Initial Value Generation for Matchmaking in Middle Agents

Zili Zhang
School of Computing and Mathematics
Deakin University
Geelong Victoria 3217, Australia
email: zzhang@deakin.edu.au

Chengqi Zhang
Faculty of Information Technology
University of Technology, Sydney
PO Box 123 Broadway, NSW 2007 Australia

Yuefeng Li
School of Software Engineering and Data Communications
Queensland University of Technology
Brisbane Queensland 4001, Australia

Abstract

One of the major challenges that agents used in open environments must face is that they must be able to find each other. This is because in an open environment, agents might appear and disappear unpredictably. To address this issue, middle agents have been proposed. The performance of middle agents relies heavily on the matchmaking algorithms used. Matchmaking is the process of finding an appropriate provider for a requester through a middle agent. The practical performance of service provider agents has a significant impact on the matchmaking outcomes of middle agents. Thus the track records of agents in accomplishing similar tasks in the past should be taken into account in matchmaking process. Considering that there are no track records available at the launching of an agent system, this paper discusses some ways to provide reasonable initial values for the track records. With the agents' history and the initial values of the track records, the performance of matchmaking algorithms can be improved significantly.

1. Introduction

An agent is an encapsulated computer system that is situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design objectives [1]. A multi-agent system can be defined as a loosely coupled network of agents that work together to make decisions or solve problems that are beyond the individual capabilities or knowledge of each agent [2].

One of the most important driving forces behind multi-agent system research and development is the Internet: agents are populating the Internet at an increasingly rapid pace. These agents invariably need to interact with one-another in order to meet their designers objectives. In such open environments, agents that would like to coordinate with each other (either cooperate or negotiate, for example) face two major challenges: first, they must be able to find each other (in an open environment, agents might appear and disappear unpredictably), and once they have done that, they must be able to inter-operate [3]. To address the issue of finding agents in an open environment like the Internet, middle agents [4] have been proposed. Each agent advertises its capability to some middle agent. A number of different agent types have been identified, including matchmakers or yellow page agents (that match advertisements to requests for advertised capabilities), blackboard agents (that collect requests), and brokers (that process both). Middle agents are advantageous since they allow a system to operate robustly in the face of agent appearance and disappearance, and intermittent communications.

The performance of middle agents relies heavily on the matchmaking algorithms used. Matchmaking is the process of finding an appropriate provider for a requester through a middle agent. The state of the art of matchmaking algorithms is as follows: Agents provide meta-information about the services they offer to middle agents. Based on relevance calculation methods (mainly distance functions) the middle agents use this information for matching and provide the best fitting offers to a requester agent. The fact that is not taken into account is that the meta-information is defined by the provider agent itself. This means that in general the advertisements of the provider agents are not proven but self defined.

On the other hand, there are usually more than one service provider agents claim that they have the same or very similar capabilities to accomplish a task in an application.

For example, in financial investment applications one usually needs to predict the interest rate. There are different techniques for interest rate prediction such as neural network (NN) technique and fuzzy logic with genetic algorithm (FLGA) [6], but their prediction performances are different. If there are two interest rate prediction agents, one based on NN, the other based on FLGA, which one should be chosen to predict the interest rate? In such cases, current matchmaking algorithms can only choose one provider agent randomly. As the quality of service of different service provider agents varies from one agent to another, even though they claimed they have the same capabilities, it is obvious that it is not a good strategy to randomly choose a provider agent for the requester agents.

To this end, we provided a solution to this issue by introducing an algorithm which can consider agents' track records [5]. The work described in [5] advances the state of the art by enhancing the existing matchmaking approaches by a quality measure which relates to the agents performance in live applications.

The track records or "credit histories" of agents are accumulated gradually during the executing process of a multi-agent system. Thus there are no track records available when the system is first launched. In such cases the improved matchmaking algorithm can only randomly choose one agent with the requested capability, which is the same as other matchmaking algorithms.

If one can find some way to provide reasonable initial values for the track records, one can choose the agents based on the initial values at the very beginning. With the track records accumulated, the initial values and the track records are combined to choose the agents. In this way, the shortcoming of the algorithm considering track records – no track records being available at the very beginning – is overcome. This is just the task of this paper.

This paper will propose some ways to provide initial values for the track records. The basic idea is to give a set of problems (called benchmark problems), and ask all the agents who claimed that they have the capabilities to solve these problems. By calculating the "distances" between the solutions to the problems the agents gave and the benchmark results, one can determine the initial values for the track records of these agents. Of course, if the benchmark results are unknown in advance, we must find the benchmark results first.

The rest of the paper is structured as follows. Section 2 gives the basic idea for initial value generation of track records. Section 3 is the description of benchmark problems and benchmark values. Section 4 presents the initial value generation process with known benchmark results. When benchmark results are unknown, an approach is proposed to find them in Section 5. Section 6 concludes the paper.

2. Basic Idea for Initial Value Generation

Suppose we want to delegate a task to a person, but we have ten persons all claimed that they have the capability to accomplish the task. In such a case whom should we choose to delegate the task? If we have the track records of the ten persons in accomplishing similar tasks in the past, we can make the decision based on their performance/accomplishment history. This is what we discussed in [5]. If there is no any information about the ten persons in accomplishing similar tasks, one simple and efficient way we usually utilize in real life is to design a set of problems and ask all the candidates to solve these problems. We assess the solutions provided by the candidates and delegate the task to the candidate with the best solutions. The simplest form of such kinds of problems is different examination papers. In this paper we call such a set of problems "benchmark problems". The "standard" solutions are called "benchmark results" or "benchmark values".

According to the above scenario, we can summarize the basic idea of initial value generation approach as follows. Before putting a multi-agent system into practical operation, the system is "trained" with a set of benchmark problems. That is, the middle agent is run with a matchmaking algorithm first (e.g., **find_nn** algorithm [7]) . The middle agent then asks the agents with the same or similar capabilities (based on the returned results of the matchmaking algorithm such as **find_nn**) to solve the benchmark problems. By testing the results provided by these agents against the benchmarks, one obtains an evaluation of these agents for their performance on solving the benchmark problems. This evaluation is then used as the initial value of these agents' track records.

3. Description of Benchmark Problems and Benchmark Values

In order to extract an appropriate description of benchmark problems and benchmark values, we take the software risk analysis as an example.

Assume there is a software project and we want to analyze the risk of this project. From software engineering risk management point of view, there are some principal software risk factors that influence the risk of a software project [8]. These software risk factors include organization, estimation, monitoring, development methodology, tools, risk culture, usability, correctness, reliability, and personnel. The software risk factor *organization* addresses risks associated with the maturity of the organization structure, communications, functions, and leadership; The software risk factor *estimation* focuses on risks associated with inaccurate estimations of the resources, schedules, and costs

Table 1. Description of Benchmark (BM) Problems

Agent	S_1			...	S_m		
	a_1	...	a_n	...	a_1	...	a_n
A_1	$b_{11}^{A_1}$	...	$b_{1n}^{A_1}$	...	$b_{m1}^{A_1}$	...	$b_{mn}^{A_1}$
A_2	$b_{11}^{A_2}$	...	$b_{1n}^{A_2}$	...	$b_{m1}^{A_2}$	...	$b_{mn}^{A_2}$
⋮		⋮		...		⋮	
A_k	$b_{11}^{A_k}$	...	$b_{1n}^{A_k}$	...	$b_{m1}^{A_k}$	...	$b_{mn}^{A_k}$
BM	b_{11}	...	b_{1n}	...	b_{m1}	...	b_{mn}

needed to develop software; The software risk factor *monitoring* refers to risks associated with identifying problems; The software risk factor *development methodology* identifies the methods by which software is developed; The software risk factor *tools* focus on risks associated with the software tools used when software is developed; and so on. To analyze the risk of a software project is to determine the values (e.g., low, medium, high etc.) of these software risk factors. The software risk factors here can be viewed as attributes describing the software risk problem. More generally, we can say that solve a problem is to find the attribute values related to the problem.

Formally, let $A = \{A_1, A_2, \ldots, A_k\}$ be the agent set with the same or similar capabilities. We use $S = \{S_1, S_2, \ldots, S_m\}$ to denote the problem set. Each problem $S_i \in S$ $(i = 1, 2, \ldots, m)$ has a related attribute set $a = \{a_1, a_2, \ldots, a_n\}$. We say agent A_i solved problem S_j if it returns the values of the n attributes related to the problem. The values can be numeric or non-numeric (linguistic values). Suppose the benchmark values of these attributes denoted by $B_i = \{b_{i1}, b_{i2}, \ldots, b_{in}\}(i = 1, 2, \ldots, m)$ and, the values returned by agent A_j denoted by $B_i^{A_j} = \{b_{i1}^{A_j}, b_{i2}^{A_j}, \ldots, b_{in}^{A_j}\}(i = 1, 2, \ldots, m, j = 1, 2, \ldots, k)$. The description of the benchmark problems is then summarized in Table 1.

The next step in the initial value generation process is to calculate the "distances" between the returned values by agent A_j and the benchmark values. There are many definitions of "distance". Here the distance is defined in terms of standard Euclidean distance. The distance between B_i and $B_i^{A_j}$ is defined to be d_j, where $d_j = \sqrt{\sum_{r=1}^{n}(b_{ir} - b_{ir}^{A_j})^2}$. Then these distances are added to the database of the middle agent as the initial values of the track records.

Considering that the initial values and the track records need to be combined when accumulated, the distances were mapped to the satisfactory degrees defined in [5]. As there are 7 levels of satisfactory degrees–*strong satisfaction, satisfaction, weak satisfaction, neutral, weak unsatisfaction, unsatisfaction,* and *strong unsatisfaction*, each level accounts for 1/7 of the distance range. Therefore, if the distance is between 0 and 0.143, *strong satisfaction* will be the initial value of the agent's track record; If the distance is between 0.143 and 0.286, *satisfaction* will be the initial value etc. The mapping results are shown in Table 2.

Table 2. Mapping Results between Distance and Satisfactory Degree

Distance Range	Satisfactory Degree
0 to 0.143	*strong satisfaction*
0.143 to 0.286	*satisfaction*
0.286 to 0.429	*weak satisfaction*
0.429 to 0.572	*neutral*
0.572 to 0.715	*weak unsatisfaction*
0.715 to 0.858	*unsatisfaction*
0.858 to 1.0	*strong unsatisfaction*

In the process of initial value generation, there are two situations that need to be considered. One is the benchmark values in Table 1 are known in advance, the other is the benchmark values are unknown. We will discuss two cases respectively in the subsequent sections with examples.

4. Initial Value Generation with Known Benchmark Results

For different applications, the benchmark problems are different. That is, the benchmark problems are application-dependent. In this section, we take the financial application as an example to discuss the initial value generation problem with known benchmark values.

In financial applications, different models (e.g., fuzzy logic and genetic algorithm model [6]) can be used for interest rate prediction. In this section, two soft computing (SC) agents for interest rate prediction (one is based on neural network, called SC_Agent_NN, the other based on fuzzy logic and genetic algorithm, called SC_Agent_FLGA) are taken as examples to show how to determine the initial values for the two agents. The initial values are based on the predictive capabilities of these two SC agents.

4.1. Construction of Benchmark Problems

When predicting the interest rate (as represented by 91-day Treasury bill rates), both of the agents take the changes of previous Treasury-bill (T-bill) rates, real gross national product (GNP), consumer price index (CPI), M2 money supply, and personal wealth (W) as inputs. Personal wealth is the accumulation of the difference between personal income and personal consumption. The M2 money supply consists of all cash in circulation and deposits in savings

and check accounts, and represents readily available liquid assets. The consumer price index is a measure of the inflation trend. The outputs are the changes of next T-bill rates (predicted interest rates). Quarterly data are used.

We use the history financial data of the five factors (from 1966 to 1987) provided in Appendix B of [6] to construct the required benchmark problems.

There is some evidence to suggest that fundamental financial market characteristics change over a period of four to five years [9]. That is, the market "forgets" the influence of data that is more than five years old. For this reason, five-year data windows are used. 15 data windows are examined, each starting in the first quarter of the years 1967 through to 1981, respectively. The ending quarters for each data window will be the fourth quarters of the years 1971 through to 1985. This means there are 15 benchmark problems. The inputs of benchmark problem S_1, for example, are the data from 1967 to 1971. The benchmark value for these inputs is the T-bill rate of the first quarter of 1972, -0.81.

4.2. Experimental Results

The 15 data windows are used to train these two agents (neural network and genetic algorithm). We then let the agents predict the interest rate of the first quarter following the training data windows. For example, for training data of 1967-1971, the outputs of the agents are the (predicted) T-bill rate of the first quarter of 1972. The prediction results of the two agents on the 15 benchmark problems are summarized in Table 3.

The average distance for the prediction values of SC_Agent_NN is 0.287. The value for SC_Agent_FLGA is 0.123. Based on the prediction results and the average distances, one can see that the prediction performance of SC_Agent_FLGA is better than that of SC_Agent_NN for the benchmark problems. Mapping the distances to satisfactory degrees (refer to Table 2), *weak satisfaction* is added to the track record of SC_Agent_NN as its initial value, and *strong satisfaction* to the track record of SC_Agent_FLGA as its initial value. Hence at this stage, if the middle agent needs to pick one agent for interest rate prediction, the result is SC_Agent_FLGA. Of course, the situation may change with the accumulation of track records of these agents.

5. Initial Value Generation with Unknown Benchmark Results

In the case discussed in the previous section, one must know a *priori* of the attribute values of the benchmark problems, but this is not always the case. In some situations, it is impossible to obtain the attribute values in advance. In such cases, one can ask the agents to solve these problems first. One can then try to cluster the attribute values returned by agents using cluster analysis methods. In this way, "heuristic" attribute values can be obtained. One can then use the "heuristic" attribute values as the benchmark values and back to the situation discussed in the previous section.

There are seven steps involved using cluster analysis algorithms to determine the benchmark results. Before presenting these steps in details, a brief introduction to fuzzy cluster analysis algorithms [10][11], which are used in our experiments, is given.

5.1. A Brief Introduction to Fuzzy Cluster Analysis

The aim of a cluster analysis is to partition a given set of data or objects into clusters (prototypes). This partition should have the following properties: (1) Homogeneity within the cluster, i.e., data that belong to the same cluster should be as similar as possible; (2) Heterogeneity between clusters, i.e., data that belong to different clusters should be as different as possible. The concept of "similarity" has to be specified according to the data. Since the data are in most cases real-valued vectors, the Euclidean distance between data can be used as a measure of the dissimilarity.

The fuzzy clustering algorithms that we will use classify the elements of the data set $X = \{x_1, x_2, \ldots, x_j, \ldots, x_n\} \subset \chi$ into classes $P = \{p_1, p_2, \ldots, p_i, \ldots, p_c\} \subset \wp$ by means of a membership matrix $U \subset [0,1]^{|P|\times|X|}$. A membership grade $u_{i,j}$ denotes the degree of belongingness of datum x_j to class p_i. The algorithms minimize the following objective function by means of alternating optimization (AO)

$$J(X, W; U, P) = \Sigma_{i=1}^{c} \Sigma_{j=1}^{n} u_{i,j}^{m} w_j d^2(k_i, x_j),$$

where $d : \chi \times \wp \to R$ measures the distance between data vectors and prototypes and w_j is the weight for the data vector x_j. The term *alternating optimization* comes from the fact that J is minimized by updating prototypes and membership alternatively. If $X = \{x_1, x_2, \ldots, x_j, \ldots, x_n\}$ and $P = \{p_1, p_2, \ldots, p_i, \ldots, p_c\}$ are finite, an analysis result $f : X \to F(P)$ can be represented as a $c \times n$ matrix U, where $u_{i,j} := f(x_j)(p_i)$. The algorithm used is shown below.

Fuzzy Clustering Algorithm

Let a data set $X = \{x_1, x_2, \ldots, x_j, \ldots, x_n\}$ be given. Let each cluster be uniquely characterizable by an element of a set P.

Choose the number c of clusters, $2 \le c < n$

Choose an $m \in R$ ($m > 1$)

Choose a precision for termination ε

Table 3. Predicting Results on Benchmark Problems

Agent	S_1	S_2	S_3	S_4	S_5	S_6	S_7
SC_Agent_NN	-0.53	0.45	0.03	-0.93	-0.67	-0.14	0.43
SC_Agent_FLGA	-0.78	0.73	-0.16	-1.36	-0.60	-0.28	0.53
Benchmark	-0.81	0.70	-0.40	-1.20	-0.69	-0.28	0.47

S_8	S_9	S_{10}	S_{11}	S_{12}	S_{13}	S_{14}	S_{15}
0.79	1.41	2.15	-0.95	-1.16	-0.33	-0.45	-0.41
1.03	1.84	2.66	-1.17	-1.00	-0.22	-0.87	-0.53
1.02	1.91	2.57	-1.10	-0.82	-0.03	-1.08	-0.11

Initialize $U^{(0)}$, $i := 0$
REPEAT
Increase i *by 1*
Determine $P^{(i)}$ *such that* J *is minimized by* $P^{(i)}$ *for fixed* $U^{(i-1)}$
Determine $U^{(i)}$ *according to certain conditions*
UNTIL $\|U^{(i-1)} - U^{(i)}\| \leq \varepsilon$

There are many ways to determine $U^{(i)}$. Different ways result in different fuzzy clustering algorithms. Refer to [10] for more details.

In the next subsection, we will discuss the steps in determining benchmark values by using this fuzzy clustering algorithm.

5.2. Determining Benchmark Values by Fuzzy Clustering

As discussed in Section 3, finding a solution to any problem can be viewed as determining the attribute values related to the problem. With this observation in mind, a set of benchmark problems can be designed, but the solutions to all these problems are unknown (e.g., we do not know what the risk exactly is for a software project). Some agents who claimed to have the capabilities are asked to solve these problems and return the solutions (the attribute values). These attribute values are then as inputs to the fuzzy clustering algorithm. The algorithm partitions the solutions into different clusters (called prototypes). We then choose the center of the cluster with the highest weight as the benchmark values for the problems. This implies that we should accept most agents opinions if their solutions are similar. It is reasonable to do so. Specifically, this process involves the following steps.

- Preparation: Given k similar problems (benchmark problems). Choose m agents having the capabilities to solve the k problems, respectively. The solution of each problem consists of n attributes.
- Use the fuzzy clustering algorithm to determine the clusters (prototypes) of the solutions for each of the k problems returned by the m agents. That is, the input of the algorithm is $m \times n$ data matrix. There are k such matrices. The outputs of the algorithm have the following format:

  (cluster

  (prototype (weight *value*) (center (*values for all the attributes*) ...

  (prototype (weight *value*) (center (*values for all the attributes*))

- Choose the prototype with the highest weight as the benchmark values for this problem. Repeat this step for k problems.
- Calculate the distances (using Euclidean distance) of the solutions given by the agents and the found benchmark values.
- Find the average distances of all the solutions provided by the m agents with the found benchmark values of the k problems.
- Normalize the distances to $[0, 1]$. Sort the m agents according to the average distances. Mapping the average distances to the seven satisfaction degrees (refer to Table 2), and use the satisfaction degrees as the initial values of these agents, respectively.

The average distances are used as the measurement of agents' performance in accomplishing benchmark problems. Such a measurement meets the "majority principle". That is, if one agent can accomplish most of the benchmark problems with high quality, another agent can only accomplish a few of the benchmark problems with high quality, the average distance of the first agent is shorter than that of the second one. Therefore, the results obtained according to the above process are convincing. The key in this process is step 2. In this step fuzzy clustering algorithm is used to cluster the solutions provided by different agents. Based on the clustering results, the "heuristic" attribute values, which are reasonable benchmark values, are determined.

5.3. Experimental Results

Recall the software project risk example in Section 3. Suppose some experts in this field are invited to assess the risks of some software projects. The experts are asked to give their assessment results by providing a number ($\in [0, 10]$) for each of the ten software risk factors. The bigger the number, the higher the risk concerning that risk factor. According to the process describing in the previous subsection, some experiments were conducted with $k = 30$, $m = 20$, and $n = 10$. That is, 20 agents with similar capabilities are delegated to assess 30 software projects (benchmark problems). The answer for each problem consists of 10 attributes. The data for the risk factors used in the experiments are randomly generated. For demonstration purpose, Table 4 shows the solutions of the 10 agents for one of the 30 benchmark problems.

Taking this as inputs, the clustering algorithm produces the following output for this problem:

(cluster
(prototype(weight 2.67437) (center (7.052511.84176 3.82815 5.14071 3.05296 1.94812 8.18994 6.23528 4.76634 4.21243)))
(prototype(weight 2.01146) (center (6.92752 2.09169 4.33178 5.17589 2.66185 2.1582 8.16927 5.69102 5.11877 3.92546)))
(prototype(weight 5.45996) (center (6.99357 1.95553 3.98165 5.02628 3.01206 2.00627 7.99989 6.01413 5.01984 3.9859))))

There are three different clusters (prototypes). Choosing the prototype with the highest weight (5.45996), we obtain the benchmark values for this problem. The attribute values are $a_1 = 6.99357$, $a_2 = 1.95553$, $a_3 = 3.98165$, $a_4 = 5.02628$, $a_5 = 3.01206$, $a_6 = 2.00627$, $a_7 = 7.99989$, $a_8 = 6.01413$, $a_9 = 5.01984$, and $a_{10} = 3.9859$, respectively. Table 5 lists the benchmark values for the first 5 benchmark problems determined by fuzzy clustering approach.

We then calculate the Euclidean distances between the solutions provided by the agents and the found benchmark values. The results are shown in Table 6. As space is limited, only the distances between 5 agents' solutions and benchmark results of 5 benchmark problems are listed.

To measure the performance in accomplishing benchmark problems, the average distances between agents' solutions and the benchmark values of all benchmark problems are used. The shorter the average distance, the better the performance. The average distance between agent A_i and the benchmark values is denoted by d_{A_i}. Based on the experimental data, these distances are $d_{A_1} = 0.2141$, $d_{A_2} = 0.0631$, $d_{A_3} = 0.0482$, $d_{A_4} = 0.0941$, $d_{A_5} = 0.1795$, $d_{A_6} = 0.3442$, $d_{A_7} = 0.2591$, $d_{A_8} = 0.3157$, $d_{A_9} = 0.1824$, $d_{A_{10}} = 0.2666$, $d_{A_{11}} = 0.2181$, $d_{A_{12}} = 0.215$, $d_{A_{13}} = 0.3946$, $d_{A_{14}} = 0.077$, $d_{A_{15}} = 0.1818$, $d_{A_{16}} = 0.1353$, $d_{A_{17}} = 0.2724$, $d_{A_{18}} = 0.2097$, $d_{A_{19}} = 0.1442$, $d_{A_{20}} = 0.4322$, respectively. From the average distances, it is obvious that agent 3 has the best performance in doing the benchmark problems, agent 2 has the second best performance, etc. Mapping these average distances to satisfactory degrees according to Table 2, the initial values of agents 2, 3, 4, 14, and 16 are *strong satisfaction*as their average distances are within ([0, 0.143]; the initial values of agents 1, 5, 7, 9, 10, 11, 12, 15, 18, and 19 are *satisfaction* as their average distances are between 0.143 and 0.286; the initial values of agents 6, 8, 13, and 17 are *weak satisfaction*; and the initial value of agent 20 is *neutral*.

6. Conclusions

Matchmaking in middle agents is essential for multi-agent systems used in open environments such as the Internet. Agents' track records have a strong impact on the outcome of matchmaking. Therefore agents' history performance/accomplishment (track records) should be taken into account in matchmaking. As the track records of agents are accumulated gradually during the executing process of a multi-agent system, there are no track records available when the system is first launched. To this end, this paper proposed ways to generate reasonable initial values for track records of agents.

The basic idea for initial value generation is to provide a set of benchmark problems and ask all provider agents claimed to have the same capabilities to solve these problems. By comparing the distances between the solutions provided by agents and the benchmark values, the initial values of these provider agents are then determined.

Two cases were identified in initial value generation. If the benchmark values are known in advance, the distances were calculated directly to determine the initial values of agents. If the benchmark values are unknown, fuzzy clustering algorithms were employed to find the benchmark values first. In both situations, experiments were conducted. The experimental results show the proposed initial value generation approaches are workable and can produce reasonable initial values. Combining the agents' history performance/accomplishment information and the initial values of track records, the performance of matchmaking algorithms can be improved significantly.

Thus far, all the discussions are based on one assumption: The track records of agents are credible. The situation with false track records is subject to further research.

Acknowledgement

The authors would like to thank Mr Hong Hu's support in conducting the experiments.

References

[1] M. Wooldridge, Agent-Based Software engineering, *IEE Proc. Software Engineering*, Vol. 144, No. 1,

Table 4. Agents' Solutions to One Benchmark Problem

Agent	S_{22}									
	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
A_1	6.94	1.85	3.89	5.38	3.24	1.99	8.21	5.84	5.02	3.85
A_2	6.93	2.04	4.07	4.96	2.97	1.95	8.06	6.0	4.97	3.97
A_3	6.99	1.99	4.0	5.0	3.0	2.0	8.0	5.99	5.0	4.01
A_4	7.12	1.9	4.08	4.87	3.14	2.01	8.07	5.99	5.15	3.87
A_5	7.06	1.93	4.22	5.21	3.31	2.14	8.14	5.76	4.88	3.8
A_6	7.15	1.65	3.65	4.82	3.42	2.01	7.66	6.03	5.16	3.57
A_7	6.7	2.41	4.44	5.14	2.65	2.27	8.42	5.54	4.9	3.8
A_8	6.93	2.03	3.69	5.28	2.99	1.75	8.45	6.46	4.45	4.34
A_9	6.86	1.9	3.73	4.78	3.21	2.15	7.87	6.02	5.17	4.29
A_{10}	6.93	1.62	3.68	4.98	2.88	1.74	8.13	6.39	4.97	4.38

Table 5. Partial Benchmark Values for Benchmark Problems

Problems	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
P_1	3.016	6.027	4.972	4.013	2.958	1.987	3.913	1.926	6.035	2.947
P_2	7.076	5.016	3.972	9.051	4.005	4.009	2.977	2.987	4.042	3.975
P_3	4.012	6.949	2.937	3.021	5.011	4.934	2.072	6.978	3.06 2	2.011
P_4	2.888	2.998	8.035	7.036	8.002	2.001	4.992	3.013	8.069	9.009
P_5	7.983	3.922	7.071	7.013	1.968	7.041	1.995	3.988	7.017	5.000

Table 6. Distances between Agents' Solutions and Benchmark Values

Agents	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}
A_1	0.2148	0.2611	0.2258	0.2229	0.2522	0.1715	0.1772	0.189	0.2533	0.1963
A_2	0.0554	0.0389	0.0622	0.0749	0.0403	0.0654	0.0579	0.0612	0.0509	0.0385
A_3	0.0467	0.0364	0.0486	0.0428	0.0394	0.0435	0.0559	0.0342	0.0341	0.0303
A_4	0.0795	0.1035	0.0904	0.088	0.0813	0.0899	0.072	0.0831	0.1101	0.0627
A_5	0.1604	0.2026	0.1785	0.1585	0.17	0.1788	0.1578	0.1864	0.1554	0.2265

1997, 26-37.

[2] E. H. Durfee and V. Lesser, Negotiating Task Decomposition and Allocation Using Partial Global Planning, in: L. Gasser and M. Huhns (Eds.), *Distributed Artificial Intelligence Volume II*, Pitman Publishing and Morgan Kaufmann, 1989, 229-244.

[3] N. R. Jennings, K. Sycara, and M. Wooldridge, A Roadmap of Agent Research and Development, *Autonomous Agents and Multi-Agent Systems*, Vol. 1, No. 1, 1998, 7-38.

[4] K. Decker, K. Sycara, and M. Williamson, Middle Agents for the Internet, *Proceedings of 15th International Joint Conference on Artificial Intelligence*, Nogoya, Japan, 1997, 578-583.

[5] Z. Zhang and C. Zhang, An improvement to matchmaking algorithm for middle agents, *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems*, ACM Press, Bologna, Italy, July 2002 (forthcoming).

[6] Stephen T. Welstead, *Neural Network and Fuzzy Logic Applications in C/C++*, Wiley, New York, 1994, 395-421.

[7] K. Arisha, F. Ozcan, R. Ross et al., Impact: A Platform for Collaborating Agents, *IEEE Intelligent Systems & Their Applications*, Vol. 14, No. 2, 1999, 64-72.

[8] D. W. Karolak, *Software Engineering Risk Management*, IEEE Computer Society Press, 1996, 43-51.

[9] E. Peter, *Chaos and Order in the Capital Markets*, John Wiley & Sons, Inc., 1991.

[10] F. Hoppner, F. Klawonn, R. Kruse, and T. Runkler, *Fuzzy Cluster Analysis*, John Wiley & Sons, 1999.

[11] F. Hoppner, Fuzzy Clustering Algorithms – A Tool Library, Open Source Project, *http://www.fuzzy-clustering.de.*