a single server system [116]. Therefore, mirroring server architecture can only improve the scalability to a certain extent.

The other method is to geographically partition the virtual world into multiple parts (i.e. cells) and one of multiple servers is assigned to support the computation within a subset of

cells. As a virtual entity of a participant moves around the virtual world, its state is updated by one of the servers that manages the cell. By continuously assigning cells to servers at run time, the workload of the system can be dynamically distributed. Different cell assignment policies are summarized as follows,

Static partitioning: In Second Life [104], NetEffect [117] and CyberWalk [118], workload of the servers are balanced by tuning the volume of virtual content or the number of users hosted by each server. No matter in what way the virtual world is partitioned, however, the cell assignment is unchangeable after the deployment of virtual world. Researchers [119] pointed out that there is no one static partitioning alone can fit well for all sceneries if considering user behavior dynamics, and simulations showed that static partitioning might have inconsistent behaviors and performed poorly in at least one scenario.

Uniform sharing: Uniform algorithm adopts a bin-packing algorithm to scan through all cells and assign workload uniformly across servers iteratively [119]. Scheduling is triggered when a server load exceeds its capacity. It starts with one empty bin for each server and successively places one cell to the bin of a server with lightest load. The process continues until all cells have been placed into bins. Then the system scheduler initiates cell migrations according to the cell assignment. The aim is a fair sharing of workload through a uniform load distribution, and this approach seeks the smallest imbalances across the system. However, uniform approach is not efficient, since it is likely to remap most servers even if only a few servers are overloaded.

Greedy approach: Approaches used in Butterfly.net [120] and [121, 122] utilize Grid architecture or mobile agents to support load migration. System scheduler directly shed load from one overloaded server to the lightest loaded node from a set of candidate nodes. The candidates must satisfy the condition that it can accommodate the migrated load without overloading itself. In this manner, greedy approach only updates the cell mapping for the overloaded server and the accommodating server.

Heuristic algorithm: [123] proposes a set of heuristics with mobile agents for LSCVE to improve system scalability. They modeled the services for data transmission and processing as mobile agents and allowed them not to bond with any fix node. These mobile agents can dynamically move to both traditional server hosts and qualified user hosts so that the system workload can be decentralized effectively. Although the idea of using client machines as additional resource is interesting, clients that volunteer their own resources would likely have downgraded experience.

Prediction-based algorithm: [124] introduces a neural network-based method for dynamic resource provisioning in massively multiplayer online games (MMOGs). A load prediction of workload estimates the user distribution of the future in the virtual world according to historical data. Based on the predicted information, a service for resource allocation is implemented in a request-offer matching manner, defined by a set of heuristics. However, this approach adds resources for mirroring popular virtual areas with additional cost to manage consistency between mirrored servers. Moreover no migration for load sharing between overloaded and light-loaded servers is supported, and it is thus possible that some servers are busy while some are left idle.

In this case study we argue that the scheduling algorithms restricted to only current state may need rescheduling at immediate future. For example, greedy algorithm only gives preference to the node with lightest load at current time. Existing algorithms do not consider whether or not a node can hold the updated workload for the next moment. Workload will not change much if users in the virtual world do not cross cell boundaries. However, users usually like to move around and even cluster at a hot-spot, where an interesting event is happening. The dynamics of user behavior may cause problems to any shortsighted algorithms that take no account of future states. For an illustration, a motivating example is given in the next section.

5.4 Distributed Motivated Agents for Dynamic Workload Prediction

In this section, we first introduce a motivating example to show the advantage of using future states of a virtual environment if this kind of information is available. To obtain the future workload of a cell, motivated agents for local information collection apply either Back Propagation (BP) or Extreme Learning Machine (ELM) as their pursuit of competence. The latter part of this section explains how neural networks are used for prediction.

5.4.1 Motivating Example

The example in Figure 5.1 shows different scheduling strategies with the workload states of every cell in a virtual world at time t and time $t + 1$ respectively. The entire virtual world is partitioned into 9 cells, and initially cells in the same row are assigned to a server. The number in each cell represents the number of concurrent players in the corresponding virtual region, and one server can accommodate up to 9 players. Assume that a hot-spot occurs within the middle cell of the first row, and then players start to crowd around the hot-spot.

Without load sharing, for example in static partitioning, Server 1 is overloaded for both time t and $t + 1$ (top row in Figure 5.1) with workload 12. To illustrate the importance of future state information, greedy approach is used in this example (middle row in Figure 5.1). The occurrence of the overloaded server (Server 5.1) at the beginning of time t triggers load distribution. By examining the current state of Server 1, the scheduler decides to move a cell out of Server 1. Since all cells of Server 1 have the same number of players (4 players in this case), Cell 3 (dashed) is selected by breaking ties randomly. After that, scheduler checks the current workload of Server 2 and Server 3. In this case, they have 4 and 5 players accordingly. Server 2 with the lightest load is chosen to take Cell 3. Following the cell assignment according to the schedule, workload of all servers is within their capacity of 9. In this case, there are 8,

Figure 5.1: A Motivating Example for Impacts of User Movements on LSCVEs

Top row for static partitioning; middle row for greedy approach; bottom row is recommended if future information is available.

8 and 5 players on Server 1, 2 and 3 at time t respectively. However, this type of scheduling makes both Server 1 and Server 2 overloaded at time $t + 1$, both 10 players.

When deciding which cell to move out from Server 1 and successively which server to accept the cell, the greedy scheduler made two inconsiderate decisions.

- Although both Cell 2 and 3 have the same workload at time t , they have different loads at time $t + 1$, 6 and 2 players. If Cell 2 is shed off, Server 1 will have 8 and 6 players for time t and $t + 1$ respectively. In this way Server 1 will not be saturated at immediate

future $t + 1$.

- Even though Cell 3 is selected, Server 3, instead of Server 2, should accommodate Cell 3. At $t + 1$ Server 2 will have loads from existing cells, 3, 3, 2 respectively. With the additional load from Cell 3, Server 2 will be overloaded. However if Server 3 took Cell 3, the load at $t + 1$ would not exceed its capacity, $0 + 1 + 0 + 2 < 9$ in this case.

As we can see, the use of future information for both migrated cell selection and accommodating server selection is essential to the efficiency of a scheduling algorithm. Furthermore if future workload states are available, a better strategy is to position Cell 2 on Server 3, such that three servers will hold 8, 4, 9 for time t and 6, 8, 7 players for time $t + 1$ accordingly, indicating no server will be overloaded.

However, in a dynamic and uncertain virtual environment, the future information is not directly available and generally hard to obtain. One way to get future information is through movement pattern analysis for individual users [125, 126]. But for virtual worlds in large scale, it is difficult to obtain a generic user model for all players, and it is computationally expensive to trace and analyze all users individually. This case study proposes to keep a track of workload for each cell in the virtual world, and exploit historical data to predict workload expected in the future.

5.4.2 Agent Competence Pursuit in Workload Prediction

The motivated agents are required to collect and store historical status data, and to make use of it through workload prediction. We model the prediction of future workload as a time series prediction problem, and specifically we want to obtain a regression model,

$$N_u^i(t + 1) = f(N_u^j(t), \dots, N_u^j(t - p)), \forall j \in adj(i) \quad (\text{Eq. 5.2})$$

where $N_u^i(t)$ denotes the number of users in Cell i at time t . For one cell, the expected workload at time $t + 1$ is a function of historical workload of its adjacent cells (including itself) from time t dated back to time $t - p$.

Generally speaking, it is usually difficult to obtain an explicit model of $N_u^i(t)$, for there exists a complex nonlinear relationship between the past and future workload as a result of user arbitrary movement. However, machine learning with neural networks is possible to discover the underlying relationships between past and future workload from historical data. Although the dynamic nonlinearity existed in a virtual world, a neural network is well known for its capability of universal approximation to arbitrary functions. For time series prediction, [124] showed that a three-layer multilayer perceptron (MLP) performed better than traditional methods such as exponential smoothing, moving average, and last value, etc. Research work [127] suggests neural networks to be a promising alternative to traditional linear methods, such as Autoregressive integrated moving average (ARIMA). In addition, neural network requires less prior knowledge and the model-free characteristic is suitable for training in different application scenarios [128]. Therefore, in this case study the agents adopt neural networks to prediction workload in the virtual world.

In order to achieve accurate predication while reducing the cost of using learning in terms of time and computations, an agile neural network, namely Extreme Learning Machine (ELM) [93], is applied for workload prediction. ELM is a fast learning algorithm for single-hidden layer feedforward neural networks (SLFNs) with arbitrarily selected input weights and hidden neuron biases. The output weights of ELM are analytically calculated by Moore-Penrose generalized inverse. The details of knowledge in the form of ELM have been explained in Section 4.3.2.

The GoalNet for workload prediction using neural networks is given in Figure 5.2. According to the regression model explained in Eq. 5.2, a scheduler of a virtual world system requires one ELM for each cell. Thus the above GoalNet is embedded in each motivated agent of each cell. The inputs of one neural network are the historical workload (dated back to time $t - p$) of four adjacent cells as well as the underlying cell itself, and the output is the predicted workload of time $t + 1$.

Figure 5.2: GoalNet for Workload Prediction Using Neural Networks

The training of ELMs in LSCVE requires numbers of simulation trials. At the beginning of one trial, one hot-spot is presented in a random location within the virtual world, and then users are allowed moving towards the hot spot for a certain steps. Workload of each cell occurred at every time step is snapshot and recorded in a log file. After a number of different trials, training data of an individual cell is prepared in pairs of input and output according to Eq. 5.2. The purpose of training is to disclose a generalized relationship between past and future workload in a pre-defined virtual environment. In this case, generalization capability of ELM greatly depends on training data. A larger number of simulation trials are preferred in order to obtain a meaningful and generic regression model.

5.5 Motivated Resource Allocation Agent for LSCVEs Load Balancing

In a typical Cloud computing environment, for example in Amazon EC2 [129], Cloud users are virtually assigned a number of machines, on which users can run their applications. Regardless of the underneath physical hardware implementation, applications built on top of virtualization layer feel as if they are running in a set of physical machines. Essentially a Cloud-based architecture is a multi-server architecture, except that Cloud has more flexibility to add new machines. Therefore, existing load balancing algorithms in the context of multiple servers may apply for Cloud-based LSCVEs. Nevertheless, they do not only lack support for incremental

scalability, but are also vulnerable to cases of immediate rescheduling as illustrated in the motivating example.

To take the advantage of features of flexibility and incremental scalability provided by Cloud computing, the central resource allocation agent that is built on a rule-based load balancing scheme is proposed with options to add new machines. Consecutively, to avoid immediate rescheduling, the motivated agent has a prediction capability embedded on top of the rule-based scheme.

In this case study, both algorithms employ multiple load balancing agents for a Cloud-based virtual environment, where one agent is assigned to one virtual cell, in charge of monitoring the workload status, initiating scheduling requests to the central schedulers, and routing traffic between clients and virtual machines. Modeling load balancing components as agents offers a great level of autonomy and convenience. Agents can be represented as a delegate of human operators, and load balancing behavior can be modeled independent from various virtual environments. The application of agents has been introduced to service oriented architectures for resource scheduling, negotiation and personalization [130–134].

Dynamic load balancing is achievable through agent migration. As the agent migrates, the workload assigned to the agent is also to be executed by the hosting virtual machine. As system workload scales up or updates, agents can move to any qualified virtual machine, which can be either an existing (named as Cloud nodes) or an incrementally added virtual machine (named as Elastic nodes). In this case study, we call both of the nodes as Service Computing nodes (SC nodes).

The proposed motivated resource allocation agent takes care of agent migration initiation, migration reasoning and node discovery. The details of each strategy are given in this section.

5.5.1 Rule-based Agent Knowledge for Load Balancing

5.5.1.1 Agent Migration Initiation

Agent migration initiation determines when it should perform an agent migration as load distribution. As discussed for the service-oriented Cloud computing, agent migration should be initialized only when the SLA is violated or specifically a virtual machine is overloaded.

At time t if the workload $w_t^{SC_i}$ of a SC node SC_i exceed its capacity c^{SC_i} , the virtual environment application hosted by the node does not have enough computational resources to power its proper services under service contract. One or more of load balancing agents hosted on the SC Node need to migrate to a lightly loaded SC nodes. We define set $P_t = \{SC_i | w_t^{SC_i} > c^{SC_i}\}$ as the set of overloaded SC nodes at time step t .

If $P_t \neq \emptyset$, the set of load balancing agents $Ag_t^{SC_i}$ running on the overloaded SC_i will send a migration request to the scheduler. Consequently, at time t an agent migration is initialized.

5.5.1.2 Agent Migration Reasoning

Migration reasoning deals with which agent needs migration. At time t , it is possible that more than one SC node is overloaded. A greedy approach is exploited by giving highest priority to nodes that is overloaded the most severely. According to the assigned priority, saturated SC nodes are considered one by one during the agent migration process. We define the SC node to be considered during an agent migration as,

$$SC_{focused} = \arg \max_{\forall SC_i \in P_t} (w_t^{SC_i} - c^{SC_i}) \quad (\text{Eq. 5.3})$$

Consecutively, the load balancing scheduler decides which agent should migrate. The migration reasoning contains multiple iterations, and for each iteration m , a set of candidate agents for this $SC_{focused}$ are identified as,

$$\mathbf{Ag}_{candidate}^{SC_{focused}} = \left\{ Ag_i | w_{t,m}^{SC_{focused}} - w_t^{Ag_i} \leq c^{SC_{focused}}, \right. \\ \left. \forall Ag_i \in \mathbf{Ag}_{t,m}^{SC_{focused}} \text{ and } w_t^{Ag_i} \leq CAP_{t,m} \right\} \quad (\text{Eq. 5.4})$$

where subscript (t, m) indicates that the underlying value is recorded at iteration m within time t . Therefore, Eq. 5.4 means that the candidate agents are those after migrating which the load of $SC_{focused}$ will be within its capacity at current step.

Since we do not want to move out an agent whose workload is too large for any SC node to take, a safety cap $CAP_{t,m}$ is applied,

$$CAP_{t,m} = \max_{\forall SC_i \in \mathbf{SC}} (c^{SC_i} - w_{t,m}^{SC_i}) \quad (\text{Eq. 5.5})$$

It guarantees that for each iteration m , in the set of all SC nodes $\mathbf{SC}$ there will be at least one qualified SC node that can accept the migrated agent workload $w_t^{Ag_i}$.

If $\mathbf{Ag}_{candidate}^{SC_{focused}} \neq \emptyset$, we know that there are choices of migrations that can satisfy current load balancing requirement. In the rule-based algorithm, an agent is chosen to maximize the utilization if $SC_{focused}$ is a Cloud node and minimize if it is an Elastic node.

$$Ag_{selected} = \begin{cases} \arg \min_{\forall Ag_i \in \mathbf{Ag}_{candidate}^{SC_{focused}}} (w_t^{Ag_i}) & \text{for Cloud nodes} \\ \arg \max_{\forall Ag_i \in \mathbf{Ag}_{candidate}^{SC_{focused}}} (w_t^{Ag_i}) & \text{for Elastic nodes} \end{cases} \quad (\text{Eq. 5.6})$$

The rationale behind Eq. 5.6 lies in an economic concern that one wishes to fully utilize the initial set of nodes (i.e. “dedicated” Cloud nodes), while reducing the usage of additional nodes (i.e. usually pay-per-use Elastic nodes) so that they can be easily turned off if demand drops.

If $\mathbf{Ag}_{candidate}^{SC_{focused}} = \emptyset$, it implies that merely migrating a single agent is not able to solve the problem of $SC_{focused}$. In that case, the agent with largest workload is migrated. Note that the selection is also constrained by the safety cap at iteration m .

$$Ag_{selected} = \arg \max_{\forall Ag_i \in \mathbf{Ag}_{t,m}^{SC_{focused}}} (w_t^{Ag_i}), \forall w_t^{Ag_i} \leq CAP_{t,m} \quad (\text{Eq. 5.7})$$

5.5.1.3 Node Discovery

Node discovery is responsible for finding a node either Cloud node or Elastic node to host the migrating agent.

- Launching new Elastic nodes

If $Ag_{selected} = \emptyset$, meaning no agent satisfies the condition $w_t^{Ag_i} \leq CAP_{t,m}$, a new Elastic node will be added into the system to host the overloaded agents. It requires that the minimum capacity of an Elastic node is at least larger than the minimum workload of an agent in the overloaded server at time t (i.e. $\min(w_t^{Ag_i})$). For simplicity, we assume the capacity of an Elastic node $c^{Elastic}$ is greater than the maximum of agent workload across the system life time.

$$c^{Elastic} \geq \max(w_t^{Ag_i}), \forall t, Ag_i \quad (\text{Eq. 5.8})$$

After launching a new Elastic node, Iteration m restarts from the beginning of agent migration reasoning.

- Migrating to existing SC nodes

If $Ag_{selected} \neq \emptyset$, the scheduler will find a candidate set of SC nodes, which is defined as follows,

$$\begin{aligned} \mathbf{SC}_{candidate}^{Ag_{selected}} = & \left\{ SC_i \mid w_{t,m}^{SC_{focused}} + w_t^{Ag_{selected}} \leq c^{SC_i} \right. \\ & \left. \text{and } SC_i \in \mathbf{SC}^{-SC_{focused}} \right\} \end{aligned} \quad (\text{Eq. 5.9})$$

where $\mathbf{SC}^{-SC_{focused}}$ denotes the set of SC nodes excluding the current SC node in consideration. Eq. 5.9 ensures that the destination node have enough resource to host the migrated agent at current situation. Note that the safety cap defined in Eq. 5.5 guarantees $\mathbf{SC}_{candidate}^{Ag_{selected}} \neq \emptyset$.

At the end of Iteration m , the scheduler greedily selects the least loaded nodes to accept the migrated agent $Ag_{selected}$.

$$SC_{selected} = \arg \min_{\forall SC_i \in \mathbf{SC}_{candidate}^{Ag_{selected}}} (w_{t,m}^{SC_i}) \quad (\text{Eq. 5.10})$$

Figure 5.3: GoalNet of Achieving Load Balancing

Following the economic concern as mentioned before, we preferably assign Cloud nodes with higher priority to be selected as the destination than existing Elastic nodes.

The GoalNet for achieving load balancing for resource allocation agent is presented in Figure 5.3. Note that three steps from initiation, migration reasoning to node discovery are elaborated in the agent knowledge model, and the knowledge is stored by the motivated agent as its pursuit of performance need. One can find that the proposed rule-based algorithm employs similar policy as greedy approach for agent migration reasoning and node discovery, except allowing Elastic nodes and distinguishing between different types of nodes. As the same reason illustrated in the motivating example, it is vulnerable to cases when rescheduling is needed. This case study future proposes a prediction-based load balancing algorithm to enhance the performance.

5.5.2 Prediction-based Agent Knowledge for Enhanced Load Balancing

As shown previously, it is difficult to avoid rescheduling without future workload. Therefore the system scheduler must analyze future states of cells. In this case study, load balancing agents are embedded with ELM neural networks, so that agents can make predictions on future workload, and then send this information to the scheduler. By collecting workload updates of

current and future states from agents, the load balancer can be improved when deciding which and where an agent should be migrated.

The prediction-based load balancing algorithm extends the rule-based scheme, and the enhanced agent migration reasoning, and node discovery are explained as follows.

- Enhanced agent migration reasoning

Existing scheme adopts a random or greedy approach to select an agent. In prediction-based approach, one more step immediately after (7) is carried out to examine the future situation.

$$\mathbf{Ag}_{candidate(t \wedge t+1)}^{SC_{focused}} = \left\{ Ag_i \mid w_{t+1,m}^{SC_{focused}} - w_{t+1}^{Ag_i} \leq c^{SC_{focused}} \text{ and } Ag_i \in \mathbf{Ag}_{candidate}^{SC_{focused}} \right\} \quad (\text{Eq. 5.11})$$

where $\mathbf{Ag}_{candidate(t \wedge t+1)}^{SC_{focused}}$ denotes the set of candidate agents after migrating which the load of $SC_{focused}$ will be within its capacity at the future step $t + 1$. The future workload $w_{t+1}^{Ag_i}$ for agents is obtained from the predicted output of ELM neural networks, and the future workload of a SC node $w_{t+1,m}^{SC_{focused}}$ is the sum of predicted workload of individual agents.

If $\mathbf{Ag}_{candidate(t \wedge t+1)}^{SC_{focused}} \neq \emptyset$, the scheduler will select an agent to migrate from $\mathbf{Ag}_{candidate(t \wedge t+1)}^{SC_{focused}}$, otherwise simply select one agent from original set $\mathbf{Ag}_{candidate}^{SC_{focused}}$. Agent selection follows Eq. 5.6 for either case, and rescheduling can be avoided for time $t + 1$ if selection is done in the non-empty set of $\mathbf{Ag}_{candidate(t \wedge t+1)}^{SC_{focused}}$.

- Enhanced node discovery

Previous scheme only considers SC nodes that have enough capacity to accommodate the shed agent for time t as in Eq. 5.9. It is possible that a selected SC node will be overloaded at time $t + 1$. Using expected future workload, the prediction-based approach takes node states at $t + 1$ into account immediately after Eq. 5.9. We define the set of candidate SC nodes which will have enough capacity to host the migrated agent at time $t + 1$ as follows,

$$\mathbf{SC}_{Ag_{selected}(t \wedge t+1)}^{Ag_{selected}} = \left\{ SC_i \mid w_{t+1,m}^{SC_{focused}} + w_{t+1}^{Ag_{selected}} \leq c^{SC_i} \text{ and } SC_i \in \mathbf{SC}_{candidate}^{Ag_{selected}} \right\} \quad (\text{Eq. 5.12})$$


```

1 foreach time  $t$  during the lifetime of a LSCVE do
2   identify the set of overloaded SC nodes  $P_t = \{SC_i | w_t^{SC_i} > c^{SC_i}\}$ 
3   initialize:  $m = 0$ ,  $w_{t,0}^{SC_i} = w_t^{SC_i}$  and  $Ag_{t,0}^{SC_i} = Ag_t^{SC_i}$ 
4   find  $SC_{focused} = \arg \max_{\forall SC_i \in P_t} (w_{t,0}^{SC_i} - c^{SC_i})$ 
5   while  $SC_{focused}$  exists do
6     identify  $Ag_{selected}$  according to Eq. 5.3- Eq. 5.7 or enhanced by Eq. 5.11
7     if  $Ag_{selected} = \emptyset$  then
8       | lunch a new Elastic node
9     else
10      select  $SC_{selected}$  and migrate  $Ag_{selected}$  to the node according to
        Eq. 5.9-Eq. 5.10 or enhanced by Eq. 5.12
11      update  $m = m + 1$ , and find next
         $SC_{focused} = \arg \max_{\forall SC_i \in P_t - \{SC_{focused}\}} (w_{t,m}^{SC_i} - c^{SC_i})$ 
12    end
13  end
14 end

```

Figure 5.4: Proposed Load balancing Algorithms for Cloud-based LSCVEs

Similarly, workload subscribed by $t + 1$ leverages the predicted output from the proposed usage of ELM neural networks as mentioned earlier. In either case of $SC_{candidate(t \wedge t+1)}^{Ag_{selected}} \neq \emptyset$ or $SC_{candidate(t \wedge t+1)}^{Ag_{selected}} = \emptyset$, the selection of SC node to accept the migrated agent $Ag_{selected}$ conforms to Eq. 5.10.

The overall load balancing algorithms for Cloud-based LSCVEs is summarized in Figure 5.4. In order to allow the proposed approach to be used in conventional multi-server architecture, for Step 7 it is possible to disable the creation of an Elastic node. In that case, some overloaded SC nodes may remain unresolved.

In next section, simulation has been designed to compare conventional and proposed situation-aware load balancing schemes. The simulation details and comparison results are discussed.

5.6 Simulation Analysis

In this section, simulations have been conducted to assess the prediction capability of ELM neural networks as well as to compare different load balancing schemes in terms of QoS delivered. The settings of experiments are firstly explained, followed by main results and discussions.

5.6.1 Simulation Settings

In the simulation, we build a two-dimensional grid as an abstracted virtual environment, and users are modelled as moving points in the 2D space. The 2D space is equally partitioned into cells, and a load balancing agent is in charge of one cell to provide the load monitoring service.

5.6.1.1 Simulation Parameters

The parameters used in the simulation are specified as follows:

Cell:The entire 2D space consists of N_c cells of equal size. Each cell had one load balancing agent, so there are totally N_c agents. Each cell corresponds to a unit of virtual region in the virtual environment. Since users are allowed to move freely, a cell may host different numbers of users at different times. In this experiment, the 2D space contains 10×10 cells.

Users:The virtual avatar of a user is modelled as a point in the 2D space, and there are totally 1000 users in the virtual world.

SC nodes:Initially there are $N_s = 5$ Cloud nodes in the virtual environment. All nodes initially host equal number of cells, so there are N_c/N_s cells in one Cloud node. This experiment assumes each Cloud node has the same capacity, $c^{Cloud} = 200$, which is the maximum number of concurrent players that can be accommodated in a Cloud node. The capacity of an Elastic node is assumed smaller than existing Cloud nodes. In the simulation, $c^{Elastic}$ equals to $c^{Cloud} \times 10\% = 20$. This assumption complies with the common approach of Cloud users who prefer dynamically added Elastic nodes to be smaller than Cloud nodes, so that required

provision of Elastic nodes is easy to be satisfied. For the same reason, we further limit the maximum number of Elastic nodes to be 10 in the simulation.

5.6.1.2 User Distribution and Movement Model

In this case study, we consider a scenario that at the beginning users are scattered randomly within the virtual environment. A hot point (an interesting event) is immediately observed by all users. Users start to approach the hot point with one's own speed and moving direction.

Based on this scenario, user nodes in the simulated 2D space are uniformly distributed. Following Random Waypoint Mobility Model which have been extensively exploited for simulations in virtual environments [135–137] we define the movement of a user with a random speed and add a small perturbation to the angle.

Figure 5.5 illustrates one example of the initial and final distribution of 1000 users in a 10×10 2D space. The red cue in the center represents the hot point, and it appears randomly within the environment in different trials. For one trial, the lifetime of the simulation is set to 120 steps.

5.6.2 Analysis of Motivated Agents with Different Learning Skills

We prefer ELM as the paradigm to do prediction for load balancing agents. In order to justify our selection, a comparison between two agent learning skills, namely back-propagation (BP) neural network and ELM, is carried out for predicting the number of users in each cell for the next time step. As mentioned previously, it is a time series prediction problem, and the parameter p in the regression model Eq. 5.2 is set to 5 for this experiment.

BP neural network is constructed using 10 hidden neurons, and the training algorithm of BP is implemented by Levenberg-Marquardt. ELM is created with 30 hidden neurons, and input weights of ELM are uniformly and randomly distributed on the interval $[-1, 1]$ and biases on $[-1, 1]$. Both BP and ELM adopt hidden neurons with sigmoid activation and linear output neurons.

(a)initial distribution

(b)final distribution

Figure 5.5: Sample User Distributions: Initial And Final

Since the load balancing agent in each cell has its own neural network, there are totally N_c neural networks. Load balancing agents are running in a parallel manner, and the training is independent of each other, so the total time required for training is as the same as one neural network.

Although training data is obtained from past records of previous simulation trials, rather than recall the generalization capability of neural networks enables load balancing agents to learn the regression model and forecast accurately in other simulations. The testing performances for both BP and ELM are shown in Figure 5.6. We select 2 cells for demonstration.

One is at top left, and the other is around center as in the 2D space of Figure 5.5.

For the top left cell, the number of users decreases since users are likely to move away from the corner, while the near center cell behaves in a non-monotonic manner. As shown in the figure, both BP and ELM can learn the mapping from the data with small errors (typically 1 user). Therefore, in terms of prediction accuracy, BP and ELM are comparable. However, in this simulation BP requires more training times than ELM in the order of 10^2 , due to thousands of training iterations used by BP. To the contrast, ELM has fast training speed, but the number of hidden neurons is three times than that of BP, requiring more memory. This means ELM has

(a)corner

(b)center

Figure 5.6: Testing Performances for 2 Selected Cells.

limited usage if memory space for training is limited. Nevertheless ELM is preferred, because speedy response is more important than memory usage for virtual world applications.

5.6.3 Analysis of Scheduling Performance

During the lifetime of a virtual world, workload in each cell changes dynamically and cannot be known apriori. This may cause problems for any scheduling scheme on deciding how to efficiently and effectively migrate a load balancing agent from one virtual machine to another. With the flexibility and incremental scalability of Cloud computing, a load balancing agent can be positioned on a newly created Elastic node so that some difficult overloading situations can be improved. In addition, with the help of a trained neural network, an agent can predict the workload one step ahead, so that occurrences of rescheduling can be reduced to avoid further violation of SLA.

This experiment has compared different approaches, namely static partitioning (**Static**), uniform load sharing (**Uniform**), greedy load shedding (**Greedy**), rule-based policy with and without Elastic nodes (**Rw/En** and **Rw/oEn**), and prediction-based scheduling with and without Elastic nodes (**Pw/En** and **Pw/oEn**).

We measure the following performance indicators to compare the performance of a load scheduling scheme.

Number of Overloaded Users (N_{OU}): The overloaded users are those situated on overloaded SC nodes. For each step, the number of overloaded users is counted, defined as N_{OU} . The average N_{OU} is used to assess the severity of violation of the SLA during the entire simulation because of overloading servers. The smaller N_{OU} , the fewer users suffer from downgraded virtual world service by saturated SC nodes.

Number of Migrated Users (N_{OU}): The migrated users are those moved from one SC node to another because of load balancing. For each step, the number of migrated users is counted, defined as N_{OU} . The average N_{OU} is used to assess the severity of side effects on users because of scheduling. The smaller N_{OU} , the fewer users are affected by a load balancing scheme.

Effectiveness of Load Distribution (Eff): The effectiveness measures how much percentage of overloaded users can be resolved by a load balancing scheme. The effectiveness for one time step is defined as,

$$\text{Eff} = (N_{OU} - N_{OU}^*) / N_{OU} \times 100\% \quad (\text{Eq. 5.13})$$

where N_{OU} is the number of overloaded users before scheduling, and N_{OU}^* is the number of overloaded users after a scheduling. Apparently, effectiveness with one implies that all overloaded users are resolved by a load distribution.

Figure 5.7 illustrates the average N_{OU} over 50 simulations before and after scheduling (dark area) for every scheduling scheme. The value of N_{OU} before scheduling evaluates the occurrence of SLA violation at the beginning of each time step, as a measure of the effectiveness of disaster prevention. To the contrast, the value of N_{OU} after scheduling evaluates the unresolved occurrence of SLA violation at the end of each time step, as a measure of the effectiveness of disaster recovery.

In terms of before scheduling, static partitioning has the largest number of overloaded users, and a great improvement is achieved by applying any dynamic load distribution. In cases of without Elastic nodes, Uniform, Greedy and Rule-based are comparable with each other about 420 N_{OU} , whereas Prediction-based outperforms the rest with improvement around 15%. Among all the scheduling schemes, both rule-based and prediction-based approaches with Elastic nodes have the smallest N_{OU} before scheduling with about 50% improvement over algorithms without Elastic nodes. Thus, adding new servers greatly reduces the happening of server saturation, and manifests itself as a strong method of disaster prevention. Nevertheless, prediction-based wins over rule-based with a difference of 26% in average.

Due to the lack of load distribution in static partitioning, overloaded users remain unchanged, and have to suffer the downgraded service until next time step. Except Static, N_{OU} is reduced for all other approaches after a load distribution. Uniform has the most unresolved problems. Although Rule-based and Prediction-based algorithms have similar performance, an

Figure 5.7: Performance Comparison for N_{OU}

improvement of more than 20% is observed for the one with predictions. Interestingly, the two approaches allowing Elastic nodes resolve all the overloaded users, indicating no server is left overloaded.

From the performance comparison in terms of N_{OU} , we can draw a conclusion that in this simulation the proposed rule-based and prediction-based load balancers best prevent the occurrence of SLA violation as a result of overloading, and they provide the most effective load distribution across the system. Moreover, prediction-based algorithm improves over the rule-based by alleviating more than 20% N_{OU} .

Figure 5.8 presents the average N_{MU} over 50 simulations for all the approaches. Since no migration is carried out, static partitioning associates with zero reallocated users. Uniform load sharing has the largest number of N_{MU} , because it aims at a fair sharing of workload across all the SC nodes, and tries to achieve the smallest imbalance among all server nodes. However for Cloud-based virtual environment, fairness is not the main concern, and from the results we can find that the cost of uniform load spreading is extremely high. In this simulation N_{MU} of Uniform is in the order of 10^3 , and the exact plot is left out for clarity. The figure clearly

Figure 5.8: Performance Comparison for N_{MU}

shows that significantly fewer users are affected by the rest algorithms than those of Uniform. Similar to Figure 5.7, proposed algorithms with Elastic nodes have the smallest N_{MU} , and prediction-based beats rule-based with average 20% improvement for cases with and without Elastic nodes. As a conclusion, the proposed approaches have less severe side effect on user activities as a result of load distribution.

We can sum up the results in Figure 5.7 and Figure 5.8 to get *total_SLA_violation* defined in Eq. 5.1. Although static partition has zero cost in migrated users, it involves the highest cost in overloaded users. The total cost defined in *total_SLA_violation* is higher than other algorithms. It is easy to observe that the rest algorithms are consistent in both criteria. For example, rule-based algorithm with Elastic nodes has both smaller overloaded users and smaller migrated users than greedy algorithm. Therefore, in terms of overall end user experience defined in *total_SLA_violation*, algorithms with Elastic nodes outperform those without, and prediction-based has improved performance over rule-based.

Figure 5.9 compares the overall effectiveness of different load distribution schemes, which is the average of scheduling effectiveness over all time steps. The two proposed approaches

Figure 5.9: Performance Comparison for Overall Effectiveness

with Elastic nodes achieve 100% effectiveness, followed by prediction-based without Elastic nodes, rule-based without Elastic nodes and Greedy, which have similar results around 75%. Uniform only has an effectiveness of 26%, and nearly one quarter of overloaded users remain affected by overloaded servers. As a conclusion, load distribution with Elastic nodes is the most effective, for which all the overload users can be resolved.

5.7 Summary

We firstly discussed the scalability issue of LSCVEs with a formulation of QoS driven load balancing. The existing multi-server based approaches were reviewed, followed by an argument on the importance of analyzing future workload state to prevent immediate re-scheduling. A workload prediction using ELM neural network was proposed for motivated load balancing agents to allow the centralized resource allocation agent to make decisions based on future system state. A rule-based load balancing algorithm on top of Cloud architecture was proposed as agent's pursuit of performance to allow the use of additional resources to support incremental scalability. Consequently, prediction-based load balancing extended the rule-based approach

by applying workload predictions. With multiple load balancing agents, resource allocation agent with the prediction-based load balancing algorithm can avoid re-scheduling in the immediate future.

The simulation results showed that without Elastic nodes rule-based scheme performs similarly to a greedy approach. However, improvements were observed if Elastic nodes were allowed. For either case, with or without Elastic nodes, the proposed usage of predictions outperformed existing and rule-based methods, resulting in fewer violations of SLA in terms of overloading and side effects due to load distribution. Therefore according to Eq. 5.1, load balancing algorithm with predictions and ability to add new servers achieves the smallest total SLA violation. It showed that proposed motivated agents can be adaptive to the dynamics of user movements within a virtual environment, and their competence and relatedness pursuit allows the resource allocation be more effective in avoiding service violation.

In current work, we treat the adverse effects of overloading and migrating equally as shown as the objective function defined in Eq. 5.1. However for different LSCVE applications, the impact of overloading may be more severe than that of migrating. Therefore, a weight assignment for the two criteria needs to be investigated for different applications in the future.

Chapter 6

Case Study II: Applying Competitive Intrinsically Motivated Agents for Resource Provisioning in a Hybrid Cloud

6.1 Why Agent-Mediated Provisioning for Hybrid Cloud

Hybrid Cloud [138] by definition of NIST (National Institute of Science and Technology) is a composition of two or more distinct cloud infrastructures. A hybrid cloud can be multiple public clouds, private clouds or a combination of both. As more and more enterprise customers consider the adoption of cloud computing, they realize that sticking with one type of cloud does not fit all their needs. For example, sensitive data and business critical process should be kept internally, such as in a private cloud. However, IT development, Testing, Client facing web sites should be put into public clouds to save money and to scale out quickly. So neither of public nor private clouds will do the job alone. A hybrid approach is the right way to adopt cloud computing for enterprise customers.

Recent market analysis shows that hybrid cloud is one of the most important trends in the adoption of cloud computing. Some of the evidences are as follows,

- Gartner predicts that by 2015, the majority of services using private clouds will evolve to leverage public clouds in a hybrid mode.

- A study by IDC in 2011 says that hybrid cloud will be widely used in the Asia/Pacific region.
- North Bridge analysts also found that 39% of both private and public cloud users say that the hybrid cloud is currently part of their strategy; that number is expected to grow to 61% in the near future.

However, there are many challenges to utilize a hybrid cloud for enterprise users [139]. One of them is the automatic resource provisioning. Automatic resource provisioning is one of key promises from conceptual Cloud computing. Topics such as scaling up and down quickly in response to workload and providing resource that is sufficient remain obstacles in recent Cloud research. RAD Lab (Reliable Adaptive Distributed System Laboratory) in UC Berkeley [140] considers elastic scaling as one of the ten obstacles, and explains the opportunity as “automatically scaling quickly up and down in response to load in order to save money, without violating service level agreements”. In another review [141], researchers state that by allocating/de-allocating resources from Cloud, a service provider achieves the objective of satisfying its service level objectives (SLOs), with operational cost minimized. However, current state of the art lacks a method to analyze the relation between high-level SLOs and low-level resource properties, such as CPU and memory size, and such mapping should be in an online fashion.

In this case study, we restrict ourselves on infrastructure resource provisioning, a high level meta scheduler. There are several essential phases within a scheduling process, explained by Schopf [81]. Schopf argues that there is currently no single system addressing all the needed features, and the most common resource selection is done by the user [81]. Resource selection requires intelligent system support for proper decision making. Resource selection starts from the time to check whether a set of resources pass the minimum requirements, and it continues to analyze the dynamic resource information at hand, and makes scheduling decisions based

on expectations of available resources. After that, a scheduler handles the submitted job over selected local resource scheduler, and monitors the job execution.

Infrastructure resources are under control of a scheduler which implements allocation and priority policies, and optimizes the execution performance for a given task. Due to the complexity of a real-life environment, scheduling has to handle information of heterogeneous resources, and fulfill various task requirements. Additionally, scheduling has to cope with resource flexibility and workload dynamics. Both complexity and uncertainty pose challenges onto resource provisioning mechanism that a scheduler needs to be both intelligent and robust in order to be competent in the real-life environment.

Competitive intrinsically motivated agents are the best choice for automatic resource provisioning, because (1) the complexity of a real-life environment is handled by motivated agent's goal-oriented through goal hierarchy and decomposition; (2) the intelligence is achievable by motivated agent's autonomy by motivated goal/action selection; (3) the adaptiveness to the dynamic environment is tackled by motivated agent's pursuit of novelty via learning/mastering new skills; and (4) robust decision making is supported by motivated agent's relatedness in seeking collaborations with other agents.

Thanks to the novelty pursuit, resource provisioning agents can learn a competent skill to estimate resource availability, predict traffic congestion, and profile workload dynamics. For most of these cases, agents are experiencing highly nonlinearity, for which there is a lack of well defined analytic solutions. Therefore, a data-driven nonlinear learning scheme is desired for agent to master. As the result of the performance pursuit, motivated agents can adaptively select a reliable and proper infrastructure resource to satisfy QoS requirements and meet task deadlines in the consideration of various constraints. Due to the limited resource and imbalanced distribution of information, the collaboration among multiple agents provides unbiased and robust decision supports, driven by agent's relatedness pursuit. In the view of organizing agent's various tasks, GoalNet is essential to manage agent's deliberative process systematically. Goal's hierarchy categorizes agent activities in various levels of details, allowing agent

planning and decision making efficient. As a summary, a full-fledged intrinsically motivated agent designed and implemented for resource selection offers great intelligence and adaptiveness to the hybrid cloud applications.

6.2 Infrastructure Provisioning in Hybrid Cloud

This section first gives a real-life case where a hybrid cloud testbed is utilized for infrastructure provisioning by enterprises. The testbed is designed by HP Labs Singapore for a convenient and easy management of heterogeneous cloud resources. The proposed agent-mediated resource allocation is leveraged by the enterprise-class solution, and a detailed problem formulation in terms of infrastructure provisioning is also given in the rest of this section.

6.2.1 Background of the Hybrid Cloud Testbed

The testbed of managing hybrid cloud for enterprise users is called “Monsoon”, which is a research prototype developed by HP Labs Singapore. The purpose of Monsoon is to provide enterprise users a comprehensive management tool for their cloud infrastructure in hybrid cloud environment.

The system architecture of Monsoon is illustrated in Figure 6.1. At the bottom, Monsoon has been integrated both private and public clouds. HP’s CloudSystem Matrix is an example of private cloud, for which an enterprise may have a localized installation and the access is strictly controlled within its own organization. For public clouds, Monsoon has been integrated with Amazon EC2, Rackspace and GoGrid, through web service protocols. In order to have a unified management for those cloud providers, in the middle layer a IaaS proxy is implemented. The proxy is implemented in a Model-based Service Integration Environment (SIE), which is a collaboration with HP Labs Palo Alto. This Service Integration Environment allows an infrastructure service to be described and invoked. The proxy has its own definition of infrastructure resources and such definitions are described as generic service models. The

Figure 6.1: System Architecture of a Hybrid Cloud Testbed

set of service models are proposed by abstracting the common features from multiple cloud providers. As a result, by talking to one single proxy, enterprises can manage their resources from multiple providers.

At the enterprise side, their applications can talk to a RESTFUL interface, or their users can log in the Monsoon portal to do resource management. Monsoon covers both basic cloud infrastructure management tools and advanced features to meet the enterprise users' requirement in hybrid cloud environment. The tool set includes,

- *User Management*: This module provides basic user and privilege management, where users are assigned with different roles and thus privileges in viewing, modifying and controlling certain set of infrastructure resources.
- *Access Control*: This module coordinates with User Management module to provide authentication, authorization, and access control to ensure the portal users only gain access to the authorized resources.

Figure 6.2: Screenshot of Cloud Resource Management in Dashboard

- *Resource Management*: This module provides users a dashboard to manage his cloud infrastructure in a hybrid cloud. It has been shown in Figure 6.2 that cloud resources from different service providers are managed in one dashboard.
- *Monitoring and Reporting*: This module enables portal users to monitor his cloud infrastructure in real time. A set of tools will provide the graphs to show statistics of the virtual resource, alert setting and report generation, as shown in Figure 6.2
- *Policy Engine*: This module is designed to allow user to define requirements such as geography, security level, government regulations and corporate policies. The policies defined by users will be enforced to conform to the compliances for the subscription and deployment on IaaS resources.
- *Optimization Engine*: This module dynamically allocation cloud resources based on QoS-centric performance analysis, and aims to select resources intelligently in the way

that user requirements are satisfied and optimized. The agent-mediated resource provisioning is proposed for making this module adaptive.

6.2.2 Problem Formulation for Dynamic Resource Provisioning

Given a hybrid cloud scenario, such as Monsoon, users are offered with a large number of infrastructure resources, varied in cloud provider, geographical location, configuration, favor, size, price, etc. Despite the extent of potential scale and variety of choices, hybrid cloud poses difficulties for users to choose appropriate resources.

For example, current Monsoon, when provisioning cloud resources, requires users to opt for resource properties such as service provider (EC2, GoGrid, Rackspace, Cloud System Matrix) and configurations (CPU, memory). Such resource selection is subjective and lacks of supportive information on resource in terms of gain and cost. Users are usually blind about what kind of resource to choose and what are the pros and cons. One way is to implement corporate policy to restrict user demands against requesting resources that hit the bottom line of the company. The other way is to consider resource selection as an optimization problem and find the resource property that maximize (or minimize) the objectives. The first method is related to minimum requirement filtering at a high level, which ignores resources that violate the corporate policy. The other method is related to multi-objective optimization with constraints, which tries to find the optimal solution in terms of the objectives. This section mainly focuses on the second aspect and is able to be coupled with the first aspect to provide a complete and sound solution.

Given a set of resources $\mathbf{R}$ and a job j with objective function f^j and constraints $\mathbf{C}^j$,

$$\text{find } r^* \text{ such that } r^* = \arg \max_{r \in \mathbf{R}} f^j(r), \text{ subject to } \mathbf{C}^j \quad (\text{Eq. 6.1})$$

The provisioning of cloud resource is changing overtime different from each case. $\mathbf{R}$ contains finite but non-static elements. The feasible solution set of optimization shifts over the

dynamics of $\mathbf{R}$. For different jobs j , the objective function f^j could vary. The optimization space is thus altering over different jobs.

The formulation of the resource selection problem comprises three key components, including demand modeling, supply modeling and optimization. Demand model targets at providing a comprehensive way to capture properties of a time-varying application system, inclusive of QoS-related objectives, constraints and dynamics of workload. Objective function f^j and constraints C^j belong to demand model. Supply model describes and differentiates Cloud resources in light of providers, capacities, configurations, etc. and deals with benchmarking and monitoring. From supply modeling, the system anticipates what the consequences in terms of SLA would be for a given set of resource supplies in response to a given amount or type of load demands. In this research, we use performance profiling to model resources. Typically, a customer specified benchmarking is applied to a relatively small set of resources to anticipate how much quality of service could be obtained from resources. Optimization takes the two models as inputs and use them for matching analysis and portfolio optimizations.

A simplified version of resource provisioning is tackled in this case study. Jobs are assumed to share one objective function, which only concerns about resource utilization. Resource utilization is defined as follows,

$$Utilization = \frac{QoS_{\hat{r}}}{\max_{r \in \mathbf{R}} QoS_r} \quad (\text{Eq. 6.2})$$

Utilization is measured as the ratio of QoS provided by the selected resource $\hat{r}$ to that by the best resources among the candidates $\mathbf{R}$. The defined utilization refers to whether a cloud user/subscriber has utilized the full capacity of computing power under his subscription. “Private clouds” involve on-premises hardware and software, for which the cost incurred happens before consuming. Therefore, there is a need to justify whether a scheduling algorithm can save organizations from wasting computing power. For “public clouds”, the ideal case lies in the assumption that cost only occur at the time of consumption, i.e. “pay per use”. However,

in reality, enterprises usually subscribe public cloud services before hand, due to the delay of on-the-fly provisioning and the difference of subscription rates.. Therefore, enterprises leveraging public clouds still need to consider the utility function defined above. We use it as the key indicator for hybrid clouds, where a higher utilization means less resource power is wasted, and thus the closer utilization is to one, the better a provision is.

Given above formulations, a resource allocation policy can be defined as

$$\Phi : \mathbb{R} \times \mathbb{J} \rightarrow j, \text{ where } j \in \mathbb{J} \quad (\text{Eq. 6.3})$$

where $\mathbb{R}$ and $\mathbb{J}$ referring to the universal spaces of resources and jobs accordingly. Therefore, the optimal policy should have the best overall utilization property, which is,

$$\Phi^* = \arg \max_{\Phi \in \Pi} \sum_{j \in \mathbb{J}} \sum_{r \in \mathbb{R}} Utilization_{j,r} \quad (\text{Eq. 6.4})$$

So the agent-mediated approach will cover how to enable intrinsically motivated agents to achieve an adaptive policy, such that the overall utilization is maximized. The agent-mediated resource provisioning is explained in the next section.

6.3 Modeling Resource Selection Service Using Motivated Agents

The proposed intrinsically motivated agent follows the main task described as the GoalNet shown in Figure 4.2. Designed with same sub-tasks for novelty, performance and relatedness pursuit, the agent is able to learn, to act and to cooperate. In this case study, performance is of our primary concern, and a detailed explanation of performance-driven design is presented in this section.

Recall the GoalNet for the fulfillment of an external request, as shown in Figure 4.7. The design of one of the sub GoalNet “Invoking Knowledge” has a particular instantiation for resource selection. The sub GoalNet is given in Figure 6.3.

Figure 6.3: GoalNet for Invoking Resource Selection

The knowledge for resource selection is invoked by collecting information of available resources $\mathbf{R}$ and job j . The set of resources are passed to a filter for minimum requirement filtering. This process is essentially another sub GoalNet, which compares user requirements specified in j with resource properties specified in $\mathbf{R}$. In particular, a checking is done by analyzing whether a resource $r \in \mathbf{R}$ could satisfy constraints $\mathbf{C}^j$ of j . If a resource passes minimum requirement filtering, then it is labeled as a candidate and stored in $\mathbf{R}^C$, which is the candidate set. The details are omitted, and we focus on the selection steps.

The selection is conducted in an iterative manner, until all candidates are processed. The important action taken by an agent is the analysis of resource score. The resource score represents the preference of one resource over the pending task. Two types of analysis are discussed, namely rule-based and learning-based approach.

6.3.1 Rule-based Competent of Resource Selection

Agents are initially designed with a rule-based knowledge, which is an example of existing schedulers [83–86]. The rule-based analysis is based on a user-defined rank expression. The expression is explicitly defined as a linear combination of one or multiple resource properties, such as $Rank = CPU_speed + Memory_size$. By the rank expression, it is straightforward to obtain the analysis score of a resource. After calculating the scores for all the candidates,

agents sorts the result and considers the resource of the highest score as the selected output.

To summarize, the rule-based policy of using ranking heuristics can be modeled as,

$$\Phi_{rule} : \hat{k} = \arg \max \text{Ranking}(r_k), \text{ for } 1 \leq k \leq |\mathbf{R}| \quad (\text{Eq. 6.5})$$

As mentioned in the example in Chapter 3, the problem with the rule-based policy lies in the fact that it requires users to have sufficient prior knowledge for a particular problem in order to determine the ranking expression. Meanwhile, the defined rank expression cannot directly reflect how a job will be executed. For example, if a resource with higher computing power is preferred, however it is not necessary that a machine with bigger sum of *CPU_speed* and *Memory_size* is more powerful. In most of the cases, a simple expression cannot meaningfully represent the complex nonlinear relationship between resource and its performance in terms of QoS. In the next section, a learning-based policy is used for the agent to provision a resource. The details of learning-based analysis are given as follows.

6.3.2 Learning-based Competence of Resource Selection

Performance profiling is a challenging issue, which aims to understand how a computing resource would behavior given a type of task. Profiling is done by analyzing job natures and resource characteristics, and usually involves extensive research of low level operations. However, the obtained profiling results are not easily reusable, because different tasks would involve different low level operations and may have a variety of dynamics and transaction patterns. Therefore, in this case study, a benchmarking approach is exploited by applying a sample of job-specific benchmark onto a set of resources. The result measured from experiments best describes the performance directly in terms of QoS. However, benchmarking cannot be done on all resources, otherwise profiling would consume most of the time. Fortunately there is a relationship between infrastructure resources and QoS they could provide for a certain type of job, but the relationship is unknown and hard to describe explicitly.

The solution is to adopt a learning-based analysis to predict the QoS that would be delivered by running a job on a resource. Learning is carried out on the performance data collected from the profiling phase. A learning-based resource allocation policy is proposed as,

$$\Phi_{learning} : \hat{k} = \arg \max \mathcal{P}(r_k, j), \text{ for } 1 \leq k \leq |\mathbf{R}| \quad (\text{Eq. 6.6})$$

where $\mathcal{P} : \mathbb{R} \times \mathbb{J} \rightarrow \text{Real}$ denotes a predicting function which maps resource and job to a performance score. The predicting function is to be achieved as a result of learning from data. One advantage is that machine learning lifts the requirement of sufficient prior knowledge from users. Another advantage lies in the fact that learning can provide accurate and timely results, and it is particularly helpful in Grid environment, which is usually dynamic and uncertain. In order to provide an optimal resource selection for hybrid cloud, a fast and intelligent learning paradigm is required. In this research, two simple neural networks are implemented as the proposed models of learning.

As introduced in Section 4.3.2, Extreme Learning Machine (ELM) is proposed as one candidate of learning. It is reasonable to assume that a resource allocation service based on ELM will inherit the feature of fast learning, and will predict resource performance with satisfaction. The other neural network is chosen as Backpropagation (BP), which is a traditional machine learning tool. The two learning schemas are used to show the capability of the proposed motivated agent in terms of the applicability of a range of data-driven adaptive techniques.

The detailed workflow is explained in Figure 6.4. The proposed learning-based resource allocation is presented in Figure 6.4. It is apparent that the proposed agent relies on the accuracy of the predicted performance values of ELM network or BP network. In order to show that ELM network and BP are competent for resource selection, and ELM is preferred due to its fast learning, numerical comparisons were conducted in the experiments.

input : Available resource $\mathbf{R}$ and a pending job j

output: A selected resource $\hat{r}$

- 1 After minimum requirement filtering, intrinsically motivated agent obtains a set of candidate resources $\mathbf{R}^C$ for a submitted job j , where $\mathbf{R}^C = \{r_1, \dots, r_K\}$ and $\mathbf{R}^C \subseteq \mathbf{R}$;
- 2 **for** $i \leftarrow 1$ **to** K **do**
- 3 Agent enquires one selected neural network with $r_i \in \mathbf{R}^C$ and j ;
- 4 The neural network responds with the predicted performance score p_i ;
- 5 **end**
- 6 Agent allocates j to the resource $\hat{r}$, where $\hat{r}$ has the index $\hat{k}$, such that $\hat{k} = \arg \max p_i, 0 \leq i \leq K$.

Figure 6.4: Learning-based Resource Allocation Workflow

6.3.3 Collaborative Resource Selection by Multiple Motivated Agents

Individual agent based on ELM randomly assigns input weights and biases, instead of explicit training as in gradient-decent based network. The idea of random assignment releases the feedforward neural network from demanding training for input weights and biases. It also ineluctably leads to certain stochastic behaviors of ELM-based agents, which bring about uncertainty and susceptibility to errors [94]. The work in [94] suggests stacking multiple ELM networks to improve the overall performance for generalization, referring to which a cooperation for multiple ELM-based agents is proposed in this case study. In Section 4.5.2 and Section 4.5.3, a collaborative model has been discussed and the numerical justification from machine learning perspective has been given. In this section, an agent cooperative decision making is proposed based on the previous knowledge model.

The cooperative decision relies on a centralized structure, in which a Collaboration Initiator (CI), a motivated agent collects decisions from each collaborative agent, and draws a conclusion as the group's final decision. Particularly, a CI proceeds as Figure 6.5. There are essentially two phases for collaborations, one for learning the assignment of decision weights for collaborative agents and the other for aggregating decisions. The assignment phase could be only conducted once, unless the environment is significantly changes or the competence

of some collaborative agents is significantly upgraded or downgraded. That means a CI can rely on previous assigned decision weights for a relatively long period, instead of conducting collaborative learning each time.

Cooperative decision making has its novelty in adapting CI to make group decision based on individual members, instead of just selecting a single best agent to represent the group idea. The decision power is distributed from a single agent to the community in order to filter off the biased decisions.

Both individual and cooperative agent-mediated services are to be assessed in the application of resource provisioning scenario, and related experiments are conducted to compare their performances. Before the analysis of experimental results, the next section gives an example on how agents are deployed as resource allocation services in the Monsoon environment.

6.4 Deploying Agent-Mediated Resource Provisioning Service in Cloud

One reason that why hybrid cloud computing is interesting to the business sector lies in the fact that cloud computing is the best choice to avoid over-provisioning of IT resources. Cloud computing's on-demand and elastic nature enables infrastructure to be closely aligned with the actual demand.

To business, such characteristic increases overall utilization and reduces operational cost. On the contrary, a business over-provisions IT resources and invests more powerful computers to meet the demand, but this approach only works to an extent. It is usual the case that either company wastes money or the demand exceeds the computational capacity before any new machine could be deployed. Besides physical resources, more human resource could be spent on installing and maintaining those servers.

The infrastructure elasticity must be supported by elastic service. In this context, service-oriented design of a hybrid cloud platform becomes necessary.

- input** : Agent CI, a resource request j , a set of available resource $\mathbf{R}$, a known history data L
- output**: A collaborative decision $r \in \mathbf{R}$
- 1 **Initialize**: A CI is motivated by the pursuit of relatedness, who later proactively identifies K other agents for collaboration, with established communication channels ;
 - 2 CI randomly splits dataset L into J equal parts, $(L_1, L_2, \dots, L_j, \dots, L_J)$;
 - 3 **for** $j = 1$ **to** J **do**
 - 4 CI uses L_j as testing data, and the rest $L^{(-j)} = L - L_j$ as training data;
 - 5 CI passes $L^{(-j)}$ to each agent for individual training ;
 - 6 CI transfers L_j to each agent, and receives corresponding decisions $(O_{i1}, O_{i2}, \dots, O_{ik}, \dots, O_{iK})$, for $i = 1, \dots, |L_j|$, where O_{ik} is the k^{th} agent's output for the i^{th} input data ;
 - 7 CI appends received dataset $\{(O_{i1}, O_{i2}, \dots, O_{ik}, \dots, O_{iK}, target_i), i = 1, \dots, |L_j|\}$ to its own training data L_{CV} , where $target_i$ is the desired output of the i^{th} input data ;
 - 8 **end**
 - 9 CI consolidates a complete set of training data L_{CV} , and starts its own cooperative learning ;
 - 10 CI adopts a simple linear combination of individual agent's prediction, in which CI makes prediction M^{CI} on input $\mathbf{x}$ of the form,
- $$M^{CI}(\mathbf{x}) = \sum_k \alpha_k M_k^{Agent}(\mathbf{x})$$
- where M_k^{Agent} is the k^{th} agent prediction, and α_k is the corresponding weight. In this case study, α is implemented as either least squares or averaging estimator [94] ;
- 11 CI coordinates all ELM-based agents to train on all the data in L ;
 - 12 CI passes $\mathbf{R}$ and j to the K agents for enquiry;
 - 13 CI receives the output from the agents, $\{o_1^j, o_2^j, \dots, o_K^j\}$;
 - 14 The finally selected r is obtained by the weighted majority vote with the calculated weights α .

Figure 6.5: Collaborative Decision Making for Multiple Intrinsically Motivated Agents

6.4.1 Components for the Agent-Mediated Service

By zooming in one level down the surface of a hybrid cloud platform, the architecture contains four basic components, namely computing elements, queues, storages and databases. In order to ensure the multi-tenant platform is scalable and resilient to failures, its basic components should be well designed at the first place. Therefore, all the components should be deployed in a service-oriented manner.

Computing elements are virtual machines for running applications or business logics (agent logics, in this case). They are the central processing units for handling user requests and supporting resource provisioning operations. Load balancing and elastic scaling are mainly referred to this component class. The easiness of duplicating computing elements is the potential solution for a scalable hybrid cloud solution.

Queues are used for decoupling and durably buffering messages between functioning components. They are employed to allow asynchronous communication and break tight dependencies between components, so that if one component may fail or be busy for some reason, the other components are able to work continuously as if nothing happens. Agent communications between environment and agent computing elements are facilitated by queues.

Storages act as a data warehouse for resource or agent knowledge data in forms of large files, documentations, scripts that are not going to change frequently. They also provide a backup repository. Images consisting of agent running engine, Operating System and agent logics scripts are stored in this category of components. Storages that are distributed and persistent ensure that virtual machines served as computing elements for agents and agent environment can be instantiated anywhere and anytime. In addition, agent status being monitored during agent life time are persistently kept in storages for performance improvement and intelligence analysis. Agent knowledge and initial agent models are saved in storages.

Databases support queries from component services to retrieve and store detailed message contents, system-wide component status, a temporary historical/log database for high perfor-

mance data such as short period monitoring information. Databases are particularly for frequently updated data with limited size, comparing with less frequently updated and bigger data in storages. Most importantly, since the communication between system components asynchronous, we need to be able to retrieve the system status at any time. In addition, for components that are autonomous and discrete we need for a data store that captures the information of the system and supports queries. Agent runtime data and internal status are kept as entries in databases.

6.4.2 System Architecture for the Proposed Provisioning Service

We have implemented a set of functional components based on the proposed resource allocation agents. Each component is wrapped up as a service, and exchanges messages through a standard interface with other components of a service-oriented environment. It accepts allocation requests from clients and returns then invocation address of a recommended resource based on resource information provided by cloud information services. For this case study, client side runs a number of independent requests on infrastructure resources. The functional view of the proposed resource allocation service is illustrated in Figure 6.6.

To seamless integrate with service-oriented architecture, the service is designed with *Globus Toolkit 4* (GT4) components. GT4 is a widely adopted software toolkit, which is based on web services mechanism for facilitating the development of service-oriented architectures [142]. In the figure, the web services based *Grid Resource Allocation and Management* (GRAM4) and *Monitory and Discovery Service* (WS MDS) are essential GT4 components, which are reused in hybrid cloud to provide execution management and information services respectively. The details can be found in [142], and in this section we mainly focus on the proposed resource allocation service.

The resource allocation service comprises four modules. The *Job Manager* receives allocation requests from clients and stores them by a queue service. After popping a request

Figure 6.6: Functional View of Agent-Mediated Resource Provisioning Service

from the queue, it is responsible for creating an internal task and monitoring task status; a task cleanup is carried out once an invocation address of the selected resource is sent back to the calling process. *Min Requirement Filtering* shortlists a set of candidate resources, which satisfy both parties' constraints. Besides, it is responsible for querying WS MDS to obtain updated resource information. After filtering, the job and candidate resources are conveyed for final screening. The *Service Selector* adopts the resource allocation agents specified in Section 6.3 and returns the winner with highest preference scores. The performance is application-dependent so that the underlying resource preference should be analyzed before consulting the *Prediction Service*. Therefore, prediction should be switchable among different preference settings. However, in our case we assume that one common preference is shared by all jobs. The Prediction Service uses the learning-based policy introduced in Section 6.3, and data samples required by the learning are to be retrieved from storages.

Event Manager is responsible for collecting and storing training data. It can have multiple

instances and each instance has several probes to infrastructure resources. Computing resource can be a single physical machine or a virtual machine, and they usually have a local scheduler which continuously generate scheduling events. The role of Event Manager is to extract and convert useful information into learning data.

Without the agent-mediated resource selection service, when the client requests reach the *Cloud Manager* (which is responsible for job execution management), users need to specify a rank expression (in rule-based policy case) or manually select a remote machine (as only supported by GT4). With the assistance of the proposed agents, we can automate the resource allocation process with timely and intelligent decisions. Moreover, the design is loosely coupled in the sense that the resource allocation service and the Cloud Manager that invokes it can be changed independently of each other, instead of re-implementation. The design also supports high scalability, since multiple independent service instances can operate concurrently, serving requests from different clients.

6.5 Performance Evaluation of Resource Allocation Service

This section contains several experiments to show agent's choices over learning-based or rule-based policies, BP or ELM learning skills, and individual or collaborative decision making.

6.5.1 Experiment Settings

Two kinds of scenarios have been evaluated for the proposed agent-mediated services: resource utilization optimization and resource feasibility. Whether resource is well utilized is central to the concern of cost reduction for enterprise. A high utilization of infrastructure implies the computing power procurement is efficient, otherwise a low utility flags a waste of capital. In addition, infrastructure in hybrid cloud may be provisioned in a heterogenous fashion, and resources may have themselves categorized according to their feasibility of client requests.

Examples are the feasibility of accomplishment before task deadline and the feasibility of infrastructure processing power. The resource selection must consider the task execution time and the task classification. Experimental data of the two scenarios are explained as follows.

Relative CPU Performance Computer Hardware dataset from UCI [99] is used in the experiments for utilization evaluation, which contains relative CPU performance for various hardware of different attributes, e.g. cycle time, memory size, etc. The relative CPU performance is a signature of a computing resource for CPU-intensive tasks. The measurement is obtained through benchmarking, and is used as a QoS merit.

The details of attributes are shown in Table 6.1. For simplicity, Attribute 1 and 2 are removed, for they contain nonvectorial data. Attribute 10 is viewed as the true relative performance of computer hardware. We further normalize all the attributes into the range to make selections scale invariant.

Table 6.1: Attributes Of Computer Hardware Dataset

Attribute ID	Attribute	Range
1	vendor name	N.A.
2	model name	N.A.
3	machine cycle time	17-1500 Nanoseconds
4	minimum main memory	64-32000 Kilobytes
5	maximum main memory	64-64000 Kilobytes
6	cache memory	0-256 Kilobytes
7	minimum channels	0-52
8	maximum channels	0-176
9	published relative performance	6-1150
10	estimated relative performance	15-1238

Two cases are designed for comparison, which are,

- Case 1: Two hardware attributes (CPU cycle time and maximum main memory) are considered, while the others are ignored.

- Case 2: All hardware attributes (CPU cycle time, min/max main memory, cache memory, min/max channels and published relative performance) are considered.

Furthermore, for Case 1 if duplicates are found, a new sample replaces them by averaging the values of CPU performance. As a result, totally 134 distinct samples are generated and 67 are used for testing. Similarly, in total 209 instances are employed for Case 2 and 104 for testing.

In order to ensure that candidate lists of different runs are as diverse as possible, the number of candidates is determined according to the probability of that no two consecutive candidate sets share a common machine,

$$p_0 = \left(1 - \frac{k}{n}\right) \left(1 - \frac{k}{n-1}\right) \cdots \left(1 - \frac{k}{n-k+1}\right) = \frac{((n-k)!)^2}{n! (n-2k)!}$$

where n is the number of testing resources and k is the number of candidates. Typically, we want $p_0 \geq 0.5$. Therefore for Case 1, we have $k \leq 6$ and for Case 2, $k \leq 8$. Simply k is set to be 5 for both cases.

Task Execution Time Profile A synthetic task from Matlab is implemented for task execution profiling. The task does *LU* matrix factorization for a full matrix of size n -by- n . Three types of machines run tasks, and the machine configurations are listed below,

- Machine 1: Intel® Pentium® 4 CPU @ 3.00GHz with 1.00G of RAM
- Machine 2: Intel® XeonTM CPU @ 3.40GHz with 2.00G of RAM
- Machine 3: Intel® CoreTM 2 Duo CPU @ 2.66GHz with 3.25G of RAM

Each machine is scheduled to run 200 tasks with different settings specified by n ranging from 200-2000, while the execution time for each task setting is computed as the average over 30 runs. Figure 6.7 describes the relationship between task workload and average execution time for the three machines. It is clear that the execution time is exponentially increased as the workload increases, and Machine 3 outperforms the other two in terms of execution time.

Figure 6.7: Relationship between Workload and Execution Time

Satellite Image Classification Regression related applications have been largely considered in computing resource selection. However, the use of classification is generally ignored in the literature, due to the assumption that either a job is so generic that any machine can run it, or all the machines are capable in running any kind of job. It soon becomes a problem in the real-life cloud application where infrastructure resources are only competent in limited types of task, and thus a task needs to be classified before selecting a resource. Therefore, a sufficient understanding of which category a task belongs to is critical to the success of a provisioning service. In this case study, a case of task classification from real world is studied.

Imagine a scenario where a landsat satellite is collecting data for remote sensing. As a request, the data needs to be analyzed immediately, and several distributed machines are standing by. Assuming different machines are specialized in analyzing different types of soils, thus we need to dispatch an image to its suitable machine. Satellite image data from UCI [99] is used for testing the classification ability of the intrinsically motivated agents.

2000 data are exploited in this experiment with 36 input attributes, and 6 decision classes. For this case, the class output is not continuous, and falls into one of the 6 classes. Thus we preprocessed a class label i into a vector of length 6, which has a one in the i -th position and

zeros otherwise.

The profiles of three machines and the segmentation of images will be used to evaluate the effectiveness of individual and cooperative decision making for intrinsically motivated agents.

6.5.2 Agent Novelty Pursuit of an Accurate Prediction

In order to justify agent's choice of learning capability, a comparison is conducted between BP and ELM based prediction approaches. The comparison takes account of prediction accuracy and training time in terms of RMSE and seconds respectively. A BP network is built using 5 hidden neurons with sigmoid activation function and linear output neurons. The training algorithm of BP is implemented by Levenberg-Marquardt with default training parameters defined in Matlab. Throughout the experiments, an ELM network with 10 hidden neurons is adopted for predicting relative CPU performance from various hardware attributes. Input weights of ELM are uniformly randomly distributed on the interval $[-1, 1]$ and biases on $[0, 1]$. Sigmoid is selected as the activation function for hidden neurons, and Moore-Penrose general inverse is calculated using Singular Value Decomposition (SVD).

A snapshot of one trial is illustrated, in which Figure 6.8.a and Figure 6.8.b present the testing performance for Case 1 and Case 2 respectively. Solid line denotes the true CPU performance and dot line denotes the estimated CPU performance. From the small RMSE that ELM networks achieved, we can find that ELMs generalized well for both cases. Although only two attributes were available in Case 1, the performance was still satisfactory because machine cycle time and maximum main memory were two valuable features for predicting relative CPU performance. In addition, since Case 2 employed more attributes, a more accurate result was obtained with smaller RMSE.

Note that approaches such as rule-based agents cannot capture the relationship between machine attributes and CPU performance, and thus no prediction can be made on how a particular machine will perform. On the other hand, if learning is adopted, agent-mediated services can overcome this limitation with minimum human effort.

(a)

(b)

Figure 6.8: Neural Network Testing Performance for Case 1 and Case 2

The results for Case 1 and Case 2 are shown in Table 6.2. The table shows that both ELM and BP achieved comparable prediction accuracy with small RMSE. However, the training times for the two algorithms were quite different. For both cases, BP required more times than ELM in the order of 10^2 . It means the improvement of accuracy using BP cannot ade-

quately compensate for the increased training time. In Cloud computing, response time is an essential factor which could determine the success or the failure of a scheduling service. As a consequence, ELM is preferred by the agent-mediated service as the learning skill due to its acceptable prediction performance as well as its fast training speed.

Table 6.2: Comparison Between BP and ELM

	RMSE		Training Time (s)	
	Case 1	Case 2	Case 1	Case 2
ELM	0.06625	0.03384	0.00063	0.00156
BP	0.06358	0.03380	0.31000	0.41094

6.5.3 Agent Performance Pursuit of High Resource Utilization

In the simulation scenario, when a client request arrives, agent-mediated resource provisioning services under investigation independently choose a machine from a given set of candidate resources. After selection, the true values of the relative CPU performance for the selected machines of each approach are compared. The agent-mediated service with higher true relative CPU performance win against the rest. As shown in Section 6.3 rule-based policy with ranking heuristics ranks CPU performance by calculating sums of heuristically selected attributes. In the other hand, the learning-based policy predicts CPU performance based on a trained neural network (either BP or ELM) which estimates CPU performance from hardware attributes.

Figure 6.9 presents the three choices for resource provisioning agent in pursuit of performance. They are rule-based and learning-based (either BP or ELM) policies. Based on the agent's autonomy goal selection, the averaged performance in terms of resource utilization and time is used to select which goal to attend. From previous section, it is clear that ELM outperforms BP in the average return, for they have similar performance, but ELM uses much less time than BP. This section compares learning-based with rule-based choices in terms of resource utilization. The experiments are explained as follows.

Figure 6.9: Choices in Performance Pursuit for Resource Provisioning Agent

50 trials are repeated to obtain reliable results and for each trial since training is involved, we arbitrarily partition all resources into training and testing sets according to the ratio of 1:1. The training resources are used to train the neural networks and the testing resources are used for performance comparison. We assume that all resources pass the minimum requirements and are available for the client request. For each job we simply randomly select 5 machines as candidates from the testing resources. In one trial, there are totally 30 jobs requesting services, and we further assume that all client requests are in favor of a machine with higher relative CPU performance.

Performance is measured in terms of computing power utilization, which is calculated as the ratio of relative CPU performance of selected machine to that of the best machine among the candidates. Figure 6.10 compares scheduling performance between rule-based policy and learning-based policy for Case 1 and Figure 6.11 for Case 2. Rule-based method ranked machines by summing up the normalized hardware attributes. For Case 1 it used $Rank = CPU\ cycle\ time + maximum\ main\ memory$, and for Case 2 rank was calculated by summing up all the normalized attributes. The proposed resource allocation algorithm predicted performance values for each candidate, and selected the machine with the highest predicted performance. The figure shows that for both cases, the agent-mediated service enhanced by learning (dot line) outperformed the rule-based ranking heuristics (solid line) for all the trials.

Figure 6.10: Utilization Comparison for Case 1

In addition, the averaged utility are shown in Figure 6.12. For Case 1 rule-based agent service only utilized half of the optimal computing power, and thus the heuristically defined rank expression was ineffective. However, the learning-based approach achieved more than 90% utilization, since a neural network achieved a satisfactory prediction. For Case 2, the per-

Figure 6.11: Utilization Comparison for Case2

Figure 6.12: Average Utility Comparison

formance of conventional algorithm was increased greatly than Case 1, due to the change of rank expression. Therefore, we can draw a conclusion that ranking heuristics is very sensitive to heuristically selected rank expressions, and a good expression requires precise and sufficient prior knowledge. On the other hand, the proposed algorithm using ELM neural network is efficient even when only two attributes are available, and learning allows that no prior knowledge is required from users.

The price paid for using the additional learning algorithm is the time for training. However, due to the simplicity of ELM learning, the training time for the experiments was almost instantaneous, for which the worst case was 0.0313 seconds.

6.5.4 Agent Relatedness Pursuit of a Robust Decision

In this section, two kinds of requirements for execution time regression and category classification are examined, and in the experiments, we will compare the performances of both individual and cooperative learning of agent-mediated services with ELM as learning algorithm. We employ real-world data to evaluate the proposed agents for resource selection. The data explanation was given in Section 6.5.1.

For all experiments, we first randomly split the entire data in L and L_{TS} according to the ratio of 9:1 for training and testing. Note that L is used in the exact way as mentioned in Section

4.3.2 for individual learning and Section 4.5.2 for cooperative learning. L_{TS} is applied to test and compare different experiment settings, and therefore only out-of-sample test is adopted. Each experiment is repeated for 10 trials to obtain reliable results, and the average over all the trials is used for comparison.

We normalize all the input attributes (except expected outputs) into the range $[0, 1]$. Agents initialize the input weights of ELM uniformly randomly on the interval $[-1, 1]$, and biases on $[0, 1]$. Sigmoid is selected as the activation function, and Moore-Penrose general inverse is implemented using Single Value Decomposition (SVD). The results and analysis are presented in the rest of this section.

Each set of data collected from the profiles of the three machines is used in the experiment, and comparisons are conducted between single agent and multiple agents. The cooperative learning is committed among 5 identical agents, and implemented with both least squares and averaging estimators. Every agent, either a single one or a member of the group, share the same structure, and 3 hidden neurons of each ELM network are employed. The results in root mean square error (RMSE) and Pearson correlation are presented in Figure 6.13 and Figure 6.14 respectively.

From the results, it is clear that multiple agents with cooperative decisions provided smaller RMSE and higher correlation for all the machines. Meanwhile, averaging estimator outperformed the least squares estimator, and thus it implies that using averaging operation is not only cheaper, but also more efficient than the least squares in this case. It is worth noting that for execution time regression in terms of RMSE, proposed cooperative learning achieved more than 10% improvement over individual learning, and the largest improvement happened on Machine 3, where a 25% improvement was observed.

Same experiment settings are adopted as the regression experiment, except that 200 hidden neurons of each ELM network are used, and agents with same and different internal structures for cooperative learning are examined. Agents with different learning structures are grouped

Figure 6.13: Comparison of Single Agent and Multiple Agents for Task Execution Time Regression in RMSE

Figure 6.14: Comparison of Single Agent and Multiple Agents for Task Execution Time Regression in Pearson Correlation

Figure 6.15: Comparison of Single Agent and Multiple Agents for Task Classification

together, and their hidden neurons were set as 100, 160, 200, 240, and 300 respectively. The experiment results are shown in Figure 6.15, in which we can see that the cooperative learning consistently outperforms the individual one. Moreover, agents with different structures are superior to those with same structures though the same total number of neurons was used in both cases. Furthermore, averaging estimator is preferred over the least squares. Note that due to the size of the problem, BP learning algorithm will fail the timing requirement due to an unacceptable training time it required [92].

6.6 Summary

In this chapter, we first expressed the needs of exploiting learning agents for resource selection to handle dynamics and uncertainties for Hybrid Cloud. After an introduction of the hybrid environment, we proposed an intrinsically motivated agent with rule-based and learning-based selection policy for competency pursuit. For the pursuit of relatedness, a cooperative learning for a group of agents was proposed.

The experiments on resource utilization maximization highlighted the advantage of the proposed agent's adaptiveness. The experiments on task execution time regression and satellite image classification demonstrated the competency of the proposed agents in resource selection, and the experiment results showed that the cooperative learning outperformed the individual learning in terms of the reliable and noise-free decisions that agents obtained in the experiment.

Chapter 7

Conclusion and Recommendations

7.1 Conclusion

The development and maturity of internet put modern industry on the edge of transforming its IT infrastructure from traditional tightly-coupled to a flexible service-oriented fashion. Services are the current best practice for developing distributed software and integrating disparate components across the internet. Infrastructures based on Service-Oriented Architecture (SOA) is on the premise of allowing all hardware resources can be accessed and utilized over a network. In this context, each resource is considered as a service, and users who know the interface and location of the service can make use of it in a decoupled manner.

The thesis has the background set in the SOA advocacy for industry sector. However, the hype of infrastructure as a service did not make enterprises adopt Cloud computing widely. The thesis identified several gaps between contemporary technology status and expectation of enterprise, and particular focuses on the performance issue of resource allocation in a uncertain and dynamic environment.

Agent and multi-agent systems were proposed as the promising solution due to agent's adaptiveness and autonomous problem solving capability. To achieve system adaptiveness for real-life problems, the thesis aimed to propose a new type of agent that would demonstrate a great level of flexibility and adaptiveness in an open and dynamic environment. The thesis identified three main challenges, including how the agent would deal with environmental

changes that was not able to be anticipated during design time; how the agent would make reliable and noise-free decisions out of information that was not uniformly distributed; and how the agent would efficiently and systematically manage a diversity of task-oriented behaviors.

Research in motivated agents has been extensively investigating agent adaptiveness. However, existing motivated agents lack the support of one or more challenges. Therefore, this thesis proposed a novel formalization of intrinsically motivated agent, namely **Competitive Intrinsically Motivated Agent** and discussed thoroughly from design to implementation. We show from the case studies that the proposed approach of **Competitive Intrinsically Motivated Agents for Resource Allocation** enables an allocation service with enhanced proactiveness and improves performance in terms of the degree of adaptiveness that the service achieves.

A summary of contributions of this research is given as follows,

Chapter 3 modeled the proposed Competitive Intrinsically Motivated Agent with a conceptual formalization. The formal model of the proposed agent possessed the essence of Self-Determination Theory (SDT), considering entity's motivations, and the trinity of intrinsic needs. The agent supported modeling relationships among multiple facets of an entity, comprising of motivations, intrinsic needs, internal states, external events and agent behaviors. In particular, agent's learning behavior demonstrated the pursuit of novelty on learning new knowledge; agent's accomplishment of routine jobs showed the pursuit of performance; agent's social interactions, such as cooperation and coordination, exposed the pursuit for relatedness; and finally agent's decisiveness on its own behaviors exhibited the pursuit of autonomy.

To make competitive agent competent for complex situations, GoalNet was leveraged to represent the proposed agent. In this way, multiple behaviors were organized in an integrated and systematic manner. The GoalNet representation decomposed any complex task in a top-down manner, and managed the hierarchy of various task levels comprehensively. Later study mapped the conceptual modules and functions of a motivated agent to the GoalNet components.

In Chapter 4, an agent architecture was introduced, together with the design of agent's main task. Several sub-tasks as the pursuit of agent needs were then introduced with a GoalNet

representation, a detailed system design and an example of implementation of agent behaviors. The numerical results for each simulation offered a better understanding on the advantage of the proposed agent. The examples of agent implementation ranged from supervised, unsupervised to reinforcement learning, in which the research showed the strength in supporting a number of categories of adaptive intelligent agents. Finally, the Multi-Agent Development Environment (MADE) was used to implement the agent running platform, which completed the cycle of motivated agent-based methodology from design to implementation.

Chapter 5 and Chapter 6 explored two case studies in the domain of resource allocation. The proposed agents were applied for the both sides of a business transaction, from the view of a service provider and the view of enterprise consumers. In Chapter 5 service providers of Large-Scale Collaborative Virtual Environment adopted motivated agents to achieve load balancing across their entire business; and in Chapter 6, enterprise consumers leveraged motivated agents to manage a hybrid Cloud infrastructure with heterogeneous resources. In both real-life cases, Extreme Learning Machine (ELM) was recommended as agent's learning algorithm due to its fast training nature, by comparing with other learning algorithms. Besides, cooperative decisions made by a group of agents outperformed a single agent. The proposed agents have been successfully employed, and exhibited a level of adaptiveness and flexibility. The evaluation results demonstrated the benefits gained by agent-mediated infrastructure services through the inherited agent's adaptiveness.

7.2 Recommendations for Further Research

The ultimate adaptiveness of an autonomous agent is the power of crafting one's own knowledge with minimal designer effort. Current research in this thesis assumes the basic design of knowledge, in the form of tasks and sub-tasks, is given by the designer, and agent's adaptiveness is constrained by the intended frame, i.e. agent can substitute parts of actions or sub-goals within the skeleton frame, instead of acquiring something novel outside the box. The need

for autonomy can also be pursued by crafting agent's own knowledge based on the unique experience obtained from the interaction with environment.

According to SDT, competence and autonomy are not separable. Agent is doing effectively on one task, but this task must be chosen from its own decision (i.e. autonomy). Autonomy must show effectiveness in dealing with task (competence), otherwise, autonomy is not recognized internally. Agent is able to create and update a knowledge GoalNet with available building blocks, and at the same time the newly formed GoalNet is evaluated against agent's effectiveness on achieving agent's interest. Only the GoalNet promotes the average effectiveness, proving its satisfaction on competence needs, will be kept and stored in the knowledge base.

There are two aspects of crafting knowledge, namely knowledge discovery and knowledge evaluation. The recommended future work for an motivated agent with extended adaptiveness is discussed in the following sections. Other than need pursuit of a single motivated agent, types of collaborations among multiple agents are also discussed, and the recommended work features a diversity of collaborations for agent's pursuit of autonomy.

7.2.1 Autonomous Knowledge Discovery for Competency Pursuit

Agent is designed to satisfy its designed objectives, for which it is essential to have a set of knowledge to execute its tasks. The knowledge should be elaborate enough such that agent has explicit directions for its next move, as well as generic enough such that agent can have choices to dynamically adjust its behaviors according to contemporary situation. In this thesis, agent knowledge is designed in the form of GoalNet, and one GoalNet allows dynamical binding of behaviors. The binding of behaviors is fully controlled by the agent, and the flexibility exhibits a degree of autonomy.

The discovery of agent's own knowledge is the most sophisticated feature of autonomy. Agent perceives knowledge workflows as a composition of individual building blocks. Organi-

```

input : A root goal  $root$  specified with  $S_{initial}$  and  $S_{end}$ , a set of  $\mathbf{S}$  and  $\mathbf{Tr}$ 
output: A discovered GoalNet  $G$ 

1 Initialize:  $G \leftarrow \{S_{initial}\}$ ;
2 while  $leaf(G) \neq S_{end}$  and  $length(G) \leq \theta$  do
3   if  $leaf(G) \in \mathbf{S}$  then
4     find  $tr \in \mathbf{Tr}$ , such that  $tr.A \in action(leaf(G))$ ;
5      $G.append(tr)$ ;
6   else /*  $leaf(G) \in \mathbf{Tr}$  */
7     find  $s \in \mathbf{S}$ , such that  $s.S_{pre} = leaf(G).S_{post}$ ;
8      $G.append(s)$ ;
9   end
10 end

```

Figure 7.1: Pseudocode for Discovering Knowledge in the Form of GoalNet

ization and re-organization of this building blocks are considered as creating novel knowledge and updating existing knowledge.

The discovery of knowledge of a motivated agent is modeled as formation of GoalNets. In light of the root goal to be achieved, an agent deliberative process is required to search through a set of known states $\mathbf{S}$, followed by connecting one state $s \in \mathbf{S}$ with another via valid transition $tr \in \mathbf{Tr}$. The overall sub-GoalNet validity and effectiveness can be accessed once the new GoalNet is formed. The formation involves a trial-and-error process, and is possibly speeded up by prior knowledge.

A pseudocode for autonomous knowledge discovery is shown in Figure 7.1. Agent is initialized with a starting state S_{start} , an ending state S_{end} and an available pool of goal states $\mathbf{S}$ and transitions $\mathbf{Tr}$. $leaf(G)$ denotes the dangling element of the GoalNet G , either a goal state $s \in \mathbf{S}$ or a transition $tr \in \mathbf{Tr}$. We incrementally construct a GoalNet G by appending an element (either s or tr) to the end of G , via invoking $G.append(s)$ or $G.append(tr)$. The function of **find** is basically a search of valid elements through semantic checking. A threshold θ is implemented to prevent infinite loops.

Instead of discovery knowledge over entire primitive action space, it is efficient to use

Figure 7.2: Illustration of GoalNet Abstraction

abstraction out of common usage patterns in a relative small subspace. During the lifetime of an agent, there may be multiple sub-GoalNets serving a common goal. Abstraction is a nature way to express a category of GoalNets that can fulfill the parent goal. Abstraction enables an more efficient search, and allows to have building blocks for more complex plans. The abstraction of plans is realized by GoalNet abstraction. The input is a pool of GoalNets acquired by agent, and the output is a parent goal, representing a set of sub-GoalNets. The process can be described as follows. Given a set of GoalNets G , we collect and order the primitive goals **Atom**, ignoring the in-between transitions. We first mark all the primitive goals that are common $\text{Atom}^{\text{Common}}$ to a subset of $G' \subset G$, which have at least two GoalNet members $|G'| \geq 2$. Incrementally, we build a sequence of primitives in $\text{Atom}^{\text{Common}}$, $seq_i = \{atom_k, atom_{k+1}, \dots\}$ where $seq_i \in Seq$ such that seq_i is common across G' . By creating $root_i$ to represent $seq_i \quad \forall i$, we can construct a set of abstracted GoalNets. The original GoalNets in G' is re-organized by substituting the created subgoal $root_i$.

Figure 7.2 illustrates the general idea of how abstraction works for efficient knowledge discovery. **G1** and **G2** contain a common sequence of goals, S_1, S_2, S_3 , and the aim of abstraction is to obtain sub-GoalNet **G3**. **G3** is thus a representative parent goal for sequence of S_1, S_2, S_3 .

7.2.2 Autonomous Knowledge Evaluation for Autonomy Pursuit

The current progress only illustrated there was a difference between agents with and without intrinsic motivations, and how an agent with intrinsic motivations could organize its behaviors from its innate and life-long values as the pursuit of autonomy. It did not explain how optimal the solution could be from the proposed approach. This is related to the discussion of agent's optimal autonomy pursuit, which tells the agent when and which action to take. Although the thesis has utilized rule-based and hierarchical reinforcement learning (as in Figure 4.18) as possible implementations for autonomy pursuit, the thesis currently did not discuss how to efficiently deliberate a plan, or how a plan can be optimal. The optimality issue was left out and discussed as part of the recommendations in this session. Nevertheless, the key contribution of the thesis is to provide a framework with a set of models that could make agent's autonomy pursuit happen. The planning efficiency and optimality can be achievable by incorporating with advanced hierarchical reinforcement learning or automated planning.

The current approach as in Figure 4.18 for agent autonomy pursuit is quite simple. The idea is simply based on the average performance of a choice, either a goal or an action. There is no discussion on the optimality the agent would achieve but rather a heuristics. The future work will emphasize utilizing techniques in Hierarchical Reinforcement Learning (HRL) to optimize agent's goal/action selection. Moreover, current motivated agents still lack an online feature which would enable agents with continues improvement through interactions with the environment. Therefore, incorporating HRL with motivated agents is important in the future work. The recommended future work of how to exploit HRL in knowledge evaluation as agent autonomy pursuit is given as follows.

Researchers in reinforcement learning (RL) realized that RL methods were seriously be-deviled by the curse of dimensionality, where the number of to-be-learnt parameters grows exponentially with the compact encoding size of a state. Theoretically speaking, the complexity of RL is estimated as follows: (1) each iteration is polynomial in the number of states $|S|$;

(2) the number of iterations to obtain an optimal policy is polynomial in $|S|$; (3) therefore the overall complexity is polynomial in $|S|$. However, for practical problems $|S|$ is exponential in the length of the feature vector of the problem domain.

We use an example to illustrate this problem of curse of dimensionality. A delivery robot is traveling through a floor of Fusionoplise to deliver a mail package. The Fusionoplise has 120,000 square meters and 24 floors. One floor is estimated as $120,000/24 = 5,000$ square meters. Assume the sensor of the robot is able to cover one square meters effectively, then the state space would be $2 * 2 * 5,000 * 5,000$, which is very large. A natural question is to ask “how do humans do the decision making”? Humans thinks in modularity, e.g. “go out of the room”, “walk in the hallway” and “go in the room”. Humans are planning in modularity at different granular levels of thinking. Getting out of a room is no difference to getting out of any room; navigation do not care whether the robot has the mail or not. Control knowledge can be put into robot, like “first pick up a mail package, then deliver it”; “navigation considers rooms and hallway, separately”, etc.

Hierarchical reinforcement learning (HRL) is motivated by the modular approach to combat the curse of dimensionality. **Temporal Abstraction** is exploited, by which decision making is not required at every step, but until termination the temporally-extended activities are executed in the way of following their own policies [143]. The partial policies are considered as temporally-extended actions, skills, options, behaviors, or modes. The policies are naturally fit into our sub-GoalNet. Within one sub-GoalNet the agent has its own policy to achieve the parent goal. The recommended future work is particularly interested in two pioneer work, namely Sutton’s Options and Dietterich’s MaxQ.

Sutton [144] formalized the hierarchical approach and defined the notion of an option to describe the temporally-extended activities in RL. An option is a well defined behavior with a set of states in which an option can be initiated, a policy when the execution of the option follows, a probability of state that this option terminates. The learning replaces the set of

primitive actions with a set of options, and the learning over options follows a Semi-Markov Decision Process (SMDP) Q-learning approach. Dietterich [145] developed an approach in HRL called the MaxQ value function decomposition. MaxQ does not directly reduce the entire original problem to a single complex SMDP, but a hierarchy of smaller SMDPs are created, of which the solutions can be obtained simultaneously. The MaxQ approach initially decomposes a core MDP into a set of sub-tasks. The sub-tasks form a hierarchy structure as a task graph. The decomposition augments the state by adding a sub-task, and defines a reward received in the sub-task after it finishes. The reward of a parent task is accumulated as the number of sub-tasks finishes.

From above descriptions, it is obvious that current goal-oriented representation of the intrinsically motivated agent is similar in terms of the option modeling and task decomposition. GoalNet provides a design tool for HRL to input control knowledge into the agent, and HRL offers a powerful tool for the agent to evaluate a choice of a goal/action effectively. We see the combination of goal-oriented representation and HRL will synthesize a powerful chemical for motivated agents to gain great autonomy in their need pursuit.

7.2.3 A Diversity of Agent Collaborations

The primary idea of the proposed cooperative reasoning in Chapter 4.5.2 is to demonstrate one of the advantages of multiple agents over a single agent. However, the form of agent collaborations is of polymorphism, and in this section we are interested in knowledge sharing and delegation among multiple agents.

Knowledge Sharing aims to provide a mechanism for sharing inductive rules so as to avoid the repeated training cycle for producing similar knowledge. To succeed in knowledge sharing between multiple agents, expert agents discover explicit rules or implicit knowledge from training, and present their knowledge in a certain format, commonly understood by different

agents before sharing. As for novice, agents should take in understandable external knowledge, absorb it to fit its learning structure, and tune to adapt in its own local settings. Above learning characters of both teachers and prentices are essential to the success of cooperation for knowledge sharing.

Knowledge should be extracted before passed. There are two forms of neural network extracted knowledge, either implicit “black box” or explicit “rule base”. We recommend IF-THEN rules as knowledge presentation, because there are well studied methods for neural networks to extract rules in this format from numeric data. Besides, rules are easy to pass among agents, and it also accords with human’s most common knowledge passing behavior when a teacher is lecturing a student. Analogue to human’s trainer-trainee interactions, declarative knowledge is employed in the knowledge passing process. This is due to the reason that this high level knowledge is explicit and highly compact. In addition, as stated in [146], declarative knowledge can speed up the learning process during an online skill learning and can also facilitate transfer of skills by accelerating training in a new setting. Moreover, it is well known in AI to use explicit knowledge for guiding agent learning. Examples can be seen in related work [147], [148], [149] which incorporates neural networks with prior explicit knowledge that is naturally expressed in a logic rule form. These connectionist models are later trained and tuned according to feedbacks from environment based on learning mechanisms like Q-learning, backpropagation, and genetic algorithms. However, previous models are heavily relied on the prior knowledge. In our proposal, an automation of rule discovery is desired for extracting knowledge from trained structures in a “bottom-up” manner.

Absorbing passed knowledge into existing learning structure is another challenging issue. The challenge lies in the needs to add redundant and yet appropriate numerical information to allow neural network to be executive. This sort of conversion from abstractness to concreteness requires a deep study. One example can be seen in [150] which introduced a methodology utilizing Linear Programming to insert symbolic rule sets into a target tree. A transformation

was done on symbolic rule sets into node weights and biases. Our proposed knowledge adaption comes from the idea that explicit rules are usually associated with some tunable linguistic variables, like the examples in fuzzy neural networks. These variables are determined based on its actual environment, which gives the learning structure flexibility to act according to uncertainty and dynamics. Furthermore, it is commonly noticeable that changing on parameters (weights) related to linguistic variables has same effect to the changing on the rule weights. This may suggest the effect of weight adaption in the linguistic layer can be propagated to the rule base, and then yields another set of rules. In this way, a newly acquired knowledge may be extractable, and consequently knowledge adapts itself into a more accurate one. Nevertheless, a deep research should be carried out to evaluate this idea. It is further noticed that received knowledge may not instruct the beneficiary to perform exactly what the teacher acts, and what have been taught thus should act as a guideline instead of rules to be obeyed. It is desirable for a knowledge receiver to evaluate, improvement over knowledge base constantly from feedbacks, and behave dynamically as the environment expects.

Involving communication between agents to facilitate agent performance has been studied in [151], [152], [153], [150]. Some are not based on knowledge sharing, but rather a form of supervised learning model, while some are communicative with knowledge, but miss explicit rules or do not have a flexible approach for different learning algorithms. Besides, none of them come out with a comprehensive agent model to incorporate such communicative learning.

Delegation is an enabling of task transfer from one to another in the multiple agent system. Delegation can happen to either an agent or a human. Delegation is needed when an agent is incompetent or not confident with a certain tasks. For example in the context of Cloud services, at a moment an agent-mediated service could receive numerous requests, and it is incapable to handle all of them before deadline. The service agent would outsource a subset of tasks to other agents, who are less loaded and willing to accept assignment. One obvious benefit of

delegation is to maintain a high level of throughput within a system, and thus more clients who are using the system could have a high level of confidence and satisfaction.

Intrinsically motivated agents elicit human-like behaviors driven by motivation. Since the agent's motivation is derived from a psychological background, this type of agents would be comfortably accepted in the process of human-computer-interaction (HCI). The pursuit of relatedness is substantially a urge of being cared, connected to and interact. Agent-mediated interaction with human rises when the agent requires human input or preference in doing its own tasks. Delegation or Keeping to one's own needs a careful tradeoff, when involving humans. Inconsiderate delegation would result in too much human intervention or too many management efforts, such that human would feel troublesome and simply get ride of the system. Motivated agents contemplate the relations between autonomy and relatedness, and could enable a well balanced interaction with humans. The future recommendation of this research work is to exploit the potential of intrinsically motivated agents in applications with HCI requirements. The intended arena is related to Serious Games, in which a project is ongoing for crafting an agent-augmented learning companion in a virtual learning environment.

Author's Publications

Conference

1. **G. Zhao**, Ailiya, and Z. Shen, "Intrinsically Motivated Agent for Service Management in Smart Homes," in the 2nd Southeast Asian Network of Ergonomics Societies (SEANES 2012), Langkawi Malaysia, 2012.
2. **G. Zhao**, Z. Shen, and C. Miao, "ELM-Based Intelligent Resource Selection for Grid Scheduling," in the 8th International Conference on Machine Learning and Applications (ICMLA 2009), Miami, 2009, pp. 398-403.
3. **G. Zhao**, Z. Shen, and C. Miao, "A Fast and Intelligent Resource Allocation Service for Service-Oriented Grid," in the 15th International Conference on Parallel and Distributed Systems (ICPADS 2009), Shenzhen, 2009, pp. 804-809.
4. **G. Zhao**, Z. Shen, C. Miao, and Z. Man, "On Improving the Conditioning of Extreme Learning Machine: A Linear Case," in the 7th International Conference on Information, Communications and Signal Processing (ICICS 2009), Macau, 2009, pp. 1-5.
5. **G. Zhao**, Z. Shen, Ailiya, and C. Miao, "ELM-Based Agents for Grid Resource Selection," in IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT 2008), Sydney, 2008, pp. 385-388.
6. **G. Zhao**, Z. Shen, C. Miao, and R. Gay, "Enhanced Extreme Learning Machine with Stacked Generalization," in IEEE International Joint Conference on Neural Networks

(IJCNN 2008), part of the IEEE World Congress on Computational Intelligence (WCCI 2008), Hong Kong, 2008, pp. 1191-1198.

7. C. Chen, S. Yan, **G. Zhao**, B.S. Lee, and S. Singhal, "A Systematic Framework Enabling Automatic Conflict Detection and Explanation in Cloud Service Selection for Enterprises," in 5th IEEE International Conference on Cloud Computing (IEEE CLOUD 2012), Honolulu, 2012, pp. 883-890.
8. S. Yan, C. Chen, **G. Zhao**, and B.S. Lee, "Cloud Service Recommendation and Selection for Enterprises," in the 6th International DMTF Academic Alliance Workshop on Systems and Virtualization Management: Standards and the Cloud (SVM12), Las Vegas, 2012.
9. Q. Liang, F. Gioachin, **G. Zhao**, and S.S. Chang, "A Framework of Scaling for Inter-tenant Collaborative Systems," in the Workshop on Emerging Issues in Cloud Computing, Co-located with the 4th IEEE International Conference on Cloud Computing Technology and Science (CLOUDCOM 2012), Taipei Taiwan, 2012.
10. S. Yan, B.S. Lee, **G. Zhao**, D. Ma, and P. Mohammed, "Infrastructure Management of Hybrid Cloud for Enterprise Users," in Systems and Virtualization Management: Standards and the Cloud (SVM11), Paris France, 2011, pp. 1-6.
11. B.S. Lee, S. Yan, D. Ma, and **G. Zhao**, "Aggregating IaaS Service," in the Annual SRII Global Conference 2011, San Jose, CA, 2011, pp. 335-338.
12. H. Song, Z. Shen, C. Miao, A.H. Tan, and **G. Zhao**, "The Multi-Agent Data Collection in HLA-based Simulation System," in the 21st International Workshop on Principles of Advanced and Distributed Simulation (PADS 2007), San Diego, 2007, pp. 61-69.

13. Z. Shen, H. Shen, C. Miao, Z. Yang, R. GAY, and **G. Zhao**, “Collaborative Workspace over Service-Oriented Grid,” in the 3rd International Workshop on Grid Computing and Applications (GCA 2007), Singapore, 2007.

Journal

1. **G. Zhao**, Ailiya, and Z. Shen, “Learning-by-Teaching: Designing Teachable Agents with Intrinsic Motivation,” *Educational Technology & Society*, vol. 15(4), 2012, pp. 62-74.
2. **G. Zhao**, Z. Shen, and Z. Man, “Robust Input Weight Selection for Well-Conditioned Extreme Learning Machine,” *International Journal of Information Technology*, vol. 17(1), 2011.
3. **G. Zhao**, and Z. Shen, “Supply Chain Platform as a Service: a Cloud Perspective on Business Collaboration,” *International Journal of Information Technology*, vol. 16(3), 2010.
4. **G. Zhao**, Z. Shen, C. Miao, and G. Bi, “QoS-Driven Load Balancing Agents for Cloud-based Collaborative Virtual Environments,” *IEEE Transactions on Industrial Informatics*, *submitted*, 2011.
5. **G. Zhao**, Z. Shen, C. Miao, and G. Bi, “Agile Reinforcement Learning with Extreme Learning Machine,” *Neurocomputing*, *submitted*, 2012.
6. **G. Zhao**, Z. Shen, C. Miao, and G. Bi, “A Survey of Game Platforms in Cloud Computing,” *ACM Computers in Entertainment*, *submitted*, 2012.
7. **G. Zhao**, Z. Shen, C. Miao, and G. Bi, “Intrinsically Motivated Agents as Learning Companions in Educational Games,” *Computer Animation and Virtual Worlds*, *submitted*, 2012.

8. **G. Zhao**, “HP Labs Singapore: Fusionopolis Singapore,” XRDS: Crossroads, The ACM Magazine for Students, vol. 18, pp. 38-39, 2011.

References

- [1] S. Franklin and A. Graesser, “Is it an agent, or just a program?: a taxonomy for autonomous agents,” *Intelligent Agents III: Agent Theories, Architectures, and Languages: ECAI’96 Workshop (ATAL), Budapest, Hungary, August 12-13, 1996: Proceedings*, 1997.
- [2] J. Holland, “Complex adaptive systems,” *A new era in computation*, pp. 17–30, 1993.
- [3] I. Foster, N. Jennings, and C. Kesselman, “Brain meets brawn: why grid and agents need each other,” *Towards the Learning Grid: advances in human learning services*, 2005.
- [4] R. M. Ryan and E. L. Deci, “Intrinsic and extrinsic motivations: Classic definitions and new directions,” *Contemporary educational psychology*, vol. 25, no. 1, pp. 54–67, 2000.
- [5] S. Singh, A. G. Barto, and N. Chentanez, “Intrinsically motivated reinforcement learning,” in *in Proc. Adv. Neural Inform. Process. Syst. 17 Conf.* Cambridge, MA: Citeseer, 2005.
- [6] K. E. Merrick, “Modelling motivation for experience-based attention focus in reinforcement learning,” Ph.D. dissertation, The University of Sydney, 2007.
- [7] A. G. Barto, S. Singh, and N. Chentanez, “Intrinsically motivated learning of hierarchical collections of skills,” in *Proceedings of the 3rd international conference on development and learning (ICDL 2004), Salk Institute, San Diego.* Citeseer, 2004.
- [8] S. Singh, R. L. Lewis, and A. G. Barto, “Where do rewards come from?” in *Proceedings of the Annual Conference of the Cognitive Science Society.* Citeseer, 2009, pp. 2601–2606.

- [9] M. Lopes and P. Y. Oudeyer, “Guest editorial active learning and intrinsically motivated exploration in robots: Advances and challenges,” *Autonomous Mental Development, IEEE Transactions on*, vol. 2, no. 2, pp. 65–69, 2010.
- [10] S. Singh, R. L. Lewis, A. G. Barto, and J. Sorg, “Intrinsically motivated reinforcement learning: An evolutionary perspective,” *Autonomous Mental Development, IEEE Transactions on*, vol. 2, no. 2, pp. 70–82, 2010.
- [11] K. Merrick, “A computational model of achievement motivation for artificial agents,” in *The Tenth International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, Taiwan, 2011, international Conference on Autonomous Agents and Multi-Agent Systems (under review).
- [12] K. Merrick and M. L. Maher, *Motivated Reinforcement Learning: Curious Characters for Multiuser Games*. Springer-Verlag, Berlin/Heidelberg, 2009.
- [13] P. Y. Oudeyer and F. Kaplan, “What is intrinsic motivation? a typology of computational approaches,” *Frontiers in Neurorobotics*, vol. 1, 2007.
- [14] P. Y. Oudeyer, F. Kaplan, and V. V. Hafner, “Intrinsic motivation systems for autonomous mental development,” *Evolutionary Computation, IEEE Transactions on*, vol. 11, no. 2, pp. 265–286, 2007.
- [15] K. Shafi and K. Merrick, “A curious agent for network anomaly detection,” in *The Tenth International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, Taiwan, 2011.
- [16] D. B. Lenat, “Am: An artificial intelligence approach to discovery in mathematics as heuristic search,” Stanford Univ Ca Dept of Computer Science, Tech. Rep., 1976.
- [17] J. Schmidhuber, “Adaptive confidence and adaptive curiosity,” in *Institut fur Informatik, Technische Universitat Munchen, Arcisstr. 21, 800 Munchen 2*. Citeseer, 1991.
- [18] —, “A possibility for implementing curiosity and boredom in model-building neural controllers,” *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior (SAB90)*, 1991.

- [19] —, “What’s interesting?” *Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale*, 1997.
- [20] —, “Artificial curiosity based on discovering novel algorithmic predictability through coevolution,” in *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, vol. 3. IEEE, 1999.
- [21] A. Sloman and M. Croucher, “Why robots will have emotions,” in *Proc 7th Int. Joint Conf. on AI*. Citeseer, 1981.
- [22] X. Huang and J. Weng, “Novelty and reinforcement learning in the value system of developmental robots,” *Lund University Cognitive Studies*, 2002.
- [23] F. Kaplan and P. Oudeyer, “Motivational principles for visual know-how development,” *Lund University Cognitive Studies*, 2003.
- [24] R. Saunders and J. S. Gero, “Artificial creativity: A synthetic approach to the study of creative behaviour,” *Computational and Cognitive Models of Creative Design V, Key Centre of Design Computing and Cognition, University of Sydney, Sydney*, pp. 113–139, 2001.
- [25] A. Baranes and P. Y. Oudeyer, “R-iac: Robust intrinsically motivated exploration and active learning,” *Autonomous Mental Development, IEEE Transactions on*, vol. 1, no. 3, pp. 155–169, 2009.
- [26] P. Y. Oudeyer and F. Kaplan, “How can we define intrinsic motivation?” 2008.
- [27] K. Merrick and K. Shafi, “Agent models for self-motivated home assistant bots,” 2009, pp. 131–150, proceedings of the International Symposium on Computational Models for Life Sciences.
- [28] K. E. Merrick, “A comparative study of value systems for self-motivated exploration and learning by robots,” *Autonomous Mental Development, IEEE Transactions on*, vol. 2, no. 2, pp. 119–131, 2010.
- [29] K. E. Merrick and K. Shafi, “Achievement, affiliation, and power: Motive profiles for artificial agents,” *Adaptive Behavior*, vol. 19, no. 1, p. 40, 2011.

- [30] E. L. Deci and R. M. Ryan, "The "what" and "why" of goal pursuits: Human needs and the self-determination of behavior," *Psychological inquiry*, vol. 11, no. 4, pp. 227–268, 2000.
- [31] M. W. Passer and R. E. Smith, *Psychology: The science of mind and behavior*, 5th ed. McGraw-Hill Higher Education, 2011.
- [32] R. M. Ryan and E. L. Deci, "Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being," *American psychologist*, vol. 55, no. 1, pp. 68–78, 2000.
- [33] J. W. Atkinson, "An introduction to motivation," 1964.
- [34] W. Weiten and D. McCann, *Concept charts for study and review for Psychology, themes and variations*. Thomas Nelson, 2006.
- [35] W. B. Cannon, "The wisdom of the body," *The American Journal of the Medical Sciences*, vol. 184, no. 6, p. 864, 1932.
- [36] C. L. Hull, "Principles of behavior: an introduction to behavior theory," 1943.
- [37] J. A. Gray, "The neuropsychology of temperament," 1991.
- [38] R. W. White, "Motivation reconsidered: The concept of competence," *Psychological review*, vol. 66, no. 5, p. 297, 1959.
- [39] A. H. Maslow, R. Frager, and J. Fadiman, "Motivation and personality," 1970.
- [40] E. L. Deci and R. M. Ryan, *Intrinsic motivation and self-determination in human behavior*. Springer, 1985.
- [41] T. Besley and M. Ghatak, "Competition and incentives with motivated agents," *LSE STICERD Research Paper No. TE465*, 2003.
- [42] Condor. [Online]. Available: <http://www.cs.wisc.edu/condor>.

- [43] Scheduling to overcome the multi-core memory bandwidth bottleneck. [Online]. Available: http://www.platform.com/NR/rdonlyres/339F8A01-3660-44E1-8714-381A99E664AE/0/multicore_scheduling_whitepaper.pdf.
- [44] PSB Pro. [Online]. Available: http://www.hipecc.wichita.edu/PBS/PBSProUG_5_4_0.pdf.
- [45] F. Berman and R. Wolski, "The AppLeS project: a status report," *8th NEC Research Symposium*, 1997.
- [46] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crumme *et al.*, "The GrADS Project: software support for high-level grid application development," *International Journal of High Performance Computing Applications*, vol. 15, no. 4, p. 327, 2001.
- [47] A. Eswaradass, X. Sun, and M. Wu, "A neural network based predictive mechanism for available bandwidth," *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pp. 33a–33a, 2005.
- [48] Nimrod/G. [Online]. Available: <http://www.csse.monash.edu.au/~davida/nimrod/nimrodg.htm>.
- [49] A. Panagakis, A. Litke, A. Doulamis, N. Doulamis, T. Varvarigou, and E. Varvarigos, "An advanced grid architecture for a commercial grid infrastructure," *The 2nd European Across Grids Conference, Nicosia, Cyprus, January, 2004*.
- [50] N. Doulamis, A. Doulamis, K. Dolkas, A. Panagakis, T. Varvarigou, and E. Varvarigos, "Non-linear prediction of rendering workload for grid infrastructure," *International Conference on Computer Vision and Graphics*, pp. 25–28.
- [51] N. Doulamis, A. Doulamis, A. Panagakis, K. Dolkas, T. Varvarigou, and E. Varvarigos, "Workload prediction of rendering algorithms in grid computing," *European Multigrid Conference*, pp. 7–12, 2002.
- [52] —, "A combined fuzzy-neural network model for non-linear prediction of 3-D rendering workload in Grid computing," *Systems, Man and Cybernetics, Part B, IEEE Transactions on*, vol. 34, no. 2, pp. 1235–1247, 2004.

- [53] J. Yuan, S. Ding, and C. Wang, "Tasks scheduling based on neural networks in grid," *Natural Computation, 2007. ICNC 2007. Third International Conference on*, vol. 3, 2007.
- [54] Nlanr measurement and netowrk analysis nlanr pma: Special traces archive. [Online]. Available: <http://pma.nlanr.net/Special/>.
- [55] X. Sun and M. Wu, "GHS: a performance system of grid computing," *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pp. 228a–228a, 2005.
- [56] R. Rahman, K. Barker, and R. Alhajj, "A predictive technique for replica selection in grid environment," *Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium on*, pp. 163–170, 2007.
- [57] J. Huang, H. Jin, X. Xie, and Q. Zhang, "Using NARX neural network nased load Pprediction to improve scheduling decision in grid environments," *Natural Computation, 2007. ICNC 2007. Volume V. Third International Conference on*, vol. 5, 2007.
- [58] F. Leu, J. Lin, M. Li, and C. Yang, "A performance-based grid intrusion detection system," *Computer Software and Applications Conference, 2005. COMPSAC 2005. 29th Annual International*, vol. 1, 2005.
- [59] F. Leu, J. Lin, M. Li, C. Yang, and P. Shih, "Integrating grid with intrusion detection," *Advanced Information Networking and Applications, 2005. AINA 2005. 19th International Conference on*, vol. 1, 2005.
- [60] S. Cho, "Incorporating soft computing techniques into a probabilistic intrusion detection system," *Systems, Man and Cybernetics, Part C, IEEE Transactions on*, vol. 32, no. 2, pp. 154–160, 2002.
- [61] F. Mora, F. Macia, J. Garcia, and H. Ramos, "Intrusion detection system based on growing grid neural network," *Electrotechnical Conference, 2006. MELECON 2006. IEEE Mediterranean*, pp. 839–842, 2006.

- [62] I. Foster and C. Kesselman, *The grid: blueprint for a new computing infrastructure*. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 1998.
- [63] H. Chen, J. Jiang, and B. Zhang, “Design of an artificial-neural-network-based application-oriented grid resource discovery service,” *Machine Learning and Cybernetics, 2004. Proceedings of 2004 International Conference on*, vol. 4, 2004.
- [64] N. Kapadia, J. A. Fortes, and C. E. Brodley, “Predictive application-performance modeling in a computational grid environment,” *Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing HPDC*, vol. 99, p. 4754.
- [65] O. Yaik, C. Yong, and F. Haron, “Time series prediction using adaptive association rules,” *Distributed Frameworks for Multimedia Applications, 2005. DFMA’05. First International Conference on*, pp. 310–314, 2005.
- [66] R. Liu, W. Zheng, and Y. Wu, “Intelligent decision making for agreement-based grid resource management,” *Proceedings of the First International Multi-Symposiums on Computer and Computational Sciences-Volume 1 (IMSCCS’06)-Volume 01*, pp. 714–719, 2006.
- [67] L. Lin, Y. Zhang, and J. Huai, “Sustaining incentive in grid resource allocation: a reinforcement learning approach,” *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*, pp. 145–154, 2007.
- [68] C. Lee and J. Suzuki, “Biologically-inspired design of autonomous and adaptive grid services,” *Proceedings of the International Conference on Autonomic and Autonomous Systems*, 2006.
- [69] C. Lee, H. Wada, and J. Suzuki, “Exploring biologically-inspired evolvable network applications with the BEYOND architecture,” *Bio-Inspired Models of Network, Information and Computing Systems, 2006. Ist*, pp. 1–8, 2006.
- [70] C. Lee and J. Suzuki, “Autonomic network applications designed after immunological self-regulatory adaptation,” *Integration of Knowledge Intensive Multi-Agent Systems, 2007. KIMAS 2007. International Conference on*, pp. 222–229, 2007.

- [71] G. Zhang and J. Sun, "Grid intrusion detection based on soft computing by modeling real-user's normal behaviors," *Granular Computing, 2006 IEEE International Conference on*, pp. 558–561, 2006.
- [72] M. Beigi, S. Calo, and D. Verma, "Policy transformation techniques in policy-based systems management," *Policies for Distributed Systems and Networks, 2004. POLICY 2004. Proceedings. Fifth IEEE International Workshop on*, pp. 13–22, 2004.
- [73] G. Xue, Z. Zhao, M. Ma, and Y. Shu, "Task scheduling by neural network with mean field annealing improvement in grid computing," *Electrical and Computer Engineering, Canadian Conference on*, pp. 554–557, 2006.
- [74] J. Adams and C. Price, "Boltzmann algorithms to partition and map software for computational grids," *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, 2004.
- [75] C. Huaihu, Y. Zhenwei, and W. Yinyan, "A chaotic neural network algorithm for task scheduling in overlay grid," *Semantics, Knowledge and Grid, 2005. SKG'05. First International Conference on*, pp. 38–38, 2005.
- [76] W. Shen, Y. Li, H. Ghenniwa, and C. Wang, "Adaptive negotiation for agent-based grid computing," *Proceedings of AAMAS2002 Workshop on Agentcities: Challenges in Open Agent Environments, Bologna, Italy*, pp. 32–36, 2002.
- [77] J. Li and R. Yahyapour, "Learning-based negotiation strategies for grid scheduling," *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)-Volume 00*, pp. 576–583, 2006.
- [78] M. Durad and Y. Cao, "A vision for the trust managed grid," *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)-Volume 00*, 2006.
- [79] S. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1995.

- [80] M. Wooldridge, “An Introduction to Multiagent Systems,” *West Sussex, England: John Wiley and Sons Ltd*, vol. 348, 2002.
- [81] J. Schopf, “Ten actions when Grid scheduling: the user as a Grid scheduler,” *Grid resource management: state of the art and future trends*, pp. 15–23, 2004.
- [82] J. M. Schopf, “Ten actions when grid scheduling,” *International Series in Operations Research and Management Science*, pp. 15–24, 2003.
- [83] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, “Condor-g: A computation management agent for multi-institutional grids,” *Cluster Computing*, vol. 5, no. 3, pp. 237–246, 2002.
- [84] R. Buyya, D. Abramson, and J. Giddy, “Nimrod/g: An architecture for aresource management and scheduling system in a global computational grid,” in *Proc. 4th International Conference on High Performance Computing in Asia-Pacific Region*, vol. 1, 2000, pp. 283–289.
- [85] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, and J. Mellor-Crumme, “The grads project: Software support for high-level grid application development,” *International Journal of High Performance Computing Applications*, vol. 15, no. 4, p. 327, 2001.
- [86] H. Casanova, G. Obertelli, F. Berman, and R. Wolski, “The apples parameter sweep template: User-level middleware for the grid,” *Scientific Programming*, vol. 8, no. 3, pp. 111–126, 2000.
- [87] M. Huber, “JAM: a BDI-theoretic mobile agent architecture,” *Proceedings of the third annual conference on Autonomous Agents*, pp. 236–243, 1999.
- [88] Wikipedia, “Goal — Wikipedia, the free encyclopedia,” 2011, [accessed 14-April-2011]. [Online]. Available: <http://en.wikipedia.org/wiki/Goal>
- [89] —, “Motivation — Wikipedia, the free encyclopedia,” 2011, [accessed 14-April-2011]. [Online]. Available: <http://en.wikipedia.org/wiki/Motivation>

- [90] “Cuesta college: Motivation and goal setting,” 2003, [accessed 14-April-2011]. [Online]. Available: <http://academic.cuesta.edu/acasupp/as/202.htm>
- [91] Z. Shen, C. Miao, and R. Gay, “Goal-oriented methodology for agent-oriented software engineering,” *IEICE Transactions on Information and Systems, Special Issue on Knowledge-based Software Engineering*, vol. 89.
- [92] G. Huang, Q. Zhu, and C. Siew, “Extreme learning machine: a new learning scheme of feedforward neural networks,” *Neural Networks, IEEE International Joint Conference on*, 2004.
- [93] G. B. Huang, Q. Y. Zhu, and C. K. Siew, “Extreme learning machine: theory and applications,” *NeuroComputing*, vol. 70, no. 1-3, pp. 489–501, 2006.
- [94] G. Zhao, Z. Shen, C. Miao, and R. Gay, “Enhanced extreme learning machine with stacked generalization,” in *Proc. International Joint Conference on Neural Networks (IJCNN 2008)*, 2008, pp. 1192–1199.
- [95] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR Upper Saddle River, NJ, USA, 1994.
- [96] G. Huang, Q. Zhu, and C. Siew, “Extreme learning machine: theory and applications,” *Neurocomputing*, vol. 70, no. 1-3, pp. 489–501, 2006.
- [97] G. Golub and C. Van Loan, *Matrix computations*. Johns Hopkins Univ Pr, 1996.
- [98] L. Breiman, “Stacked regressions,” *Machine Learning*, vol. 24, no. 1, pp. 49–64, 1996.
- [99] C. Blake and C. Merz, “Uci repository of machine learning databasets,” Department of Information and Computer Science, Irvine, CA: University of California. [Online]. Available: <http://www.ics.uci.edu/mllearn/MLRepository.html>.
- [100] D. Wolpert, “Stacked generalization,” *Neural Networks*, vol. 5, no. 2, pp. 241–259, 1992.
- [101] K. Ting and I. Witten, “Issues in stacked generalization,” *Journal of Artificial Intelligence Research*, vol. 10, pp. 271–289, 1999.

- [102] M. Leblanc and R. Tibshirani, "Combining estimates in regression and classification." *Journal of the American Statistical Association*, vol. 91, no. 436, 1996.
- [103] L. Torgo, "Regression data sets," University of Porto, Porto, Portugal. [Online]. Available: http://www.niaad.liacc.up.pt/~ltorgo/Regression/ds_menu.html.
- [104] S. Kumar, J. Chhugani, C. Kim, D. Kim, A. Nguyen, P. Dubey, C. Bienia, and Y. Kim, "Second life and the new generation of virtual worlds," *Computer*, vol. 41, no. 9, pp. 46–53, 2008.
- [105] N. Gupta, A. Demers, J. Gehrke, P. Unterbrunner, and W. White, "Scalability for virtual worlds," in *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*. Piscataway, NJ, USA: IEEE, 2009, pp. 1311–1314.
- [106] A. Abdelkhalek, A. Bilas, and A. Moshovos, "Behavior and performance of interactive multi-player game servers," *Cluster Computing*, vol. 6, no. 4, pp. 355–366, 2003.
- [107] M. Ye and L. Cheng, "System-performance modeling for massively multiplayer online role-playing games," *IBM Systems Journal*, vol. 45, no. 1, pp. 45–48, 2006.
- [108] J. Watts and S. Taylor, "A practical approach to dynamic load balancing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 9, no. 3, pp. 235–248, 2002.
- [109] J. W. Barrus, R. C. Waters, and D. B. Anderson, "Locales: Supporting large multiuser virtual environments," *Computer Graphics and Applications, IEEE*, vol. 16, no. 6, pp. 50–57, 2002.
- [110] M. R. Macedonia, M. J. Zyda, D. R. Pratt, D. P. Brutzman, and P. Barham, "Exploiting reality with multicast groups," *Computer Graphics and Applications, IEEE*, vol. 15, no. 5, pp. 38–45, 2002.
- [111] E. Lety, T. Turetti, and F. Baccelli, "Score: a scalable communication protocol for large-scale virtual environments," *Networking, IEEE/ACM Transactions on*, vol. 12, no. 2, pp. 247–260, 2004.
- [112] E. Frecon and M. Stenius, "Dive: A scaleable network architecture for distributed virtual environments," *Distributed Systems Engineering*, vol. 5, p. 91, 1998.

- [113] C. Diot and L. Gautier, “A distributed architecture for multiplayer interactive applications on the internet,” *Network, IEEE*, vol. 13, no. 4, pp. 6–15, 2002.
- [114] J. S. Steinman, “Scalable parallel and distributed military simulations using the speedes framework,” 1995, proc. of 2nd Electronic Simulation Conference (ELECSIM95), Internet.
- [115] M. Mauve, J. Vogel, V. Hilt, and W. Effelsberg, “Local-lag and timewarp: Providing consistency for replicated continuous applications,” *Multimedia, IEEE Transactions on*, vol. 6, no. 1, pp. 47–57, 2004.
- [116] E. Cronin, A. R. Kurc, B. Filstrup, and S. Jamin, “An efficient synchronization mechanism for mirrored game architectures,” *Multimedia Tools and Applications*, vol. 23, no. 1, pp. 7–30, 2004.
- [117] T. K. Das, G. Singh, A. Mitchell, P. S. Kumar, and K. McGee, “Neteffect: a network architecture for large-scale multi-user virtual worlds.” ACM, 1997, pp. 157–163, proceedings of the ACM symposium on Virtual reality software and technology.
- [118] J. Chim, R. W. H. Lau, H. V. Leong, and A. Si, “Cyberwalk: A web-based distributed virtual walkthrough environment,” *Multimedia, IEEE Transactions on*, vol. 5, no. 4, pp. 503–515, 2003.
- [119] J. Chen, B. Wu, M. Delap, B. Knutsson, H. Lu, and C. Amza, “Locality aware dynamic load management for massively multiplayer games.” ACM, 2005, pp. 289–300, proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming.
- [120] Butterfly.net, “Butterfly.net: Powering next-generation gaming with computing on-demand computing,” *IBM: An IDC e-Business Case Study*, 2004.
- [121] T. Wang, C. L. Wang, and F. Lau, “Grid-enabled multi-server network game architecture.” Citeseer, 2004, pp. 26–27, proceedings of the 3rd International Conference on Application and Development of Computer Games.

- [122] J. Stender, S. Kaiser, and S. Albayrak, "Mobility-based runtime load balancing in multi-agent systems," in *Proceedings of the 17th Software engineering and knowledge engineering conference (SEKE'06), KSI*, vol. 3420, 2006.
- [123] L. Zhang and Q. Lin, "Macve: A mobile agent based framework for large-scale collaborative virtual environments," *Presence: Teleoperators and Virtual Environments*, vol. 16, no. 3, pp. 279–292, 2007.
- [124] R. Prodan and V. Nae, "Prediction-based real-time resource provisioning for massively multiplayer online games," *Future Generation Computer Systems*, vol. 25, no. 7, pp. 785–793, 2009.
- [125] H. Liang, R. N. De Silva, W. T. Ooi, and M. Motani, "Avatar mobility in user-created networked virtual worlds: measurements, analysis, and implications," *Multimedia Tools and Applications*, vol. 45, no. 1, pp. 163–190, 2009.
- [126] H. Sarkel, J. Takatalo, P. May, M. Laakso, and G. Nyman, "The movement patterns and the experiential components of virtual environments," *International Journal of Human-Computer Studies*, vol. 67, no. 9, pp. 787–799, 2009.
- [127] G. P. Zhang, "Time series forecasting using a hybrid arima and neural network model," *NeuroComputing*, vol. 50, pp. 159–175, 2003.
- [128] G. Zhao, Z. Shen, and C. Miao, "Elm-based intelligent resource selection for grid scheduling," in *Machine Learning and Applications, 2009. ICMLA '09. International Conference on*, 2009, pp. 398–403.
- [129] Amazon, "Elastic compute cloud, [online]. available:<http://aws.amazon.com/ec2/>," 2011.
- [130] G. Zhao, Z. Shen, and C. Miao, "ELM-Based Agents for Grid Resource Selection," in *Web Intelligence and Intelligent Agent Technology, 2008. WI-IAT'08. IEEE/WIC/ACM International Conference on*, vol. 2. IEEE, 2008, pp. 385–388.
- [131] B. An, K. Sim, L. Tang, C. Miao, Z. Shen, and D. Cheng, "Negotiation agents' decision making using markov chains," *Rational, Robust, and Secure Negotiations in Multi-Agent Systems*, pp. 3–23, 2008.

- [132] C. Miao, J. Weng, A. Goh, Z. Shen, and B. An, “Fuzzy cognitive maps for dynamic grid service negotiation,” *Multiagent and Grid Systems*, vol. 2, no. 2, pp. 101–114, 2006.
- [133] C. S. Kuay, Z. Shen, R. Gay, and C. Miao, “Goal net agent framework for web service integration,” *language*, vol. 117494, p. 054.
- [134] Z. Q. Shen, R. Gay, C. Y. Miao, and Q. Wang, “Goal oriented modeling for agent mediated e-learning grid,” vol. 3. IEEE, 2005, pp. 2280–2285, control, Automation, Robotics and Vision Conference, 2004. ICARCV 2004 8th.
- [135] J. C. S. Lui and M. F. Chan, “An efficient partitioning algorithm for distributed virtual environment systems,” *Parallel and Distributed Systems, IEEE Transactions on*, vol. 13, no. 3, pp. 193–211, 2002.
- [136] K. Lee and D. Lee, “A scalable dynamic load distribution scheme for multi-server distributed virtual environment systems with highly-skewed user distribution,” in *Proceedings of the ACM symposium on Virtual reality software and technology*. ACM, 2003, pp. 160–168.
- [137] C. E. B. Bezerra and C. F. R. Geyer, “A load balancing scheme for massively multiplayer online games,” *Multimedia Tools and Applications*, vol. 45, no. 1, pp. 263–289, 2009.
- [138] N. I. of Science and Technology, “The nist definition of cloud computing,” 2011, [accessed 24-July-2011]. [Online]. Available: <http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>
- [139] S. Yan, B. Lee, G. Zhao, D. Ma, and P. Mohamed, “Infrastructure management of hybrid cloud for enterprise users,” in *Proceedings of the 5th International DMTF Academic Alliance Workshop on Systems and Virtualization Management: Standards and the Cloud*, 2011.
- [140] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, “Above the clouds: A berkeley view of cloud computing,” Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Tech. Rep., 2009.

- [141] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, 2010.
- [142] I. Foster, "Globus toolkit version 4: Software for service-oriented systems," *Journal of Computer Science and Technology*, vol. 21, no. 4, pp. 513–520, 2006.
- [143] A. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Dynamic Systems*, vol. 13, no. 4, pp. 341–379, 2003.
- [144] R. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1, pp. 181–211, 1999.
- [145] T. Dietterich, "Hierarchical reinforcement learning with the maxq value function decomposition," *Arxiv preprint cs/9905014*, 1999.
- [146] 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. 203–44, 2001.
- [147] D. Gordon and D. Subramanian, "A multistrategy learning scheme for agent knowledge acquisition," *Informatica*, vol. 17, pp. 331–346, 1994.
- [148] G. Towell, J. Shavlik, and M. Noordewier, "Refinement of approximate domain theories by knowledge-based neural networks," *Proceedings of the Eighth National Conference on Artificial Intelligence*, pp. 861–866, 1990.
- [149] R. Maclin and J. Shavlik, *Incorporating advice into agents that learn from reinforcements*. University of Wisconsin-Madison, Computer Sciences Dept, 1994.
- [150] R. Forlenza, "Knowledge sharing in multiagent environments: neural networks as intelligent agents," Ph.D. dissertation, University of Miami, 1997.
- [151] J. Clouse, "Learning from an automated training agent," *Working Notes of the ICML95 Workshop on Agents that Learn from Other Agents*, 1995.
- [152] P. Kar and S. Sen, "Agent teaching agent framework," *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pp. 1028–1029, 2003.

- [153] Q. Stephen, “Communicating neural networks in a multiagent system,” Ph.D. dissertation, University of Maryland Baltimore County, 2000.