SANet: A Service-Agent Network for Call-Center Scheduling

Qiang Yang, *Member, IEEE,* Yong Wang, and Zhong Zhang

***Abstract*—We consider a network of service-providing agents, where different agents have different capabilities, availability, and cost to solve problems. These characteristics are particularly important in practice for semi-automated call centers which provide quality customer service in real time. We have developed SANet, a service agent network for call center automation, to serve as an experimental testbed for our research. SANet can select appropriate agents to provide better solutions for customer problems according to the changing capabilities and availability of service agents in the network. It can also add or delete appropriate agents to balance problem-solving quality, efficiency, and cost according to the number and types of incoming customer problems. On this network, each service agent can be a human service agent, an automated software service agent, or a combination of the two. This paper describes the architecture, a problem scheduling algorithm and an agent assignment algorithm on the SANet. We highlight an application in which we apply SANet to a call-center scheduling problem for a cable-TV company. Finally, we show the efficiency and adaptability of our system via experimental results and discuss related works.**

***Index Terms*—Agent-based systems, call-center applications, intelligent scheduling and planning.**

I. Introduction

OUR research is motivated by a realistic problem in call center environments. To solve customers' problems, many telecommunications companies, such as cable-TV and telephone companies, maintain large call centers that are aimed at providing real-time solutions to their customers. In a cable-TV call center environment, for example, a customer may phone in to ask about a solution to his fuzzy-picture problem. A human agent is selected for answering the question and interactively diagnoses the source of the problem. In this environment, the human agent's expertise is distributed and changing. They are not always available either due to time shifts, or changing interests and training. A solution is to create a number of software agents which can provide subsets of the expertise that human agents can provide, and cater to customers through a network of human and software agents.

From the agent research point of view, a call center is a multiagent system which includes customer service agents and an agent who schedules customer problems to service agents. Using agent development techniques based on IBM's Aglet package [8], we have developed SANet—a service agent network for call center automation. SANet integrates both human and software service agents in providing customer service in real time. It employs a broker to schedule customer problems to service agents for better solutions according to the nature of the input problems as well as the changing capabilities and availability of service agents in the network. A main feature is an agent scheduling and assignment algorithm based on problem type, capabilities, and availability of service agents. SANet employs a manager to assign appropriate agents in the network according to the number and types of the incoming customers' problems. The manager balances between problem solving quality and cost in real time, by dynamically adding and deleting agents to the network.

In SANet, agent assignment plays an important role. Usually, it costs more to get higher service quality for customers. The cost may include hiring more experienced service agents, spending more time with customers, etc. In contrast, reducing the cost may reduce the service quality. Hence, it is a key issue to balance quality of service and cost. The management agent in SANet is responsible for this task: based on the incoming problems and the system performance from the near past, it estimates the system performance in the near future and adjusts the service agents by adding or deleting appropriate agents. In this way, the network can be adapted dynamically to fit the current situation for a better balance between service quality and cost.

This paper describes the architecture, an agent assignment algorithm and a problem-type learning method for SANet. As we will see, SANet is a *flexible* network on which agents can be added or deleted on demand at any time and its capability to solve problems can be tracked and utilized. Our work contributes to artificial intelligence and distributed problem solving in several aspects. First, our system represents an innovative attempt to adjust the number and type of agents to in order to tradeoff quality with time. Second, our proposed architecture integrates multiple human and software agents into a unified framework. Our system relates scheduling, multiple-agent communication, learning and reasoning approaches in artificial intelligence.

There are at least three ways in which we envision our framework can be applied. First, the agents in the network provide internal help-desk assistance functions for human customer-service personnel. When an incoming problem arrives, the agent assigned to help a human agent can act to provide solutions to the agent in real time in response to the type of incoming problem. The human agent, equipped with the real-time solu-

Manuscript received November 30, 2000; revised March 26, 2003. This work was supported by grants from a Canadian IRIS-III Project and Hong Kong RGC. This paper was recommended by Associate Editor L. Fang.

Q. Yang is with the Department of Computer Science, Hong Kong University of Science and Technology, Clearwater Bay, Kowloon, Hong Kong (e-mail: qyang@cs.ust.hk).

Y. Wang is with the School of Computing Science, Simon Fraser University, Burnaby, BC, V5A 1S6, Canada (e-mail: yongwang@canada.com).

Z. Zhang is with the School of Computing Science, Simon Fraser University, Burnaby, BC V5A 1S6, Canada (e-mail: zzhang@cs.sfu.ca).

Digital Object Identifier 10.1109/TSMCA.2003.817047

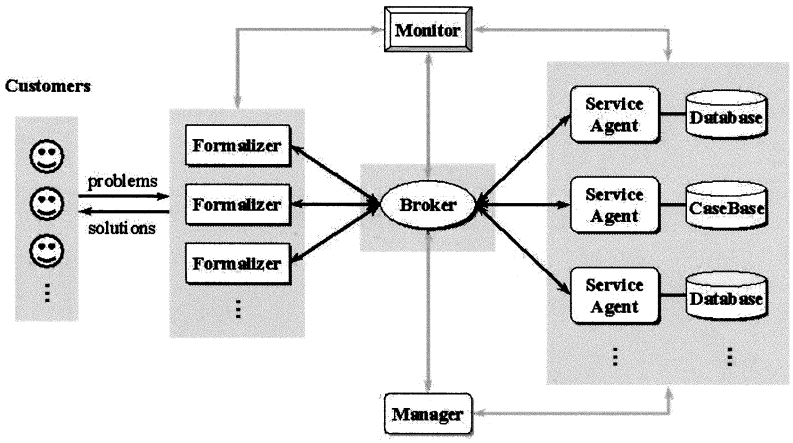

Fig. 1. Architecture of SANet.

tion, can provide higher quality solutions more efficiently and uniformly. This can lead to a reduction in customer service costs to the company. Second, the agent system can be used as a simulation and designing tool for setting up a call center. Based on past and anticipated records of incoming calls, designers of a call center can invoke the system and test for the need for human agents in each specific problem area. Such a simulation and design tool would be invaluable in the continual operation of a call center as well. Third, the system can be used as part of an automated call-center solution for customer service. The software agents and managers can be installed over a network of computers in a call center. These agents can then be dynamically managed to answer customers' questions. Because all agents are automated, this solution will use less resources than human agents. However, the number of incoming calls can easily out-pace the available computer resources. To deal with this, a typical method in network research is replication, by duplicating the resources. However, as we point out in this paper, to know which agents to replicate requires that we know the distribution and the growth of different problem types as a function of time. This is addressed by the schedule management algorithm discussed in the paper.

The organization of the paper is as follows. In Section II we first describe the architecture of SANet. Then, in Section III, we describe the scheduling algorithm for assigning incoming problems to agents. We then develop the agent assignment algorithm and the problem-type learning method in Sections IV and V, respectively. In Section VI, we present an application in which we simulate a call center environment of a cable-TV company and demonstrate our the experimental results that show that the proposed architecture and algorithms can provide high-quality service in real time. We discuss related works in Section VII and conclude in Section VIII.

II. SANet Architecture

To begin with, our work makes the following assumptions.

1) There is a collection of service agents who each specialize in one or more problem solving areas. Furthermore, agents' capability can change with time and an agent may increase his/her capability in solving a type of problem through more practice. Alternatively, an agent may decrease his/her capability in solving a problem due to other assignments.
2) Service agents overlap in their capabilities in problem-solving, and the amount of the overlap is not known by the broker agent ahead of time. This makes it important for the broker agent to learn and track the change in capabilities.
3) A service agent may or may not be available at any point in time. An example of an unavailable software agent is one such agent that is offered through a Web site where the server is temporarily out of order.
4) Each software agent freely maintains its own database, which in turn affects its capability. Also, at all times, the system functions can be monitored by a monitor agent for management purposes.
5) If a problem can be solved by an agent, then the agent can solve the entire problem rather than portions of it.

Our goal is to maintain high problem-solving quality and efficiency when large numbers of problems arrive.

A. Agents in the SANet

Fig. 1 shows the architecture of SANet. SANet has the following components:

1) Software and Human Service Agents: A software service agent is a problem solver which is capable of solving one or more types of problems. This agent may be an expert system, a rule-based reasoning system, a case-based reasoning system, or other type of reasoning system. In the experiment that we present, we used a case-based reasoning system as a software agent. Software agents are coupled with the human agents in the network, forming the basic components in the SANet. Each software agent is usually associated with a knowledge base or case base. Software agents have different capabilities in solving problems because of the difference in the content of the data bases and case bases associated with them. They are distributed on a network of computers in a call center.

2) Broker Agent: The broker is responsible for selecting the appropriate agents to solve incoming problems according to the availability and capabilities of the agents. It is trained to identify problem types from problem descriptions and hence can decide on the type of a given problem automatically. Fig. 2 shows the architecture of the broker agent. The broker agent also consists

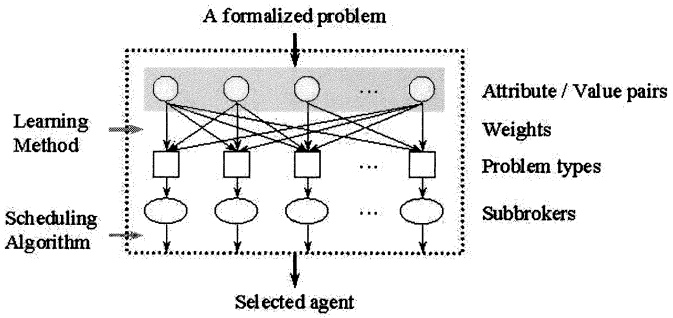

Fig. 2. Architecture of the broker agent.

of sub-brokers which help divide the problem set according to problem types. This helps scale-up the problem domain. We will describe the sub-brokers in more detail in the next several sections.

To divide the problems to problem types, a set of subbrokers maintain a table of availability and capabilities of the agents who can solve the problems of the corresponding type. To avoid being a bottleneck, the broker can also be split into several duplicates that can work in parallel. However, in the experiments in this paper, we test the architecture with only one broker agent.

3) Manager: To reduce the problem-solving time and get the higher problem-solving quality for customer's problems, sufficient service agents in the network are needed. However, hiring more service agents will increase the service cost. Therefore, a balance between cost and efficiency should be maintained. For example, if all agents in the network are very busy and many problems are still coming in, then the management agent should add more appropriate agents automatically. In contrast, if some agents in the network are near-idle, then they should be temporarily laid off. The manager is responsible for service agent assignment of this type. It maintains an information table for all service agents who can be added to the network. We discuss our agent-assignment algorithm in Section IV.

4) Formalizer: Problems in a call center environment may come in many ways, through telephone, fax, letter, email etc. The formalizer transforms the input problems into a standard representation—a set of attribute/value pairs. More than one formalizer can register with the broker agent. At this point, our formalizers are associated with the user interfaces (UIs) which are used by users to input the problems and display the solutions. To get the attribute/value pairs, each attribute is designed as a question and then the user's answer becomes the value of the attribute. For example, the problem type "picture problem" may be the value of attribute "what's the problem type?" When a formalizer receives a problem, a problem package is created, which includes a problem ID, a problem description, solutions suggested by agents and their respective quality.

5) Monitor: The monitor watches the working situation of the network and presents to the human user and the broker information such as the availability of the service agents, problem delivery procedure, etc. for the purpose of system management.

III. Overview of Network Algorithms

A. System Function and Parameters

We provide an overview of the operation of the SANet system in this section. The system is implemented in Java using the Aglet package from IBM [8]. Communication between different agents is done through KQML [4]. Initially, the broker should be launched at a certain location. As soon as the broker is created, it creates the manager and the monitor agent. Then we can create service agents and formalizers which can be located anywhere. After a service agent or formalizer is created, it sends out a registrar to the location of the broker. When the registration of a service agent is announced by the registrar, the broker will check the problem types which the service agent can solve, as well as updating the list of registered service agents. If there is a new problem type, the broker will create a new subbroker for this type. Then it sends the service agent ID and the capability for each problem type to the corresponding subbroker. When the problems are sent to the broker by the formalizers, the broker will deliver these problems to the subbrokers according to their types. The subbrokers then use the scheduling algorithm described in the next section to send each problem to the selected service agent according to the availability and capabilities of the service agents.

As soon as a service agent receives a problem, it sends a "not available" message to the broker and then the broker can inform both the monitor and the subbroker. When a service agent is done, the list of suggested agents and the list of the agents who have worked on the problem are updated. Also, the service agent updates the quality of solving this type of problem by the following quality-updating formula:

$$q'(t) = \frac{100q(t) + q(p)}{100 + 1} \tag{1}$$

where $q(t) \in [0,1]$ is the average quality of solving the problems of type t before solving the problem $p, q'(t) \in [0,1]$ is the average quality after solving p and $q(p) \in [0,1]$ is the quality of solving p. Essentially, this formula is a running average of the quality of the agent. In deciding on the new weight after solving the problem p, we allocate a weight ratio of 100 to one for the historical average $q(t)$ over the new quality value $q(p)$. The choice of 100 as the weight here is determined by the designer of the SANet network; in general the assignment of the weight should reflect how drastic a new quality value will change the running average, and the weight can be tuned by the designer. Finally, the denominator $(100 + 1)$ is used for keeping the resulting quality value in the range of zero to one.

One simple way to decide $q(p)$ is

$$q(p) = \begin{cases} 1, & \text{if } p \text{ is solved} \\ 0, & \text{if } p \text{ is not solved.} \end{cases} \tag{2}$$

Another way is to ask the user to provide a feedback in the form of a ranking score on the quality of the solution provided by the agent.

An important feature of our SANet system is its ability to continuously update the capabilities and availability of different agents in real time. After a problem is solved, the solution and the quality at this time are added to the problem package. Then the agent sends back the problem and the new quality to the subbroker, along with an "available" message to the broker. When a subbroker receives an availability changing message of a service agent, it updates its availability table. When a subbroker receives a problem with a new quality returning from a service agent, it

updates its capability table. If the problem was not solved, then the subbroker delivers it to a different agent. If the problem was solved, the subbroker sends it back to the broker and then the broker sends it back to the user through the corresponding formalizer.

B. SANet's Problem Scheduling Algorithm

We now describe SANet's scheduling method for the subbroker to deliver problems to agents. We assume that the type of each problem can be decided based on the result of problem-type learning (see Section V), whereby each problem is sent to a corresponding subbroker.

The problems a subbroker may receive can be divided into three kinds:

New problem	A new problem is one delivered from the broker.
Problem with solution	A problem with solution is one returned from an agent which solved the problem.
Problem without solution	A problem-without-solution is one returned by an agent which did not solve the problem.

Besides the capability table, the subbroker also maintains a problem queue. Each problem, which cannot be sent to any agent at current time will be put into the queue for future delivery. To select an agent to solve the problem, we define a capability function for each problem type.

The *capability function* for a problem type k is a function that maps the problem type to a real value in [0, 1]. It is assumed that the quality value of zero is considered undesirable and a quality value of one is the most desirable. We would like to relate this function to both the average quality of the solution provided by the agent for a k problem and the average time it takes to provide such solutions. Let a be an agent under discussion. Let $q(a,k) \in [0,1]$ be the average quality for this agent to solve the problems of type k, and $t(a,k) \in [0,\infty]$ be the average solution time. We denote the capability of the agent by $\mathrm{ca}_k(q(a,k), t(a,k))$.

The capability of an agent a is proportional to the quality $q(\,)$ of solution it provides. In addition, the longer it takes for the agent to provide a solution, the worse off the agent's capability. Depending on the specific problem domain and business case, different companies may define the delay cost differently. For example, in a large telephone company, a delay in answering a customer's questions may cause the customer to go to another company. Let $f(t)$ be a function of time that maps the time delay to the quality of solution in interval $[-1,0]$. $f(t)$ should be a monotonically decreasing function. For example, in our experiments described below, we set $f(t) = (1/t+1)$, where the 1 is added to t in the denominator to prevent a division by zero problem.

Given the above definitions, the capability of an agent for problem type k is defined as

$$\mathrm{ca}_k(q(a,k), t(a,k)) = \frac{q(a,k) + f(t(a,k))}{\max_k\{q(a,k) + f(t(a,k))\}}. \quad (3)$$

In our scheduling algorithm, we assume that we are given N problems, each associated with a problem type $k = 1, \ldots, K$.

```
Algorithm Problem-Schedule
Input: A problem P at subbroker B and a set of service agents
       under B. and a limit N.
Output: an agent A assigned to P

1. Let S be a set of available agents for solving P
2. If P has been routed through sub-broker N times, then
3.   Commit P to an agent A with maximal capability
4. Else,
5.     If S is empty, then re-route P through B
6.     Else
7.       Assign P to agent A in S with maximal capability value
8. End
```

Fig. 3. Problem scheduling algorithm. We assume that some agents has not worked on P yet. Otherwise, we send P to a special human agent.

There are M agents $A_l, l = 1, 2, \ldots M$. Agents can have overlapping capabilities as well that are changing with time. We wish to assign problems P_k to agents A_l such that the total sum of the capability values for all agents is maximized.

We adopt a greedy, first-come first-serve algorithm. The basic idea is to assign problems to the next available agent who can solve it with the highest quality. The scheduling algorithm is careful to avoid cycles by assigning an unsolved problem to the next capable and available agent. The greedy scheduling method is shown in Fig. 3.

This algorithm is designed to have the following two properties. First, we can ensure that a problem cannot wait forever before it is solved by an available agent. This is ensured in *step 3* of the algorithm, which commits a problem to an agent. Second, we claim that this greedy algorithm returns solutions with good quality and solution time. We verify this point experimentally in Section VI.

IV. Agent Assignment

To balance between service quality and cost, the agents in the network need to be adjusted according to the number and type of problems. We call the adjustment *agent assignment*. This includes adding and deleting some agents. The assignment is performed once in a period of time. Based on the number of incoming problems for each type and the agents in the network one period before, the manager estimates how many agents are needed in the near future to maintain the balance between service quality and cost. Then it adds or deletes appropriate agents based on the result. This section describes the cost model and the agent assignment algorithm used by the manager agent.

A. Cost Model

To evaluate the balance between service quality and cost of the network, we define a benefit function based on the difference between the reward and expense of using the network to solve problems. If the benefit is maximized, we have the best balance between service quality and cost. Our goal in agent assignment is then to maximize the benefit by adding/deleting appropriate agents.

TABLE I
AGENT ASSIGNMENT ALGORITHM BASED ON LOCAL SEARCH

Algorithm Agent Assignment Algorithm
Input: A period Δt; a problem set of P; an agent set A; an agent set $\tilde{A}$ in which each agent is possible to be added to the network; and an estimated benefit function $\tilde{b}(A, P)$;

Output: Two agent sets $A^+ \subseteq \tilde{A}$ and $A^- \subseteq A$, where $\forall a \in A^+$ should be added to A, and $\forall a \in A^-$ should be deleted from A.

1. let A^- and A^+ be empty sets;
2. **loop**
3. calculate $\tilde{b}(A, P)$;
4. find a^+ from $\tilde{A}$ with the biggest benefit $\tilde{b}(A \cup \{a^+\}, P)$;
5. find a^- from A with the biggest benefit $\tilde{b}(A - \{a^-\}, P)$;
6. **if** $\tilde{b}(A \cup \{a^+\}, P) \geq \tilde{b}(A - \{a^-\}, P)$ and $\tilde{b}(A \cup \{a^+\}, P) \geq \tilde{b}(A, P)$ **then**
   add a^+ to A and A^+;
   else if $\tilde{b}(A - \{a^-\}, P) \geq \tilde{b}(A \cup \{a^+\}, P)$ and $\tilde{b}(A - \{a^-\}, P) \geq \tilde{b}(A, P)$
   then
   delete a^- from A and add it to A^-;
   else
   exit loop
7. Update agents by A^+ and A^-.

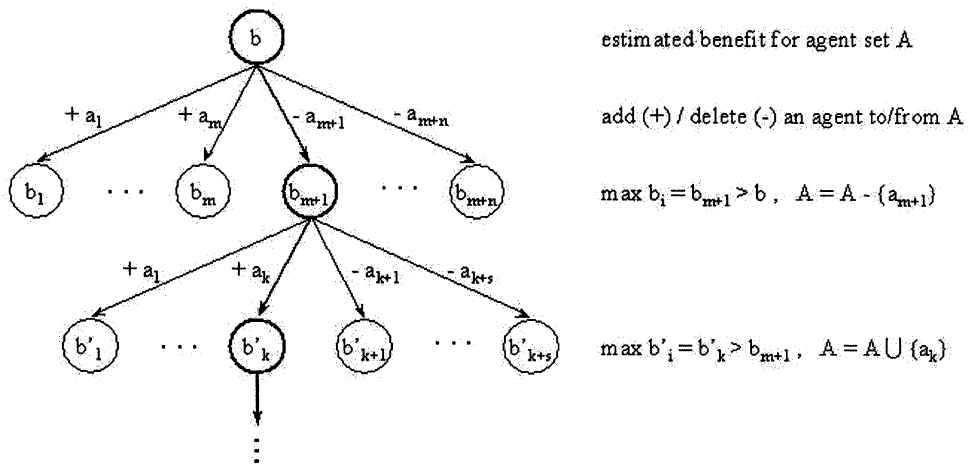

Fig. 4. Diagram of agent-assignment algorithm.

Let $P(k)$ be the set of all problems of type k in a problem set P. The *benefit* for an agent set A to solve all problems in $P(k)$ is denoted by $b(A, P(k))$ and is defined as follows:

$$b(A, P(k)) = R(A, P(k)) - E(A, P(k)) \tag{4}$$

where $R(A, P(k))$ and $E(A, P(k))$ are the average reward and the expected expense for agents in A to solve all problems in $P(k)$ during a previous period of time, respectively.

We estimate $R(A, P(k))$ and $E(A, P(k))$ as follows:

$$R(A, P(k)) = \bar{q}(A, P(k)) \tag{5}$$

$$E(A, P(k)) = \frac{(\bar{c}(A, P(k)) + \beta * \bar{t}(A, P(k)))}{\max_{a,p}\{c(A, P(k)) + \beta * t(A, P(k))\}} \tag{6}$$

where the maximum is taken over all agents and problems. The final benefit value is a number between plus and minus one. In these estimations, $\bar{q}(A, P(k))$ is the average solution quality and $\bar{c}(A, P(k))$ is the average solution cost per problem of type k using agents in set A. $\bar{t}(A, P(k))$ is the average solution time.

The parameter β, which converts time to cost, deserves some explanation. This parameter represents the cost to the call-center if a problem solution is delayed over a period of time. For example, suppose that every second a customer's problem is not solved from the time the customer enters the queue, the company loses \$10. Then β will be set to 10. In general, this constant can be set by the specific call-center company. For example, in a telecommunications company, the cost of delay is usually quite high because any delay in solving a customer's problem may cost the company the customer altogether.

B. Agent Assignment Algorithm

Using the definitions described above, the agent assignment algorithm is shown in Table I.

As we can see, this greedy algorithm is based on the *local search algorithms*. In each iteration, we try to find an agent to be added to or deleted from the network so that the estimated benefit has the maximal value. We keep doing this until no such agent can be found. Fig. 4 shows the diagram of this algorithm.

TABLE II
EXAMPLE OF AGENTS.

Agent	Type	Average Solution Quality	Average Solving Time (s)	Cost ($)
a_0	No Picture	0.8	60	200
a_1	No Picture	0.5	150	300
b_0	Poor Reception	0.8	60	200
b_1	Poor Reception	0.5	150	300
c_0	No Picture	0.8	60	200
	Poor Reception	0.8	60	
c_1	No Picture	0.5	150	300
	Poor Reception	0.5	150	

TABLE III
EXAMPLE OF AGENT ASSIGNMENT

	A	$\tilde{b}(A,P)$
	$\{a_1, b_1, c_1\}$	0.15
delete a_1	$\{b_1, c_1\}$	0.11
delete b_1	$\{a_1, c_1\}$	0.19
delete c_1	$\{a_1, b_1\}$	0.15
add a_0	$\{a_1, b_1, c_1, a_0\}$	0.23
add b_0	$\{a_1, b_1, c_1, b_0\}$	0.15
add c_0	$\{a_1, b_1, c_1, c_0\}$	0.22
add a_1'	$\{a_1, b_1, c_1, a_1'\}$	0.17
add b_1'	$\{a_1, b_1, c_1, b_1'\}$	0.14
add c_1'	$\{a_1, b_1, c_1, c_1'\}$	0.19

When an agent is added to agent set A, it adds its cost to the overall cost of the set, but reduces the problem solving time of the agents that share the same type by a certain amount.

Consider an example where there are six kinds of agents as shown in Table II. For instance, a_0 can solve the problems of type "*no picture*" with the initial solving quality 0.8 and the initial solving time 60 s. The cost of a_0 is $200. For agents c_0 and c_1, they can solve the problems of both types "*no picture*" and "*poor reception*."

Consider a period of time $\Delta t = 3600$ s (1 hour), after which agent assignment is performed. Let the problems P include 100 problems of type "*no picture*" and ten problems of type "*poor reception*." That means the network received these problems during the last period of agent assignment (i.e., the last one hour). Let $A = \{a_1, b_1, c_1\}$, which means three agents are in the network before the assignment. Let $\tilde{A}$ be a set of available agents which include enough agents of each kind as shown in Table II. That means the same agents in Table II can be added to the network more than once.

Before the loop, we set A^+ and A^- to empty.

Loop 1: We try to add or delete an agent in A and calculate the estimated benefit $\tilde{b}(A,P)$ for each A after the deleting/adding. The results are shown in Table III.

The biggest value for the estimated benefit is 0.23 when add a_0 to A. Hence, we have

$$A+ = \{a_0\} \quad A = \{a_1, b_1, c_1, a_0\} \quad \tilde{b}(A,P) = 0.23.$$

The rest of the calculation simply repeats until no improvements can be observed. Finally, we get $A+ = \{a_0, a_0', a_0''\}$ and $A- = \{a_1, c_1\}$ for this time of agent assignment. Based on this result, the manager will delete a_1 and c_1 from the network, and add three a_0's to the network.

V. MATCHING PROBLEMS WITH SERVICE AGENTS THROUGH LEARNING

In order to pass a given problem to an appropriate subbroker, the broker has to decide what type the problem is. We can relay this task to the outside end users, but this will impose an extra requirement that an end user have to know the problem type *a priori*. This will not only burden the users but also, more importantly, force a user to choose a problem type when in fact the user may not know exactly what the problem type is. Furthermore, the situation is made more complex by the dynamic nature of the capabilities and availability of the distributed agents. Obviously, we wish to alleviate such a burden, making automatic the decision-making process of problem types in the broker agent.

As discussed before, after a problem is input, the formalizer will convert it into a standard representation form—a set of attribute/value pairs. Essentially, the broker will decide the problem type based on these attribute/value pairs. Therefore, we can view the relationship between attribute/value pairs and problem types as that shown in Fig. 2. From the figure, it can be seen that each problem type is connected to a set of attribute/value pairs. The input attribute/value pairs will map to a problem type. The mapping is done through a weighted sum of the attribute-values that are present in a problem description. However, in real applications, not all attribute/value pairs have equal importance in this decision-making process. We thus consider assigning different weights to the connections between attribute/value pairs and problem types. These weights can be learned through a training phase.

In [16], a learning model is applied to a network which is very similar to the one shown in Fig. 2. The learning model is a two-level architecture following a the learning algorithm that is a variation of the perceptron learning algorithm using a back-propagation algorithm to adjust the weights [18].

In particular, the training process of a problem type in the broker can be described as follows. After an input problem is formalized into a set of attribute/value pairs, these pairs will be fed into the network shown in Fig. 2. The ranking scores for individual problem types will be computed. During the training, an expert critiques whether the highest-ranking problem type is the correct one. This will, in turn, be taken by the learning policy to update the relevant weights. In our broker agent, the training process is offline assisted by domain experts. Our experiments show that after learning, the broker agent will choose, on behalf of the user, an appropriate problem type to be sent to the corresponding subbroker.

The learning algorithm can also deal with occasional misclassification of incoming problems. When a problem is misclassified into a wrong type, it is sent to an agent who has low capability in solving the problem. The quality of the solution will be used to complete a feedback loop for retraining the weights. This will cause the classifier to improve its classification ability the next time it sees the same or a similar problem during retraining.

Consider the overall computational complexity of the SANet system, we note that the training time complexity is mainly dominated by the training time for the weights of the classifier. In our subsequent experiments, we show that this cost is around

3–5 s per training problem in each training round, and it usually takes around three training rounds for the errors to diminish. In real-time operations of the system, we also note that the assignment for an incoming problem takes linear time in the number of problem features. In addition, the assigned agent will solve the problem using a certain amount of time. If we assume that the agents use case bases to organize their knowledge bases, and that the knowledge base is indexed based on the problem features, then the problem solving time is again linear in the number of problem features.

VI. Experiments

This section describes the experiments we have conducted and discusses the experimental results. We have done two series of experiments. One is for testing the performance of SANet's agent assignment algorithm, and the other is for testing the problem-type learning for the broker.

A. Cable-TV Call Center Example

We have applied SANet to a simulated call-center environment of a local cable-TV company. In this application, a software service agent is a case-based reasoning (CBR) system [9] that can solve customer's cable-TV problems automatically by retrieving similar cases in a case base. A live agent is a staff in the call center with an assistant CBR system. At this point, the system can solve 16 types of problems. Each problem type has one to ten cases in one or more case bases. To input a problem, the user has to answer some questions. The question list is as follows.

Q1 What's the problem type?
Q2 Which channels have the problems?
Q3 Is the account an active cable account?
Q4 Is the problem affecting more than 1 outlet?
Q5 What does the picture on the screen look like?
Q6 Is the customer in an affected area?

These questions are optional to answer. If the user answered question 1, which means the user knows the problem type, then the broker just passes this problem to the corresponding subbroker. Otherwise, the broker has to decide the problem type. In other words, the user has two choices—let the problem type be decided by himself or the broker. For instance, the user may have answered questions 1, 2, and 3. Then the formalizer creates a problem package such as the following.

Problem ID	111
Description	the set of q/a pairs
Solution	Null
Suggested Agents	ϕ
Agent/Time List	ϕ

The set of q/a pairs is as follows

Q1	Poor reception
Q2	Channels 2 to 6
Q3	Yes

If the problem is sent to a live agent, the solution part may be filled by a staff who solved this problem. If it is routed to a CBR agent, a CBR system will then solve this problem and the solution part is filled by the most similar case in the case base. In this case, when the problem returns to the user, the solution part looks like the following.

TABLE IV
Settings of Agents for a Subset of Problem-Type Set K

Kind of Agent	Solving Quality	problem solution time (s)	Cost
A_1	1.0	300	500
A_2	0.8	60	200
A_3	0.5	150	300
A_4	0.2	30	100

Case Name	Poor reception low band
Q1	Poor reception
Q2	Channels 2 to 6
Description	Channels 2 to 6 have poor reception
Solution	Sometimes caused by a loose connection. 1. Check channel outage note pad; 2. Check for a possible loose connection; 3. Try unplug power cord for television; 4. Try hook cable direct to Television;

Also, the agent/time list will be filled. This list provides all agents who have worked on this problem and the corresponding problem-solving time. The quality of solution can then be provided by the end customer based on his or her satisfaction with the solution. This feedback value is used to update the capability function for each agent. According to the solution quality, the end customer can also provide an evaluation value of the problem-type decision to the broker to adjust the learning result of the problem types.

This application system was developed for simulation by using the Java JDK 1.1.6 and ASDK V 1.0.3 (IBM's aglets software development kit) on an P200 NT Server.

B. Experiments for the Performance of SANet

We have conducted experiments to verify the performance of SANet's agent-assignment algorithm. Our expectation is that using the agent-assignment algorithm will keep the benefit of the network much higher than not using the algorithm (which corresponds to a static network of agents). This expectation is tested through a comparison of the network performance under the two situations. The application domain of the experiment is the call center of a cable-TV company, as described above.

In our experiment, we have a set of three problem types $K = \{\text{poor reception, no picture, no sound}\}$. For each subset $K' \subset K, K'$ is not empty, we have four kinds of CBR agents who can solve the problems of the types in K'. Table IV shows the solving time, quality and costs associated with the four types of CBR agent.

Because the number of subset K' is 6, we have $6 \times 4 = 24$ kinds of agents that are possible to be added to the network. We assume that there are enough agents for each kind at our disposal, and study the effect of assigning or removing them to and from the network. For every kind of agent, we choose one agent from A_2, A_3, and A_4 respectively to initialize SANet. Then there are $6 \times 3 = 18$ problem-solving agents on SANet at the beginning. The time interval of agent assignment, which is

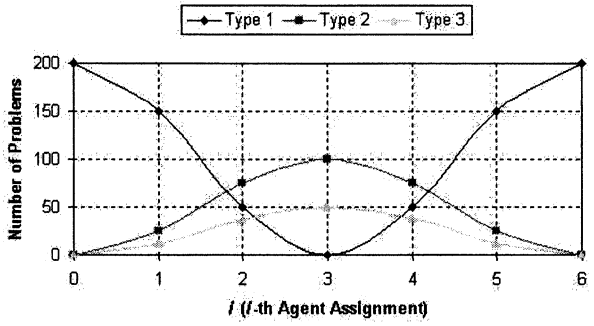

Fig. 5. Experiment 1— Incoming problem patterns.

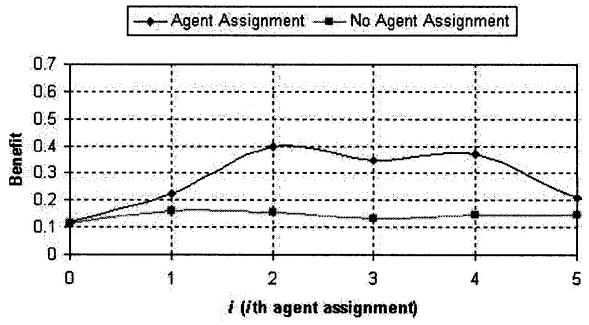

Fig. 6. Experiment 1—Benefit values.

denoted by Δt, is set to one hour, i.e., the agent assignment is done once every one hour.

1) Experiment 1: During each Δt, the total number of randomly created problems for each problem-type is based on the following *incoming problem pattern*:

$$\text{number of problems} = \frac{m}{2}\left(1+\cos\left(\theta+\frac{\pi}{3}i\right)\right)$$

where i denotes the ith agent assignment. $m = 200, 100, 50$ and $\theta = 0, \pi, \pi$ for problem-type 1, 2, 3, respectively. The sending pattern determines the number and type of incoming problems to the call center, as a function of time. In the cable-TV call-center domain, this assumption on the incoming problem pattern is reasonable because the problems usually arrive in a nonuniform way. The number of incoming problems is a function of time. For example, during a sporting event, there are usually a larger number of calls regarding how to tape a certain sports channel using the VCR. Likewise, when a certain region experiences an outage, a larger number of calls will pour in in a big wave.

Fig. 5 shows the graph of incoming problem patterns for each type. Those problems are divided into parts on average and sent to SANet every five min during the period from ith to $(i+1)$th agent assignment. For example, during the period from the first to the second periods, the number of problems for the type 1, 2, and 3 are 150, 25, and 12, respectively.

For each problem, we calculate the problem solution time which begins at the time it is sent to SANet and ends at with the time when it is returned with the solution. When a problem is solved, we get the quality of the agent which solved this problem. For each problem-type k, we calculate the average problem solution time $\bar{t}(A,k)$ and the average quality $\bar{q}(A,k)$ of all problems during each Δt. Then we calculate the value of the benefit function $b(A,P)$. The experiment was done twice with the same setting and incoming problem patterns to compare agent assignment with no agent assignment. Fig. 6 shows the result.

We can see that the value of the benefit function increases in situations where there is agent assignment as compared to no agent assignment. The statistical mean for the differences between agent assignment and no agent assignment is 0.13, with a 95% confidence, the p-value is 0.039.

Note that the agent assignment is based on the information gathered exactly one time-period before the current period, and

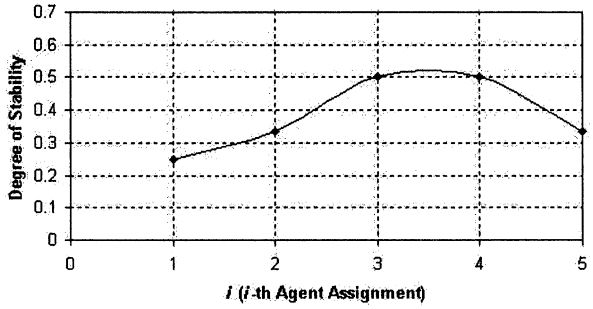

Fig. 7. Experiment 1—Degree of stability.

the number of problems during the next period is always different from the past with our incoming problem patterns. Hence, the result of our agent assignment may provide the best estimate for the current situation and the benefit we get varies depends on how different the incoming problem patterns are. This variation is also clear from the test results.

To make this concept clear, we define and calculate the *degree of stability* for the problems from the current agent-assignment period to the next one. Let n_i^k be the number of problems with type k during the ith agent-assignment period. Let r_i^k be the degree of stability for the problems with type k from the $(i-1)$th to the ith agent-assignment period. We define

$$r_i^k = 1 - \frac{\left|n_{i-1}^k - n_i^k\right|}{\max\left\{n_{i-1}^k, n_i^k\right\}}.$$

For example, the number of problems for type 2 during the second and the third agent-assignment period are 75 and 100, respectively. Let k be the type 2, we get $r_3^k = (|75-100|)/100 = 0.25$.

The degree of stability for the all problems from the $(i-1)$th to the ith agent-assignment period is denoted by r_i and is defined as the average degree of stability for all problem types during these two periods.

$$r_i = \frac{\sum_k r_i^k}{\text{number of types}}.$$

From the incoming problem patterns, we calculate r_i for each i. Fig. 7 shows the result.

Higher stability means that between the two adjacent agent-assignment periods, the difference is small. Higher

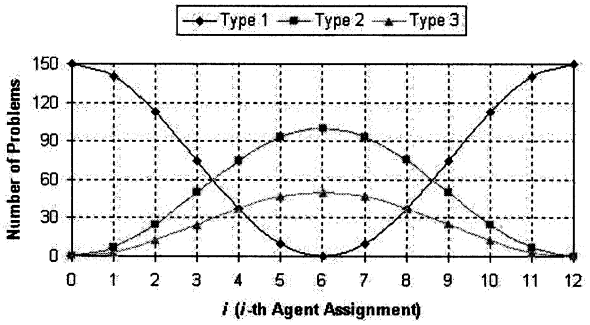

Fig. 8. Experiment 2—Incoming problem patterns.

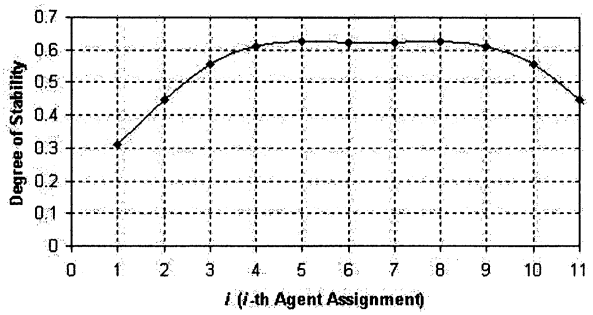

Fig. 9. Experiment 2—Degree of stability.

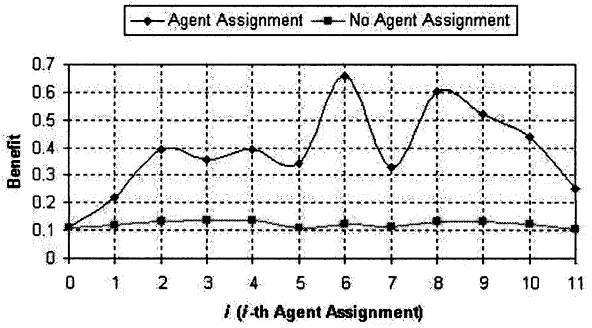

Fig. 10. Experiment 2—Benefit values.

stability implies a better benefit result. We can see this clearly by comparing Fig. 7 to Fig. 6.

2) Experiment 2: In Experiment 2, we shorten the length of time periods and perform more sampling. This is expected to increase the stability of the network and provide higher benefit due to more accurate estimate of the incoming problem types and size.

The incoming problem pattern is shown as follows:

$$\text{number of problems} = \frac{m}{2}\left(1+\cos\left(\theta+\frac{\pi}{6}\mathrm{i}\right)\right)$$

Where i denotes the ith agent assignment. $m = 150, 100, 50$ and $\theta = 0, \pi, \pi$ for problem-type 1, 2, 3, respectively.

Fig. 8 shows the graph of incoming problem patterns for each type.

Fig. 9 shows the stability. We can see that these rates are greater than that in Experiment 1.

Except the incoming problem pattern, the other settings in Experiment 2 are the same as in Experiment 1.

Fig. 10 shows the benefits for both agent assignment and no agent assignment.

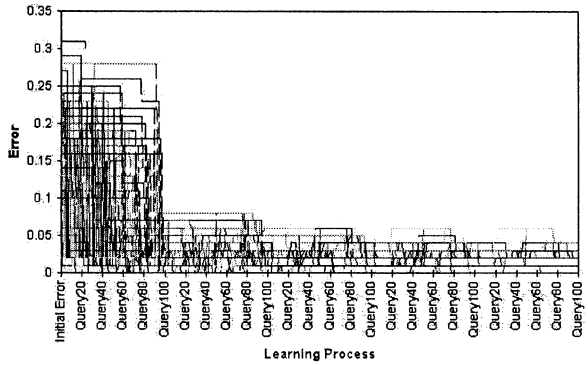

Fig. 11. Error convergence chart for 100 queries.

The statistical mean for the difference between agent assignment and no agent assignment is 0.27, with a 95% confidence, the p-value is 0.035. The t-test value is 3.6×10^{-5}, implying a large difference between the two populations shown in the figure. Comparing to Experiment 1, it is clear that the benefits are higher with the increasing degree of stability.

However, the benefits vary with time. There are at least two reasons for this. First, when making agent assignment, we are only performing local search. Thus, the agent set we get from agent assignment may not be the best one. Second, our agent assignment is based on the estimated benefit and hence there is a difference between the real benefit and the estimated one.

C. Experiment for the Problem-Type Learning

We introduced a learning model into our broker agent with the intention that it would automatically choose the most appropriate problem type on behalf of the user. In the following we demonstrate our experiments with the learning model.

The learning process is offline and mainly occurs in the broker agent. We create 20 problem types with each type being associated maximally with 20 attributes. In the experiment, on average each attribute can have two to five values. Accordingly, we also create a set of 100 user queries or problems represented by attribute/value pairs. The ranking of a problem type is between 0.0 and 1.0.

We feed these queries one by one into the learning model in the broker agent. The learning model then computes the rankings for the problem types and compares them with the desired ones specified by the queries. If there is a discrepancy between the computed ranking and desired ranking of a problem type, the learning process will be triggered and the corresponding weights will be updated. In the experiment, the learning process takes five rounds. Therefore, there is a total of 500 sample data points.

In Fig. 11, we show the error convergence chart for all the 100 queries after five learning rounds, where the X-axis represents the learning process that is composed of 500 learning data points while the Y-axis represents the error which is defined as the absolute distance between the computed ranking and desired ranking of a problem type in individual queries.

As can be seen from the figure, after five learning rounds, most of the queries have an error of the problem type falling within an acceptable range. In the experiment, we set this range to be 0.02 (which is the difference between desired and actual

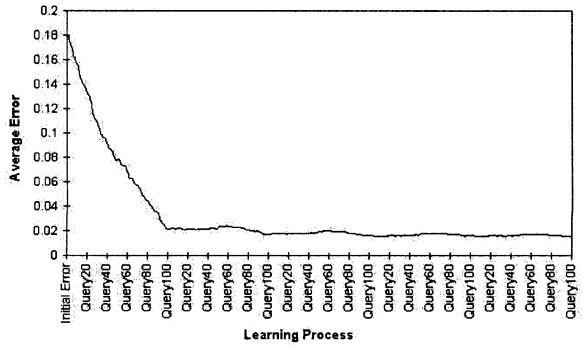

Fig. 12. Average error convergence chart.

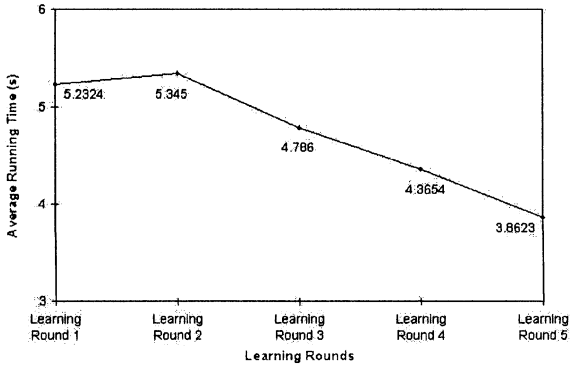

Fig. 13. Average running time for each learning round.

rankings, where the rankings are within zero and one range). However, we also observe that, due to the interactions among different queries, some problem types in the queries oscillate in their desired rankings, forming periodic waves, as shown in the figure.

In order to better understand Fig. 11, in Fig. 12, we plot the average error of problem types in individual queries, after the system is trained with 100 sample problems. In the figure, the two axes have the same meaning as before. However, the error is now averaged among the 100 problem types in all the queries after each learning data point is learned. There is still a total of 500 learning points along the X-axis. It can be seen that along the learning process, the average error for all the problem types tends to zero and will be stabilized at learning rounds 4 and 5. We also see that because of the interactions among different queries, the learning error could not be zero no matter how long the learning process undertakes.

We plot in Fig. 13 the average running time for each learning round. The X-axis represents the learning process while the Y-axis represents the average running time in CPU s for individual learning data points in a learning round. As shown in the figure, the running time decreases along with the learning process. This can be better explained that the more the learning model learns, the smaller the average error will be and the shorter the learning time it will take.

After learning the model has reached its optimal state, i.e., it satisfies almost all the queries. If the capabilities and availability of the distributed agents change, the queries should be changed accordingly. Therefore, the optimal state will be destroyed; another learning will be necessarily triggered to capture these changes. Therefore, every time when given an input problem, the learning model in the broker will try to select the most suitable problem type and send it to the corresponding subbroker.

VII. Related Work

Several research areas are related to ours in the SANet framework. First, our work is closely related to the scheduling problems in artificial intelligence (AI) and operations research (OR) literature. In the past, scheduling has been widely studied by researchers from both AI [3], [7], [11], [14], [17] , and OR [6]. In OR, the resource assignment problem has been studied in depth. A typical example is the transportation problem, where given m sources, n destinations and demand, supply as well as cost functions, the goal is to find an optimal assignment of sources to destinations such that the total cost function is minimized. Solutions to this problem and its variations can be found in the OR literature; for example, see [5] and [6, ch. 7]. Compared with the problem studied in the SANet framework, we observe that the transportation problem assumes that all inputs are given. The sources and destinations are both well mapped out. However, in SANet, there is a problem of "guessing" which problem type an incoming problem should belong to, a problem that is addressed by classification methods in machine learning. In addition, in SANet, the parameters such as capability of agents and agent costs are adaptively acquired as opposed to being given as part of the input. Finally, in the transportation problem, there is a fixed number of sources and destinations, whereas in SANet, the number is dynamically changing. In the latter the broker has to intelligently make its decisions in real time. Despite these differences, there is also a lot in common between the problems studied in OR and in SANet. In particular, SANet can employ the optimization techniques to solve the agent-assignment problems optimally. In fact, similar to some OR literature for dynamically changing problems [13], the solution adopted in SANet is a local-search method. This method has the advantage of being able to cope with dynamically expanding nodes in the search tree.

Another related work is multiagent research. In this field, Wellman [15] proposed a quantitative market model for a well-defined class of distributed configuration design problems. The model defines a computational economy to allocate basic resources to agents participating in the design. This idea is similar to our definition of the benefit function to maintain a balance between the service quality and cost. However, Wellman's method assumes that the agents participating in the design are fixed and try to get a optimal design result by allocating basic resources to the agents. In contrast, our method assumes that agents in the network are not fixed and try to get a optimal solving result by adding/deleting appropriate agents.

The broker in SANet is closely related to *information broker* research. The basic idea of an information broker is to gather information and find solutions from heterogeneous resources on a network of resources. An example of information-gathering agent systems is the BIG system [10], which integrates many AI techniques in one system in a scalable manner. Many information-gathering systems employ brokers as their central com-

ponents. For example, brokers are used in InfoSleuth [12] and SIMS [1], [2]. These systems take a user query and translate it into subqueries that can be executed by various information agents attached to information sources. Most of these systems mainly focus on the integration of different knowledge representations. Therefore, they do not focus on the scheduling problems, at least for now, as their main features. In our application domain, the agent-scheduling issue is critical because a good solution can ensure that a call center provides high quality service with low service cost.

VIII. Conclusion and Future Work

This paper described SANet—a service agent network for call-center scheduling which is capable of providing services to customers by selecting the most appropriate agents according to the availability and capabilities of the agents in the network, as well as environmental conditions. Our main contribution is an agent-assignment algorithm that allows agents to be added and deleted on demand in real time, based on the tradeoff between quality and cost. Our scheduling algorithm takes into account the agent availability, capability and cost when assigning problems to agents. The mapping between problems and agents can be learned through classification training.

The system is fully developed as a Java-based simulation system for call-center design. Currently we are engaged in research collaborations with a local cable-TV company to put the system on field trial.

In the future, we plan to incorporate other efficient task-scheduling methods and compare with our own. We also plan to study how to decompose a large problem into smaller sub-problems and then integrate the solutions that are returned by agents.

Acknowledgment

The authors wish to thank the anonymous reviewers for their comments.

References

[1] Y. Arens, C. Y. Chee, C.-N. Hsu, and C. A. Knoblock, "Retrieving and integrating data from multiple information sources," *Int. J. Intell. Coop. Inform. Syst.*, vol. 2, no. 2, pp. 127–158, 1993.

[2] Y. Arens, C. A. Knoblock, and C.-N. Hsu, "Query processing in the sims information mediator," in *Advanced Planning Technology*, A. Tate, Ed. Menlo Park, CA: AAAI, 1996.

[3] J. Dey, J. Kurose, and D. Towsley, "On-line processor scheduling for a class of iris real-time tasks," *IEEE Trans. Comput.*, vol. 45, no. 7, July 1996.

[4] T. Finin, Y. Labrou, and J. Mayfield, "KQML as an agent communication language," in *Software Agents*, J. Bradshaw, Ed. Cambridge, MA: MIT Press, 1997.

[5] G. N. Frederickson, "A note on the complexity of a simple transportation problem," *SIAM J. Comput.*, vol. 22, no. 1, pp. 57–61, 1993.

[6] F. S. Hiller and G. J. Lieberman, *Introduction to Operations Research*. New York: McGraw-Hill, 1990.

[7] J. Jonsson and J. Vasell, "Evaluation and Comparison of Task Allocation and Scheduling Methods for Distributed Real-Time Systems," Dept. Comput. Eng., Comput. Arch. Lab. (CAL), MicroMultiProcessor Group (MMP), 1996. Tech. Rep., CTH.

[8] D. B. Lange and M. Oshima, *Programming and Deploying Java Mobile Agents with Aglets*. Reading, MA: Addison-Wesley, 1998.

[9] D. B. Leake, "Cbr in context: The present and future," in *Case-Based Reasoning, Experiences, Lessons & Future Directions*, D. B. Leake, Ed. Menlo Park, CA: AAAI, 1996, pp. 1–30.

[10] V. R. Lesser, B. Horling, F. Klassner, A. Raja, T. A. Wagner, and S. X. Zhang, "Big: A resource-bounded information gathering agent," in *Proc. 15th National Conf. Artificial Intelligence*, Jan. 1998.

[11] S. Minton, M. D. Johnston, A. B. Philips, and P. Laird, "Solving large scale constraint satisfaction and scheduling problems using a heuristic repair method," in *Proc. 8th Nat. Conf. Artificial Intelligence*, MA, 1990, pp. 17–24.

[12] M. Nodine, B. Perry, and A. Unruh, "Experience with the infosleuth agent architecture," in *Proc. Workshop Software Tools Developing Agents*, 1998.

[13] O. Martin, S. W. Otto, and E. W. Felten, "Large-step Markov chains for the TSP incorporating local search heuristics," *Oper. Res. Lett.*, vol. 11, pp. 219–224, 1992.

[14] P. Sparaggis and D. Towsley, "Optimal routing and scheduling of customers with deadlines," *Prob. Eng. Inform. Sci.*, vol. 8, no. 1, Jan. 1994.

[15] M. P. Wellman, "A computational market model for distributed configuration design," in *Reading in Agents*, M. N. Huhns and M. P. Singh, Eds. San Francisco, CA: Morgan Kaufmann, 1998, ch. 4.

[16] Z. Zhang and Q. Yang, "Toward lifetime maintenance of case based indexes for continual case based reasoning," in *Artificial Intelligence: Methodology, Systems, and Applications. 8th International Conference, AIMSA'98. Sozopol, Bulgaria, September 1998. Proceeedings, Lecture Notes in Artificial Intelligence*, F. Giunchiglia, Ed. New York: Springer, 1998, vol. 1480, pp. 489–500.

[17] Y. C. Zhuang, C. K. Shieh, and T. Y. Liang, "Centralized load balance on distributed shared memory systems," in *Proc. 4rth Workshop Compiler Techniques High-Performance Computing*, Mar. 1998, pp. 166–174.

[18] J. M. Zurada, *Introduction to Artificial Neural Systems*. St. Paul, MN: West, 1992.

Qiang Yang (M'03) received the Ph.D. degree from University of Maryland, College Park, in 1989.

He is Associate Professor, Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong. His research interests are planning, case based reasoning, and data mining. He had been a Faculty Member at University of Waterloo, Waterloo, ON, Canada, and Simon Fraser University, Burnaby, BC, Canada.

Yong Wang received the B.Sc. degree in mathematics from Hunan Normal University, China, the M.Sc. and Ph.D. degrees in electrical and computer engineering from Nagoya Institute of Technology, Nayoya, Japan, in 1983, 1995, and 1998, repectively.

He worked as a Postdoctoral Researcher at Simon Fraser University, Burnaby, BC, Canada, from August 1998 to July 1999. His research interests include case-based reasoning, genetic algorithms, planning and information agent. Since 1999, he has worked for several software companies and is currently a software consultant.

Zhong Zhang received the M.Sc. degree in artificial intelligence from the School of Computing Science, Simon Fraser University, Burnaby, BC, Canada, in 1988. He is currently pursuing the Ph.D. degree at the same university.

Mr. Zhang's research interests include artificial intelligence, data mining, and computational geometry. He has published papers in several international conferences and journals.