Learning in multi-agent systems

Chen, Junhwa

2005

Chen, J. (2005). Learning in multi-agent systems. Master's thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/3759>

<https://doi.org/10.32657/10356/3759>

Nanyang Technological University

Downloaded on 26 Apr 2023 18:36:01 SGT

Learning in Multi-agent Systems

Chen Junhwa

School of Electrical & Electronic Engineering

A thesis submitted to the Nanyang Technological University
in fulfillment of the requirement for the degree of
Master of Engineering

2005

Acknowledgement

I would like to express my sincere appreciation to my supervisor, Associate Professor Yang Zhonghua. He gave me the freedom to develop on my own direction. At the same time he provided me valuable guidance and support through out the research.

I would also like to thank the staff of ICIS Research Lab for their assistance. I am grateful to the help of all my friends and other people who helped me with the research in one or another way.

Table of Content

<u>ACKNOWLEDGEMENT</u>	<u>I</u>
<u>TABLE OF CONTENT</u>	<u>II</u>
<u>LIST OF FIGURES</u>	<u>IV</u>
<u>SUMMARY</u>	<u>VI</u>
<u>CHAPTER 1 INTRODUCTION</u>	<u>1</u>
1.1 MOTIVATION	1
1.2 OBJECTIVES	3
1.3 CONTRIBUTIONS	4
1.4 ORGANIZATLON OF THE DISSERTATION	5
<u>CHAPTER 2 LITERATURE REVIEW</u>	<u>6</u>
2.1 MULTI-AGENT SYSTEMS	6
2.1.1 AGENTS	7
2.1.2 MULTI-AGENT SYSTEMS	8
2.1.3 ADVANTAGES OF MULTI-AGENT SYSTEMS	9
2.1.4 TAXONOMY	11
2.2 REINFORCEMENT LEARNING	12
2.2.1 BACKGROUND	12
2.2.2 REINFORCEMENT LEARNING FRAMEWORK AND ELEMENTS	12
2.2.3 THREE ELEMENTARY ALGORITHMS	19
2.2.4 SAMPLE REINFORCEMENT LEARNING APPLICATIONS	31
2.3 BACKPROPAGATION NEURAL NETWORK	32
2.4 SUMMARY	34
<u>CHAPTER 3 AGENT LEARNING FOR PERSONALIZED INFORMATION FILTERING: OUR APPROACH</u>	<u>35</u>
3.1 WEB DOCUMENT CLUSTERING	36
3.1.1 TRADITIONAL DOCUMENT CLUSTERING FAILS	37
3.1.2 SUFFIX TREE CLUSTERING	39
3.1.3 SAMPLE SUFFIX TREE CLUSTERING APPLICATIONS	42
3.2 PERSONALIZED INFORMATION FILTERING	43
3.2.1 THE NEED FOR PERSONALIZED ~NFORMATION FILTERING	44
3.2.2 EXISTING INFORMATION FILTERING SYSTEMS	45
3.3 AGENTS IN THE PROPOSAL	46
3.4 WHY MULTI-AGENT APPROACH?	47

3.5 THE MODEL: FOUR-AGENT SYSTEM	49
3.6 HYBRID LEARNING ALGORITHMS	52
3.7 THE JADE-BASED MULTI-AGENT SYSTEM	58
3.8 THE MULTI-AGENT SYSTEM DESIGN	61
3.8.1 USER AGENT	61
3.8.2 INFORMATION RETRIEVAL AGENT	62
3.8.3 CLUSTERING AGENT	63
3.8.4 LEARNING AGENT	64
3.9 DISCUSSION AND COMPARISON WITH EXISTING APPROACHES	67
 CHAPTER 4 GREENLEAF: A PROTOTYPE	 69
 4.1 USER AGENT	 69
4.2 INFORMATION RETRIEVAL AGENT	70
4.3 CLUSTERING AGENT	71
4.4 LEARNING AGENT	73
4.5 EVALUATION	80
4.6 SUMMARY	81
 CHAPTER 5 COMPETITIVE MULTI-AGENT LEARNING	 82
 5.1 DESCRIPTION OF THE GAME	 82
5.2 RULES OF THE GAME	84
5.3 DESIGN A LEARNING AGENT	85
5.3.1 EVALUATION FUNCTION	86
5.3.2 LEARNING ALGORITHM	87
5.4 IMPLEMENTATION	88
5.5 EVALUATION	95
5.6 SUMMARY	96
 CHAPTER 6 CONCLUSION AND RECOMMENDATIONS	 98
 6.1 CONCLUSION	 98
6.2 RECOMMENDATIONS FOR FURTHER RESEARCH	99
 AUTHOR'S PUBLICATION	 101
 REFERENCES	 102
 APPENDIX	 111

List of Figures

FIG 1 REINFORCEMENT LEARNING FRAMEWORK	13
FIG 2. VALUE OF TAKING ACTION A	17
FIG 3 POLICY ITERATION ALGORITHM	22
FIG 4 VALUE ITERATION ALGORITHM	23
FIG 5 MONTE CARLO ALGORITHM	25
FIG 6 Q-LEARNING ALGORITHM	28
FIG 7 SARSA ALGORITHM	29
FIG 8 BACKPROPAGATION ALGORITHM	33
FIG 9 LOCAL OPTIMUM	39
FIG 10 EXAMPLE OF A SUFFIX TREE	41
FIG 11 EXAMPLE OF A GENERALIZED SUFFIX TREE	42
FIG 12 THE FOUR AGENTS IN PROPOSED MULTI-AGENT SYSTEM	49
FIG 13 PROCEDURAL FORM OF OUR PROPOSED APPROACH	51
FIG 14 PROPOSED MODEL	52
FIG 15 OUR PROPOSED ALGORITHMS	56
FIG 16 ALGORITHM REASONING	58
FIG 17 USER PROFILE UPDATING ALGORITHM	66
FIG 18 USER AGENT	70
FIG 19 TAP INTO GOOGLE WEB SERVICE	71
FIG 20 A SAMPLE CLUSTER VIEW OF CLUSTERING RESULTS	72
FIG 21 A SAMPLE INSIDE CLUSTER VIEW OF CLUSTERING RESULTS	72
FIG 22 USER AGENT WITH EXPLICIT RATE FUNCTION	73
FIG 23 CODING BP NETWORK 1	74
FIG 24 CODING BP NETWORK 2	75
FIG 25 BP NETWORK ARCHITECTURE	75
FIG 26 DATA SET	76
FIG 27 WEIGHTS OF ALL THE LINKS	78
FIG 28 ORIGINAL USER PROFILE	79
FIG 29 UPDATED USER PROFILE	79
FIG 30 CHINESE CHECKERS BOARD	84

FIG 31 MARBE MOVE	85
FIG 32 THE GAME BOARD ALLOWING TWO PLAYERS	89
FIG 33 THE IMPLEMENTATION OF THE BOARD	90
FIG 34 WAYS OF GAME BOARD IMPLEMENTATION	91
FIG 35 YELLOW LOOSE 3 STEPS	93
FIG 36 UPDATED PARAMETERS	94
FIG 37 YELLOW LOOSE 2 STEPS	94
FIG 38 YELLOW WINS FOR THE FIRST TIME	95
FIG 39 PROPOSED HEURISTIC FUNCTION VS ALWAYS LONGEST JUMPS	96

Summary

In the recent years, multi-agent systems have gained increasing attention. Such systems can be cooperative or competitive. When designing multi-agent systems, designers are generally not able to tell an agent what to do in advance since it is impossible to predict all the situations an agent may experience. Agents have to learn. Although there are standard learning techniques, they need to be customized for a specific application domain. The main objective of this research is to investigate learning techniques available in a multi-agent setting, understand the cooperative and competitive learning, and develop and apply a new learning technique in a multi-agent area.

For learning in cooperative multi-agent system, this research considers the problem of personalized information filtering in the World Wide Web. The user's interests should be captured for personalized information filtering task. To realize efficient personalized information filtering, three issues should be solved. The first is how to capture the user's interest without repeatedly asking for his/her explicit rates. The second is to ensure the captured user profile is reasonably close enough to the user interest. The third is how to converge to user interest quickly. The term "converge" here means a gradual process of user profile becoming as close as possible to the user interest. Addressing these issues using conventional multi-agent learning techniques, author notes that the supervised learning technique alone failed to solve the first problem and the reinforcement learning technique alone failed to solve the third problem. After studying existing learning techniques and analyzing the problem, author proposes a new multi-agent learning approach. The learning technique is a set of hybrid learning algorithms which base on modified Monte Carlo method and combine with features of unsupervised Suffix Tree Clustering and supervised Backpropagation neural network. Author argues that the proposed approach can reasonably capture the user's interest close enough without repeatedly asking for his/her explicit rates and converge to the user's interest quickly. Author shows that the proposed approach is efficient, reasonably close enough to the user interest and converges more quickly compared with previous approaches.

For learning in competitive multi-agent system, a Chinese Checkers application is developed to demo the competitive learning. A variation of Q learning is applied to the agent to learn a heuristic evaluation function. The experimental results prove that the proposed algorithm works well.

Chapter 1 Introduction

This chapter gives an overview of this research. It starts from the motivation of this research, followed by research objectives and issues, then outlines author's major contributions and ends with introduction of report organization.

1.1 Motivation

In the recent years, multi-agent systems have gained increasing attention. Some researches in multi-agent systems involve the investigation for autonomous, rational and flexible behavior of entities such as software programs or robots, and their interaction and coordination in such diverse areas as robotics [4]. When designing agent systems, it is difficult to tell an agent what to do in advance since it is impossible to predict all the situations an agent may experience. Therefore, agents have to adapt to and learn from environment, especially in a multi-agent system. Learning ability is the most important feature of intelligent agents.

An agent is a computerized entity like a computer program or a robot. A multi-agent system differs from single agent system in that several agents exist which share

common resources and interact with one another [31]. Single agent learning is independent of the presence of other agents. In contrast, multi-agent learning relies on the presence of other agents and interaction among them. Learning in multi-agent system consists of both single agent learning and multi-agent learning. The term "multi-agent learning" has two meanings. In its stronger meaning, multi-agent learning refers only to situations in which several agents collectively pursue a common learning goal; In its weaker meaning, multi-agent learning additionally refers to situations in which an agent pursues its own goal, but is affected in its learning by other agents, their knowledge, beliefs, intentions, and so on [1].

There are a few criteria that can be used to classify many forms of learning in multi-agent systems. Feedback based learning is one of them. Following this criterion, three categories of learning are formed. They are reinforcement learning, unsupervised learning and supervised learning. Reinforcement learning agent receives an input and a reward of the action selected by the agent, and the learning algorithm has to learn a policy which maps inputs to actions resulting in the best performance. Unsupervised learning agent receives only input data and uses an objective function (such as a distance function) to extract clusters in the input data or particular features which are useful for describing the data. Supervised learning algorithms receive inputs and the correct outputs, and searches for a function which approximates the unknown target function [1].

Although standard learning techniques such as supervised, unsupervised, and reinforcement learning can be used as starting points for exploring effective learning

techniques in multi-agent situations, they need to be customized to match problem features for a specific application domain.

1.2 Objectives

The objectives of this research are to:

- Investigate learning techniques available in a multi-agent setting
- Understand the cooperative and competitive learning
- Develop a new learning technique.
- Apply the new learning technique to a multi-agent area.

Multi-agent systems are categorized as cooperative multi-agent systems and competitive multi-agent systems [31]. I investigate learning techniques for both cooperative multi-agent system and competitive multi-agent system. I consider personalized information filtering in the World Wide Web as the cooperative multi-agent area and Chinese Checkers as competitive multi-agent area.

Information preferences vary greatly across users. Hence personalized filtering systems should capture a user's individual interests to serve the user. Several problems occur in capturing user's individual interests. The first problem is how to capture the user's interest without repeatedly asking for his/her explicit rates such as asking user to rate the page from 1 to 5 in which 5 is best. The second problem is to ensure the captured interest is reasonably close enough to the actual user interest. The

third problem is how to converge to the user's interest quickly. The faced issues of multi-agent learning are:

- The supervised learning technique alone failed to solve the first problem.
- The reinforcement learning technique alone failed to solve the third problem.

In this research, I address all these issues and problems.

1.3 Contributions

The work presented in this report facilitates techniques of multi-agent learning to the problem of personalized information filtering. The user's interests are learned. The proposed system automates filtering tasks for the user. The contributions made by this report are as follows.

- A multi-agent model for personalized information filtering system
- Two hybrid learning algorithms and mechanisms
- A user profile updating algorithm
- A prototype of personalized information filtering system with multi-agent learning

Other contributions made by this report are a Chinese Checkers playing algorithm and a program to demonstrate the learning in competitive multi-agent learning.

1.4 Organization of the Dissertation

The rest of this report is organized as follow: **Chapter 2** provides literature review. The term "Multi-agent systems" is defined and multi-agent systems are introduced. Three elementary algorithms of reinforcement learning and supervised Backpropagation neural network learning are explained. In **Chapter 3**, first, unsupervised web document clustering is introduced. Then an example of learning in cooperative multi-agent system is presented that is a multi-agent approach to personalized information filtering. The need for personalized information filtering is pointed out and some existing filtering systems are outlined. It proposes a multi-agent hybrid learning approach to personalized information filtering problem. The reason for employing multi-agent system is explained. A multi-agent model is presented. Hybrid learning algorithm is modified Monte Carlo combined with features of Suffix Tree Clustering and Backpropagation network. Multi-agent system design is done by analyzing functions of each of four kinds of agents in the model and designing techniques for them. Algorithmic reasoning of proposed algorithms is presented. Comparison with previous approaches to personalized information filtering is outlined. **Chapter 4** describes a prototype of the proposed model in Chapter 3. Implementation of agents is listed. **Chapter 5** presents a Chinese Checkers application to demonstrate learning in competitive multi-agent system. A variation of Q learning is applied to the agent to learn a heuristic evaluation function. The experimental results prove that the proposed algorithm works well. **Chapter 6** draws conclusion and lists a few points for future research.

Chapter 2 Literature Review

Agents and multi-agent systems have been received increasing attention. In this chapter, I will first define the term agent and multi-agent systems, summarize the characteristics of agents and multi-agent systems, give the advantages of multi-agent systems and propose a taxonomy for multi-agent systems.

There are many learning methods available for each of reinforcement learning, unsupervised learning and supervised learning in multi-agent domain. I could not cover all of them in this report. For my purpose, reinforcement learning and supervised Backpropagation neural network are discussed in this chapter and unsupervised web document clustering is discussed in chapter 3. The rest of this chapter describes algorithms and applications of each of the learning methods.

2.1 Multi-agent Systems

This section introduces multi-agent systems. The term "Multi-agent systems" is defined. Advantages of multi-agent systems are outlined. A multi-agent systems' taxonomy is proposed.

2.1.1 Agents

Several formal definitions of multi-agent systems have been proposed by researcher. Before retaining these definitions, I present the widely adopted definitions of agent first.

This research is about learning in multi-agent systems, we must first agree on what an agent is. But there is no such agreement and there is no universally accepted definition of the term agent [42]. Here I give a general definition of agent.

Definition 2.1

A piece of software which performs a given task using information gleaned from its environment to act in a suitable manner so as to complete the task successfully. The software should be able to adapt itself based on changes occurring in its environment, so that a change in circumstances will still yield the intended result." [43]

Definition 2.2

"An agent is a computer system, situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives " [42].

An agent is characterized by the following five properties [44][45][47]. The first property is Autonomy. An agent is autonomous if it has some control over its actions and operates without the direct intervention of humans or other agents. The second property is social ability. If an agent is able to communicate and interact with other agents, we say it has social ability. The third property is reaction. An agent perceives

its environment and responds to change that occurs in it. The forth property is pro-activity. An agent does not simply act in response to its environment. It is goal-oriented. It is able to handle high-level tasks. It can make decisions to complete such high-level tasks. Like human, an agent can split the task into several sub tasks and finish the sub tasks in a reasonable sequence. The fifth property is mobility. If an agent can transport itself from one machine to another, retaining its current state, we call it mobile agent.

2.1.2 Multi-agent systems

Having agreed on what an agent is, I can present the definition of the term multi-agent systems (MASs).

Definition 2.3 *A Multi-Agent System (MAS) is a system composed of a population of autonomous agents, which cooperate with each other to reach common objectives, while simultaneously each agent pursues Individual objectives. [44][45]*

The characteristics of MASs are as follows. First, each agent has a limited viewpoint because it has incomplete information or capabilities for completing the task. Second there is no system global control. Third, data are decentralized. Forth, computation is asynchronous [39]. Multi-agent System issues and challenges can be found in [39].

There are several reasons why multiple agents need to be coordinated [44][45][46].

The first reason is to prevent chaos. Each agent has a limited viewpoint and has only

local goals and knowledge. When it communicates with other agents or tries to support other agents, there may be conflicts due to the limited knowledge. Coordination is critical to prevent chaos during conflicts.

The second reason is that agents in MAS possess different capabilities and expertise. They play different roles to contribute to the common objective. Like in an academic conference, all researchers communicate and share the knowledge to further development. Such knowledge sharing needs coordination.

The third reason is that agent's actions are frequently interdependent. Actions need to be governed by the common objective. Such agents' actions need coordination.

The agents have to interact and exchange information in order to achieve this coordination. But unlike for human, it's a difficult task for computer agents. Computer agents need some common language among them. KQML[49], Knowledge Query and Manipulation Language, is a good example of such communication language [49]. FIPA-ACL[48], FIPA Agent Communication Language, is another good example.

2.1.3 Advantages of Multi-agent Systems

A MAS has the following advantages over a single agent or centralized approach:[39]

First, A MAS distributes computational resources and capabilities across a network of interconnected agents [39]. A MAS is decentralized and thus does not have problems

associated with centralized systems, such as resource limitations, performance bottlenecks etc.

Second, A MAS allows for the interconnection and interoperation of multiple existing legacy systems [39]. An agent society can be formed by building an agent wrapper around such systems. For example, expert systems or decision support systems.

Third, A MAS models problems in terms of autonomous interacting component-agents [39]. That means MAS provide solutions that can be easily understood because the agents in the solutions naturally represents the roles in the real society. For example, in appointment booking for a clinic system, general practice doctor in a clinic attempts to book a time slot of a specialist. An appointment booking agent that manages the calendar of the general practice doctor can be regarded as autonomous and interacting with the appointment booking agent that manage calendars of the specialist. Appointment booking agent naturally represents the general practice doctor and discusses and books a time slot with the agent naturally represents the specialist. Appointment booking agents also can be customized to reflect the preferences and constraints of their users.

Forth, A MAS retrieves, filters, and coordinates information from spatially distributed sources efficiently [39]. One example of such domain is sensor networks as described in [59].

Fifth, A MAS provides solutions for situations where expertise is spatially and temporally located [39]. Spatially and temporally located indicates different expertise has its own venue and time of availability/presence. One example of such domain is

health care, in which specialists working time may differ and they work at different places.

Sixth, A MAS enhances overall system performance, in particular along the computational efficiency, reliability, extensibility, robustness, maintainability, responsiveness, flexibility, and reuse dimensions [39].

2.1.4 Taxonomy

Multi-agent systems can broadly be divided into two categories: Cooperative multi-agent system and competitive multi-agent system [50][51].

Cooperative multi-agent system is a system in which all agents share the same goals. Each agent may have a local goal, but this goal will always support the global goal of the all the agents. Each agent acts with the global goal in mind. All agents perform tasks in order to achieve the global goal of the MAS. An example of a cooperative multi-agent system is a team of robots play football with a global goal of the team -- win.

Competitive multi-agent system is a system in which agents' interests are different or opposite. An example of a competitive multi-agent system is an agent based auction system.

2.2 Reinforcement Learning

This section covers background of reinforcement learning (RL), RL framework and elements, three elementary algorithms and some successful applications, e.g. Elevator dispatch [10][32] and Dynamic channel allocation [2].

2.2.1 Background

Two main threads form the history of reinforcement learning [2]. They developed independently before intertwining in modern reinforcement learning. The first thread started in the psychology of animal learning. It concerns learning by trial and error. The second thread concerns optimal control problem. The two threads were largely independent until a third, less distinct thread concerning temporal-difference methods. The modern field of reinforcement learning came into being when all three threads came together in the late 1980s.

2.2.2 Reinforcement Learning Framework and Elements

This section is a detailed survey on the framework and elements of reinforcement learning, especially the state-value function and action-value function. The state-value function and action-value function will be discussed soon in this section. The state-value function and action-value function are important because they will be frequently used in the next section which is a survey on three elementary reinforcement learning algorithms.

Fig 1 graphically illustrates the framework of reinforcement learning. At each time step n , agent receives some representation of the environment's state x_n , and on that basis selects an action a_n , which is one of the actions available in state x_n . This selection is guided by a policy π . One time step later, in part as a consequence of its action, the agent receives a numerical reward, r_{n+1} , which is given by reward function, and finds itself in a new state, x_{n+1} . Note that the time steps in the framework need not refer to fixed intervals of real time. They can be arbitrary successive stages of making decision and taking action.

Fig 1 Reinforcement Learning Framework

From the framework of RL, six elements in reinforcement learning named agent, environment, policy, reward function, value function, and optionally, a model of the environment [3][6] can be identified. Agent is the one who learns and makes decision. Environment comprises everything outside the agent which agent interacts with.

Policy maps states of environment to actions taken by agent when agent is in those states. It defines agent's selection of action at a given time. Reward function gives agent a numerical reward, when agent is in a state, or when agent takes an action in a state. Value function gives the value of a state (or state-action pair) which is the summation of expected rewards starting from that state (or state-action pair and thereafter follow the same policy). Reward function indicates the intrinsic desirability of the state (or state-action pair), i.e. which event is good and which event is bad. Value function indicates the long-term desirability of state (or state-action pair) after taking into account the states (or state-action pairs) that are likely to follow and the rewards available in those states (or state-action pairs). Agent's objective is to maximize the total reward it receives in the long run. Given a state and an action, the model, which is transition possibility among states of environment, might predict the resultant next state and next reward.

Value function, which is the fifth element in reinforcement learning, gains our interest. Almost all RL algorithms are based on estimating value functions [2]. State-value function and action-value function are two kinds of value functions [2]. Value function represents the value of being in a state with respect to the goal is known as state-value function. Value function represents the value of taking an action in a state with respect to the goal is known as action-value function. State-value function and action-value function are used together to find the best policy.

The value of a state s under policy π , $V^\pi(s)$, is the expected discounted summation of reward r , R , starting from that state. From the definition of value of state, state-value function can be written as in equation (1) [2], where γ is discount factor, γ^k indicates

the later reward has less effect on the current state value, r_{t+k+1} is reward at $(k+1)$ th time step from the current state. As explained at the beginning of section 2.2.2, the time step needs not refer to fixed intervals of real time. It can be arbitrary successive stage of making decision and taking action.

$$V^\pi(s) = E_\pi \{R_t | s_t = s\} = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right\} \quad [2] \quad (1)$$

Equation (1) is only a simple equation resulting from definition. If equation (1) can be modified and written as the relationship between the current state and the next state, i.e. current state is a function of next state, then state-value function will be useful. Equation (2) [2] makes such idea real. The expected discounted summation of reward; R_t starting from time step t is the accumulation of multiplication of discount factor γ^k by r_{t+k+1} . The conclusion of equation (2) is the expected discounted summation of reward, R_t starting from time step t is equals to r_{t+1} plus discount factor γ multiplies the expected discounted summation of reward, R_{t+1} , starting from time step $t+1$. Definition of γ^k and r_{t+k+1} in equation (2) are same as those in equation (1).

$$\begin{aligned} R_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \dots \\ &= r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} + \dots) \\ &= r_{t+1} + \gamma R_{t+1} \end{aligned} \quad [2] \quad (2)$$

Substitute equation (2) into equation (1) we get equation (3) [2], which tells us the value of state s under policy π , $V^\pi(s)$, is the summation of immediate reward r_{t+1} and

discounted value of the state follows, $\gamma V(s_{t+1})$. Definition of y in equation (3) is same as that in equation (1).

$$V^\pi(s) = E_\pi \{R_t | s_t = s\} = E_\pi \{r_{t+1} + \gamma V(s_{t+1}) | s_t = s\} \quad [2] \quad (3)$$

When in state s , take an action a may result in different next states, e.g. s' , s'' , shown in Fig 2. The value of *taking action a and resulting s'* is the summation of immediate reward $R_{ss'}^a$ and discounted value of the state s' , $\gamma V^\pi(s')$. The value of *taking action a and resulting s''* is the summation of immediate reward $R_{ss''}^a$ and discounted value of the state s'' , $\gamma V^\pi(s'')$. If given the possibility of transiting from current state s to a next state s' when taking action a , $p_{ss'}^a$, as shown in Fig 2, multiplication of $p_{ss'}^a$ by the value of *taking action a and resulting s'* will be the value of *taking action a and resulting s'* . The total value of taking action a is the summation of all the multiplications of possibility and the value of *taking action a and resulting correspond state*. Given the possibility of selecting action a in current state s , $\pi(s, a)$, multiplication of this possibility by the value of *taking action a* will be the contribution of *taking action a* to the current state value. The value of the current state is the accumulation of such multiplication over different action a . This idea is shown in equation (4) [2].

Fig 2. Value of taking action a

$$V^{\pi}(s) = \sum_a \pi(s, a) \sum_{s'} p_{ss'}^a [R_{ss'}^a + \gamma V^{\pi}(s')] \quad [2] \quad (4)$$

The definition of value of taking an action from [2] is: the value of taking an action a in a state s under policy π , $Q^{\pi}(s, a)$ is the expected discounted summation of reward r , R , starting from that state, taking that action, and thereafter following π . From the definition of value of action, action-value function can be written as in equation (5) [2], where γ is discount factor, γ^k indicates the later reward has less effect on the current action value, r_{t+k+1} is reward of time step $k+1$ from the current state-action pair. γ and γ^k in equation (5) explain the "discounted summation" in the definition of value of taking an action; $s_t = s$, $a_t = a$ in equation (5) which explains "starting from that state, taking that action" in the definition of value of taking an action; k is from 0

to ∞ in equation (5) which explains the "and thereafter following π " in the definition of value of taking an action.

$$Q^\pi(s, a) = E_\pi \{R_t | s_t = s, a_t = a\} = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a \right\} \quad [2] \quad (5)$$

Using equations (3), (4) and (5) value of state and value of action can be calculated. These values are used to find an optimal policy. The original state-value function is $V^\pi(s) = \sum_a \pi(s, a) \sum_{s'} p_{ss'}^a [R_{ss'}^a + \gamma V^\pi(s')]$ For each state, there will be at least one action that leads to maximum action-value. A policy that assigns non-zero probability to such an action and zero probability to all others will let the state value of state s be maximum. And we call this an optimal policy since optimal policy is a policy which will lead to maximum value. Equation (6) [2] can be written down, which is the Bellman optimality equation for optimal state-value function, V^* . The optimal policy is obtained from the optimal state-value function by one-step look-ahead search.

$$\begin{aligned} V^*(s) &= \max_a Q^\pi(s, a) \\ &= \max_a E_\pi \{R_t | s_t = s, a_t = a\} \\ &= \max_{a \in A(s)} E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a \right\} \\ &= \max_{a \in A(s)} E \{r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a\} \\ &= \max_{a \in A(s)} \sum_{s'} p_{ss'}^a [R_{ss'}^a + \gamma V^*(s')] \end{aligned} \quad [2] \quad (6)$$

Analogous to the Bellman optimality equation for optimal state-value function, for each next s' resulting from state s and action a , there will be at least one action that leads to maximum action-value in state s' . A policy that assigns non-zero probability

to such an action and zero probability to all others in state $\mathbf{s}'$ will be an optimal policy. Equation (7) [2] can be written down. Equation (7) is the Bellman optimality equation for optimal action-value function Q^* .

$$Q^*(s, a) = E\left\{r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \mid s_t = s, a_t = a\right\} = \sum_{s'} p_{ss'}^a \left[R_{ss'}^a + \gamma \max_{a'} Q^*(s', a') \right] \quad [2] \quad (7)$$

The next section is a detailed survey on three elementary reinforcement learning algorithms. The state-value function and action-value function discussed in the above section will be frequently used in the three elementary algorithms.

2.2.3 Three Elementary Algorithms

Reinforcement learning has three elementary algorithms. They are Dynamic Programming, Monte Carlo methods and Temporal Difference learning. All the three elementary algorithms adopt Generalized Policy Iteration (GPI) [2] technique, which is an interaction of policy evaluation and policy improvement. Policy evaluation is concerned with estimating the values of states (or state-action pairs) when the agent acts according to some fixed policy. Policy improvement algorithm improves the current policy. A sequence of policy evaluation and policy improvement is $\pi_0 \rightarrow V^{\pi_0} \rightarrow \pi_1 \rightarrow V^{\pi_1} \rightarrow \pi_2 \rightarrow V^{\pi_2} \rightarrow \dots$ till the optimal policy is found. The sequence started with a policy π_0 , with policy evaluation, we get states' value V^{π_0} . Using these states' value, policy improvement is performed, which resulting a better policy π_1 . In some cases, GPI can be proved to converge, most notably for the classical Dynamic Programming (covered in section 2.2.3.1) methods [2]. In other cases convergence has

not been proved [2]. The essential idea is to combine search and memory. Search is to try and select among many actions in each state; and memory is to remember what action worked best in each state. In the following sections, we briefly describe each of these elementary algorithms.

2.2.3.1 Dynamic Programming

The term "Dynamic Programming"(DP) was due to Bellman in 1957 [57]. DP refers to a collection of algorithms those can compute optimal policies when Markov decision process (MDP) environment is given. A reinforcement learning task that satisfies the Markov property[2] is called a Markov decision process (MDP)[2]. It is called a *finite Markov* decision process (finite MDP) if the state and action spaces are finite [2]. DP always assumes finite MDP [2].

Given a perfect model of the environment, DP methods can compute optimal policies, by using the value function and Bellman equations in equation (6) and (7) to guide the search for optimal policies. Rather than solving the Bellman equation directly, DP methods treat them as recursive update rules. Two most popular Dynamic Programming algorithms are policy iteration and value iteration,

Policy iteration follows GPI tightly. An agent evaluates the current policy π by calculating all the states' values $V^\pi s$ and memorizing them. Equation (4) is used to calculate values of states. An agent improves policy π using $V^\pi s$ to yield a better policy π' . Agent then starts evaluate policy π' and improves it again to yield an even

better policy π' . Equation (6) is used to find a better policy. The process iterates till the agent reaches the optimal policy which means convergence. It guarantees the successive policy to be a strict improvement over the previous policy (unless the previous policy is already optimal). Since a finite MDP has only a finite number of policies, the algorithm given in Fig 3 must converge to an optimal policy and optimal value function in a finite number of iterations. The complete algorithm of policy iteration is shown in Fig 3.

```

1. Initialize  $V(s) \in \mathbb{R}$  and  $\pi(s) \in A(s)$  arbitrarily for all  $s \in S$ 

2. Policy Evaluation

   Repeat

      $\Delta \leftarrow 0$ 

     For each  $s \in S$ :

        $v \leftarrow V(s)$ 

       
$$V(s) \leftarrow \sum_{s'} p_{ss'}^{\pi(s)} [R_{ss'}^{\pi(s)} + \gamma V(s')]$$


        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$ 

   until  $\Delta < \theta$  (a small positive number)

3. Policy Improvement

   policy is stable (means no change in policy)

   For each  $s \in S$ :

      $b \leftarrow \pi(s)$ 

```

$$\pi(s) \leftarrow \arg \max_a \sum_{s'} p_{ss'}^a [R_{ss'}^a + \gamma V(s')]$$

If $b \neq \pi(s)$, then policy is not stable.

If policy is stable, then stop; else go to 2

Fig 3 Policy Iteration Algorithm

As shown in Fig 3, each cycle of policy iteration involves policy evaluation, which may take long time to converge to V^π . Can an agent stop before the exact convergence? Fortunately, the answer is yes. The policy evaluation is to provide the state value information for policy improvement. Because of the finite MDP, policy evaluation does not necessary to converge before policy improvement can be done. The policy evaluation step of policy iteration can be truncated in several ways without losing the convergence guarantees of policy iteration [2]. Value iteration is an important special case when policy evaluation is stopped after just one cycle. One cycle of policy evaluation and one cycle of policy improvement are combined. The complete algorithm of value iteration is shown in Fig 4.

Initialize V arbitrarily, e.g., $V(s)=0$, for all $s \in S$

Repeat

$\Delta \leftarrow 0$

For each $s \in S$:

$v \leftarrow V(s)$

$$V(s) \leftarrow \max_a \sum_{s'} p_{ss'}^a [R_{ss'}^a + \gamma V(s')]$$

```


$$\Delta \leftarrow \max(\Delta, |v - V(s)|)$$


until  $\Delta < \theta$  (a small positive number)

Output a deterministic policy,  $\pi$ , such that:


$$\pi(s) \leftarrow \arg \max_a \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V(s')]$$


```

Fig 4 Value Iteration Algorithm

The optimal policy does not change with time.

2.2.3.2 Monte Carlo Methods

The term "Monte Carlo" dates from the 1940s when physicists at Los Alamos devised games of chance that they could study to help understand complex physical phenomena relating to the atom bomb [2]. "Monte Carlo" sometimes refers to any estimation method whose operation involves a significant random component. In a pole balancing problem, Michie and Chambers first introduced Monte Carlo methods in a reinforcement context in 1968 [2]. They used averages of episode durations to assess the worth, which is the expected balancing time before failure, of each possible action in each state, and then used these assessments to control action selections.

Monte Carlo methods require only experience---sample sequences of states, actions, and rewards from actual or simulated interaction with an environment. Although a model is still required, the model needs only generate sample transitions, not the

complete probability distributions of all possible transitions that are required by DP methods.

Monte Carlo methods are based on averaging sample returns. To ensure that well-defined returns are available, we define Monte Carlo methods only for episodic tasks. That is, we assume experience is divided into episodes, and that all episodes eventually terminate no matter what actions are selected. It is only upon the completion of an episode that values are estimated and policies are changed. Monte Carlo methods are thus incremental in a sense of episode-by-episode, but not in a sense of step-by-step. In an episode, there may have many steps. The episode-by-episode increment means value estimation and policy change are only done when an episode is completed. The step-by-step increment means value estimation and (or) policy change are done when a step is completed as discussed in Dynamic Programming. For step-by-step increment, if there are 10 steps in an episode, the policy will be updated 10 times in an episode.

Just as DP, MC follows GPI. Firstly, policy is evaluated by the computation of P and Q^π for an arbitrary policy π , which is average of sample returns. Each time an agent visits a state and takes an action, return of state-action pair R is taken down in a list. For example, if an agent visits the same state s and takes the same action a for 3 times, action value $Q(s,a)$ is equal to $(R_1+R_2+R_3)/3$, where R_i denotes return of the i th visit, stored in the i th place in the list. Secondly, the policy is improved using equation (6). A complete list of the most basic Monte Carlo algorithm can be found in Fig 5. Other Monte Carlo algorithms such as Monte Carlo ES, First-visit MC, Every-visit MC and so on are not covered in this report.

```

1. Initialize, for all  $s \in S$ ,  $a \in A(s)$ :

     $Q(s,a) \leftarrow \text{arbitrary}$ 

     $n(s) \leftarrow \text{Arbitrary}$ 

     $\text{Returns}(s,a) \leftarrow \text{empty list}$ 

2. Repeat Forever:

    a. Generate an episode using  $n$ 

    b. For each pair  $s,a$  appearing in the episode:

         $R \leftarrow \text{return following the first occurrence}$ 

        Append  $R$  to  $\text{Returns}(s,a)$ 

         $Q(s,a) \leftarrow \text{average}(\text{Returns}(s,a))$ 

    c. For each  $s$  in the episode:

         $\pi(s) \leftarrow \text{argmax}_a Q(s,a)$ 

```

Fig 5 Monte Carlo Algorithm

An Agent with MC algorithm learns value functions and optimal policies from experience in the form of sample episodes. This gives Monte Carlo methods at least three kinds of advantage over DP methods. First, Monte Carlo methods can be used to learn optimal behavior directly from interaction with the environment, with no model of the environment's dynamics. Second, Monte Carlo methods can be used with simulation or sample models. For surprisingly many applications it is easy to simulate

sample episodes even though it is difficult to construct the kind of explicit model of transition probabilities required by DP methods. Third, it is easy and efficient to focus Monte Carlo methods on a small subset of the states.

2.2.3.3 Temporal-Difference Learning

The idea of Temporal-Difference(TD) learning has its early roots in animal learning psychology and artificial intelligence, most notably the work of Samuel in 1959 [2]. Samuel's checker player program suggests the essential idea of temporal-difference learning--- the value of a state should equal the value of likely later states starting from that state. But modern TD learning is not conceptualized until Richard Sutton. TD(0) algorithm and the term "temporal-difference learning" are due to Sutton in 1988, who proved Tabular TD(0) convergence and established the optimality of the TD algorithm under batch training [2].

TD learning is the combination of Monte Carlo and Dynamic Programming ideas. Like Monte Carlo, TD methods can learn from raw experience without a model of the environment's dynamics. Like DP, TD methods update estimates based in part on other learned estimates, without waiting for a final outcome.

When researchers update value of a state in a problem where no clear transition probabilities available, equation (8) is usually used. Equation (8) means the state value $V(s_t)$ is updated with summation of $(1-\alpha)$ times of the original state value and α times of expected rewards from time t , R_t .

$$V(s_t) \leftarrow (1 - \alpha)V(s_t) + \alpha R_t \quad (8)$$

Rearrange the operands and operators of equation (8), equation (9) is easily written down.

$$V(s_t) \leftarrow V(s_t) + \alpha(R_t - V(s_t)) \quad (9)$$

When the idea applied to TD learning, return R_t is not known. An estimation of return R_t , $r_{t+1} + \gamma V(s_{t+1})$, is used to replace R_t . The estimated value is immediate reward plus the discounted value of following state. This is what meant by the sentence "temporal difference learning is a prediction on prediction [30]", because the state value $V(s_t)$ is a prediction on the $V(s_{t+1})$ and $V(s_{t+1})$ is a prediction too. Therefore, the state-value function used for policy evaluation in the simplest TD method, TD(0), is $V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$, where α is learning rate.

With policy improvement, there are two famous algorithms named Q-Learning [2][33], and SARSA [2]. Q-learning was introduced by Watkins in 1989 [2]. Q-Learning is an off-policy algorithm for Temporal Difference learning. Algorithm stores separate values for each action in each state in look-up table, so that the optimal action can be selected directly for each state.

Q-learning uses greedy policy to update action values, which is the reason for *max* appearing in formula in Fig 6. The term greedy is used in computer science to refer to any search or decision procedure that selects alternatives based only on local or immediate considerations, without considering the possibility that such a selection may prevent future access to even better alternatives [2]. The action a' , whose action

value is used for update, may not be selected and taken in next step. This is off-policy means. Formal definition of off-policy will be covered later.

```

Initialize  $Q(s, a)$  arbitrarily
Repeat (for each episode):
    Initialize  $s$ 
    Repeat (for each step of episode):
        Choose  $a$  from  $s$  using policy derived from  $Q$  (e.g.  $\epsilon$ -greedy)
        Take action  $a$ ; observe reward,  $r$ , and next state  $s'$ 
         $Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$ 
         $s \leftarrow s'$ 
    until  $s$  is terminal

```

Fig 6 Q-Learning Algorithm

The Sarsa algorithm was first explored by Rummery and Niranjan in 1994 [2]. The Sarsa algorithm is an on-policy algorithm for TD-Learning. The major difference between it and Q-Learning is that the maximum reward for the next state is not necessarily used for updating the Q-values. Value of a new action a' , which is selected using the same policy that determined the original action, is used for updating. The action a' will actually be taken by agent in next step. Sarsa is only concerned with the policy agent is following, i.e., the policy which determined actions a, a' , comparing to Q-Learning, a, a' may determined by different policies.

The name Sarsa actually comes from the fact that the updates are done using the quintuple $Q(s, a, r, s', a')$. Where s, a are the original state and action, r is the reward observed in the following state and s', a' are the new state-action pair. The update formula is similar to that of Q-learning except the difference described above. The procedural form of Sarsa algorithm given in Fig 7 is comparable to that of Q-Learning. The parameters α and γ have the same meaning as they do in Q-Learning.

```

Initialize  $Q(s, a)$  arbitrarily
Repeat (for each episode):
    Initialize  $s$ 
    Choose  $a$  from  $s$  using policy derived from  $Q$  (e.g.  $\epsilon$ -greedy)
    Repeat (for each step of episode):
        Take action  $a$ , observe  $r, s'$ 
        choose  $a'$  from  $s'$  using policy derived from  $Q$  (e.g.
 $\epsilon$ -greedy, which is explained after the Figure;)
         $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$ 
         $s \leftarrow s'$ 
    until  $s$  is terminal (means all states are visited, no
more unvisited states)

```

Fig 7 SARSA Algorithm

ϵ -greedy policy is a policy in which most of the time an action that has maximal estimated action value is chosen, but with probability ϵ a random action is chosen instead.

TD methods have advantage over DP methods: TD methods do not require a model of the environment. TD methods also have advantage over MC methods: TD methods are naturally implemented in an online, fully incremental fashion. With the MC, one must wait until the end of the episode, because only then the return is known, whereas with TD methods, one need wait only one time step. Some applications have very long episode, so that delaying all learning until an episode's end is too slow.

Q-learning is an off-policy algorithm of Temporal Difference learning while Sarsa is on-policy. We present the definitions of off-policy and on-policy as following.

On-Policy methods learn the value of the policy that is used to make decisions [3]. These policies usually ensure sufficient exploration. An agent updates the value functions using results from executing actions determined by some policy. Updating value functions is strictly based on experiences of an agent.

Off-Policy methods can learn different policies for behavior and estimation [3]. The behavior policy guarantees sufficient exploration going on. Agent updates the estimated value functions using the maximum action value for the next state in a look-up table. Agent may not take the action used for updating in the next step.

2.2.4 Sample Reinforcement Learning Applications

Elevator dispatch

Crites and Barto applied reinforcement learning techniques to a four-elevator, ten-floor system [10][32]. Each elevator has a position, direction, speed and a set of buttons indicating where passengers want to get off. Roughly quantizing the continuous variables, Crites and Barto estimated that the system has over 10^{22} states [10]. This large state set rules out classical dynamic programming methods such as value iteration. The objective is to keep the time passenger waits before getting on an elevator as low as possible while serving all the passengers fairly. Crites and Barto applied modified Q-learning and Backpropagation Neural Network (as covered in section 2.3) to this problem. Each elevator is corresponding to an agent. Q-learning is to learn agent operation. BP network is trained to evaluate Q values, because the traditional tabular method fails to store such large amount of Q values. These reinforcement learning dispatchers performed very well. By all of the performance measures, the learned dispatchers compare very favorably with the others.

Dynamic channel allocation

An important problem in the operation of a cellular phone system is how to efficiently use the available bandwidth to provide quality service to customers as many as possible. This problem gained more attention with the rapid growth in the use of cellular telephones. Singh and Bertsekas applied reinforcement learning to this problem [2]. If there is no channel for a call, the call will be blocked. Singh and

Bertsekas considered the problem of allocating channels so that the number of blocked calls is minimized. It uses Temporal Difference learning. Comparing to other methods, result shows reinforcement learning method is much better than other methods.

2.3 Backpropagation Neural Network

Numerous neural network algorithms appear in literature and there are good monographs available, In this subsection, we only introduce Backpropagation.

The Backpropagation algorithm was developed for training multi-layer perceptron networks. It was popularized by Rumelhart, Hinton and Williams in 1986 [58], although similar ideas had been developed previously by others [20]. Output errors are propagated backward through the layers so that network is trained. The errors serve to evaluate the derivatives of the error function with respect to the weights, which can then be adjusted [20].

Backpropagation algorithm for a one-hidden-layer network, using sigmoid functions activate function, consists of three steps [21]. First, initialize all link weights w_{ji} from neuron U_i to neuron U_j . For example, we can set all of them to 0. Second, for each neuron j in network, calculate output of the neuron o_j , using activation function. In our case, we use sigmoid function as activation function. Parameter passed to sigmoid function is the summation of $w_{ji}o_i$ over i . For example if neuron 1, 2, 3 are connected to neuron 4, output o_4 is sigmoid function of $w_{41}o_1 + w_{42}o_2 + w_{43}o_3$. Third, update all

link weights by add ∇w_{ji} , which is change of link weight w_{ji} , to the previous weight. Formula for calculating the change for links from input neuron to hidden neuron and those from hidden neuron to output neuron are different. These formulas are shown in Fig 8, which is an outline of the complete algorithm of Backpropagation.

1. Initialization: all w_{ji} (from U_i to U_j)
2. Calculation of Activation
 - a) For input units: o_j =inputs presented to the net
 - b) For hidden and output units:

$$o_j = f(\sum_i w_{ji}o_i + \theta_j) \text{ where } f(x) = 1/(1+e^{-x})$$
3. Weight training:

$$w_{ji}(t+1) = w_{ji}(t) + \nabla w_{ji}$$

$$\nabla w_{ji} = \eta * \delta_j * o_i$$

The error gradient δ_j on U_j :

 - a) for output units: $\delta_j = o_j(1 - o_j)(t_j - o_j)$ where t_j is target of unit j .
 - b) for hidden units: $\delta_j = o_j(1 - o_j) \sum_k \delta_k w_{kj}$

Repeat iterations until convergence in terms of the selected error criterion.

Fig 8 Backpropagation Algorithm

2.4 Summary

In this chapter, I have defined the terms agent and multi-agent systems, summarized the characteristics of agents and multi-agent systems. Three elementary reinforcement learning algorithms are introduced. They are Dynamic Programming, Monte Carlo Methods and Temporal-Difference Learning. The difference among the three elementary algorithms is explained and some of the respective applications are covered. At last, supervised Backpropagation neural network is discussed.

Chapter 3 Agent Learning for Personalized Information Filtering: Our Approach

Research in multi-agent systems has been carried out in diverse areas such as robotics and online auctions. Personalized information filtering for information on World Wide Web is an area too. This chapter starts with reviewing the technologies of web document clustering, followed by explaining the need for personalized information filtering and introduction of some existing personalized information filtering systems. Next, the proposed multi-agent approach to personalized information filtering is proposed.

The rest of this chapter covers four kinds of agents in our proposal followed by explaining why I use multi-agent based system for personalized information filtering system. After that a simple version of the system---a four-agent hybrid learning model is described. Working procedure and hybrid learning algorithms are proposed. JADE is selected as runtime framework. Then multi-agent system design is described. Functions and algorithms of each of the four agents are discussed. Finally, a comparison with traditional approaches is presented.

The terms "user profile" and "user interest" will be frequently used. An early understanding will help. A user profile is a vector of terms representing the user's interest on a specific topic. For implementation, user profile can be a .txt file containing phrases describing the user's interest on the specific topic. User's Interest is described by a vector of terms. A user knows his/her interest in a topic but he/she may fail to express the interest precisely in terms of English words.

3.1 Web Document Clustering

This section starts with defining clustering and categorizing clustering methods. However, my focus is web document clustering. Traditional document clustering fails to classify web documents. A new algorithm called Suffix Tree Clustering, from University of Washington, is suitable for web document clustering [7].

Clustering is a process in which a set of data or objects are unsupervised partitioned into a set of meaningful sub-classes, called clusters. Clustering is unsupervised classification. A good clustering algorithm will produce clusters such that objects from the same cluster are "similar" to each other and any two objects from different clusters are "dissimilar" to each other.

There are many clustering methods. They are partitioning algorithm, hierarchical clustering, density based methods, grid based methods and model based methods [24].

Web document clustering is different from classical clustering. For example, as we know, web documents can have several topics; therefore we should put one document to several clusters, instead of just one cluster. Traditional document clustering methods fails to classify web documents.

3.1.1 Traditional Document Clustering Fails

Document clustering is the process in which a collection of text is clustered into groups or clusters that have similar contents. One of the most pervasive document clustering approaches is the Hierarchical Agglomerative Clustering (HAC) algorithm [34]. HAC algorithms are belonging to hierarchical algorithms. These algorithms start with each document in a cluster of its own, iterate by merging the two most similar clusters, and terminate when some halting criterion is reached [28], e.g. when 5 clusters remains. According to different definition of cluster similarity, single-link, complete-link and group-average algorithms are identified. The single-link defines the similarity of two clusters as the minimal similarity of any pair of items, one from each cluster; while complete-link algorithms define the similarity as the maximal similarity. Average similarity of all pairs of items is used as the definition of similarity in the group-average algorithm. These algorithms are not suitable for web document clustering for the following reasons. First, they are slow. Single-link and group-average can be implemented in $O(n^2)$ time, whereas complete-link requires $O(n^3)$ time [28][35]. The proof is given in [35]. Second, HAC algorithms are very sensitive to the halting criterion [28]. Therefore, halting criterion plays an important role in getting good clustering result. For example, if a set of documents belongs to 6

meaningful clusters, when the halting criterion is stop when 5 clusters remain, the resulting 5 clusters for the set of documents could be meaningless. For example, the first document is about Chinese. The second document is about French. The third document is about Japanese. The fourth document is about English. The fifth document is about German. The sixth document is about Malay. Each document belongs to a meaningful cluster. If the algorithm is forced to form 5 clusters for the 6 documents, the resulting 5 clusters are not meaningful to the user. Several halting criteria for HAC algorithms have been suggested, but in practice predetermined constants are typically used [28]. In the Web applications, since the searching results for different queries could be extremely varied and we could not predefine a good halting criterion for such varied documents, poor results are often caused by the sensitivity to halting criterion [28].

Another class of algorithms is the partition algorithms. Most of these algorithms are also known as model-based algorithms as they have a prior assumption as the model describing the data. These algorithms search for the most probable model parameters, given the data and a priori assumptions regarding the model. K-mean [29] is the most basic partition algorithm. K-means is a fast, linear time algorithm [28]. But it has several drawbacks. First, it often terminates at local optimum [28]. A common technique used in algorithms is to make small change and observe whether the result is better or not. If not, the change is reversed and another change is tried. A sequence of such explorations may result in a situation in which no small change will improve the result. This is called a local optimum, since it is still possible that some major change will result in a better outcome. Point B in Fig 9 is a local optimum. Second, number of clusters has to be given in advance [28]. It's difficult to predefine clusters

in web domain. Third, it is unable to handle noisy data and outliers [28]. Web documents contain noisy data such as formatting and navigational information. We know that in HTML there is no clear separation among the content, navigational information, and formatting. Therefore, k-means is not suitable for web document clustering.

Fig 9 Local Optimum

Other methods such as neural models [25] have been used for document clustering. Its long computation time makes it unsuitable for web applications. More recent, fast algorithm, Suffix Tree Clustering [23], suitable for web domain, has been introduced. This method creates clusters based on phrases shared between documents.

3.1.2 Suffix Tree Clustering

The Suffix Tree Clustering (STC) algorithm has four important characteristics [28]. First, in STC, a document can belong to more than one cluster. Most other clustering

algorithms force each document into a single cluster. Second, in STC, phrases are used to identify documents similarities and consequently to construct clusters. In most other clustering algorithms, a document is treated as a bag of words, i.e. the ordering of the words is not considered. Third, STC algorithm is fast and incremental, i.e. suffix tree can be constructed incrementally as documents read in. Forth, the number of cluster is not required to be pre-specified. Therefore, STC is well suited for Web document clustering.

STC is a linear time clustering algorithm (linear in the size of the document set). STC is based on identifying phrases that are common to groups of documents. A phrase is an ordered sequence of one or more words. For example the suffix tree of the sentence "I know you know I know" is shown in Fig 10, and every label in the suffix tree is a phrase and the concatenation of labels on the same path is a phrase too. STC has three logical steps [7] [8]: document "cleaning", identifying base clusters using a suffix tree, and merging these base clusters into clusters. In the document "cleaning" step, the string of text representing each document is transformed using a light stemming algorithm. Sentence boundaries are marked and non-word tokens (such as numbers, HTML tags, and most punctuation) are stripped. The second step, the identification of base clusters, is done efficiently using a data structure called a suffix tree. This data-structure can be constructed linearly and incrementally. Each base cluster is assigned a score that is a function of the number of documents it contains, and the number of words that make up its phrase. The final step of the STC algorithm merges base clusters with a high degree of overlap in their document sets.

3.1.2.1 Suffix Trees

A suffix tree is a data structure. A suffix tree T for an m -word string S is a rooted directed tree with exactly m leaves numbered 1 to m . Each internal node, other than the root, has at least two children and each edge is labeled with a nonempty sub-string of words of S . No two edges out of a node can have edge labels beginning with the same word. The key feature of the suffix tree is that for any leaf i , the concatenation of the edge labels on the path from the root to leaf i exactly spells out the suffix of S that starts at position i , that is it spells out $S[i..m]$ [16]. Fig 10 taken from [28] illustrates the suffix tree of a document which contains string "I know you know I know". Internal nodes are marked as circles, leaves as rectangles. There are six leaves in this example, numbered from 1 to 6.

Fig 10 Example of A Suffix Tree

In a similar manner, a suffix tree of a set of strings, called a generalized suffix tree. A generalized suffix tree T for a set S of n strings S_n , each of length m_n , is a rooted directed tree with exactly $\sum m_n$ leaves marked by a two number tuple (k, l) where k ranges from 1 to n and l ranges from 1 to m_k . Each internal node, other than the root, has at least two children and each edge is labeled with a nonempty sub-string of words

of a string in S . No two edges out of a node can have edge labels beginning with the same word. For any leaf (i,j) the concatenation of the labels on the path from the root to leaf (i,j) exactly spells out the suffix of S_i that starts at position j , i.e. $S_i[j..m_i]$ [16]. Fig 11 taken from [28] is an example of the generalized suffix tree of a set of three documents which contain the following three strings respectively – "cat ate cheese", "mouse ate cheese too" and "cat ate mouse too". There are 11 leaves in this example (the sum of the lengths of all the strings in documents) drawn as rectangles. Internal nodes marked as circles represent base clusters.

Fig 11 Example of A Generalized Suffix Tree

3.1.3 Sample Suffix Tree Clustering Applications

Here I present two applications which using STC: Grouper and Carrot.

Grouper

Grouper [23] was the first clustering engine, developed by graduate student Oren Zamir at University of Washington. Grouper is a document clustering interface to the HuskySearch meta-search service. HuskySearch retrieves results from several popular Web search engines, and Grouper clusters the results as they arrive using the STC algorithm. Grouper partitioned the results into clusters, which contain similar content, compared to showing results in a linear list ranked by relevance or sorted by URL in most search services. By generating qualified clusters with simple descriptions for novice users, Grouper provided an effective way of browsing.

Carrot

Carrot [22] is a meta search engine with clustering capabilities from Poznan University of Technology, Poland. It acts as a filter between a user and a search engine, with goal of enhancing readability and comprehension of search results by the manner of good presentation. In order to achieve that goal, Carrot uses STC technique, making an attempt to group documents with similar subjects into clusters.

3.2 Personalized Information Filtering

Existing personalized Information Filtering systems have some limitations and there is no successful application of multi-agent learning in this field. I thus intend to propose a multi-agent learning approach to personalized information filtering, which

learns user's interests by observing his or her browsing behaviors during the interaction with the system to eliminate the drawback of asking for users' explicit feedback. The need for personalized information filtering and technologies and limitations of existing information systems are discussed in this section.

3.2.1 The Need for Personalized Information Filtering

With the rapid progress of computer and network technology in recent years, information on the Web has been explosively increased. As the volume of information increases, the need for finding more relevant information on the Web is growing. For example, in a non personalized information filtering system such as normal search engine, sometimes the results may be so irrelevant to our interest that we are frustrated. The main reason causing the situation may be that the user is not able to describe his/her interest in a precise manner. A topic has many sub topics. For example, "machine learning" has sub topics such as "research groups of machine learning", "papers on machine learning", "methods of machine learning" and so on. It is not easy for the user to describe their interest precisely. Another reason is that the user may not wish to invest a great deal of effort in creating such a query [13]. In order to improve the user's information services experience, his/her individual interest should be captured by system to optimize the relevance of search results.

3.2.2 Existing Information Filtering Systems

Some approaches have been tried to capture user preferences. I discuss them in this subsection.

Browse [27] is an X windows neural network based USENET news filtering system by Andrew Jennings and Hideyuki Higuchi. It has a neural network model of user's interest. Each message presented to user is classified as either being accepted or rejected. It emphasizes the difference between accepted and rejected articles. One message can only belong to one cluster. It eliminates the degree of how the user is interested in the information.

GroupLens [18][19][27] is an experimental collaborative filtering service based on "Better Bit Bureaus" which is itself a collaborative venture between Paul Resnik of the Center for Coordination Science at MIT and Brad Miller and others at the University of Minnesota. The project began to explore automated collaborative filtering in 1992, but is most well known for its world wide trial of an automated collaborative filtering system for Usenet news in 1996. GroupLens requires users to rate the articles as they are read. The time user spent reading is also stored but information is not used yet. The synergy achieved by having articles rated by many readers is that the rating servers can correlate users with similar preferences. This allows the rating server to hypothesize the rating a user will give to an unread article.

NewT [36][37] is a personalized information filtering system for USENET news, from MIT media technology lab. It uses keyword-based filtering and machine

learning (relevance feedback and genetic algorithms) to personalize the presentation of USENET news. The agents are adaptive as they learn the preferences of the user and adapt as they change over time. The learning mechanism used by the agents is relevance feedback and genetic algorithm. Users are asked to provide positive or negative integer indicating the amount of feedback for documents.

Letizia [26] is a recommend system from MIT media technology lab. The agent tracks the user's browsing behavior -- following links, initiating searches, requests for help -- and tries to anticipate what items may be of interest to the user. It uses a simple set of heuristics to model what the user's browsing behavior might be. Upon request, it can display a page containing its current recommendations, which the user can choose either to follow or to return to the conventional browsing activity. It can only roughly capture user's interest.

WAIR [11][12][38] is a personalized information filtering system from artificial intelligence lab, Seoul National University. It models the problem as TD(0) reinforcement learning. It learns user's preference without user's explicit feedback. WAIR tries to display documents most similar to user profile.

3.3 Agents in the Proposal

In order to provide the user with a more efficient and easy using filtering system, I propose a multi-agent hybrid learning approach. There are three issues I intend to solve. The first is how to capture the user's interest without asking for his/her explicit

rates repeatedly. The second issue is to ensure the captured user profile is reasonably close enough to the user interest. The third is how to converge to user interest quickly.

After analyzing the problem and three issues mentioned in the above paragraph, I identified four kinds of agent in our proposal according to their functionalities. They are User Agent, Information Retrieval Agent, Clustering Agent and Learning Agent.

- **User Agent** is an interface agent and represents a user. User interacts with the system through User Agent; System presents results and information to the user through User Agent. User Agent also observes the user's behavior, i.e. reaction to the displayed information.

Information Retrieval Agent retrieves web-documents from search engine using queries created from user profile, which is a vector of terms representing the user's interest on a specific topic. One user profile is for one topic. Every user may maintain many profiles.

- **Clustering Agent** classifies web-documents retrieved by Information Retrieval Agent into several clusters.
- **Learning Agent** updates user profile according to the user's behaviors observed by User Agent.

3.4 Why Multi-agent Approach?

Why should I use multi-agent approach? The idea is illustrated in Fig 12. A User Agent can communicate with any agent from Information Retrieval Agent layer. Different Information Retrieval Agents may retrieve information from different search

engines. In an analogous manner, an Information Retrieval Agent can interact with any agent from Clustering Agent layer. A User Agent can also communicate with any agent from Learning Agent layer. There are more than one Information Retrieval Agent and more than one Learning Agent serving a User Agent. If some of the Information Retrieval Agents or Learning Agents are busy, jobs can be allocated to fewer jobs queuing agents. In a similar way, an Information Retrieval Agent asks a fewer jobs queuing Clustering Agent to cluster its results. On the flow, all agents need know the User Agent ID they are serving currently. With multi-agent system, the User agent sits in the client and all the other three agents can be spatially distributed. With multi-agent system, each agent plays its role clearly and this enables the flexibility in implementation. Different developers for different agents can choose to implement the agents in different languages. With multi-agent system, the reusability of every agent is improved. Another great advantage of the proposed multi-agent approach is that if one or more agents in the same layer fail, the system will continue to function reliably. In instances of a information retrieval agent failure, the system is designed to reroute tasks to other information retrieval agents.

Fig 12 The Four Agents in Proposed Multi-agent System

Fig 12 The Four Agents in Proposed Multi-agent System

3.5 The Model: Four-Agent System

In this section I show a simple version of the multi-agent system presented in Fig 12. I assume there are only one User Agent, one Information Retrieval Agent, one Clustering Agent and one Learning Agent in our model. The resulting model is given in Fig 14.

Before the overall working procedure starts, there is an optional step---creating a new user profile for a new user interested topic. Topic is created by the user. If the user prefers the multi-agent system to help him/her on getting information he/she really interested for certain topic, the user creates a topic. To the user, creating a topic is to

give a title for the topic. The title may be different from the real meaning of a topic. It is up to the user to give the title of a topic. The system never uses the title for forming searching query. It only uses the terms in the user profile to form search query. To the system, creating a topic is to create a user profile. The title of the topic and topic never change after creation. But the terms in the user profile changes in terms of the terms themselves and their weights. The terms in the user profile represents the user's interest in the topic. For the following searching on an existing topic, the user should tell the agent which topic is the current search relates to. If the user creates a topic, User Agent gets the initial query terms from the user and saves them in a new user profile. If the user searches an existing topic, this optional step is omitted. Procedure starts when Information Retrieval Agent selects highly scored terms from user profile and sends them as a query to search engine. After the search results come back to Information Retrieval Agent, Information Retrieval Agent forwards the results to Clustering Agent. These results are classified into clusters. This clustering result, including how many clusters, which documents belong to which cluster, which common terms represent which cluster, is sent to User Agent. User Agent watches what the user does to the clusters. These are the users' browsing behavior information, including bookmarking, scrolling, following hyperlinks etc. The users' behavior information will be supplied to Learning Agent. Learning Agent will use this information to calculate the user's utility on documents as number. The user's utility plays an important role in updating user profile. Outline of procedure is given in Fig 13. Fig 14 graphically illustrates the model proposed and the communication among agents.

1. User Agent gets the initial query terms on certain topic from the user.
 2. User Agent saves initial query terms as initial user profile.
 3. Information Retrieval Agent generates a query from user profile.
 4. Information Retrieval Agent forwards the query generated in step 3 to search engine.
 5. Information Retrieval Agent retrieves search results from search engine.
 6. Clustering Agent classifies search results retrieved by Information Retrieval Agent into clusters.
 7. User Agent is told with the clustering result.
 8. User Agent presents the result to the user and observes the user's browsing behavior---reaction to the displayed information.
 9. User Agent forwards the user's behavior information to Learning Agent.
 10. Learning Agent calculates the user's utility using his/her browsing behavior information from User Agent. It updates user profile using utility calculated.
 11. Go to step 3.
- (Algorithms inside each agent will be discussed in detail in the following sections.)

Fig 13 Procedural Form of Our Proposed Approach

Fig 14 Proposed Model

3.6 Hybrid Learning Algorithms

I use a hybrid learning approach to the problem of personalized information filtering. My hybrid learning technique is modified Monte Carlo (MC) combined with the features of Backpropagation neural network and Suffix Tree Clustering. In my proposed model, there are no clear probabilities of transition from one state to another state, which rules out Dynamic Programming. Each time a user makes a query, a new episode of learning process starts. My model only learns at the end of each episode since each episode is not long. Obviously, it's not Temporal Difference, since

Temporal Difference is used when each episode is long and learns at the end of each step of episodes.

The goal of my proposal is to find an optimal policy, same as that of MC. State is defined as the population of documents which are within the range of Google's search. Since World Wide Web (WWW) is dynamic and documents are updated (including update existing documents, remove existing documents from WWW and add new documents to WWW) all the time, I assume the chance for agent visiting the same state for a second time is approximately zero. Agent finds itself in a different state each time a new search processed. Fortunately, at the moment of searching, there is only one single state. That means, the agent only explores one state in each episode. In episodic reinforcement learning problem, states are considered as status of environment from the beginning to ending of an episode. For example, in my application, when the user is not issuing a query, and agent is doing nothing, the idle state of agent; is not considered as one state of our system. Agent's action is to display relevant documents to the user. Reward of the action is the utility of the user. This utility is expressed in a number. Policy is used to guide the agent to select an action, which will maximize the user's utility. In my application, policy is to generate query from user profile.

When I say "fixed user interest" means the user's interest on a specific topic never changes over time. "not fixed user interest" indicates the user's interest on a specific topic may change over time. For example, if a user searches a topic "fruit", he is always interested in only "apple", this is a fixed interest. If another user searches the topic "fruit", for the first few days the user is always interested in only "apple", but after the few days he is totally or partially interested in "orange", this is an not fixed interest.

Two different algorithms are presented for situations in which the user's interest on specific topic is fixed and not fixed, respectively. The former is shown in Fig 15(a), latter in Fig 15(b). The algorithms are based on MC algorithm and combined with the features of Backpropagation neural network and Suffix Tree Clustering. My algorithms differ from classic Monte Carlo method (given in Fig 5) in another two aspects. First, my action value is equals to the current return. As we know, action value is calculated as average returns in classic MC. In my application, since with the assumption of the chance for visiting the same state for a second time is approximately zero, average return will be equal to the current return. Second, my action values are not stored in look up table. In classic MC, action values are kept in look up table. In my application, these values are not kept. Agent may have many actions to select in one state. For example, agent may display documents as result of searching terms a,b, agent also may display documents as result of searching terms b,c. But when agent selects the action according to the current profile, this action will lead to the maximum value of state-action pair if the profile is close to the user's interest. Since we are trying to find action leading to maximum user's utility in a state and agent is designed to select action according to user profile, it's not necessary to find, calculate and store values of state-action pairs.

There are two steps in each algorithm. First, initialize action value $Q(s,a)$ arbitrarily and set user profile to initial user's query. When a user tells User Agent he/she wants to create a new user profile for a new topic, he/she will be asked to input initial user's query. Second, a new episode starts when query executed, agent's action is to display relevant documents to the user. Agent chooses the action which uses query created

from highly scored terms in user profile to display documents to the user. STC will classify the documents retrieved into clusters. The user's utility on all these clusters are the reward of this action, denoted $Q(s,a)$. User profile is updated using the user's utility. If the user's interest is fixed, the process will be terminated when terminate condition reached. The term "terminate condition" in Fig 15(a) refers to the moment when rewards of each cluster are roughly equal provided that the rewards of each cluster are greater than zero. "roughly equal" means the difference among the rewards is within a tolerant range, such as 0.5. Rewards of each cluster are greater than zero is to guarantee that the user did view the pages. If the user did not read any pages, of course the rewards of each cluster are equal because they are also zero. If reward of each cluster is roughly equal, then the user is roughly equally interested in the clusters. At this time, the system can merge clusters and no longer update user profile, because the user's interest is found already. If the user's interest is not fixed, the process will be repeated forever.

Our algorithm (interest fixed)	Our algorithm (interest not fixed)
<ol style="list-style-type: none"> 1. Initialize, for all $s \in S$, $a \in A(s)$: <ol style="list-style-type: none"> 1.1. $Q(s,a) \leftarrow \text{arbitrary}$ 1.2. Initial user profile $\leftarrow$ initial user query 1.3. $\text{Returns}(s,a)=0$; 2. Repeat until terminate condition <ol style="list-style-type: none"> 2.1. Episode starts when query made 2.2. Select action a according to user profile 2.3. $\text{Returns}(s,a)=0$; 2.4. For each cluster given by STC classifies resulting documents of action a in state s <ol style="list-style-type: none"> 2.4.1. $R \leftarrow$ reward of cluster from BP NN 2.4.2. $\text{Returns}(s,a) += R$ 2.5. $Q(s,a) \leftarrow \text{Returns}(s,a)$ 2.6. $\pi(s) \leftarrow$ updated user profile 	<ol style="list-style-type: none"> 1. Initialize, for all $s \in S$, $a \in A(s)$: <ol style="list-style-type: none"> 1.1. $Q(s,a) \leftarrow \text{arbitrary}$ 1.2. Initial user profile $\leftarrow$ initial user query 1.3. $\text{Returns}(s,a)=0$; 2. Repeat forever <ol style="list-style-type: none"> 2.1. Episode starts when query made 2.2. Select action a according to user profile 2.3. $\text{Returns}(s,a)=0$; 2.4. For each cluster given by STC classifies resulting documents of action a in state s <ol style="list-style-type: none"> 2.4.1. $R \leftarrow$ reward of cluster from BP NN 2.4.2. $\text{Returns}(s,a) += R$ 2.5. $Q(s,a) \leftarrow \text{Returns}(s,a)$ 2.6. $\pi(s) \leftarrow$ updated user profile
(a)	(b)

Fig 15 Our Proposed Algorithms

I evaluate hybrid learning algorithms' correctness. If user profile is the user interest, user utility will be maximized and the goal of our system is achieved. Therefore, the system only need to make sure user profile is as close as possible to user interest.

If the system can get truthful user's behavior information, reward of documents will be calculated correctly and user profile will be properly updated and growingly closer

to the user's interest. Even if the system gets fake user's behavior information, user profile will be still growingly closer to the user's interest. The proof is outlined in Fig 16.

- 1 If the system can get the truthful user's behavior information, reward of documents will be calculated correctly and user profile will be properly updated, which results a better user profile closer to the user's interest.
- 2 Even if the system gets fake user's behavior information, user profile will be still growingly closer to the user's interest. Assume terms a, b, c are the user interest.
 - a) Assume terms a, b, c are in original user profile. When the system get fake user's behavior information (e.g. the system think the user is reading the document while in fact he/she leaves with the document open), reward of documents will be calculated incorrectly and user profile will be improperly updated. Assume the system changes the terms weights of a, b, c and inserts a new term d to the original user profile, as consequence of the improper update. When user searches this topic again, if term d is weighted a lot, the system creates a query which is a, b, c, d not $a+b+c+d$ (a, b, c, d means a or b or c or d , $a+b+c+d$ means a and b and c and d). That means the user still has the chance to correct the user profile. If term d is weighted a little, the system creates a query as if term d was not in user profile.

b) Assume terms a, b, c are not in original user profile. When the system get fake user's behavior information, reward of documents will be calculated incorrectly and user profile will be improperly updated. Assume the system changes inserts a new terms d, e, f to the original user profile, as consequence of the improper update. When user searches this topic again, the system creates a query which is d, e, f . However, the user should not be always missing the relevant pages. The next time when the user browses the relevant page, the user profile will be inserted into one or all of the terms of a, b, c .

Therefore, The proposed system is not able to capture the user's interest for just one search. The proposed system captures the user interest in terms of a process. User profile will be still growingly closer to the user's interest.

Fig 16 Algorithm Reasoning

3.7 The JADE-based Multi-agent System

I adopt Java Agent DEvelopment Framework (JADE) [15] as a software development and runtime framework, which is fully implemented in Java language. JADE simplifies the implementation of multi-agent systems through a middleware that claims to comply with the FIPA specifications. FIPA, The Foundation for Intelligent Physical Agents (FIPA), is a consortium formed in 1996 to produce standards for the interoperation of heterogeneous software agents [48]. FIPA specifications are a

collection of such standards. The agent platform can be distributed and the configuration can be controlled via a remote graphic user interface. In JADE, agent behaviors are implemented using behavior classes. JADE also implemented non-preemptive multitasking of executing an agent's different behaviors.

The communication among the agents in the model is accomplished by exchanging messages expressed in the FIPA-ACL language. This is different from [14] in which messages are expressed in the KQML language.

JADE provides a conventional way of constructing of agents and communication among agents. Basic steps of building agents, sending and receiving messages are summarized as follows.

Building an agent:

1. Define an agent class which extends from `jade.core.Agent`,
2. Process startup parameters using optional method `Object[] getArguments()`
3. In mandatory method `setup()`
 - 1) Register content languages
 - 2) Register ontologies
 - 3) Start behaviours

Sending a message:

1. Create classes, which represents content of your message
2. Create ontology that describes these classes
3. Create instance of class representing your message content

4. Create instance of class ACLMessage
5. Fill set of receivers into ACLMessage
6. Fill name of ontology and language into ACLMessage
7. Create instance of ContentManager class
8. Use method Content Manager.fillContent(ACLMessage m, ContentElement content) to format content
9. Use method send (ACLMessage m) to send message to receivers

Receiving a message:

1. Create behavior for receiving messages
2. Receive a message (if any) from the message buffer in the mandatory method action() of the behavior

```
1) Build instance of the content message using  
ContentElement ce =  
ContentManager.extractContent(ACLMessage m)
```

- 2) Process the content class

```
if (ce instanceof AObject) {  
    AObject o = (AObject) ce;  
}
```

- 3) If necessary, send back some response

3.8 The Multi-agent System Design

3.8.1 User Agent

User Agent is an interface agent. Each user has his/her own user agent. User agent should be able to accept queries from a user and present results to the user. It is supposed to be able to "watch" and take detail note of what the user is doing. For example, if the user is reading a document, User Agent should be able to take note of the ID or name of the document, how long the user is keeping reading it, whether the user bookmarks it, whether the user follows hyperlinks inside it, whether the user scrolls mouse and whether the user saves it. User agent could communicate with any Information Retrieval Agent the user likes. User profiles could be saved and be displayed to the user through User Agent. A Java browser, a Java Swing Frame, meets the requirement.

Not all the user browsing behaviors indicate the user's interest. Sometimes, user takes a tour in search result pages to either check whether it is a relevant page or out of curiosity, he/she may visit a page. This is not an accurate ranking. Sometimes, even user picks few of the results randomly and may escape a relevant page. However, the system still gradually approximates the user profile as close as possible to the user interest. Reasons lie in two points.

First, a serious user for expected topic is assumed. He/she really hope the system will do some help to locate the information he/she really interested. The term "serious

user" means a user who is searching information on movie does not browsing the information on music. Even the user does browse information on music while he tells the agent he is searching movie information, the system is still capable to capture the movie information gradually. This is explained in the second point.

Second, the system captures the user interest in terms of a process. If the user misses the relevant page and (or) browses something else, the resulting user profile is not matching the user interest. However, the user should not be always misses the relevant pages. The next time when the user browses the relevant page, the user profile will be closer to the user interest. Finally, the system is able to capture the user interest close enough to satisfy the user. At that time, the user will not be bored by the overwhelmed information on the internet.

3.8.2 Information Retrieval Agent

The reason why Information Retrieval Agent is separated from user Agent lies in the consideration of implementation flexibility. Information Retrieval Agent can be implemented in any language the developer like since the design take this issue into account. The User Agent resides in the client pc which usually is not powerful enough, and User Agent already designed to perform to capture user behaviors, so the information retrieval task is designed to complete by the Information Retrieval Agents reside in the Server. Information Retrieval Agent retrieves web-documents from search engine using queries created from user profile. All information of the retrieved web-documents, such as URL of document, summary of document, date of document etc, should be able to be handled freely and quickly by Information Retrieval Agent. Normally, parser can parse the documents to get information the agent asks for, but

even a complex parser does not guarantee all html documents can be parsed. In our design, we tap into Google web service. With Google web service, the results come back in the form of SOAP message [9], an XML based mechanism for exchanging information. SOAP stands for Simple Object Access Protocol, which is used for information exchange over HTTP. More information can be found at [56]. The SOAP messages can be manipulated easily and freely. Google API [9], programmer's guide to tap into Google web service, is available from Google web site as well. Tapping into Google web service fully meets the requirements of Information Retrieval Agent.

3.8.3 Clustering Agent.

Clustering Agent classifies web-documents retrieved by Information Retrieval Agent into several clusters. Each cluster is a subtopic of the topic the user is searching. One document can belong to several clusters, since it may have more than one subject. Web document clustering should be fast, i.e. linear time, online, tolerant to noisy data and does not require predefining number of cluster. Here the noisy data refers to the navigational information and formatting information in the web document since these information can no be clearly separated from the content of the web document. Suffix Tree Clustering is most suited for web-documents ,clustering. Therefore, the Clustering Agent employs Suffix Tree Clustering.

3.8.4 Learning Agent

Learning Agent gets the user's browsing behavior information on each document from User Agent. How many aspects of user browsing behaviors and what aspects of user browsing behaviors to be used is always a design issue. I attempted to use six aspects of user browsing behavior. I believe the six aspects of user browsing behavior form reasonable a starting point for capturing user's interest. Other aspects can be added on.

The six aspects of user browsing behavior numbered from A to F are described below:

- A. Whether the user bookmarks the document. Bookmarking a document shows the user is really showing his/her interest in the document. Bookmarking a document indicates the user will visit the same document again in later stage.
- B. Whether the user follows hyperlinks inside the document. If the user follows the hyperlinks inside the document, the user tries to find out more about the information inside the document. This shows the user has certain level of interest in the document.
- C. Whether the user scrolls. The user scrolls indicates the user reads through the document. The scrolling shows the user has certain interest in the document.
- D. Whether the user opens the document. If the user opens the document, the user is attracted by the title or snippet of the document. This indicates the user has some interest in the document.

- E. Whether the user saves the document. Here the term "save" refers to either the user prints out the document or saves the document on the hard drive. If the user saves a document, the user will probably open and study the document again. This indicates user is attracted by the content of the document.
- F. And how long the user reads the document. If the user browses properly, I can assume the longer the user spends on a document, the more interest he shows on the document.

Reward of each document, the user's utility on the document, will be computed using a three layer Backpropagation (BP) Neural Network. Generally BP Neural Network is a three layer neural network. In the system, I adopt the general design of BP neural network. There are researches which add more hidden layers to the BP neural network. However, in the proposed system, three layer BP neural network is shown to be good enough for my purpose. BP can approximate non-linear function. Six input neurons in the first layer, one output neuron in third layer. The six aspects of user browsing behavior numbered from A to F are inputs for the network, reward is output.

I have my own algorithm to update user profile. The user's utility on certain document j , r_j , can be calculated using BP neural network. User's utility on certain document represents how much the user is interested in the document. The user's utility on cluster i , R_i , is the accumulation of all r_j s for j inside cluster i . For each cluster i , for each term k in user profile, if term k is a common term of cluster i , term weight of term k is updated. There are two cases for updating: if R_i is equal to zero, that means the user is not interested in this cluster, term weight is set to zero; if R_i is greater than

zero, term weight is updated by adding αR_i , where α is learning rate. When term weight is reduced to pass certain threshold, it can be removed from user profile. If any common term of top scored cluster is not in user profile, it will be inserted into user profile. By this algorithm, the system can insert new terms into user profile, remove terms from user profile and update terms' weights. The algorithm is outlined in Fig 17, where R_i denotes the user's utility on cluster i and r_j denotes user utility on certain document j .

1. Get user's browsing behavior information of each document from User Agent.
2. for each cluster i , $R_i=0$
 - 2.1. for each document j inside cluster i
 - 2.1.1. calculate user's utility r_j using BP Neural Network
 - 2.1.2. $R_i += r_j$
3. for each cluster i
 - 3.1. for each term k in user profile
 - 3.1.1. if term k is a common term of cluster i

If $R_i = 0$, then set $W_k = 0$

else $W_k = W_k + R_i$
4. For top scored cluster, extract common terms and insert into user profile if the common term is not in current user profile.

Fig 17 User Profile Updating Algorithm

3.9 Discussion and Comparison with Existing Approaches

After presenting the approach and design of a multi-agent system, a few comparisons with traditional approaches are made in terms of three issues to be addressed.

The first issue is how to capture the user's interest without asking for his/her explicit rates repeatedly. User's explicit rates are rates from user about the document targeting. For example a user may mark a document as 1 to 5 in terms of the document's relevance to the user's interest. The agent observes users' reaction on the displayed documents. Users' behavior information is used to calculate users' utility. User profile is updated using user's utility. This is the way the agent captures the user's interest without asking for his/her explicit rates repeatedly. Modified Monte Carlo and supervised BP network in my approach play the most important roles in solving this issue. The most distinctive difference between proposed approach and NewT, GroupLens and Browse is that this approach does not ask for user's rates repeatedly. NewT, GroupLens and Browse elicit for user's explicit rates for documents.

The second issue is to ensure the captured user profile is reasonably close enough to the user interest. The proposed system is not able to capture the user's interest for just one search. The proposed system captures the user interest in terms of a process. If the user misses the relevant page and (or) browses something else, the resulting user profile for this visit is not matching the user interest. However, the user should not be always misses the relevant pages. The next time when the user browses the relevant

page, the user profile will be closer to the user interest. Gradually, the system is able to capture the user interest close enough to satisfy the user. At that time, the user will not be bored by the overwhelmed information on the internet. Letizia only captures user's preference roughly. Browse treats document as either accepted or rejected and eliminates the degree of how the user is interested in documents. Browse does not capture user preferences precisely.

The third issue is how to converge to user interest quickly. The proposed approach converges to the user's interest more quickly than WAIR and NewT. The proposed approach is quite similar to WAIR. It models the problem as TD(0), I use modified Monte Carlo method combined with supervised learning and unsupervised learning. Three actions in updating user profile are updating, inserting and removing. WAIR inserts all the terms in a document into user profile because WAIR does not use clustering agent and nowhere else show it can get only the user interested terms, my system only inserts the user interested terms. Therefore, my approach converges to the user's interest more quickly. WAIR did not model the problem clearly as TD(0) learning, I model the problem clearly as Monte Carlo in section 3.6. NewT also assumes the user's feedback applying to all the terms in a document when user rates the document. It inserts all terms into user profile if the user rates the document highly. This slows down the convergence. Suffix Tree Clustering algorithm in my approach plays a most important role in solving this issue.

Chapter 4 GreenLeaf: A Prototype

In this chapter, I describe implementation of the four agents in the proposed model. GreenLeaf is the name of the system. First, Interface agent is implemented as a Java browser. Next, Retrieval Agent taps into Google web service to retrieval information. Google is used in the system is not because it is assumed to provide best search results. Google is used because of its web service which the Retrieval Agent can easily tap into. Third, Clustering Agent clusters information into clusters using STC. Finally, Learning Agent is successfully implemented to calculate the user utility and update user profile.

4.1 User Agent

A Java Browser is implemented, which is a JFrame. Refer to Fig 18. A user can initialize the Information Retrieval Agent address in URL textfield in top panel. Backward and forward functions will be implemented. Left panel is used to store user profiles of topics. Right Panel presents information. The user can click the root of the tree to add a new user profile of a new topic. If the user searches an existing topic, just click the node in the left in Fig 18, which is representing the topic, to let the agent

know the topic the user is searching. In this JFrame, the user's behavior information can be captured.

Fig 18 User Agent

4.2 Information Retrieval Agent

Information Retrieval Agent was implemented to tap into Google Web Service. It can get information easily from Google. Implementation was done by referring to Google web API [17]. Fig 19 shows the first 10 results of searching “multi-agent”. Estimated number of result is 232000. With web service, we can easily manipulate information to display what we want to show. In Fig 19, we display title, URL and snippet of documents. The topic title displayed in Fig 19 is “topic 1”, but a more meaningful topic title should be given such as “multi-agent”.

Fig 19 Tap into Google Web Service

4.3 Clustering Agent

Clustering Agent employs Suffix Tree Clustering algorithm. Clustering implementation is based on *Carrot's* implementation. Clustering result of searching "machine learning" is given in Fig 20. Common terms of the first cluster are "computer, "science""collection of computer Science Bibliographies", "Machine Learning Online" and ""Kluwer". This cluster is quite different from the third cluster whose common terms are "research", "Group", "Members" and "Machine Learning Research Group". This difference helps the user to focus on his/her real interest in machine learning.

Fig 20 A Sample Cluster View of Clustering Results

When the user clicks "Display all" of the first cluster; in Fig 20, an inside cluster view given in Fig 21 will be presented to him/her. All the documents shown in this view are classified to the first cluster in Fig 20. Following the links, the user can read the full text,

Fig 21 A Sample inside Cluster View of Clustering Results

4.4 Learning Agent

Implementation of learning Agent includes two parts as indicated by algorithm given in Fig 17. One is implementation of the neural network and trains the network to calculate the user utility. The other is to update user profile. In the first part, Backpropagation network is used to compute user utility on documents. This BP network has been implemented as show in Fig 23 and Fig 2.4, B P is trained before the user starts to use the system. It is trained to let the system know the relationship between the user behaviors and the user utility. When user visit a document the user agent shown in Fig 22 can get the user browsing behaviors and can get user rates for the document as 1 to 5. The user browsing behaviors will be the input for the BP Neural Network and the user rates is the output of the Neural Network. This is the source of the training data.

Fig 22 User Agent With Explicit Rate Function

```

class SigmoidalNeuron{
    private double inputSignal, outputSignal;
    private SigmoidalFunction f;
    public SigmoidalNeuron(){
        f = new SigmoidalFunction(); ...
    }
    public double calc(double input){
        //given input to this neuron, calculate output the neuron.
    }
    public double getOutput() { //return output; }
    public double getInput() { //return input; }
}

class SigmoidalFunction{
    public double calc(double x){
        return(1.0/(1.0 + Math.exp(-x)));
    }
    public double differential(double x){
        return(calc(x)*(1.0 - calc(x)));
    }
}

```

Fig 23 Coding BP Network 1

In the BP network, there are 6 input neurons, 1 bias neuron, 1 output neuron and 4 hidden neurons. Each input, hidden, output neuron is an object of SigmoidalNeuron. Refer to Fig 24.

```

inputNeuron = new SigmoidalNeuron[numberOfInput + 1];
for(i = 0; i < numberOfInput + 1; i++) {
    inputNeuron[i] = new SigmoidalNeuron();
}

```

```

hiddenNeuron = new SigmoidalNeuron[numberOfHidden + 1];
for(i=0; i < numberOfHidden + 1; i++) {
    hiddenNeuron[i] = new SigmoidalNeuron();
}
outputNeuron = new SigmoidalNeuron[numberOfOutput];
for(i = 0; i < numberOfOutput; i++) {
    outputNeuron[i] = new SigmoidalNeuron();
}

```

Fig 24 Coding BP Network 2

Fig 25 shows the NN network. In the first layer, there are 6 input neurons and 1 bias neuron. 4 hidden neurons reside in the second layer, and 1 output neuron.

Fig 25 BP Network Architecture

Fig 26 gives the data set that is used to train the above BP network. The "target" in the data set is the rates user gives. User can choose the rates among 1, 2, 3, 4 and 5. And the "learningData" in the data set is the user behaviors the User Agent can capture.

The first set of input data, {1.0, 1.0, 1.0, 1.0, 1.0, 20 } means user bookmarks the document (A), user follows hyperlinks inside the document (B), the user scrolls (C), the user opens the document (D), the user saves the document (E) and the user reads the document for 20 seconds (F). The user's utility on the document, the output of the network, is 5.0 points, this 5.0 points is given by user. The second set of learning data, { 1.0, 1.0, 1.0, 1.0, 1.0, 15 } means similar with the first of learning data, except that this time the user reads the document for 10 seconds. The user's utility on the document, the output of the network, is still 5.0 points. The fourth set of learning data, { 1.0, 1.0, 0.0, 1.0, 1.0, 15 }, indicates the user does not scroll, the corresponding user utility is 4.0.

The value for A, B, C, D, E aspects of the user browsing behavior is either 0 or 1. The value for F aspect of the user browsing behavior should be greater than or equal 0.

The last one in an input data set is the time user spends on reading the current page. Since user knows he/she is training the system to function well and he/she is assumed to cooperate, we do not give any limitation on the time user spends on the page.

```
hiddenNeuron = new SigmoidalNeuron[numberOfHidden + 1];
for(i=0; i < numberOfHidden + 1; i++) {
    hiddenNeuron[i] = new SigmoidalNeuron();
}
outputNeuron = new SigmoidalNeuron[numberOfOutput];
for(i = 0; i < numberOfOutput; i++) {
    outputNeuron[i] = new SigmoidalNeuron();
}
```

Fig 26 Data Set

The data set given in Fig 26 is small, but it is typical, and here we only show the idea of how the system can be built. Therefore, no large training data sets were used for experiment. In fact, for different user, since user browsing behaviors and user utilities are different, the training data sets can be largely varied.

The convergence threshold is set to 0.2. When the Neural Network passes the threshold, the user agent will show a message to the user saying the training is completed. The source code for the constructing and training the BP Neural Network is appended in appendix. Fig 27 shows the result of the training. Applying the above learning data, the weights of all the links are learned. With all the link weights, the BP network can calculate user utility on every document.


```

C:\WINDOWS\System32\cmd.exe
dif=1.0045132710197935
Weight of hidden output link[0][0]=2.846778744969624
Weight of hidden output link[1][0]=0.9525058508026633
Weight of hidden output link[2][0]=-0.13325345031083788
Weight of hidden output link[3][0]=1.489198801419094
Weight of hidden output link[4][0]=2.5709908992627346
Weight of input hidden link[0][0]=-0.9264096317025363
Weight of input hidden link[1][0]=0.19809300896461737
Weight of input hidden link[2][0]=0.8814691563306503
Weight of input hidden link[3][0]=-0.9388231970339322
Weight of input hidden link[4][0]=-0.32574713098666414
Weight of input hidden link[5][0]=0.6790483626853713
Weight of input hidden link[6][0]=0.9511083377633359
Weight of input hidden link[0][1]=0.5801773167897073
Weight of input hidden link[1][1]=-0.8214494199980872
Weight of input hidden link[2][1]=-0.33633903688735567
Weight of input hidden link[3][1]=-0.7770030224212594
Weight of input hidden link[4][1]=0.044049690712905626
Weight of input hidden link[5][1]=-0.6849139545432631
Weight of input hidden link[6][1]=-0.7787042155370943
Weight of input hidden link[0][2]=-0.7158576389901814
Weight of input hidden link[1][2]=0.5085087892776221
Weight of input hidden link[2][2]=-0.6147828904272942
Weight of input hidden link[3][2]=-0.6482314540926938
Weight of input hidden link[4][2]=0.8456644652077606
Weight of input hidden link[5][2]=-0.6957644869813
Weight of input hidden link[6][2]=-0.8345113160422394
Weight of input hidden link[0][3]=-0.05323309940787704
Weight of input hidden link[1][3]=-0.5974225541755965
Weight of input hidden link[2][3]=-1.0361432569726028
Weight of input hidden link[3][3]=-0.6818952088159158
Weight of input hidden link[4][3]=0.15282158885065258
Weight of input hidden link[5][3]=-1.576088646591696
Weight of input hidden link[6][3]=-0.9863385673993473
*****
dif=0.004593742907020748
dif=0.004717759484117345
training finished.

```

Fig 27 Weights of All The Links

For the second part, the following is the example of updating user profile. Fig 20 shows 3 clusters of searching results of "machine learning". Fig.20 is a result given by the clustering agent. Common words of the first cluster are computer, science, Collection of Computer Science Bibliographies, Machine Learning Online, Kluwer. Common words of the second cluster are Application, Methods, Applications of Machine Learning, list of Machine Learning resources, Concept Book Summary Machine Learning and Data. Common words of the third cluster are research, Group, Members, Machine Learning Research Group. Let's assume the original user profile contains words and corresponding weights as given in Fig 28.

Fig 28 Original User Profile

Now, the system has all the user browsing behaviors information, and the system can use the above trained BP network to get the user utility for all the documents. For each cluster, the system sums up the user utilities on documents under this cluster. Assume a user utility for the first cluster is 0, 25 for the second cluster and 5 for the third cluster. Since "Collection of Computer Science Bibliographies" is a common term of the first cluster which is zero rewarded, the weight of the term will be set to zero and can be removed from user profile. "Machine Learning Research Group" is a common term of the third cluster which is rewarded 5 points, the weight of the term will be increased by 5. Same for "Application", the term weight will be increased by 25 points. And all the common terms of the second cluster, which is the top scored cluster, will be inserted into the user profile. The updated user profile is given in Fig 29.

Fig 29 Updated User Profile

4.5 Evaluation

Ideally, an experiment with a reasonable scale of users such as 20 users should be conducted. Due to constraints of resource limitation of a student project, I try to build the features in the design and the algorithm.

As described previously, the proposed approach is efficient in terms of capturing the user's interest without asking for his/her explicit rates repeatedly. Learning Agent calculates users' utility and updates user profile transparently to users. As stated in section 4.1, the user agent is my own Java browser, which is able to capture user behavior inside the browser instead of asking for user rates repeatedly.

Gradually, the system is able to capture the user interest close enough to satisfy the user. The proposed system captures the user interest in terms of a process. If the user misses the relevant page and (or) browses something else, the resulting user profile for this visit is not matching the user interest. However, the user should not be always misses the relevant pages. The next time when the user browses the relevant page, the user profile will be closer to the user interest. Finally, the system is able to maintain a user profile reasonably close to the user interest. The example in section 4.4 shows that how the system calculate the user's utility and update user profile to show user's interest. It indicates that the proposed approach can maintain a user profile closely to the users' interest.

The proposed approach uses Clustering Agent to get the user interested terms and inserts them into the user profile. But in WAIR, there is no clustering agent in the

design and nowhere shows how the design can get only the user interested terms and insert into the user profile. The proposed approach converges to the user's interest more quickly than WAIR and NewT in theory. However, no data from WAIR and NewT are available, no specific comparison can be made.

4.6 Summary

In this chapter, I have described implementation of the four agents in my proposed model. Interface agent is a Java browser. Retrieval Agent taps into Google web service to retrieval information. Clustering Agent groups information into clusters using STC. Learning Agent is successfully implemented to calculate the user utility and learned to update user profile.

Chapter 5 Competitive multi-agent Learning

In previous chapters, I considered the multi-agent learning in a cooperative setting, now I turn my attention to a competitive environment and study the technique and algorithm in this setting. I use an example called Chinese Checkers to demonstrate the learning algorithm and theory. It starts with a review of the history of Chinese Checkers followed by a description of the game. Next, the rules of the game are described. The discussion on the difficulty of designing an agent to play the Chinese Checkers is followed. Next, the design of a learning agent is presented. Lastly, a program of playing Chinese Checkers is implemented.

5.1 Description of The Game

Chinese Checkers game is a game in which players move round marbles on a board with holes, where the marbles can rest. Chinese Checkers board is a six-pointed star shaped board as shown in Fig 30 [55]. Chinese Checkers is played by 2 to 6 players. The board can be viewed as six triangles attached to a hexagon. The six triangles are color marked in Fig 30 and the hexagon is in the center. Except holes on the edge,

each hole on the board is connected to six adjacent holes. Each hole is in the center of a hexagon formed by the six adjacent holes.

To begin a game, each player puts his 10 same-colored marbles in the 10 holes of one of the six triangular areas. We call the triangular area as that player's home position. The triangular area opposite a player's home position is the player's goal position. For example, if player A's home position is the yellow triangle area, then the goal position of player A is the blue triangle area. The player who first finishes moving his marbles from home position to his goal position is the winner.

Fig 30 Chinese Checkers Board

5.2 Rules of The Game

A move in Chinese Checkers can be either a non-jump move or a sequence of contiguous jumps. Rolling a marble from one hole to an adjacent hole is a non-jump move. For example, in Fig 31 the marble marked as 1 can roll to hole p1. This kind of roll is considered as non-jump move. Jumping over an adjacent marble, of any color, into an unoccupied hole, collinear with the first two, is considered a jump. A single

jump move can consist of several contiguous jumps. For example, in Fig 31, the marble marked as 3 can jump over marble 2 to hole p1 and then jump over marble 1 to hole p4. This is considered as one jump. The player can not make a non-jump move and a jump move at the same turn, When the player has moved one of his marbles, the turn passes onto the next player, Unlike chess, the player can never remove any game pieces from the board [53][54],

Fig 31 Marble Move

5.3 Design A Learning Agent

We know that in reinforcement learning, there must be a return after an action taken, For Chinese Checkers, if we know the accurate return of taking an action, we can design an agent to play Chinese Checkers, So, my approach here is to learn a heuristic evaluation function. In the rest of this section, I will discuss the factors that affect the heuristic value and formulate heuristic function.

5.3.1 Evaluation Function

State of the game is the state of the game board, the current positions of the marbles on the game board. The action is the movement of a marble. There are several actions a player can take in a particular state. Player has to know the result of an action in the current state before he can decide which action to make. A human player can do this easily but it's not easy for a computer agent.

We should formulate an evaluation function, or heuristic function to evaluate the value of a state so that the agent will know which action to choose among the possible moves. For Chinese Checkers, there are many factors we can take into account in the evaluation function. For this report, the following factors are considered.

First, one problem in Chinese Checkers is trailing marbles. If one or two marbles left behind, they will need much more moves than the number of moves they need when move together with the rest of the marbles to reach their goal positions. Therefore, this must be taken into account in the evaluation function.

Second, we should consider the distance of marbles' positions to the center of the board. It is easier and faster to reach the goal for the marbles in the middle of the board than at the edges.

Third, it is strategically important to make the marble jumps several times in a jump move. Long jumps make faster move to goal position. Therefore, the ability to recognize the chance to make a long jump is useful.

Forth, although the ability to recognize the opportunity to make a long jump is critical, the ability to create such opportunity for the player itself is even more useful.

Fifth, when the player creates opportunities of making long jumps, he may introduce opportunities for the opponent at the same time. If the move introduces such opportunities for the opponent, it's not a good move.

Sixth, we should also consider the distance of the marble to the goal position. This is reasonable. The longer the distance to your goal, the worse the situation is. For Chinese Checkers, the longer the distance to the goal position, the worse the move is.

5.3.2 Learning Algorithm

As discussed in the above subsection, there are six factors we should take into account in the heuristic function. The overall heuristic function is a weighted combination of all the six factors.

$$h(n) = w_1 h_1(n) + w_2 h_2(n) + w_3 h_3(n) + w_4 h_4(n) + w_5 h_5(n) + w_6 h_6(n)$$

The agent's task is to learn the values of weight w_1 , w_2 , w_3 , w_4 , w_5 , w_6 . A temporal difference learning is applied. The agent starts with some arbitrary values of the six weights. Play the game again and again. The weight update function is as follows.

$$w_1 \leftarrow w_1 + a[r + \gamma \max h(n+1) - h(n)] * \delta h / \delta w_1$$

It's a variation of the original Q-learning. Since we should distribute the change among all the weights, $\delta h / \delta w_1$ is multiplied. The update functions for other weights are similar as that of w_1 . For example, the update function for w_2 is as follows.

$$w_2 \leftarrow w_2 + a [r + \gamma \max h(n+1) - h(n)] * \delta h / \delta w_2$$

Similarly we can write

$$w_m \leftarrow w_m + \alpha [r + \gamma \max h(n+1) - h(n)] * \delta h / \delta w_m$$

where m is from 1 to 6.

5.4 Implementation

My code allows only two players. As given in Fig 32, each player will hold yellow marbles and blue marbles respectively.

Fig 32 TheGame Board Allowing Two Players

For discussion, I use the game board show in Fig 33 to explain the various ways of implementation of the board. Both of the ways of game board implementation in Fig 34 have 120 positions. The positions are numbered from the upper left corner to the bottom right corner. Each position can be one of four possibilities: outside the board, empty, occupied by a yellow marble or occupied by a blue marble. This state space is much larger than our implementation of the game board shown in Fig 33. In Fig 33, there are 64 positions, and each position can be one of the three possibilities: empty,

occupied by a yellow marble or occupied by a blue marble. And for the real implementation of the game board, shown in Fig 32, we have only 25 positions.

Fig 33 The Implementation of The Board

Fig 34 Ways of Game Board Implementation

After we have the game board in hand, we can start implement the learning of the heuristic function. I use a simplified version of the heuristic function, which is

$$h(n) = w_1 h_1(n) + w_3 h_3(n)$$

If this simplified function works, then the complete heuristic function should also works and the latter should give more accurate heuristic. Just a reminder, $h_1(n)$ is calculating the bad effect from a trailing marble, $h_3(n)$ is encouraging the long jumps. If n is inputted, what is of output of functions $h_1(n)$, $h_3(n)$? The detail designs of the two functions are described below.

It requires more efforts to move the trailing marble. How to let the computer know which one is trailing marble, which one is not? There must be some standards for the computer to judge. I define a reasonable standard here. When a marble's sum of x-axis and y-axis is 4 less than the most nearest marble which is nearer to the goal position, I consider this marble and all the marbles behind this marble as trailing marbles. But in my implementation the agent only check whether the last marble is a trailing marble. That means if the sum of the last marble's x-axis and y-axis is 4 less than the second last marble, it is a trailing marble. If it is trailing marble, the value of $h_1(n)$ is -1, otherwise it's 1.

Whenever there are possibilities for long jumps, the agent should take advantage of the long jumps. Long jumps make a player move his marbles quickly towards the goal position. If a long jump consists 3 separate jumps, the agent is rewarded each separate jump 1 point, then the long jump has a positive 3 points.

I choose the initial weights of w_1 , w_3 as 0.6 and 0.4 respectively. And I set both α and τ 0.6. The yellow Marbles are played using the Q learning. The blue marble is played by human. The result of the first match is given in Fig 35. Blue marbles win and yellow marbles loose 3 steps.

Fig 35 Yellow Loose 3 Steps

Fig 36 shows the update of the parameters w_1 and w_3 . The updated w_1 is 1,342 and updates w_3 is 1.296.

```
C:\WINDOWS\System32\cmd.exe

15
w1 = 0.81600004; w3 = 0.928
w1 = 0.98016006; w3 = 1.20256
w1 = 1.1049217; w3 = 1.3453312
w1 = 1.1997405; w3 = 1.4195722
w1 = 1.4158028; w3 = 1.7461776
w1 = 1.4360101; w3 = 1.509847
w1 = 1.4513677; w3 = 1.5051205
w1 = 1.4630395; w3 = 1.5026627
w1 = 1.47191; w3 = 1.5013846
w1 = 1.4066516; w3 = 1.35672
w1 = 1.4290552; w3 = 1.3911072
w1 = 1.446082; w3 = 1.4433757
w1 = 1.3870224; w3 = 1.3265554
w1 = 1.342137; w3 = 1.296182
Average w1 = 1.2633265; w3 = 1.3247732
```

Fig 36 Updated Parameters

Now the yellow marbles are played using the updated parameters. Fig 37 shows the result. This time, Blue still wins. However, there is improvement in yellow marbles. Yellow marbles only loose 2 steps.

Fig 37 Yellow Loose 2 Steps

After about 10 matches, Yellow wins for the first time as shown in Fig 38. The current values of the parameters are 1 and 0.6.

Fig 38 Yellow Wins for The First Time

5.5 Evaluation

After implementing the learning agent using proposed evaluation function in section 5.3.1 and learning algorithm in section 5.3.2, we get the experimental data - the parameter values for the heuristic function in section 5.4.

In Fig 39, both yellow and blue are played by software agents. The yellow is using the heuristic function with the experimental data from section 5.4— the parameter values 1 and 0.6, and the blue is always taking the longest jumps. From Fig 39, we can see that the trailing marble of the blue is got stuck, It's obvious that the heuristic function works.

Fig 39 Proposed Heuristic Function vs Always Longest Jumps

5.6 Summary

In this chapter, I have used Chinese Checkers to demonstrate the learning algorithm and theory in a competitive multi-agent system. I have reviewed the description of Chinese Checkers followed by the rules of the game. Then the six factors of my

evaluation function are described and applying a variation of Q-learning to learn the parameters of evaluation function. At last, the experimental result indicated that the evaluation function and the variation of Q-learning work.

Chapter 6 Conclusion and Recommendations

This chapter concludes the research and gives recommendations for future research. It summarizes my findings particularly the approaches to the personalized information filtering and Chinese Checkers, I also list a few directions for future research.

6.1 Conclusion

In this report, I present the popular definition of “multi-agent learning”. Reinforcement learning, unsupervised web document clustering and supervised Backpropagation neural network are described. Their brief history, algorithms and applications are discussed.

The need for information filtering is obvious. Technologies and limitations of existing information filtering systems are shown. An approach to personalized information filtering is proposed. The reason why multi-agent system is employed for the proposal is given. Working procedure of a four-agent model running on JADE, a simple version of the proposed multi-agent system, is outlined. Its learning algorithm is hybrid learning technique, which is modified Monte Carlo algorithm combined with

features of Backpropagation neural network and Suffix Tree Clustering algorithm. A proof of this algorithm's correctness is outlined. Function of each agent and algorithm and method used in each agent are presented. User agent is a Java Swing browser. Information Retrieval Agent taps into Google web service. Clustering Agent employs Suffix Tree Clustering. Learning Agent uses a proposed method.

A description of the game is reviewed. The rules of the game are explored. And an effort has been made to formulate a heuristic evaluation function. Q-learning is applied for learning the heuristic function.

In conclusion, cooperative multi-agent learning and competitive multi-agent learning are explored in this research. Basic learning algorithms and theories are studied. And modified learning algorithms are applied to cooperative and competitive agent systems. A multi-agent model for personalized information filtering system, two hybrid learning algorithms and mechanisms, a user profile updating algorithm and a prototype of personalized information filtering system with multi-agent learning are developed for the personalized information filtering. A Chinese Checkers playing algorithm is formulated and a program is developed to demonstrate the learning in competitive multi-agent learning.

6.2 Recommendations for Further Research

This research opens a few directions for further research. We can explore more user behaviors for capturing user interests. For example, other user behaviors can be the

frequency of a user to visit the same webpages, or whether a user blog the content or the URL.

Another area of future research is to apply ontologies to personalized information filtering. In the proposed approach presented in the thesis, phrases are used to identify user interest. Phrases can have different meanings in different context. Context plays an important role for accurate and effective information access. The semantic knowledge about the domain being investigated thus can help a lot in personalized information filtering. If a user's query is integrated with semantic knowledge to assist the user in information retrieval, the system should be able to locate and provide the most appropriate result for users' information needs. And if user chooses to share user profile, the user profile of other users, who have similar semantic knowledge, can do the help to identify the current user's interest more quickly.

An experiment with reasonable scale of users such as 20 users should be conducted to get the experimental data.

On a prototyping part, we can extend the implementation of the heuristic evaluation function for the Chinese Checkers to the full evaluation function. The agent would be more intelligent if we implement all the six factors discussed in section 5.3.1.

The study on competitive agents can be applied to many domains such as internet auction, negotiation and playing football.

Author's Publication

Junhua, Chen and Zhonghua Yang. "A Learning Multi-agent System for Personalized Information Filtering". the Fourth International Conference on Information, Communications & Signal Processing(ICICS 2003), Singapore, 15-18 December 2003.

References

- [1] Gerhard Weiß, "Adaptation and Learning in Multi-Agent Systems: Some Remarks and Bibliography", *Adaption and learning in multi-agent systems : IJCAI '95 workshop, Montréal, Canada, August 21, 1995, proceedings*, pp. 1-21, **1996**.
- [2] Richard Sutton and Andrew Barto, *Reinforcement Learning: An Introduction*, MIT Press, **1998**.
- [3] Leslie Pack Kaelbling, Michael Littman, and Andrew Moore, "Reinforcement Learning: A Survey", *JAIR (Journal of AI Research)*, vol. 4, pp. 237-285, **1996**.
- [4] Eduardo Alonso, Mark d'Inverno, Daniel Kudenko, Michael Luck, and Jason Noble, "Learning in Multi-Agent Systems", *Knowledge Engineering Review*, vol. 16, No. 3, pp. 277-284, **2001**.
- [5] Mance E. Harmon and Stephanie S. Harmon, "Reinforcement Learning: A Tutorial", available from <http://citeseer.nj.nec.com/harmon96reinforcement.html>, **1996**.
- [6] Richard Sutton, "Reinforcement Learning", *The MIT Encyclopedia of the Cognitive Sciences*, MIT Press, **1999**.

- [7] Oren Zamir and Oren Etzioni, "Web document clustering: a feasibility demonstration", *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval, Melbourne, Australia*, pp.46-54, August 24-28, 1998.
- [8] Oren Zamir and Oren Etzioni, "Grouper: a dynamic clustering interface to Web search results", *In Proceedings of the Eighth International World Wide Web Conference, Toronto, Canada*, May 1999.
- [9] IBM, "WS applications with the Google API", IBM Developer Works, ibm.com/developerWorks.
- [10] Robert Crites and Andrew Barto, "Elevator Group Control Using Multiple Reinforcement Learning Agents", *Machine Learning*, vol. 33, pp. 235–262, 1998.
- [11] Young-Woo Seo and Byoung-Tak Zhang, "Personalized web-document filtering using reinforcement learning", *Applied Artificial Intelligence*, 2001.
- [12] Young-Woo Seo and Byoung-Tak Zhang, "Learning user's preferences by analyzing web-browsing behaviors", *Proc. of Int'l Conf. on Autonomous Agents 2000 (AA '2000)*, pp. 381–387, 2000.
- [13] Javed Mostafa, Snehasis Mukhopadhyay, Wai Lam and Mathew Palakal, "A Multilevel Approach to Intelligent Information Filtering: Model, System and

Evaluation." ACM Transactions on Information Systems, vol. 15, No. 4, pp. 368-399, 1997.

- [14] S. Kebreau, S. Pierre, and W. G. Probst, "Modeling Intelligent Agents for Information Filtering", IEEE Canadian Conference on Electrical and Computer Engineering, vol. 2, pp. 569-572, 1998.
- [15] JADE, Java Agent DEvelopment Framework API, available from <http://sharon.cselt.it/projects/jade/>
- [16] Dawid Weiss, "A Clustering Interface for Web Search Results in Polish and English", Master thesis, Poznan University of Technology, Poznan, Poland, 2001.
- [17] Google, "Google Web APIs", available from <http://www.google.com/apis/>
- [18] GroupLens Research, "GroupLens Home Page", Department of Computer Science and Engineering, University of Minnesota, available from <http://www.cs.umn.edu/Research/GroupLens/>
- [19] Joseph Konstan, Bradley Miller, David Maltz, Jonathan Herlocker, Lee Gordon, and John Riedl, "GroupLens: Applying collaborative filtering to Usenet news", Communications of the ACM, vol. 40, No. 3, pp. 77-87, March 1997.

- [20] "Supervised learning in a single-layer neural network", Laboratory of Computational Neuroscience, School of Computer and Communication Sciences, Swiss Federal Institute of Technology Lausanne, available from <http://www.epfl.ch/mantra-tutoria-eng/liswapb-tml1theo.html>
- [21] Laurene Fausett, Fundamentals of Neural Networks architectures, algorithms, and applications, The Prentice Hall, 1994.
- [22] "Carrot", Laboratory of Intelligent Decision Support Systems, Institute of Computing Science, Poznan University of Technology, Poland, available from <http://www.cs.put.poznan.pl/dweiss/site/research-carrot/index.html>
- [23] "MetaCrawler/HuskySearch/Grouper" Department of Computer Science and Engineering, University of Washington, available from <http://www.cs.washington.edu/research/projects/WebWare/metacrawler/>
- [24] "Knowledge Discovery and Data Mining", School of Electrical Engineering and Computer Science, University of Newcastle, Australia, available from http://www.cs.newcastle.edu.au/~ijlee/li_open.html
- [25] K. MacLeod, "An application specific neural model for document clustering", Proceedings of the Fourth Annual Parallel Processing Symposium, vol.1, pp. 5-16, 1990.

- [26] Henry Lieberman, "Letizia: An Agent That Assists Web Browsing",
Proceedings of the Fourteenth International Joint Conference on *Artificial*
Intelligence, August 1995.
- [27] Kilander Fredrik, "A Brief Comparison of News Filtering Software", The
IntFilter Project, Dept. of Computer and Systems Science, Stockholm
University, available from http://www.dsv.su.se/~fk/if_Doc/IntFilter.html,
1995
- [28] Oren Eli Zamir, "Clustering Web Documents: A Phrase-Based Method for
Grouping Search Engine Results", Ph.D. thesis; University of Washington,
1999.
- [29] Rocchio Joseph John, "Document retrieval systems - optimization and
evaluation", Ph.D. Thesis, Harvard University. Report ISR-10 to National
Science Foundation, Harvard Computation Laboratory, 1966.
- [30] Richard Sutton, "Learning to predict by the methods of temporal-
differences", Machine Learning, vol. 3, pp. 9-44, 1988.
- [31] Peter Stone and Manuela Veloso, "Multiagent Systems: A Survey from a
Machine Learning Perspective", Autonomous Robotics, vol. 8, No.3, 2000.
- [32] Satinder Singh, Peter Norvig, and David Cohn, "How to Make Software
Agents Do the Right Thing: An Introduction to Reinforcement Learning",

Adaptive Systems' Group, Harlequin Inc, 1996.

- [33] Christopher Watkins and Peter Dayan, "Technical note q-learning", Machine Learning , vol. 8, pp. 279-292, 1992.

- [34] Peter Willett, "Recent Trends in Hierarchic Document Clustering: A Critical Review", Information Processing and Management, Vol. 24, No 5, pp. 577-597, 1988.

- [35] E. M. Voorhees, "Implementing agglomerative hierarchical clustering algorithms for use in document retrieval", Information Processing and Management, vol.22, No. 6, pp. 465-476, 1986.

- [36] Beerud Sheth and Pattie Maes, "Evolving agents for personalized information filtering", In Proceedings of the ninth IEEE Conference on *Artificial Intelligence for applications*, pp. 345-352, 1993.

- [37] Beerud Sheth, "A learning approach to personalized information filtering", Master's thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, available from <http://citeseer.ni.nec.com/sheth94learning.html>, 1994.

- [38] Young-Woo Seo and Byoung-Tak Zhang, "A reinforcement learning agent for personalized information filtering", Proceedings of International

Conference on Intelligent User Interfaces (*IUI'2000*), pp. 248-251, 2000.

[39] Katia Sycara, "Multiagent Systems", *AI Magazine*, Vol. 19, No. 2, pp. 79-92, 1998.

[40] Lesser V. "Multiagent Systems: An Emerging Subdiscipline of AI". *ACM Computing Surveys*, Volume 27 Issue 3, pp340-342, September 1995.

[41] Wooldridge, M. *An Introduction to Multiagent Systems*. John Wiley & Sons, 2002

[42] Gerhard Weiss, *Multiagent Systems, a modern approach to distributed artificial intelligence*. Edited by Gerhard Weiss, MIT Press, 2000.

[43] Hermans, Björn. "Intelligent Software Agents on the Internet: an inventory of currently offered functionality in the information society & a prediction of (near-)future developments". University at Tilburg, Holland, Tilburg, Holland, 1996. Available from <http://www.hermans.org/agents/>.

[44] Wooldridge M., *Intelligent Agents: Theory and Practice*, Knowledge Engineering Review, 1995.

[45] Jacques Ferber. *Multi-Agent Systems. An introduction to Distributed Artificial Intelligence*. Addison Wesley 1999.

- [46] R. Lesser. "Co-operative multi-agent systems: a personal view of the state of the art". IEEE Transactions on Knowledge and Data Engineering, Vol. 11, No 1,1999.
- [47] "Intelligent agents and multi-agents". Available from <http://www.asap.cs.nott.ac.uk/themes/ma.shtml>
- [48] "FIPA Agent Communication Language Specifications". Available from <http://www.fipa.org/repository/aclspecs.html>
- [49] "UMBC KQML Web". Available from <http://www.cs.umbc.edu/kqml/>
- [50] Kevin Leyton-Brown, "Resource Allocation in Competitive Multiagent Systems", Doctor dissertation, Stanford University department of computer science, 2003
- [51] Walid ElSayed Aly Gomaa, "A New Learning Technique for Planning in Cooperative Multi-Agent Systems", master thesis, Alexandria University Faculty of Engineering, 2002
- [52] "Halma & Chinese Checkers – History". Available from <http://home.no.net/vkp/histon/.html>

- [53] "Chinese Checkers - Rules for Chinese Checkers and **Halma**". Available from <http://home.no.net/vkp/rules.html>
- [54] "The Rules of Chinese Checkers from Masters Games". Available from <http://www.mastersgames.com/rules/chinese-checkers-rules.htm>
- [55] "Programmer's Challenge". Available from <http://www.mactech.com/progchallenge/9902Challenge.html>
- [56] "**SOAP** Specifications", Available from <http://www.w3.org/TR/soap/>
- [57] Bellman, R, Dynamic Programming, Princeton University Press, 1957.
- [58] Rumelhart, D. E., Hinton and G. E. and Wiliams, R. J, "Learning representations by back-propagating errors", Nature, vol. 323, pp. 533-536, 1986.
- [59] Corkill, Daniel, **and** Lesser, Victor. The use of meta-level control for coordination in a distributed problem solving network. Proceedings of the Eighth International Joint Conference on *Artificial* Intelligence, pp. 748-756. August 1983.

Appendix

-----Browser.java-----

```

import javax.swing.*.*;
import javax.swing.text.html.*;
import javax.swing.text.*;
import javax.swing.tree.*;
import javax.swing.event.*;
import java.awt.*.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;

public class Browser extends JFrame implements HyperlinkListener, ActionListener,
AdjustmentListener{

    public static void main(String[] args) {
        if (args.length == 0)
            new Browser("http://127.0.0.1:8080/mal/google.jsp");
        else
            new Browser(args[0]);
    }

    public void initializer(){
        File creator=new File("UrlHistory");
        creator.mkdir();
        File OldFiles=new File("UrlHistory\\");
        String ListOfOldFiles[] = OldFiles.list();
        int NumberOfOldFiles = ListOfOldFiles.length;
        for(int OldFilesCount=0; OldFilesCount< NumberOfOldFiles;
        OldFilesCount++){
            File DeleteFiles=new File("UrlHistory\\" +
            ListOfOldFiles[OldFilesCount]);
            DeleteFiles.delete();
        }
    }

    public Browser(String initialURL) {
        super("GreenLeaf :: Multi-agent based information filtering system");
        this.initialURL = initialURL;
        // System.setProperty("http.proxyHost", "proxy.ntu.edu.sg");
        //System.setProperty("http.proxyPort", "80");
        ImageIcon im_bg=new ImageIcon("leaf.jpg");
        Image img = im_bg.getImage();
        setIconImage(img);
        addWindowListener(new ExitListener());
        WindowUtilities.setNativeLookAndFeel();

        hSplitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT);
        vSplitPane = new JSplitPane(JSplitPane.VERTICAL_SPLIT);
        vSplitPane.setDividerLocation(150);
        ImageIcon bg = new ImageIcon("bg.gif");
        ImagePanel topPanel= new ImagePanel(bg);
    }

```



```

homeButton = new JButton("home.gif");
homeButton.addActionListener(this);
backButton = new JButton("back.gif");
backButton.setEnabled(false);
backButton.addActionListener(this);
forwardButton = new JButton("forward.gif");
forwardButton.setEnabled(false);
forwardButton.addActionListener(this);

JLabel urlLabel = new JLabel("URL:");
urlField = new JTextField(50);
urlField.setText(initialURL);
urlField.addActionListener(this);
topPanel.add(urlLabel);
topPanel.add(urlField);
topPanel.add(homeButton);
topPanel.add(backButton);
topPanel.add(forwardButton);
JLabel upLabel = new JLabel("User Profile");
getContentPane().add(topPanel, BorderLayout.NORTH);
JMenuBar menubar = new JMenuBar();
JMenu menu = menubar.add(new JMenu("File"));
save = (JMenuItem)menu.add(new JMenuItem("Save"));
save.addActionListener(this);
JMenuItem exits = (JMenuItem)menu.add(new JMenuItem("Exit"));
exits.addActionListener(this);
menubar.add(menu);
setJMenuBar(menubar);
initializer();

try {
    htmlPane = new JEditorPane(initialURL);
    htmlPane.setEditable(false);
    htmlPane.addHyperlinkListener(this);
    JScrollPane scrollPane = new JScrollPane();
    scrollPane.setViewportView(htmlPane);
    vSplitPane.setTopComponent(scrollPane);
    bottomPane = new JEditorPane();
    bottomPane.setEditable(false);
    bottomPane.addHyperlinkListener(this);
    JScrollPane scrollPane1 = new JScrollPane();
    scrollPane1.setViewportView(bottomPane);
    h_bar = scrollPane1.getHorizontalScrollBar();
    h_bar.addAdjustmentListener(this);
    v_bar = scrollPane1.getVerticalScrollBar();
    v_bar.addAdjustmentListener(this);
    vSplitPane.setBottomComponent(scrollPane1);
    DefaultMutableTreeNode top = new DefaultMutableTreeNode("topics");
    DefaultMutableTreeNode a = new DefaultMutableTreeNode("topic 1");
    top.add(a);
    DefaultMutableTreeNode a1 = new DefaultMutableTreeNode("topic 2");
    top.add(a1);
    DefaultMutableTreeNode a2 = new DefaultMutableTreeNode("topic 3");
    top.add(a2);
    JTree tree = new JTree(top);

```

```

        int v = ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED;
        int h =
            ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED;
        JScrollPane treeScrollPane = new JScrollPane(tree, v, h);
        hSplitPane.setLeftComponent(treeScrollPane);
        hSplitPane.setRightComponent(vSplitPane);
        getContentPane().add(hSplitPane, BorderLayout.CENTER);
    } catch (IOException ioe) {
        warn("Can't build HTML pane for " + initialURL + ": " + ioe);
    }

    Dimension screenSize = getToolkit().getScreenSize();
    int width = screenSize.width * 8 / 10;
    int height = screenSize.height * 8 / 10;
    setBounds(width/8, height/8, width, height);
    setVisible(true);
}

public void actionPerformed(ActionEvent event) {

    String url="";
    boolean proceed=false;
    boolean which_pane=false;
    if (event.getSource() == save){

        Frame SaveFrame=new Frame("frames");
        FileDialog SaveFileDialog=new FileDialog(SaveFrame,"p",1);
        SaveFileDialog.show();
        String SavedFileName;
        String SavedFileDirectory;
        SavedFileDirectory=SaveFileDialog.getDirectory();
        SavedFileName=SaveFileDialog.getFile();
        String urlfield=urlField.getText();
        // System.out.println(urlfield);
        try{
            HTMLDocument doc =
                (HTMLDocument)htmlPane.getDocument();
            String file=SavedFileDirectory+SavedFileName;
            Writer writer=new FileWriter(file);
            HTMLWriter htmlWriter = new HTMLWriter(writer, doc);
            htmlWriter.write();
            writer.flush();
            writer.close();
        }catch(Exception io)
        {
            System.out.println(io);
        }
    }
    else{
        if (event.getSource() == urlField){
            url = urlField.getText();
            proceed=true;
        }
        else if(event.getSource()==homeButton){
            proceed=false;
            url = initialURL;
        }
    }
}

```

```

        which_pane=true;
    }
    else if(event.getSource() == backButton){
        proceed =false;
        increment =increment - 1;
        // System.out.println("back:....."+increment);
        if(increment == 2 || increment == 1)
            backButton.setEnabled(false);
        forwardButton.setEnabled(true);

url="file:///g:/agent/tool/browser/UrlHistory/one"+increment+".html";
    }
    else if(event.getSource() == forwardButton){
        proceed=false;
        backButton.setEnabled(true);
        increment = increment+1;
        if(increment==count){
            url=current_url.toString();
            forwardButton.setEnabled(false);
        }
        else

url="file:///g:/agent/tool/browser/UrlHistory/one"+increment+".html";
        //System.out.println("for:....."+increment);
    }

    try {
        if(proceed){
            save(new URL(url));
            scroll_times=0;
        }
        if(which_pane) htmlPane.setPage(new URL(url));
        else bottomPane.setPage(new URL(url));
        urlField.setText(url);
    } catch(Exception ioe) {
        warn("Can't follow link to " + url + ": " + ioe);
    }
}

}

public void hyperlinkUpdate(HyperlinkEvent event) {

    if (event.getEventType() == HyperlinkEvent.EventType.ACTIVATED) {
        if(event.getSource()==htmlPane){
            try{
                scroll_times=0;
                save(event.getURL());
                current_url=event.getURL();
                bottomPane.setPage(event.getURL());
                urlField.setText(event.getURL().toExternalForm());
            } catch(Exception ioe){
                warn("Can't follow link to "+
                    event.getURL().toExternalForm()
                    + ": " + ioe);
            }
        }
    }
}

```

```

    }
    else if(event.getSource()==bottomPane){
        try{
            save(event.getURL());
            scroll_times=0;
            current_url=event.getURL();
            bottomPane.setPage(event.getURL());
            urlField.setText(event.getURL().toExternalForm());
        } catch(Exception ioe){
            warn("Can't follow link to "+
                event.getURL().toExternalForm() + ": " + ioe);
        }
    }
}

}

public void adjustmentValueChanged(AdjustmentEvent event)
{
    JScrollBar sb = (JScrollBar)event.getSource();
    if (sb == h_bar){
        int current_hvalue=h_bar.getValue();
        if(current_hvalue-old_hvalue!=0){
            scroll_times++;
            //System.out.println("horizontal scroll"+scroll_times);
        }
        old_hvalue=current_hvalue;
    }
    else{
        int current_vvalue=v_bar.getValue();
        if(current_vvalue-old_vvalue!=0){
            scroll_times++;
            //System.out.println("vertical scroll"+scroll_times);
        }
        old_vvalue=current_vvalue;
    }
    //if(scroll_times>=8) System.out.println("*****scrolled is true.");
}

public void behavior()
{
    //if(layer==2&&scroll_times>=8) scrolled=true;
}

private void warn(String message) {
    JOptionPane.showMessageDialog(this, message, "Error",
        JOptionPane.ERROR_MESSAGE);
}

public void save(URL url){

    count = count +1;
    // System.out.println("***** "+count+"
        *****");

    increment = count;
    //System.out.println(increment+"....."+url);
    try{

```

```

        HTMLDocument doc = (HTMLDocument)bottomPane.getDocument();
        String file="UrlHistory\\one"+(increment-1)+".html";
        Writer writer=new FileWriter(file);
        HTMLWriter htmlWriter = new HTMLWriter(writer, doc);
        htmlWriter.write();
        writer.flush();
        writer.close();
    } catch(Exception io){
        System.out.println(io);
    }
    if((count-1) == 1)
        backButton.setEnabled(true);
}
private JButton homeButton;
private JButton backButton;
private JButton forwardButton;
private JTextField urlField;
private JEditorPane htmlPane;
private JEditorPane bottomPane;
private String initialURL;
private JSplitPane hSplitPane;
private JSplitPane vSplitPane;
JScrollbar h_bar;
JScrollbar v_bar;
JMenuItem save;
JPanel topPanel;
int old_hvalue=0;
int old_vvalue=0;
int scroll_times=0;
int count=1;
int increment;
URL current_url=null;
int TextAreaFileNameContent;
}

```

ExitListener.java

```

import java.awt.*;
import java.awt.event.*;

public class ExitListener extends WindowAdapter {
    public void windowClosing(WindowEvent event) {
        System.exit(0);
    }
}

```

ImagePanel.java

```

import javax.swing.*;
import java.awt.*;

public class ImagePanel extends JPanel{

```



```

private ImageIcon im_bg;
public ImagePanel(ImageIcon background){
    super();
    im_bg = background;
}
public void paintComponent(Graphics g){
    super.paintComponent(g);
    Image img = im_bg.getImage();
    g.drawImage(img, 0, 0, this.getWidth(), this.getHeight(), this);
}
}

```

----- JIconButton.java -----

```

import javax.swing.*;
public class JIconButton extends JButton {
    public JIconButton(String file) {
        super(new ImageIcon(file));
        setContentAreaFilled(false);
        setBorderPainted(false);
        setFocusPainted(false);
    }
}

```

----- MLP.java -----

```

import java.awt.*;

public class MLP{
    private int numberOfInput, numberOfHidden, numberOfOutput;
    private SigmoidalNeuron[] inputNeuron, hiddenNeuron, outputNeuron;
    public double[][] Wih, Who;

    public MLP(int numinput, int numhidden, int numoutput){
        numberOfInput = numinput;
        numberOfHidden = numhidden;
        numberOfOutput = numoutput;

        int i;
        inputNeuron = new SigmoidalNeuron[numberOfInput + 1];
        for(i = 0; i < numberOfInput + 1; i++) {
            inputNeuron[i] = new SigmoidalNeuron();
        }
        hiddenNeuron = new SigmoidalNeuron[numberOfHidden + 1];
        for(i = 0; i < numberOfHidden + 1; i++) {
            hiddenNeuron[i] = new SigmoidalNeuron();
        }
        outputNeuron = new SigmoidalNeuron[numberOfOutput];
        for(i = 0; i < numberOfOutput; i++) {
            outputNeuron[i] = new SigmoidalNeuron();
        }

        //initialize input-hidden weight
        int j;
        Wih = new double[numberOfInput + 1][];
        for(i = 0; i < numberOfInput + 1; i++){

```

```

        Wih[i] = new double[numberOfHidden];
        for(j = 0; j < numberOfHidden; j++){
            Wih[i][j] = 0.0;
        }
    }
    //initialize hidden-output weight
    Who = new double[numberOfHidden + 1][];
    for(i = 0; i < numberOfHidden + 1; i++){
        Who[i] = new double[numberOfOutput];
        for(j = 0; j < numberOfOutput; j++){
            Who[i][j] = 0.0;
        }
    }
}

public int getNumberOfInput() { return(numberOfInput); }
public int getNumberOfHidden() { return(numberOfHidden); }
public int getNumberOfOutput() { return(numberOfOutput); }

public double getOutput(int k) { return(outputNeuron[k].getOutput()); }
public double getHiddenOutput(int k) { return(hiddenNeuron[k].getOutput()); }
public double getInputOutput(int k) { return(inputNeuron[k].getOutput()); }

public double getInput(int k) { return(inputNeuron[k].getInput()); }
public double getHiddenInput(int k) { return(hiddenNeuron[k].getInput()); }
public double getOutputInput(int k) { return(outputNeuron[k].getInput()); }

public boolean calc(double[] inputVector){
    int i, h, o;
    if(inputVector.length != numberOfInput) { return(false); }

    for(i = 0; i < numberOfInput; i++){
        inputNeuron[i].through(inputVector[i]);
    }

    inputNeuron[numberOfInput].through(1.0); //bias

    for(h = 0; h < numberOfHidden; h++){
        double sum = 0.0;
        for(i = 0; i < numberOfInput + 1; i++){
            sum += inputNeuron[i].getOutput()*Wih[i][h];
        }

        hiddenNeuron[h].calc(sum);
    }

    hiddenNeuron[numberOfHidden].through(1.0); //bias

    for(o = 0; o < numberOfOutput; o++){
        double sum = 0.0;
        for(h = 0; h < numberOfHidden + 1; h++){
            sum += hiddenNeuron[h].getOutput()*Who[h][o];
        }
    }
}

```

```

        outputNeuron[o].calc(sum);
    }

    return(true);
}

public boolean backpropagation(double[] inputVector, double[] targetVector, double
learningConst, double tolerance{

    if(targetVector.length != numberOfOutput){
        throw(new IllegalArgumentException());
    }
    if(calc(inputVector) == false) { return(false); }

    int i, j, k;

    boolean convergeState = true;
    double dif;
    for(k = 0; k < numberOfOutput; k++){
        dif = targetVector[k] - getOutput(k);
        System.out.println("dif="+dif);
        if(dif > tolerance) { convergeState = false; }
        if(dif < -tolerance) { convergeState = false; }
    }
    if(convergeState == true){
        return(true);
    }

    double[] delta_o, delta_h, sigma_o;
    double[][] preWho;
    delta_o = new double[numberOfOutput];
    delta_h = new double[numberOfHidden];
    preWho = new double[numberOfHidden + 1][numberOfOutput];

    for(k = 0; k < numberOfOutput; k++){
        delta_o[k] = (targetVector[k] - getOutput(k))*getOutput(k)*(1.0 -
            getOutput(k));
        for(j = 0; j < numberOfHidden + 1; j++){
            preWho[j][k] = Who[j][k];
            Who[j][k] += learningConst*delta_o[k]*getHiddenOutput(j);
            System.out.println("Weight of hidden output
                link["+j+"]["+k+"]="+Who[j][k]);
        }
    }

    for(j = 0; j < numberOfHidden; j++){
        delta_h[j] = 0.0;
        for(k = 0; k < numberOfOutput; k++){
            delta_h[j] += delta_o[k]*preWho[j][k];
        }
        delta_h[j] *= getHiddenOutput(j)*(1.0 - getHiddenOutput(j));
        for(i = 0; i < numberOfInput + 1; i++){
            Wih[i][j] += learningConst*delta_h[j]*getInputOutput(i);

            System.out.println("Weight of input hidden

```



```

        link["+i+"["+j+"]="+Wih[i][j]);
    }
}
System.out.println("*****");
return(convergeState);
}
}

```

```

class SigmoidalNeuron{
    private double inputSignal, outputSignal;
    private SigmoidalFunction f;

    public SigmoidalNeuron(){
        f = new SigmoidalFunction();
        inputSignal = 0.0;
        outputSignal = 0.0;
    }

    public double calc(double input){
        inputSignal = input;
        outputSignal = f.calc(inputSignal);
        return(outputSignal);
    }

    public double through(double input){
        inputSignal = input;
        outputSignal = input;
        return(input);
    }

    public double getOutput() { return(outputSignal); }
    public double getInput() { return(inputSignal); }
}

```

```

class SigmoidalFunction{

    public double calc(double x){
        return(1.0/(1.0 + Math.exp(-x)));
    }

    public double differential(double x){
        return(calc(x)*(1.0 - calc(x)));
    }
}

```

Random.java

```

class Random{
    private int seed;
    private int A = 109;
    private int C = 1021;
    private int M = 32768;
}

```

```

public Random() { init(1); }
public void init(int s){
    if(s < 0) { s = -s; }
    seed = s%M;
}
public double uran(){
    seed = ((seed*A) + C)%M;
    return((double)seed/(double)M);
}
public double uran(double lowerLimit, double upperLimit){
    double range = upperLimit - lowerLimit;
    if(range > 0.0){
        return(uran()*range + lowerLimit);
    }
    else{
        throw(new IllegalArgumentException());
    }
}
}

```

Test.java

```

//use jdk1.4
//roughly 3.8seconds.
import java.util.*;
public class Test
{
    public static void main(String[] args){
        BpdemoData data = new BpdemoData();
        MLP mlp = new MLP(data.NUM_INPUT, data.NUM_HIDDEN,
            data.NUM_OUTPUT);
        Random rnd = new Random();
        for(int i = 0; i < mlp.getNumberOfInput() + 1; i++){
            for(int j = 0; j < mlp.getNumberOfHidden(); j++){
                mlp.Wih[i][j] = rnd.uran(-1.0, 1.0);
            }
        }

        for(int j = 0; j < mlp.getNumberOfHidden() + 1; j++){
            for(int k = 0; k < mlp.getNumberOfOutput(); k++){
                mlp.Who[j][k] = rnd.uran(-1.0, 1.0);
            }
        }
        boolean res = false;
        int num=0;
        //Calendar cal = Calendar.getInstance();
        //long start_time = cal.getTimeInMillis();
        while(!res){
            for(int c = 0; c < data.target.length; c++){
                for(int n = 0; n < data.learningData[c].length; n++){
                    mlp.calc(data.learningData[c][n]);
                }
            }
        }
    }
}

```

```
    }  
    for(int c = 0; c < data.target.length; c++){  
        for(int n = 0; n < data.learningData[c].length; n++){  
            if(mlp.backpropagation(data.learningData[c][n],  
                data.target[c],  
                data.DEFAULT_LEARNING_CONST,  
                data.CONVERGE_TOLERANCE) == false){  
                res = false;  
            }  
            else res=true;  
        }  
    }  
    num++;  
}  
//Calendar call = Calendar.getInstance();  
//long end_time = call.getTimeInMillis();  
if(res) System.out.println("training finished. ");  
else System.out.println("need more samples.");  
System.out.println("round="+num);  
}  
}
```