Multi-agent Coordination through Coalition Formation *

Onn M. Shehory, Katia Sycara and Somesh Jha
The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213, U.S.A.
onn,katia,sjha@cs.cmu.edu

Abstract

Incorporating coalition formation algorithms into agent systems shall be advantageous due to the consequent increase in the overall quality of task performance. Coalition formation was addressed in game theory, however the game theoretic approach is centralized and computationally intractable. Recent work in DAI has resulted in distributed algorithms with computational tractability. This paper addresses the implementation of distributed coalition formation algorithms within a real-world multi-agent system. We present the problems that arise when attempting to utilize the theoretical coalition formation algorithms for a real-world system, demonstrate how some of their restrictive assumptions can be relaxed, and discuss the resulting benefits. In addition, we analyze the modifications, the complexity and the quality of the cooperation mechanisms. The task domain of our multi-agent system is information gathering, filtering and decision support within the WWW.

1 Introduction

Theories of cooperation among computational intelligent agents have been developed in the last decade, providing methods which enable, theoretically,

*This material is based upon work supported in part by ARPA Grant #F33615-93-1-1330, by ONR Grant #N00014-96-1-1222, and by NSF Grant #IRI-9508191.

low complexity of the cooperation mechanisms as well as high performance of the multi-agent systems. Although they seem promising, most of these mechanisms were not tested in a *real-world* multi-agent environment.

Cooperating groups of agents, referred to as coalitions, were thoroughly investigated within game theory (e.g., in [9]). There, issues of solution stability, fairness and payoff disbursements were discussed and analyzed. The formal analysis provided there can be used to compute multi-agent coalitions, however only in a centralized manner and with exponential complexity. DAI researchers have adopted some of the game-theoretical concepts and upon them developed coalition formation algorithms, to be used by agents within a multi-agent system (e.g., [18, 5]). These algorithms concentrate on distribution of the computations, complexity reduction, efficient task allocation and communication issues. Nevertheless, some of the underlying assumptions of the coalition formation algorithms, which are essential for their implementation, do not hold in real-world multi-agent systems.

In this paper we report on coalition formation as a means for coordinating agents. The coalition formation method we present is appropriate for dozens of agents[1]. We begin with a brief overview of the multi-agent system into which the algorithms are applied in section 2. We continue by presenting the theoretical coalition formation method, in section 3. We then present the implementation requirements (section 4) and relaxation of theoretical assumptions (section 5). In section 6 we analyze the properties and modifications of the implemented method, both theoretically and via simulations. Finally we conclude in section 7.

2 The information multi-agent system

The problem of locating information sources, accessing, filtering, and integrating information, as well as interleaving information retrieval and problem solving has become a very critical task, due to the increasing amount of distributed, dynamically changing information.

Most work in intelligent software agents that gather information from Internet-based sources, e.g., [7, 8, 1] focussed on a single agent with simple

[1]For hundreds of agents, coalition formation methods are usually too complex. However, several cooperation methods were developed for such cases (e.g., market oriented solutions [16]).

knowledge and problem solving capabilities whose main task is information filtering to alleviate the user's cognitive overload. Another type of agent is the *Softbot* ([3]), a single agent with general knowledge that performs a wide range of user-delegated information-finding tasks. A single general agent would need an enormous amount of knowledge to effectively deal with user information requests that cover a variety of tasks. In addition, a centralized system constitutes a processing bottleneck and a "single point of failure". Finally, because of the complexity of the information finding and filtering task, and the large amount of information, the required processing would overwhelm a single agent. To resolve the above problems, a multi-agent system is necessary.

We have developed a multi-agent system named RETSINA (REusable Task-based System of Intelligent Networked Agents) [15, 13, 12] to integrate information gathering from web-based sources and decision support tasks. The agents in RETSINA compartmentalize specialized task knowledge, organize themselves to avoid processing bottlenecks, and can be constructed specifically to deal with dynamic changes in information, tasks, number of agents and their capabilities.

2.1 The system infrastructure

In RETSINA, the agents are distributed and run across different machines. Based on models of users, agents and tasks, the agents decide how to decompose tasks and whether to pass them to others, what information is needed at each decision point, and when to cooperate with other agents. The agents communicate with each other to delegate tasks, request or provide information, find information sources, filter or integrate information, and negotiate to resolve inconsistencies in information and task models. The system consists of three classes of agents (see Figure 1): *interface* agents, *task* agents and *information* agents.

Interface agents interact with users receiving their specifications and delivering results. They acquire, model and utilize user preferences. Task agents formulate plans and carry them out. They have knowledge of the task domain, and which other task agents or information agents are relevant to performing various parts of the task. In addition, task agents have strategies for resolving conflicts and fusing information retrieved by information agents. They decompose plans and cooperate with appropriate task agents or

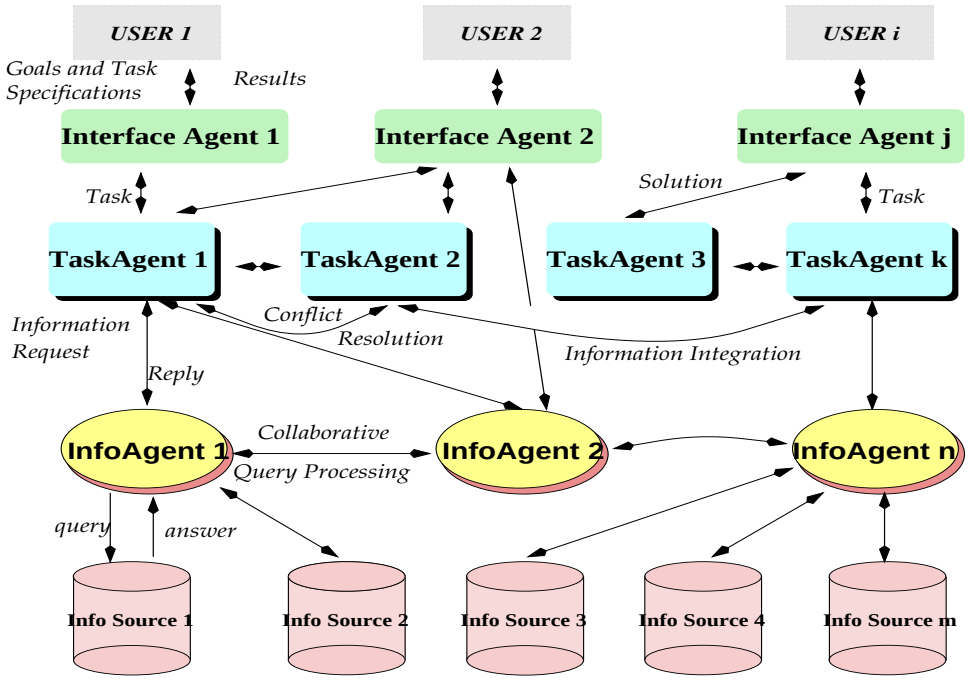

Figure 1: Illustration of the infrastructure of the agent system

information agents for plan execution, monitoring and results composition. Information agents provide intelligent access to a heterogeneous collection of information sources. They have models of the information resources and strategies for source selection, information access, conflict resolution and information fusion. Information agents are active, in the sense that they actively monitor information sources and *proactively* deliver the information.

2.2 Agent matchmaking

One of the basic design problems of cooperative, *open*, multi-agent systems for the Internet is the connection problem [2]. That is, each agent must be able to locate the other agents who might have capabilities which are necessary for the execution of tasks, either locally or via coalition formation. The fact that the system is open (participating agent may dynamically enter and leave) and distributed over the entire Internet precludes broadcast communication solutions.

The solution to this problem in our system relies, instead, on some well-known agents and some basic interactions with them – matchmaking [6, 14]. In general, the process of matchmaking allows an agent with some tasks, the requester, to learn the contact information and capabilities of another agent, the server, who may be able to execute some of the requester's tasks. This process involves three different agent roles:

- Requester: an agent that holds a set of tasks and wants them to be performed (at least partially) by other agents who possess relevant capabilities.
- Matchmaker: an agent that knows the contact information, capabilities, and other service characteristics (e.g. cost, availability, reliability) of other agents.
- Server: an agent that has committed to the execution of a task or at least part of a task delegated to it by a requester.

During the operation of the multi-agent system, agents that join the system advertise themselves and their capabilities to a matchmaker, and when they leave the agent society, they un-advertise (for more details, see [12]). Requesters, in search of agents with which they may possibly form coalitions, approach a matchmaker and ask for names of relevant servers. After having acquired the information about other agents they can directly contact these agent and initiate cooperation as needed. Note that there may be several matchmaker agents to relax the problem of unavailable or overwhelmed single matchmaker.

3 Cooperation via coalition formation

Coalition formation methods among multiple agents (e.g., in [18, 10, 5, 11]) refer to cases in which groups of agents work jointly in order to accomplish their tasks. The RETSINA system can receive several tasks from several users. Our hypothesis is that incorporation of a coalition formation mechanism shall increases the efficiency of groupwise task execution, resulting in near-optimal task performance. We report such results in section 6. In addition, this mechanism will enable agents to decide upon the importance (and thus – the order) of tasks to be performed. Such decision making is important in real-world domains, where there may be situations in which a system cannot fulfill all of its tasks.

We provide below a brief description of the coalition formation model presented in [11], which we later modify to enable its implementation and take into consideration additional requirements of an open and dynamic agent environment. There is a set of n agents, $\{A_1, A_2, \ldots, A_n\}$. Each agent A_i has a vector of real non-negative capabilities $B_i = \langle b_1^i, \ldots, b_r^i \rangle$. Each capability[2] is a property of an agent that quantifies its ability to perform a specific type of action[3]. In order to enable the assessment of coalitions and task-execution, an evaluation function shall be attached to each type of capability. There is a set of $|T|$ independent tasks $T = \{t_1, t_2, \ldots, t_m\}$. For the satisfaction of each task t_j, a vector of capabilities $B_j = \langle b_1^j, \ldots, b_r^j \rangle$ is necessary. The benefits gained from performing the task depend on the capabilities that are required for its performance. Benefits are measured from the whole system viewpoint.

A coalition is defined as a group of agents who have decided to cooperate in order to perform a common task. The model assumes that a coalition can work on a single task at a time, and that agents may be members of more than one coalition. A coalition C has a vector of capabilities B_c which is the sum of the capabilities that the coalition members contribute to this specific coalition. C has a value V which is the joint benefits of the members of C when cooperatively satisfying a specific task.

The model assumes that the agents are group-rational. That is, they join a coalition only if they (jointly) benefit as a coalition at least as much as the sum of their personal benefits outside of it [4, 9]. Group rationality is necessary to ensure that whenever agents form a coalition, they always increase the system's global benefits, which is the sum of the coalitional values. It is also assumed that the agent-population does not change during the coalition formation; all of the agents must know about all of the tasks and the other agents[4]; the details of intra-coalitional activity are not necessary for agents outside of the coalition; there is no clock synchronization among the agents. The coalition formation algorithm consists of two main stages[5]:

1. A preliminary stage – all possible coalitions are distributively calculated.

[2]In the context of information agents, an example of a capability is the type and the amount of information that an agent can provide.

[3]Action is the most fundamental task, virtually un-divisible.

[4]Since RETSINA operates in an open, dynamic environment, it does not satisfy these two assumptions (see section 5).

[5]For additional details with respect to the algorithm see [11].

This distribution is achieved by having each agent A_i compute only coalitions in which it is a member (put these in a list L_i).

2. A main stage – an iterative greedy procedure in which two sub-stages occur:

   - The coalitional values are calculated[6] such that for each task, all of the coalitions that can satisfy it are considered. The distribution of these calculations is done by having each agent A_i approach the agents which are members of the coalitions in L_i and commit to the calculation of the values of coalitions in which they are both members. Consequently, each coalition value calculation will be committed to only once (this distribution depends on the communication order).
   - The agents decide upon the preferred coalition (according to its maximal calculated value with respect to a specific task) and form it, and perform the respective task.

Since the number of the possible coalitions is exponential (2^n), such is the complexity of the algorithm. We reduce it by limiting the permitted coalitions. Such heuristics were implemented in the algorithm by using an integer k which denotes the highest coalitional size allowed. This restriction limits the number of coalitions to $O(n^k)$ (polynomial in n). We later explain why such a limitation is justified.

The algorithm has a ratio bound $\rho = \frac{c_{tot}}{c^*_{tot}} \leq \sum_{i=1}^{max(|C_j|)} \frac{1}{i}$ where c_{tot} denotes the total cost[7] of all coalitions derived by the algorithm, c^*_{tot} denotes the optimal total cost, and $max(|C_j|)$ is the maximal coalition size. The ratio bound ρ is the worst case bound, and the average case is significantly better (shown in section 6). The two processes of calculating coalitional values and choosing coalitions may be repeated up to $|T|$ times. Therefore, the worst case complexity per agent is $O(n^{k-1} \cdot |T|)$ computations and $O(n \cdot |T|)$ communication operations.

[6] Note that the value calculation must be repeated on every iteration, since the execution of tasks may change these values.

[7] Note that the notion of cost (and not value) is used here, however the translation of the first to the latter is rather simple. In our system costs, which are part of an agent's advertisement to a match maker, stem from the computation and communication efforts associated with the agent activity.

4 The cooperation component

The architecture of each agent in the RETSINA framework includes a generic cooperation component. We shall elaborate on the architecture, the functionality and the advantages of this component. The role of the cooperation component is to enable close cooperation among agents. An agent should consider cooperation if one of the following holds:

- The agent cannot perform a specific task by itself.
- The agent can perform a specific task, but other agents are more efficient in performing this task (e.g., they require less resources or perform faster).
- The agent can perform a specific task, but working on it collaboratively will increase the benefits from the task (or reduce the costs).

The last two conditions are not necessarily easy for an agent to perceive, especially in cases of incomplete information with regard to the capabilities of other agents and the expected benefits from task execution by them. Nevertheless, reasoning about the global utility of cooperation and the application of cooperation strategies strongly relies on such expected benefits. Measurable expected benefits can be compared to decide upon the preferable ones and the cooperation activities that may achieve them. Benefits are commonly assessed and expressed by utility functions[8]. We are interested only in the payoff gained by the whole agent-system as the result of agent activity, and not in the individual payoff of an agent. While the other parts of the cooperation component of the agent are reusable[9], the utility functions must be determined and implemented specifically for each task domain.

Cooperation strategies mainly depend on two parameters: the environment type with respect to payoffs (super-additive vs. non-super-additive) and the agent rationality (self-rationality vs. group-rationality). Each of the four combination of these requires different cooperation strategies to increase the payoffs, either of single agents in the self-rationality case or of the whole system in the group-rationality case. However, since they do not depend on the specific task domain, cooperation strategies can be formulated in a generic manner, and instantiated by the agents for each specific task domain.

[8] Utility functions are frequently referred to as cost functions or payoff functions.

[9] That is, they can be used for various domains with no modifications.

5 Coalition re-design

In RETSINA we implement coalition formation mechanisms for group-rationality cases. We rely on the theoretical methods presented in [11] as a basis for the algorithms implemented, however modify them due to fundamental differences between the agent-systems discussed there (see section 3) and those discussed here:

- The number of goals and agents in [11] is fixed, while RETSINA is a dynamic system where agents appear and disappear and tasks vary constantly.
- The size of coalitions in [11] is bounded by a pre-defined constant k, independent of the n and $|T|$. k has a significant effect on the complexity of the solution. Such an artificial constraint may prohibit solutions even in cases where these not only exist but are also feasible and beneficial.
- The algorithm in [11] does not discuss the effect of two cases which are typical in our system: how to choose from among two agents (or more) that can provide the same service with the same expected payoff; how to deal with the case of reusable or non-depleting capabilities.
- Tasks with complex time dependencies, such as partial overlapping use of a resource, which are typical in our dynamic system, are not referred to in [11].
- The method in which the information with regards to the existence of tasks and their details is distributed in not discussed in the original algorithm.

To resolve the above restrictions, we made various modifications to the algorithm.

Since the communication- and computation-time for value calculation and coalition design (section 3) are significantly small as compared to the task execution time, and tasks can dynamically appear, the modified algorithm includes a re-design process. When a new task is received by the system, we require:

- When an agent receives a new task, it finds through matchmaking relevant agents that can execute the task.
- If tasks that were assigned to coalitions have not been performed yet within the current iteration of the coalition formation algorithm, the agents will re-calculate the coalitional values to take into consideration the arrival of the new task.

  - If inclusion of the newly arrived task in coalition recalculations in the current iteration raises the value of a coalition, then the agents shall re-design coalitions, selecting again the best among the actual, re-designed, coalitions[10].
  - Otherwise, the agents shall avoid coalition re-design, and consider the new task for inclusion in coalitions at the next iteration.

- If all previous tasks are in process, the new task will be added to the group of tasks T and be dealt with in the next coalition formation iteration.

- In case of a rapid high-frequency stream of new tasks, the re-design process may be dis-enabled. If such a rate of new tasks is expected in advance, or the agents statistically infer such a rate by sampling the task stream and interpolating the statistical data, the re-design process shall be avoided.

The dynamic addition of tasks to the agent system does not change the overall order of complexity of the algorithm, it however adds a factor to it. This is since the complexity is linear in $|T|$, and the maximal number of re-design processes is bounded by the number of the dynamically received tasks $|T_d|$, and $|T_d| < |T|$. Hence, the worst case complexity will be less than twice the non-dynamic complexity.

The communication requirements of the original algorithm are in the worst case $O(n)$ per agent per task (however the average is $O(1)$). In the new algorithm, however, there is an additional complexity due to the dynamic task advertisement. Hence, while the worst case remains unchanged, the average case becomes $O(|T_d| \cdot k)$. Yet this is a low linear complexity. Nevertheless, in WWW information gathering (in which our agent system operates), the high network latency causes the computation time for coalition formation to be dwarfed by comparison. This was observed in the course of our experiments.

[10] Note re-design may not be allowed if the expected task flow is rapid, lest the system will constantly re-design and not perform its tasks.

6 Algorithm modification and analysis

6.1 Computational complexity

In section 3, the number of agents n was assumed to be constant. However, in our agent system, n may dynamically change. Given this difference, the analysis of the complexity must be modified. We introduce[11] $N = max(n)$. Using N, the complexity can be expressed by a similar expression as in section 3, where n is substituted by N, resulting in $O(N^{k-1} \cdot |T|)$. Since $N = \text{const} \cdot n$, the complexity will remain of the same order. The k limitation, that enable polynomial complexity is disturbing. The limitation it represents with respect to the size of coalitions must either be justified and adjusted to our system or omitted. We show that some restrictions can be applied in our system, without reduction in its functionality, as described below.

An important property of a RETSINA agent is its ability to perform task reductions [17]. In practice, the internal complexity of a sub-task is determined within the plan library. The plan library is domain-specific, hence the designers of the domain-specific components have control over the complexity of sub-tasks. In the information domain of RETSINA sub-tasks, each sub-task can typically be performed by a small number of agents. This implies that the coalition formation procedure will concentrate on the formation of small coalitions of agents with particular expertise to perform a task.

For example, one of the domains in which RETSINA was implemented is satellite tracking. One of the tasks that the agents can cooperatively perform is finding if and when a specific satellite will be observable in a specific location. For this, up to 4 information agents are involved in the information gathering, and up to 4 other agents are involved in other related tasks. This means that the maximal coalition size for this task type is 8. Since other tasks of the system are of same order of complexity, the sizes of coalitions are limited as well. The system may include other active agents, however these will be involved in other tasks or be idle.

Each agent in our system is specialized in a specific type of task-performance. We do not incorporate complex, multi-purpose agents, since most of the capabilities of such agents may remain unused most of the time, while their size

[11] Since $max(n)$ may be unknown, N shall be decided upon according to the expectations of the designers with regards to their agent system.

and complexity consume computational resources, reducing the system's performance. This specialization results in the incorporation of agents into coalitions according to their specialty/capability (necessarily, when more than one agent with the same specialty are present, their utility functions enables comparison which results in the choice of the one with the highest payoff). Thus, a coalition size is limited to the number of different specialties which are necessary for the execution of the task that this coalition performs. The number of specialties which are necessary for a given task execution is small and hence such is also the size of coalitions. Denoting the maximal number of specialties necessary for sub-task execution by k, we obtain the required restriction on the size of coalitions[12]. However, since different decompositions of tasks along different specialty dimensions are possible, a specific agent system may have several k's. Among them, the maximal will determine the worst case complexity. There is a trade-off between the complexity of sub-tasks and the complexity of task reduction: a more complex task reduction will result in simpler sub-tasks thus reducing their complexity, and vice versa.

6.2 Quality analysis

Recall the theoretical ratio bound $\rho = \frac{c_{tot}}{c^*_{tot}} \leq \sum_{i=1}^{max(|C_j|)} \frac{1}{i}$ in [11]. This is a logarithmically increasing expression with respect to $max(|C_j|)$, i.e., $\rho \sim \log k$. A logarithmically increasing cost (c_{tot}) entails a logarithmically decreasing value. This means that according to the ratio bound analysis the overall payoff from task execution may be less than half of the optimal payoff of the system. This is far from being satisfactory. We have shown via simulations that the average case is close to the optimal case (see figure 2)[13].

The figure shows that the average performance (in terms of task allocation and execution) reached via simulation, depicted by the solid line, is around 0.9 of the optimal performance[14], while the worst case (the ratio bound), depicted by the broken line, declines fast to less than 0.5 of the optimal performance. Since the ratio bound depends on k, and in our system k does

[12]The k restriction can be further relaxed e.g., by designing task decomposition that avoids $k \sim n/2$.

[13]In order to get statistically significant results, we performed controlled experiments through simulation.

[14]We calculated the optimal performance explicitly, off line, when a small number of agents are involved in coalition formation. Small here means up to 20.

not depend on n or N, the logarithmic expression holds, and its magnitude can be determined by the designers. Thus the worst case performance can be traded-off with the computational complexity of task reduction. For instance, by simplifying the task reduction process, the reduced tasks remain rather complicated and will probably require more agents to perform each (since each involves more capabilities). Coalitions will therefore have to be larger. Hence, while the computational complexity for task reduction was reduced, the coalition formation complexity increased. The analysis of this trade-off may allow to further improve the performance of the system.

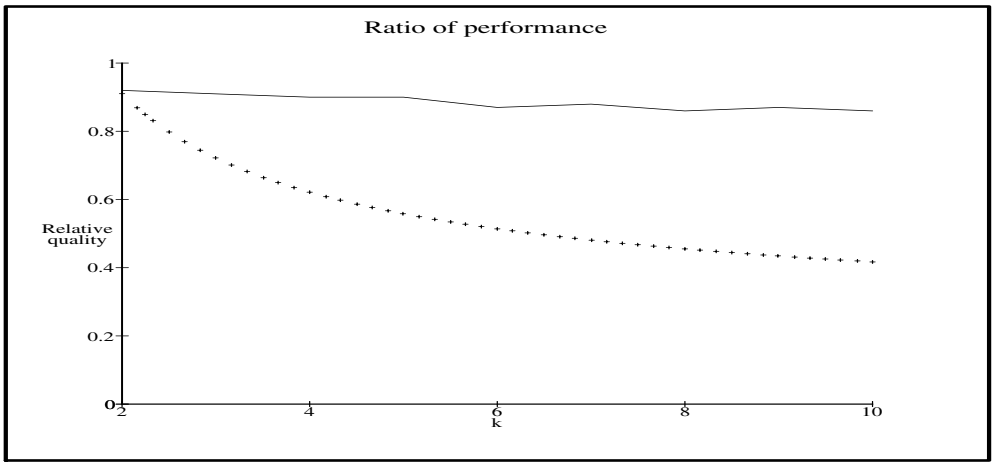

Figure 2: The average performance with respect to the worst case

The simulations performed for checking the performance of the implemented algorithm were done as follows. A dynamic set of agents which included up to 20 agents ($N = 20$), where each provided with a vector of capabilities $B_j = \langle b_1^j, \ldots, b_r^j \rangle$. The agents received an initial set of tasks T, and additional tasks were provided dynamically in a random manner (i.e., the frequency of tasks, their type and the required capabilities were randomly chosen). Each task was associated with a vector of capabilities necessary for its execution[15] and a payoff function for calculating the value of the task. During the simulation, coalitions of agents were formed, where a task was

[15]Note that in the simulation we avoided the planning phase of task reduction. This simplification does not affect the properties of the coalition formation mechanism.

allocated to each, and the value of its execution by this coalition was calculated. The sum of these values was calculated to find the total payoff. When new tasks arrived, the re-design procedure was followed. We have performed this simulation several hundreds of times, and compared the total payoffs to the optimal payoffs (calculated off-line) and to the theoretical ratio bound, as depicted above.

7 Discussion

In this research we have utilized coalition formation methods to improve multi-agent coordination (in terms of the joint payoff) of a real-world, information multi-agent system. The coalition formation algorithm takes into consideration requirements and constraints arising from the dynamic nature of the environment in which the system operates. We have shown through simulation that the obtained efficiency and quality of the allocation of groups of agents for task execution is close to the optimal. In addition, the incorporation of coalition formation algorithms into an open, multi-agent system creates a decision mechanism for cases in which a subset of the tasks cannot be performed, allowing the choice of the more beneficial ones for execution. To enable the implementation of the coalition formation methods within a working, real-world agent system, we had to relax several binding assumptions and limitations which are common in the theoretical coalition formation theory, and provide solution to problems arising from these relaxations and from the dynamics and uncertainty to which our system is subject. We have analyzed the complexity and the quality, and shown that the incorporation of the coalition formation method induces a near-optimal task allocation while not significantly increasing the execution time. The algorithm implementation described in the paper is most appropriate for group-rationality cases. We currently work on the implementation of coalition formation methods for self-rationality cases.

References

[1] Robert Armstrong, Dayne Freitag, Thorsten Joachims, and Tom Mitchell. Webwatcher: A learning apprentice for the world wide web. In *Proceedings*

of AAAI Spring Symposium on Information Gathering from Heterogenous Distributed Environments, 1995.

[2] R. Davis and R. G. Smith. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20(1):63–109, January 1983.

[3] Oren Etzioni and Daniel Weld. A softbot-based interface to the internet. *Communications of the ACM*, 37(7), July 1994.

[4] J. C. Harsanyi. *Rational Behavior and Bargaining Equilibrium in Games and Social Situations*. Cambridge University Press, 1977.

[5] S. P. Ketchpel. Forming coalitions in the face of uncertain rewards. In *Proc. of AAAI94*, pages 414–419, Seattle, Washington, 1994.

[6] D. Kuokka and L. Harada. On using KQML for matchmaking. In *Proceedings of the First International Conference on Multi-Agent Systems*, pages 239–245, San Francisco, June 1995. AAAI Press.

[7] K. Lang. Learning to filter netnews. In *Proceedings of the Machine Learning Conference 1995*, 1995.

[8] P. Maes. Agents that reduce work and information overload. *Communications of the ACM*, 37(7):31–40, 1994.

[9] A. Rapoport. *N-Person Game Theory*. University of Michigan, 1970.

[10] T. W. Sandholm and V. R. Lesser. Coalition formation among bounded rational agents. In *Proc. of IJCAI-95*, pages 662–669, Montrèal, 1995.

[11] O. Shehory and S. Kraus. Formation of overlapping coalitions for precedence-ordered task-execution among autonomous agents. In *Proc. of ICMAS-96*, pages 330–337, Kyoto, Japan, 1996.

[12] K. Sycara, K. Decker, A. Pannu, and M. Williamson. Designing behaviors for information agents. In *Proceeding of Agents-97*, pages 404–412, Los Angeles, 1997.

[13] K. Sycara, K. Decker, A. Pannu, M. Williamson, and D. Zeng. Distributed intelligent agents. *IEEE Expert – Inteligent Systems and Their Applications*, 11(6):36–45, 1996.

[14] K. Sycara, K. Decker, and M. Williamson. Middle-agents for the internet. In *Proceeding of IJCAI-97*, Nagoya, Japan, 1997. To appear.

[15] K. Sycara and D. Zeng. Coordination of multiple intelligent software agents. *International Journal of Intelligent and Cooperative Information Systems*, 1996.

[16] M. P. Wellman. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research*, 1:1–23, 1993.

[17] M. Williamson, K. Decker, and K. Sycara. Unified information and control flow in hierarchical task networks. In *Proceedings of the AAAI-96 workshop on Theories of Planning, Action, and Control*, 1996.

[18] G. Zlotkin and J. S. Rosenschein. Coalition, cryptography, and stability: Mechanisms for coalition formation in task oriented domains. In *Proc. of AAAI94*, pages 432–437, Seattle, Washington, 1994.