2002

Negotiation-Based Coalition Formation Model for Agents with Incomplete Information and Time Constraints

Leen-Kiat Soh
University of Nebraska, lsoh2@unl.edu

University of Nebraska-Lincoln, Computer Science and Engineering
Technical Report # TR-UNL-CSE-2002-0002

A Negotiation-Based Coalition Formation Model for Agents with Incomplete Information and Time Constraints

Leen-Kiat Soh
Department of Computer Science and Engineering
University of Nebraska, Lincoln, NE 68588
Tel: (402) 472-6738 Fax: (402) 472-7764 E-mail: lksoh@cse.unl.edu

Abstract

In this paper we describe a coalition formation model for a cooperative multiagent system in which each agent has incomplete information about its dynamic and uncertain world and must respond to sensed events within time constraints. With incomplete information and uncertain world parameters while lacking time, an agent cannot afford organizing a rationally optimal coalition formation. Instead, our agents use a two-stage methodology. When an agent detects an event in the world, it first compiles a list of coalition candidates that it thinks would be useful, and then negotiates with the candidates. A negotiation is an exchange of information and knowledge for constraint satisfaction until both parties agree on a deal or one opts out. Each successful negotiation adds a new member to the agent's final coalition. The agent that initiates the coalition needs to determine the task distribution among the members of the coalition and designs its coalition strategy to increase the chance of successfully forming a working coalition. Since the environment is dynamic, noisy, and the agents are resource-constrained, agents must form the working coalition to react to events as soon as possible and with whatever partial information they currently hold.

1. Introduction

In this paper we describe a coalition formation model for agents with incomplete information and time constraints within a dynamic and uncertain world. A coalition is a group of agents that collaborate to perform a coordinated set of tasks, that may be a response to an event that has occurred in the environment. A dynamic coalition is one that is formed as a response to an event and dissolved when the event no longer exists or when the response is completed. A coalition is necessary when an agent cannot respond to an event all by itself due to lack of information, knowledge, or functional capabilities. Ideally, the agent would prefer to form an optimal coalition to maximize the yield of the system as a whole. However, such optimal rationalization requires the agent to have complete information about its world and its neighboring agents, and also about the uncertainty associated with all factors related to the multiagent infrastructure. When that information is not readily available and the collection of that information is too costly, an agent cannot afford such optimality. In the following, we elaborate on some of the problem characteristics.

First, our model applies to an environment where each agent has incomplete information about its world. Incomplete information may be due to (1) polling and updating costs, (2) constrained resources, and (3) decentralized information base. In a time-critical domain, agents may not afford to poll for information or update the changes in their perceived environments constantly. Moreover, in cases where resources are constrained such as the size of a centralized blackboard, or shared communication channels, agents can only exchange information when necessary. Further, in a domain where each agent senses its local world and maintains a local

information base at remote sites, it is prohibitive for every agent to share all local information bases as some information may not be needed and keeping the information concurrent may impose constraints on local agents unnecessarily. As a result, when an agent needs to rationalize based on its profile of other agents to form a successful coalition, it can only do so based on partial or outdated information. In a coalition formation process, that means the coalition-initiating agent may know which other agents can be useful but can only guess at their willingness to help. Our goal is to provide a model that increases the chance of a successful coalition formation.

Second, an optimal rationalization for coalition formation may not be possible due to (1) noise and uncertainty in the environment, and (2) time constraints. For example, the communication channels among the agents may be congested or faulty, messages may be noisy or lost, perceived events may be qualified inaccurately, and so on. These uncertainties as a whole render rationally optimal (but otherwise arduous) planning less cost efficient compared to one that is more reactive, since the longer the initiating agent takes to respond to an event, the more likely it is going to change in the dynamic environment. Thus, an initiating agent has to compromise on the optimality of its coalition formation strategy before a change in the world pre-empts its effort. Moreover, the domain environment that we want to address is highly time-critical. A coalition formation process has limited time to complete depending on the time available to respond to an event.

Third, in our problem domain, we assume all agents are peers—there is no hierarchy among the agents. Each agent is able to sense its environment, revise its own perceptions, and form its own coalitions. This allows the agents to be reactive to environmental changes, without having the directives passed from a higher-up agent while encouraging diversity in information stored at each agent. We also assume agents are cooperative as opposed to self-interested. Each agent is implicitly motivated to cooperate for the common good of the system while managing the usage of its allocated resources. This allows our model to relax the computation of cost-benefit ratio requirements, which would have to be adhered to in an optimal rationalization.

Fourth, we propose using negotiations to refine a coalition. We see negotiation as an exchange of necessary information pertinent to individual constraints, perceptions, and commitments. This exchange of information is performed only when the coalition-initiating agent approaches potential coalition partners to request for help. Thus, the exchange is efficient. It also allows both parties of a negotiation to update their viewpoints of each other, and that results in better-informed decision making in the future. The motivation of a negotiation is for the initiating agent to persuade a potential coalition partner to agree to help. Our negotiation-based coalition formation also implies that even though the coalition-initiating agent is in charge of the entire formation process, the responsibility of making the decision to join lies on the shoulders of the potential coalition partners. All the coalition-initiating agent can do is to prepare an initial coalition—based on whatever the information that it currently has—that it thinks has a high chance of success and proceed from there. This negotiation allows the initial coalition to be less than optimal and to be computed *hastily*.

Fifth, unlike traditional coalition formation work that assume potential coalition members are readily willing to help, our model expects coalition members to refuse to join in a coalition, especially in a resource-constrained environment and also plans for failed communication due to congestion, noise, or message loss. Thus, the initial coalition may not survive after negotiations as the working coalition is finalized. Our model is also designed to withstand noise and uncertainty by incorporating *insurance policies* and algorithms that are *greedy* or *worried*.

Briefly, our proposed model works as follows. When an event is detected in a multiagent system, one of the agents initiates the coalition formation process in hope of organizing a group of cooperative agents to perform tasks in response to the event. This initiating agent (also known as the "computing agent" (Sandholm and Lesser 1995)) shoulders the responsibility of designing the best coalition given the situated information to increase the chance of forming a working and useful coalition at the end of the process. The model consists of two stages. First, during the *coalition initialization*, the initiating agent extracts a ranked list of useful agents. Then, the initiating agent approaches the potential coalition partners and requests for negotiations during a *coalition finalization* step. Our negotiation is based on a case-based reflective argumentative model (Soh and Tsatsoulis 2001). Finally, the agent re-designs its coalition if it fails to satisfy its response to the triggering event and if time permits. This three-step model allows an agent to form an initial coalition quickly to react to an event and to rationalize to arrive at a working final coalition as time progresses.

In the following, we first discuss the agent characteristics of multiagent system that our model assumes. Then, we present our dynamic negotiation-based coalition formation model in Section 3. Subsequently, we describe our current work using the model in a multiagent sensor tracking problem domain and discuss some experimental results. In Section 4 we describe some related work in coalition formation. Finally, we conclude.

2. Agent Characteristics

Readers are referred to (Wooldridge and Jennings 1995) for a detailed discussion on various agent characteristics. In general, our model assumes that agents have the following characteristics:

(1) Autonomous – Each agent runs without interactions with human users.
(2) Rational – Each agent is rational in that it knows what its goals are and can reason and choose from a set of options and make an advantageous decision to achieve its goal. In the resource allocation domain, each agent is selfish as each attempts to preserve its control of CPU resources and conserve its power usage, for example. Yet, each agent is bounded by a global objective that demands each agent *cooperate* to share resources. The two balancing goals drive the rationality of our agents.
(3) Communicative
(4) Reflective – According to Brazier and Treur (1996) in their DESIRE framework, a reflective agent reasons based on its own observations, its own information state and assumptions, its communication with another agent and another agent's reasoning, and its own control or reasoning and actions.
(5) Honest – Each agent does not knowingly lie or intentionally give false information. This characteristic is also known as veracity (Galliers 1988).
(6) Cooperative -- Each agent is motivated by a set of global directives to help each other when possible to achieve global goals. Each agent is honest and sincere during a negotiation—does not intentionally provide faulty information just to convince another to perform a task or give up a certain resource. The initiating agent trusts the responding agents that they must have their reasons for not agreeing to a deal during a negotiation, and also trusts the responding agents to commit to an agreement as best as possible. There are in general three reasons why agents cooperate (Shehory *et al.* 1997). First, an agent cannot perform a specific task by itself. Second, an agent can perform a specific task, but other agents are

more efficient in performing the task. Third, an agent can perform a specific task, but working on it collaboratively will increase the benefits from the task (or reduce the costs).

We also assume that each agent exists in a neighborhood (Section 3.1) where it knows some basic properties of its neighbors (such as functional capabilities) and can communicate to them directly. Each agent has one neighborhood, and each can exist in multiple neighborhoods of others. It is from this neighborhood that an agent forms a coalition.

3. Dynamic Negotiation-Based Coalition Formation Model

In this section, we present our dynamic negotiation-based coalition formation model. As discussed in Section 1, our agents operate in a dynamic, real-time and uncertain world. When an agent detects an event, it selects from its neighborhood a subset of neighbors as the initial coalition candidates. It determines the task allocation among the candidates and how it should approach these candidates via negotiations. Thus, it shoulders the initial computation cost for the coalition formation. Subsequently, the agents negotiate to exchange information on their respective constraints, commitments, and perceptions. This allows the coalition to be refined as time progresses. Eventually, all negotiations complete with either a success or a failure. The initiating agent and the candidates that agree to help become members of the final coalition.

In our domain, due to the uncertainty in message loss and noise and the dynamic nature of the system, not all coalition candidates become the eventual coalition members. Thus, an initiating agent must design its coalition formation strategy with that in mind. Our coalition formation model consists of two stages: coalition initialization (Section 3.3) and coalition finalization via negotiations (Section 3.4). An interruptible mechanism can be installed in the initialization stage; for example, to stop coming up with feasible coalitions when the time allocated to the step runs out. The coalition finalization stage is interruptible and adaptive as an agent can elect to stop negotiations when it sees fit.

3.1. Neighborhood

As previously discussed in Section 2, each agent, a_i, has a neighborhood, η_{a_i}. It knows some intrinsic information about all neighbors, $\eta_{k,a_i} \in \eta_{a_i}$, in this neighborhood such as a neighbor's physical location, its functional capabilities, and so on. The functional capabilities of an agent a_i are denoted as f_{a_i}. An agent can belong to different neighborhoods concurrently; however, it does not necessary have knowledge about those neighborhoods except its own. An agent can communicate directly with all its neighbors, and each neighbor can communicate with the agent directly as well. However, those neighbors may not be able to communicate with each other directly because they are not necessarily neighbors of each other. Suppose we denote the ability to communicate directly by an agent, a_i, with another, a_j, as $Comm(a_i, a_j)$. Then in a neighborhood of a_i, $Comm(a_i, a_j)$ and $Comm(a_j, a_i)$ are true for all $a_j \in \eta_{a_i}$.

3.2. Events

When an agent senses an event, it measures and collects its properties to perceive it. It is this perception that quantifies the event to facilitate the subsequent coalition design. Suppose an

event is denoted as e_i. It has a time stamp when it was detected, $t_{detected,e_i}$, a time stamp when it is no longer valid, t_{end,e_i}, and a categorical type of the event, $type_{e_i}$. The agent has knowledge about events of the type $type_{e_i}$, denoted as $\mathrm{K}(type_{e_i} = \tau)$. It contains three basic items: $\delta_{expected,\tau}$ for the expected duration during which the event will be valid, Θ_τ for the set of tasks devised as the standard response to the event, and Ω_τ for the coalition formation strategy.

Θ_τ determines the sequence of tasks that an agent performs in reaction to an event. For example, suppose an agent senses that it is currently using close to the level of CPU resource that it has been allocated with, then it declares an event, $\tau = \mathrm{CPU\_SHORTAGE}$. $\Theta_{\tau=\mathrm{CPU\_SHORTAGE}}$ may specify the following:

(1) re-organize the CPU distribution within all the processing threads of the agent,
(2) check to see whether the CPU shortage still persists
  (2.1) if yes, then go to step (3);
  (2.2) otherwise, exit from Θ_τ,
(3) compute the desired additional CPU allocation,
(4) request from the operating system for additional CPU allocation,
  (4.1) if granted, then exit from Θ_τ;
  (4.2) otherwise, go to step (5),
(5) obtain from the operating system all the agents (including their respective CPU allocations) sharing the same CPU resource on the same platform,
(6) perform dynamic, negotiation-based coalition formation using Ω_τ,
(7) carry out all deals resulted from successful negotiations,
(8) check to see whether the CPU shortage still persists
  (8.1) if yes, then go back to step (1);
  (8.2) otherwise, exit successfully from Θ_τ.

Equipped with this knowledge, an agent can react to events consistently and plan out its tasks.

Ω_τ specifies the coalition formation strategy which we discuss in detail in the ensuing sections.

3.3. Coalition Initialization

The first stage of the dynamic, negotiation-based coalition formation algorithm is the determination of the set of the initial coalition candidates, denoted as $\Lambda_{ini}(a_i, e_j)$ for agent a_i and event e_j. This notation allows an agent to have concurrent multiple coalitions, one for every event that it is currently handling. We denote a candidate as α_k. In this section, we first discuss different approaches to coalition initialization. Then, we discuss the ranking of coalition members, and even coalitions. Subsequently, we present some task allocation algorithms.

3.3.1. Approaches to Coalition Formation

In our model, we identify four general scenarios of coalition formation: resource-driven, task-driven, function-driven, and utility-driven.

In a resource-driven coalition formation process, the underlying objective is resource allocation among the agents. For an initiating agent to generate $\Lambda_{ini}(a_i, e_j)$, it needs to know the list of agents currently using the same resource. Let us denote this list as $\Psi_{r,a_i}(t)$ for agent a_i, resource *r*, and at time *t*. If the event e_j calls for the re-allocation of a resource *r*, then $\Lambda_{ini}(a_i, e_j) = \Psi_{r,a_i}(t) \cap \eta_{a_i}$.

In a task-driven approach, the agents are homogeneous in their functional capabilities and only differ in what they know based on their experience, perception, and database. For example, in a multiagent database query system, some information sharing or exchange needs to take place to satisfy a query. The agents in this example are homogeneous in their functional capabilities but differ in what information they can maintain and provide. As a result, an initiating agent needs to obtain a $\Psi_{q,a_i}(t)$, for example, that contains the candidates with the necessary information for responding to the query *q*. As such, the flexibility in the coalition design is reduced compared to the resource-driven approach discussed above. IThus, the initiating agent obtains $\Psi_{f,a_i}(t)$ in which all members in the list have the same functional capability *f* that is needed to respond to event e_j. We also denote all functional capabilities that an agent a_i knows how to perform as F_{a_i}. This information may be stored in each agent during start up, or be furnished on request, or be registered with a centralized service, or advertised at a common blackboard. The initiating agent then obtains $\Lambda_{ini}(a_i, e_j) = \Psi_{f,a_i}(t) \cap \eta_{a_i}$. Note that the set of functions needed to tackle an event is documented in $\mathrm{K}(type_{e_i} = \tau)$. We also denote such a set of functions for an event type τ as F_τ.

In a function-driven approach, the agents are heterogeneous in their functional capabilities—each knows how to perform a different set of functions. As a result, the coalition design is even less flexible compared to the task-driven approach discussed above. For example, suppose there is a multiagent system that maintains a large database with three different agents: an interface agent, a search agent, and a reporting agent. When a query comes in through the interface agent, a coalition is formed by the interface agent that calls upon the search agent to retrieve the queried results from the database and the reporting agent to output in a format specified by the user. Since each agent is specialized to do a particular set of functions, the interface agent must pick the other two agents as the coalition members. In this restrictive manner, the coalition formation is rather straightforward. Once again, we denote a set of functions required to respond to an event type τ as F_τ, and in this function-based approach, $|F_\tau| > 1$. An initiating agent obtains the list of useful agents as $\Psi_{F_\tau,a_i}(t)$ and $\Lambda_{ini}(a_i, e_j) = \Psi_{F_\tau,a_i}(t) \cap \eta_{a_i}$.

When there is flexibility in the coalition design, one can have a utility-driven approach to coalition initialization. For example, in the resource-driven or the task-driven approaches, the initiating agent may want to narrow down $\Lambda_{ini}(a_i, e_j)$ to include only what it considers the most helpful agents. In a strict function-based approach, there is not much flexibility to facilitate

utility-based optimization. However, in a system where each agent is capable of some overlapping set of functions, one can then improve the coalition initialization using utility. Suppose an event type τ with F_τ is detected, and thus the initiating agent knows that the eventual coalition must perform $F_\tau = \{f_1, f_2, \cdots, f_N\}$ to respond to the event. It also knows that all agents in its neighborhood know how to perform all the functions but does not know whether they can do it as requested at the current time and situation. As a result, for each function, it forms a sub-coalition such that $\Lambda_{ini}(a_i, e_j) = \{\Lambda_{ini}(a_i, f_1), \Lambda_{ini}(a_i, f_2), \cdots, \Lambda_{ini}(a_i, f_N)\}$. Each sub-coalition can then adopt a task-driven approach and enjoys greater flexibility in its design. We will discuss general inter-coalition issues in Section 3.6.

3.3.2. Evaluation of Coalitions and Coalition Members

In our dynamic, negotiation-based coalition formation model, the initiating agent a_i first generates the initial coalition candidates, $\Lambda_{ini}(a_i, e_j)$, to deal with an event e_j. $\Lambda_{ini}(a_i, e_j)$ represents the neighbors that it thinks can be of help to respond to e_j. To find out whether these candidates are *willing* to help, the initiating agent needs to negotiate. Negotiation is a process of exchange of information on individual commitments, constraints, and perceptions and may be lengthy and time-consuming. Hence, the initiating agent must think twice about whom to approach first. This motivates the agent to evaluate its coalition members. The objective is to rank the candidates on their potential utility values to the coalition so that the initiating agent can negotiate with the agents with the highest utility values first.

For a candidate $\alpha_k \in \Lambda_{ini}(a_i, e_j)$, we base its potential utility, PU_{α_k, a_i}, on three sets of attributes: (1) the past relationship between the initiating agent and the candidate, $rel_{past, a_i}(\alpha_k, t)$, where *t* is the point in time when the set of attribute-value pairs in the relationship is collected, (2) the current relationship between the initiating agent and the candidate, $rel_{now, a_i}(\alpha_k, t)$, and (3) the ability of the candidate in handling the event, $ability_{a_i}(\alpha_k, e_j, t)$. All these sub-utility measures map into $\Re : 0 \ldots 1$ and each is asymmetric such that $rel_{past, a_i}(\alpha_k, t) \neq rel_{past, \alpha_k}(a_i, t)$.

Now, we define the past relationship between an agent a_i and a candidate α_k. First, suppose that the number of negotiations initiated from an agent a_i to α_k is $\Sigma_{negotiate}(a_i \to \alpha_k)$, the number of successful negotiations initiated from an agent a_i to α_k is $\sum_{negotiate}^{success}(a_i \to \alpha_k)$, the number of negotiation requests from α_k that a_i agrees to entertain is $\sum_{negotiate}^{entertain}(\alpha_k \to a_i)$, the total number of all negotiations initiated from a_i to all its neighbors is $\Sigma_{negotiate}(a_i \to \eta_{a_i})$, and the total number of all successful negotiations initiated from a_i to all its neighbors is $\sum_{negotiate}^{success}(a_i \to \eta_{a_i})$. In our model, $rel_{past, a_i}(\alpha_k, t)$ includes the following:

(a) the helpfulness of α_k to a_i: $\dfrac{\sum_{negotiate}^{success}(a_i \to \alpha_k)}{\Sigma_{negotiate}(a_i \to \alpha_k)}$,

(b) the importance of α_k to a_i: $\dfrac{\sum_{negotiate}(a_i \to \alpha_k)}{\Sigma_{negotiate}(a_i \to \eta_{a_i})}$,

(c) the reliance of a_i on α_k: $\dfrac{\sum_{negotiate}^{success}(a_i \to \alpha_k)}{\sum_{negotiate}^{success}(a_i \to \eta_{a_i})}$,

(d) the friendliness of a_i to α_k: $\dfrac{\sum_{negotiate}^{entertain}(\alpha_k \to a_i)}{\sum_{negotiate}(\alpha_k \to a_i)}$,

(e) the helpfulness of a_i to α_k: $\dfrac{\sum_{negotiate}^{success}(\alpha_k \to a_i)}{\sum_{negotiate}^{entertain}(\alpha_k \to a_i)}$, and

(f) the relative importance of a_i to α_k: $\dfrac{\sum_{negotiate}(\alpha_k \to a_i)}{\sum_{negotiate}(a_i \to \alpha_k)}$.

The higher the value of each of the above attributes, the higher the potential utility the agent a_j may contribute to the coalition; i.e., each is proportional to $rel_{past,a_i}(\alpha_k, t)$. The first three attributes tell the agent how helpful and important a particular neighbor has been. The more helpful and important that neighbor is, then it is better to include it in the coalition. On the other hand, the second last attributes tell the agent the chance of having a successful negotiation. The agent expects the particular neighbor to be *grateful* and more willing to agree to a request based on the agent's friendliness, helpfulness and relative importance to that neighbor. Note that the above attributes are based on data readily collected whenever the agent a_j initiates a request to its neighbors or whenever it receives a request from one of its neighbors. To further the granularity of the above attributes, one may measure them along different event types: for each event type, the initiating agent records the above six attributes. This allows the agent to better analyze the utility of a neighbor based on what type of events that it is currently trying to form a coalition for. In that case, an event type would qualify all the above attributes.

Now, we define the current relationship between an agent a_i and its neighbor α_k. Suppose the number of concurrent negotiations that an agent can conduct is *#negotiation_threads*, and the number of tasks that the agent a_i is currently executing as requested by α_k is $\sum_{execute}(task : initiator(task) = \eta_{k,a_i})$. Suppose $\sum_{negotiate}^{success}(a_i \to \alpha_k)$ is the number of ongoing negotiations initiated from a_i to α_k. In our model, $rel_{now,a_i}(\alpha_k, t)$ includes the following:

(a) negotiation strain between a_i and α_k: $\dfrac{\sum_{negotiate}^{ongoing}(a_i \to \alpha_k)}{\#negotiation\_threads}$,

(b) negotiation leverage between a_i and α_k: $\dfrac{\sum_{negotiate}^{ongoing}(\alpha_k \to a_i)}{\#negotiation\_threads}$, and

(c) degree of strain on a_i from α_k: $\dfrac{\sum_{execute}(task : initiator(task) = \alpha_k)}{\sum_{execute}(task : initiator(task) \in \eta_{a_i})}$.

The first attribute is inversely proportional to $rel_{now,a_i}(\alpha_k,t)$ and the other two are proportional to $rel_{now,a_i}(\alpha_k,t)$. The first attribute approximates how demanding the agent is of a particular neighbor. The more negotiations an agent is initiating to a neighbor, the more demanding the agent is and this strains the relationship between the two and the negotiations suffer. The last two attributes are used as a leverage that the agent can use against a neighbor that the agent is negotiating with, about a request initiated by the neighbor.

Now, we deal with the ability of the candidate to handle an event e_j, $ability_{a_i}(\alpha_k,e_j,t)$. While $rel_{past,a_i}(\alpha_k,t)$ and $rel_{now,a_i}(\alpha_k,t)$ are both domain-independent utilities, $ability_{a_i}(\alpha_k,e_j,t)$ is domain-specific. For example, in a database system, if a coalition requires a reporting agent and α_k is a reporting agent, then it has a high ability measure. In a computing system, if an agent α_k has a high CPU allocation and the coalition formed is for CPU re-allocation to alleviate a computing crisis for agent a_i, then $ability_{a_i}(\alpha_k,e_j,t)$ is high. Note also that both $rel_{past,a_i}(\alpha_k,t)$ and $rel_{now,a_i}(\alpha_k,t)$ are time-dependent because the measures change over time as the agent interacts with its neighbors and world. $ability_{a_i}(\alpha_k,e_j,t)$ is also time-dependent though not as obvious. An event is dynamic and thus may require different responses depending on its characteristics even if its is of the same type. $ability_{a_i}(\alpha_k,e_j,t)$ is further influenced by the current status of the agent a_i. Depending on the ability of the agent itself to handle an event due to its current schedule of tasks and computing resources, a candidate α_k that can perform approximately what a_i wants may be better than another candidate that can perform exactly what a_i wants but for a shorter duration[1]. For example, suppose an event requires $F_\tau = \{f_1, f_2, f_3\}$ and a_i knows how to perform all three functions, candidate α_k knows how to perform f_2, and candidate α_l knows how to perform f_3. Suppose that a_i is currently performing f_2 and f_3 for another event, and thus it needs its neighbors to perform f_2 and f_3 for the current event while it shoulders the responsibility for f_1. On the other hand, suppose that a_i has only one negotiation thread available, meaning that it can only negotiate with one coalition candidate. Thus, it needs to decide between α_k and α_l. When $ability_{a_i}(\alpha_l,e_j,t)$ is further analyzed, the agent realizes that it will soon finish its own execution of f_3, hence the adjusted ability of α_l decreases since the agent a_i can rely on itself to perform the function in a short time. In addition, functions or tasks can be prioritized. Here are some priority heuristics that add to the ability of a candidate: (a) if a candidate can provide a functional capability of high uniqueness to the coalition, (b) if a candidate can provide a functional capability of high importance (with inflexible constraints) to the coalition, (c) if a candidate can provide a functional capability that is very time consuming, or (d) if a candidate can provide a functional capability that is resource taxing.

[1] Assuming that it is more likely to reach a deal when the request is flexible from the initiating agent, it is then easier to form a coalition when approximated abilities are acceptable.

Finally, the potential utility, PU_{α_k,a_i}, of a candidate α_k is a weighted sum of $rel_{past,a_i}(\alpha_k,t)$, $rel_{now,a_i}(\alpha_k,t)$, and $ability_{a_i}(\alpha_k,e_j,t)$[2]:

$$PU_{\alpha_k,a_i} = W_{\Lambda_{ini}(a_i,e_j)} \bullet \left[rel_{past,a_i}(\alpha_k,t) \quad rel_{now,a_i}(\alpha_k,t) \quad ability_{a_i}(\alpha_k,e_j,t) \right]$$

where $W_{\Lambda_{ini}(a_i,e_j)} = \begin{bmatrix} w_{past,a_i,e_j} \\ w_{now,a_i,e_j} \\ w_{ability,a_i,e_j} \end{bmatrix}$ and $w_{past,a_i,e_j} + w_{now,a_i,e_j} + w_{ability,a_i,e_j} = 1$. Note that ultimately these weights may be dynamically dependent on the current status of a_i and the event e_j. A higher resolution of PU_{α_k,a_i} is the following. Suppose that the functions needed to be implemented as the response to the event type τ are $F_\tau = \{f_1, f_2, \cdots, f_N\}$. The ability, $ability_{a_i}(\alpha_k,e_j,t)$, as viewed by the initiating agent a_i, of a candidate α_k includes a matrix of scores when α_k is multi-functional such that

$$ability_{a_i}(\alpha_k,e_j,t) = \left[ability'_{a_i}(\alpha_k, f_1, t) \quad ability'_{a_i}(\alpha_k, f_2, t) \quad \cdots \quad ability'_{a_i}(\alpha_k, f_N, t) \right].$$

As a result, PU_{α_k,a_i} can be further specified as $PU_{\alpha_k,a_i} = \{PU_{\alpha_k,f_1,a_i}, PU_{\alpha_k,f_2,a_i}, \cdots, PU_{\alpha_k,f_N,a_i}\}$. In a homogeneous system, all such sub-utility values will be non-zero. But in a heterogeneous system where an agent may not have all the functions needed in $F_\tau = \{f_1, f_2, \cdots, f_N\}$, some of the sub-utility values will be zero. This resolution allows the initiating agent to perform task-based selection and assignment.

In a scenario where there are multiple coalitions, the agent needs to rank them before negotiations. The potential utility of a coalition is the total sum of all its candidates' potential utilities. In addition, we have the following heuristics to further qualify the potential utility of a coalition: (a) if the number of candidates is too large, then the overall utility goes down, (b) if the demand/request by the initiating agent of one or more candidates is overwhelming, then the overall utility goes down, (c) if the demands/requests are evenly distributed, then the overall utility goes up, (d) if the demands/requests are inflexible, then the overall utility goes down; and vice versa, (e) if the expected time for the coalition formation to be completed is high—judging from its past communication behavior, then the overall utility goes down, (f) if the expected time for the solution to be completed is high, or the expected cost of the solution to be completed is high, then the overall utility goes down, and (e) in the case of multiple coalitions of different events competing for execution, if the event for which the coalition is designed for is of a high priority, then the overall utility goes up. Equipped with these heuristics, an agent can choose among its set of initial coalitions to react to a specific event immediately.

---

[2] Strictly, the notation for PU_{α_k,a_i} should be PU_{α_k,a_i,e_j}. But to simplify our discussions here, we use PU_{α_k,a_i} since we deal with only one event at a time. In Section 3.7 where we talk about multiple coalitions, we will use PU_{α_k,a_i,e_j}.

Note that the design of the coalition evaluation is motivated by two concerns. First, we want to have a good-enough and soon-enough coalition. In a dynamic and time-critical environment, agents do not have time or resources to obtain and maintain perfect information that is needed to rationalize about options for an optimal coalition, as assumed by most work in coalition formation (Section 5). Hence, we use (1) domain-independent data readily collected when agents negotiate and the current status of an agent to estimate that optimality value, and (2) utility-based heuristics designed to promote more efficient coalition formation. We can afford to use this sub-optimal strategy since our model subsequently performs negotiations to finalize the coalition and other coalition-related refinements that are time aware, making the coalition more reflective of the dynamic and uncertain characteristics of the events and the environment. This also allows the agents and the system as a whole to be both reactive and rational. More importantly, an agent does not spend unnecessary computational resources in the rationalization of its coalition that may end up inadequate or useless after negotiations and finalization. Second, we want to have a good-enough and soon-enough coalition that has the best chance to survive. This second concern means that the model is designed to try to retain as much as possible the initial coalition when the coalition is finalized. Whether the coalition will be able to carry out its tasks and achieve its goals successfully becomes a secondary issue.

Note that since our coalition finalization is negotiation-based, that means the initial coalition is biased towards increasing the chance of having successful negotiations among the initiating agent and its candidates. This is evidenced in our use of heuristics to compute the potential utility of the candidates. We will talk about the negotiations in Section 3.4.

Finally, note that the above evaluation approach is a form of reinforcement learning, in which the initiating agent learns to rank neighbors that have been helpful higher in its coalition initialization stage. We will discuss experiments from the viewpoint of learning for coalition formation in Section 4.4.

3.3.3. Task Allocation and Assignment

After $\Lambda_{ini}(a_i, e_j)$ is determined, the initiating agent needs to design a task allocation plan. For a task-driven or function-driven approach, the plan is dictated by the knowledge stored for the type of the event, $\mathrm{K}(type_{e_i} = \tau)$. Based on the potential utility PU_{α_k, a_i} of a candidate α_k, the initiating agent matches a particular task in the plan to a candidate. If there is at most one task assigned to a candidate, we call the assignment *1-to-1*; otherwise, *many-to-1*.

First we address the trivial scenarios. In a homogeneous, uni-functional multiagent system for a task-based approach, every agent knows how to perform exactly the same and the only one function, *f*. In this case, the initiating agent simply negotiates with each candidate to perform their respective tasks—i.e., executing *f* on the contexts that each individual agent knows. In a heterogeneous, uni-functional multiagent system for a function-based approach, every agent, α_k, knows how to perform only one unique function f_{α_k}. The initiating agent here simply assigns the task to the candidate that can perform it.

In a multi-functional system, however, the initiating agent has more flexibility in its task allocation and assignment. First, if the agents are functionally homogeneous and if the task allocation is 1-to-1, then the initiating agent computes the PU_{α_k, a_i} value for each candidate α_k, including the individual $PU_{\alpha_k, f_n a_i}$ values for $n = 1 \ldots N$. In reality, it is likely to have at least

one agent that scores the highest abilities for more than one task. In this scenario, if the task allocation is 1-to-1, then we adopt the following algorithm:

> *Algorithm Priority-Based 1-To-1: (1) rank all prioritized sub-utility* PU_{α_k, f_n, a_i} *values by* their scores, (2) select the top score PU_{α_k, f_n, a_i}, assign f_n to the candidate α_k, and remove all α_k-related scores from the candidate pool, and (3) go back to step 2 until all tasks in $F_\tau = \{f_1, f_2, \cdots, f_N\}$ have been assigned to a unique candidate.

On the other hand, if the task allocation is many-to-1, meaning that the initiating agent can assign more than one task to a single candidate as long as the tasks do not conflict each other in resource usage, time constraints, and goals, then we adopt the following algorithm:

> *Algorithm Priority-Based Many-To-1*: Initialize *n* = 1. (1) rank all prioritized sub-utility PU_{α_k, f_n, a_i} values by their scores, (2) assign the task f_n to the candidate with the top PU_{α_k, f_n, a_i}, and (3) increment *n* and go back to step (2) until *n* = *N*.

Second, if the agents are functionally heterogeneous, then the task allocation and assignment algorithm follows that for the case in which the agents are functionally homogeneous in both the 1-to-1 and many-to-1 scenarios. Note that the candidates with unique functional capabilities will have high PU_{α_k, f_n, a_i} values, allowing those tasks to be assigned first.

At the end of task allocation and assignment, the initiating agent has a list of task-candidate pairs or the assignment. We denote this assignment as $assign_{a_i}(e_j, t) = \{\rho_1, \rho_2, \cdots, \rho_P\}$ where *P* is the total number of assignments, and $\rho = \langle \alpha_\rho, f_\rho \rangle$ states the candidate with its assigned task. This list is also sorted, with the top-prioritized assignments first.

What we have discussed above are clear-cut, simplified approaches to task allocation and assignment. However, to further increase the optimality of the coalition and task distribution, we have the following guidelines.

Bounded Flexibility

The number of coalition members that an initiating agent can approach is bounded by its available resource. For example, suppose an agent has a set of negotiation threads, $\gamma = \{\gamma_1, \gamma_2, \cdots, \gamma_R\}$. Each γ_r is spawned at the startup of an agent and is capable of conducting a negotiation with another negotiation thread of another agent. Then, the number of coalition members to be approached is determined by (1) the number of negotiation threads that are currently available, and (2) the availability of computational resource that the agent currently has to support the eventual negotiations. As a result, together the two factors determine a hard constraint, $\lceil \Lambda_{approached}(a_i, e_j) \rceil$, that specifies the number of coalition members to be approached.

If $|F_\tau| \le \lceil \Lambda_{approached}(a_i, e_j) \rceil$, then the initiating agent simply uses the above task allocation and assignment algorithms.

In a 1-to-1 task allocation case, if $|F_\tau| > \lceil \Lambda_{approached}(a_i, e_j) \rceil$, then the coalition cannot be successfully formed. The initiating agent may quit and ignore the event, or continue doing whatever it can: approaching the top-$\lceil \Lambda_{approached}(a_i, e_j) \rceil$ candidates on its list with requests to perform the top-prioritized tasks. This is not entirely irresponsible, as it is possible that the candidates may spawn off their own coalitions afterwards, causing a chain reaction (Section 3.6). The modified algorithm thus becomes:

Algorithm Priority-Based 1-To-1 Bounded: (1) rank all prioritized sub-utility PU_{α_k, f_n, a_i} values by their scores, (2) select the top score PU_{α_k, f_n, a_i}, assign f_n to the candidate α_k, and remove all α_k-related scores from the candidate pool, and (3) go back to step (2) until (a) all tasks in $F_\tau = \{f_1, f_2, \cdots, f_N\}$ have been assigned to a unique candidate or (b) the number of candidates assigned so far is equal to $\lceil \Lambda_{approached}(a_i, e_j) \rceil$.

In a many-to-1 task allocation scenario, if $|F_\tau| > \lceil \Lambda_{approached}(a_i, e_j) \rceil$ and it is possible to assign non-conflicting tasks to one candidate, then we have the following algorithm:

Algorithm Priority-Based Many-To-1 Bounded: Initialize *n* = 1. (1) rank all prioritized sub-utility PU_{α_k, f_n, a_i} values by their scores, (2) assign the task f_n to the candidate with the top PU_{α_k, f_n, a_i}, (3) increment *n* and go back to step (2) until *n* = *N*, (4) if the number of candidates assigned is greater than $\lceil \Lambda_{approached}(a_i, e_j) \rceil$ then perform *Algorithm Task Shuffle*, with the assignment $assign_{a_i}(e_j, t) = \{\rho_1, \rho_2, \cdots, \rho_P\}$ as the argument, and (5) if the algorithm returns with a failure, then remove the last members of $assign_{a_i}(e_j, t)$ until $P = \lceil \Lambda_{approached}(a_i, e_j) \rceil$.

Algorithm Task Shuffle (Lazy): Initialize *i* = 1. (1) if *i* = *P*, then return with a failure, (2) *absorb* ρ_P into ρ_i and re-organize $assign_{a_i}(e_j, t)$, (3) if the absorption fails, then increment *i* by 1 and go back to step (1), (4) otherwise, if new $P > \lceil \Lambda_{approached}(a_i, e_j) \rceil$, then go back to step (1), (5) otherwise, return with a success and a new $assign_{a_i}(e_j, t)$.

The function $absorb(\rho_j, \rho_i)$ returns true if the agent is able to absorb the task-candidate pair ρ_j into ρ_i, making α_{ρ_i} performing both f_{ρ_i} and f_{ρ_j}. When the assignment is re-organized, the task-candidate pair that has been absorbed is removed from the list, and the new task-candidate pair has become $\rho_i = \langle \alpha_{\rho_i}, \{f_{\rho_i}, f_{\rho_j}\} \rangle$. As a result, the total number of assignments decreases by 1, resulting in a new *P*. Note that the function $absorb(\rho_j, \rho_i)$ is domain-specific guided by

domain-independent rules. Basically, absorption is feasible only if the two tasks do not compete for the same resource and do not have conflicting goals. In addition, the above task-shuffling algorithm is lazy as it tries to dump all the extra assignments into the first (and top-prioritized) task-candidate pair. This may be rational, as the first candidate associated with the top-prioritized task-candidate pair is more likely to become a useful coalition member. Variants of the task-shuffling algorithm involve modifications to the fourth step. Instead of going to the same task-candidate pair, one may want to increment *i* by 1.

Soft-Bounded Flexibility

It is also practical to include soft constraints such that the number of coalition members that an initiating agent *wants* to approach is bounded by its beliefs and intentions. For example, if the initiating agent is anticipating a highly important event to occur in the next time step, then it may want to reserve some of its resources to respond to that event. As a result, it may not want to respond fully to the current event. This anticipatory behavior introduces soft-bounds. If an agent decides to soft-bound its $\lceil \Lambda_{approached}(a_i, e_j) \rceil$, then it may want to remove the last members of $assign_{a_i}(e_j, t)$ that are of low PU_{α_k, a_i} values since the chance of negotiating successfully with each of these candidates is low and the utility of its contribution is low as well.

Imperfect Coalition and Greedy Algorithms

As mentioned previously, when an initiating agent is faced with a dilemma where it has more negotiations to perform than it has available resources to conduct negotiations with its coalition members, it either (1) quits or (2) continues with as many negotiations as possible to recruit as many coalition members as possible. This is feasible since each agent in our system is capable of forming a coalition dynamically on its own. Each agent has the knowledge on how to deal with events that it senses and reacts to it. In a way, this implies that *if the initiating agent can get the message out, then hopefully the coalition members will pass the message along to their own coalition members.* So, an initiating agent does not necessarily have to plan for a perfect coalition solution for an event. Moreover, it is unlikely to obtain a perfect coalition solution even with a perfect plan since the coalition formation process is subjected to dynamic changes in the environment, noise, message loss, refusals to negotiate, and failed negotiations. Thus, we have lazy and greedy algorithms that are opportunistic. Our task-shuffling algorithm above is lazy, as it does not solve for the optimal absorption, for example.

Here, we introduce a greedy algorithm for task allocation and assignment, for a multi-functional, heterogeneous multiagent system, in a 1-to-1 task allocation scenario. First, we define a modified prioritized utility score called the *focused utility*. We denote it as:

$$PU'_{\alpha_k, f_n, a_i} = W_{\Lambda_{ini}(a_i, e_j)} \bullet \left[rel_{past, a_i}(\alpha_k, t) \quad rel_{now, a_i}(\alpha_k, t) \quad \frac{ability_{a_i}(\alpha_k, e_j, t) + ability'_{a_i}(\alpha_k, f_n, t)}{2} \right].$$

So the utility value has an emphasis in what particularly the candidate α_k knows how to do, from the point of view of the initiating agent a_i, in an initial coalition of $\Lambda_{ini}(a_i, e_j)$. If an agent has *n* functional capabilities that suit the tasks that the initiating agent wants done, then it has *n* such focused utility values. Then we have the following algorithms:

Algorithm Greedy Priority-Based 1-To-1 Bounded: Initialize *n* = 1. (1) rank all focused utility values, (2) assign the task f_n to the candidate with the top PU'_{α_k, f_n, a_i}, (3) remove all utility values of that candidate from the ranking, (4) increment *n* and go back to step (2) until *n* = $\lceil \Lambda_{approached}(a_i, e_j) \rceil$.

Algorithm Greedy Priority-Based Many-To-1 Bounded: Initialize *n* = 1. (1) rank all focused utility values, (2) assign the task f_n to the candidate with the top PU'_{α_k, f_n, a_i}, and (3) increment *n* and go back to step (2) until *n* = $\lceil \Lambda_{approached}(a_i, e_j) \rceil$.

An initiating agent becomes greedy when practicing the above algorithms because (1) it tries to minimize its own rationalization and computing process, (2) it selects the candidate with the higher overall utility values to approach hoping for a successful negotiation, (3) it cares mostly about high-priority tasks, (4) it tries to maximize its chance of getting a particular task done—by including sub-utilities in the focused utility evaluation, and (5) it hopes to shift its responsibility (partially) to the candidates via successful negotiations—expecting the candidates to spawn their own coalitions to help respond to the event.

Insurance and Worried Algorithms

Since negotiations cannot be guaranteed to be always successful, that means some initial candidates may be dropped from the final coalition. This also implies that if an initiating agent over-relies on one particular candidate, then the initiating agent may lose a large portion of the coalition's utility. So, in the task allocation and assignment process, we can build in some insurance policies—some alternative plans—to at least absorb the impact of such disasters. Of course, an initiating agent considers these plans only when it has enough computational resources to do so, i.e., $\lceil \Lambda_{approached}(a_i, e_j) \rceil > |F_\tau|$. As such, we have the following *worried* algorithms:

Algorithm Worried Priority-Based 1-To-1 Bounded: (1) rank all prioritized sub-utility PU_{α_k, f_n, a_i} values by their scores, (2) select the top score PU_{α_k, f_n, a_i}, assign f_n to the candidate α_k, and remove all α_k-related scores from the candidate pool, (3) go back to step (2) until all tasks in $F_\tau = \{f_1, f_2, \cdots, f_N\}$ have been assigned to a unique candidate, (4) repeat steps (2)-(3) until number of candidates assigned so far is equal to $\lceil \Lambda_{approached}(a_i, e_j) \rceil$.

In a many-to-1 task allocation scenario, if $\lceil \Lambda_{approached}(a_i, e_j) \rceil > |F_\tau|$ and it is possible to assign non-conflicting tasks to one candidate, then we have the following algorithm:

Algorithm Worried Priority-Based Many-To-1 Bounded: Initialize n = 1. (1) rank all prioritized sub-utility PU_{α_k, f_n, a_i} values by their scores, (2) assign the task f_n to the candidate with the top PU_{α_k, f_n, a_i}, (3) increment n and go back to step (2) until the number of candidates assigned so far is equal to $\lceil \Lambda_{approached}(a_i, e_j) \rceil$.

Note that the *insurance* assignments as a result of the worried algorithms will be aborted once the initiating agent has achieved a satisfactory coalition (e.g., as different negotiations complete with successes). We will discuss this further in Section 3.4.2.

Over-Demanding and Caps

Of course, the lazy and greedy algorithms may end up assigning all tasks to a single agent. This becomes an over-demanding scenario that complicates the negotiation, and, as a result, the coalition may suffer. Hence, the number of assignments for a candidate has to be bounded when the computational resource of the initiating agent can afford it. The cap can be determined dynamically. For example, if the primary task that the initiating agent wants the candidate to perform is extremely important, or highly unique, then it is better for the initiating agent to not over-demand in its approach to the candidate. On the other hand, if the candidate has been very helpful and friendly, then the initiating agent may be able to take advantage of that relationship by over-demanding. Suppose we denote the cap for an assignment ρ for candidate α_k as $\lceil f_\rho \rceil_{\alpha_k}$, then $\lceil f_\rho \rceil_{\alpha_k} \propto rel_{past, a_i}(\alpha_k, t)$, $\lceil f_\rho \rceil_{\alpha_k} \propto rel_{now, a_i}(\alpha_k, t)$, and $\lceil f_\rho \rceil_{\alpha_k} \propto 1/ability_{a_i}(\alpha_k, e_j, t)$. And these caps can be inserted into all the algorithms above to prevent too many assignments to a single agent.

3.3.4. Resource-Driven Allocation

In this subsection, we deal with resource allocation and assignment. In a resource-driven approach, we still have

$$PU_{\alpha_k, a_i} = W_{\Lambda_{ini}(a_i, e_j)} \bullet \left[rel_{past, a_i}(\alpha_k, t) \quad rel_{now, a_i}(\alpha_k, t) \quad ability_{a_i}(\alpha_k, e_j, t) \right].$$

However, we have a higher resolution of PU_{α_k, a_i}. Suppose that the resources to be obtained by the initiating agent as the response to the event type τ are $R_\tau = \{r_1, r_2, \cdots, r_N\}$. The ability, $ability_{a_i}(\alpha_k, e_j, t)$, as viewed by the initiating agent a_i, of a candidate α_k includes a matrix of scores when α_k is multi-resource-based such that

$$ability_{a_i}(\alpha_k, e_j, t) = \left[ability'_{a_i}(\alpha_k, r_1, t) \quad ability'_{a_i}(\alpha_k, r_2, t) \quad \cdots \quad ability'_{a_i}(\alpha_k, r_N, t) \right].$$

As a result, PU_{α_k,a_i} can be further specified as $PU_{\alpha_k,a_i} = \{PU_{\alpha_k,r_1,a_i}, PU_{\alpha_k,r_2,a_i}, \cdots, PU_{\alpha_k,r_N,a_i}\}$. In a homogeneous system where each agent is allocated and uses the same set of resources, all such sub-utility values will be non-zeros. But in a heterogeneous where an agent may not have all the resources needed in $R_\tau = \{r_1, r_2, \cdots, r_N\}$, some of the sub-utility values will be zeros.

Now, the initiating agent needs to determine how much to ask for from each candidate for each resource. Suppose we denote the amount of resource r_n to ask for from a candidate α_k as $|r_n|_{\alpha_k,a_i}$ and the total amount of resource that the initiating agent a_i needs as a response to an event e_j as $\lfloor r_n \rfloor_{e_j,a_i}$. We have

$$|r_n|_{\alpha_k} = \lfloor r_n \rfloor_{e_j,a_i} \cdot \frac{PU_{\alpha_k,r_n,a_i}}{\sum_{\alpha_m \in \Lambda_{ini}(a_i,e_j)} PU_{\alpha_m,r_n,a_i}}.$$

Hence, we determine the amount of resource proportional to the potential utility of a candidate in its ability to provide that particular resource.

Since not all negotiations are guaranteed to be successful and not all coalition candidates approached by the initiating agent are guaranteed to join the coalition, once again, we can incorporate some insurance policies such as multiplying $|r_n|_{\alpha_k,a_i}$ by a factor greater than 1. This factor may be computed dynamically based on the importance of the event, the criticality of the resource and other issues, as stored in $\mathrm{K}(type_{e_i} = \tau)$. Note that the resource assignments will be refined (increased or decreased) once the initiating agent starts to receive various accepted deals, reported from its negotiation threads. We will discuss this further in Section 3.4.2.

The resource allocation and assignment algorithms are similar to the priority, bounded, greedy, and worried algorithms presented above for the task-driven and function-driven approaches.

3.4. Coalition Finalization: Negotiation

Agent-based negotiations involve information exchanges between two agents where the initiating agent desires to persuade the responding agent to accept a task. Two agents need to exchange information as each has only a partial set of the information or knowledge contained in the entire multi-agent environment, especially where information cannot be updated and distributed quickly and accurately to all agents. Thus, when an initiating agent encounters a task that it believes that another agent can perform, it negotiates by informing that agent of the description of the task and its own constraints. In this way, each agent maintains a local information base and information is exchanged only when necessary. In our model, we use a real-time case-based logical negotiation protocol to dictate the rules of encounter or the negotiation strategies between two agents. Interested readers are referred to (Soh and Tsatsoulis 2001) for a detailed presentation of the logical protocol. Some important work in negotiation can be found in (Durfee and Lesser 1991; Lâasri et al. 1992; Kraus *et al*. 1995; Zlotkin and Rosenschein 1996; Doran *et al.* 1997; Kraus 1997; Faratin *et al*. 1998; Kraus *et al.* 1998; Parsons *et al.* 1998).

Here we briefly describe our negotiation protocol. An agent has a set of negotiation threads. When an agent needs to negotiate with one of its neighbors, it uploads the information necessary for the negotiation and activates one of its threads. That negotiation thread downloads the information and then carries out its negotiation autonomously with the responding agent. The negotiation is iterative. The negotiation thread composes and sends out messages to the responding agent, and receives and parses the incoming messages from the responding agent. At each step, the negotiation thread decides what to do next, following a negotiation strategy customized for itself by the parent agent. Also, the negotiation thread checks its data cache to monitor its negotiation pace, to see whether the parent agent has additional instructions such as "Abort now", "increase your demand", and so on. During a negotiation, an initiating agent basically tries to convince a responding agent to perform a set of tasks or give up a certain set of resources. To do so, it sends over arguments—information pertinent to its constraints, commitments, and perception. The responding agent evaluates these arguments and updates its own view of the situation. If the arguments are persuasive enough, then the responding agent agrees to the request. The responding agent also has the ability to counter-offer, for example, when the initiating agent has exhausted all its arguments, or when the responding agent grows impatient and is about to quit the negotiation.

Since our negotiation protocol is iterative, it facilitates interactions between negotiation threads. An initiating agent can invoke a host of concurrent negotiations, one to each of its coalition members in $\Lambda_{ini}(a_i, e_j)$ bounded by $\lceil \Lambda_{approached}(a_i, e_j) \rceil$. While the negotiation threads are actively engaged in their respective negotiations, the initiating agent continues to monitor its world, examine its tasks, communicate with other agents, and watch the status of its negotiation threads. Since each of these threads knows how to negotiate on its own, all it needs from time to time is for the parent agent to update the agent's current beliefs and intentions that might interrupt the negotiation or change the negotiation issues. The parent agent thus is able to infuse a high-level of awareness in the negotiation threads, relax the negotiation issues (less demanding or more conceding, for example), and abort negotiations with diminishing returns.

In the following subsections, our discussions focus on the facilities that the agent provides for its negotiation threads such that there are indirect inter-thread activities to achieve a real-time working coalition.

3.4.1. Awareness

Since the environment is dynamic, an ongoing negotiation may become useless. For example, if the negotiation is part of a response to an event e_j and e_j becomes false, then the negotiation has to be terminated. Since a negotiation thread handles the negotiation semi-autonomously, it must be aware of such a situation, and the parent agent has to provide such awareness. This coalition awareness has several benefits. First, it allows an agent to free up its negotiation threads, communication channels, and communication bandwidth for other negotiation tasks. Second, it allows an agent to immediately abandon failing coalition, re-assess its environments, and start another coalition formation. Third, by terminating useless negotiations, an agent is able to base its reasoning on updated, more correct status profile.

A negotiation thread conducts its negotiation following a logical real-time protocol that spells out what it should do in each negotiation iteration, and a negotiation strategy that dictates how the thread should negotiate—how much time it has, how conceding it should be, what kind of

arguments it has, which arguments it should send first, and so on. It also needs to know the context of the negotiation—the request, the amount of resource to give up, the counterpart agent, and so on. When activated, a negotiation thread downloads the negotiation context and strategy from the parent agent. Then, if the thread does not hear from the parent agent, it knows how to negotiate on its own and report back to the parent agent when the negotiation is completed.

The parent agent, on the other hand, carries out its normal tasks such as monitoring the world, actuating its sensors, and so on. It holds a shared data object with each negotiation thread. When the event changes or the current status of the coalition changes, the agent evaluates the current status of each negotiation thread and makes a decision as to whether to relax, to terminate, or refine the negotiation. This decision together with its pertinent information is stored at that data object. We call this shared data object the *awareness link*, or AL_{a_i,β_z} for the β_z negotiation thread of agent a_i, where a_i has X negotiation threads, $\{\beta_1, \beta_2, \cdots, \beta_X\}$. Both the agent and the negotiation thread can store and access data on this awareness link. With this design, the parent agent shoulders the task of feeding its negotiation threads additional instructions. There are several reasons why we adopt this awareness link design in our model. First, the environment is dynamic and real-time critical. It does not make sense for each negotiation thread to setup its own sensors, monitors, and even decision makers to determine the current status of the agent and changes its negotiation behavior accordingly, since it would have to sieve through unrelated information and data and that would be time consuming. Second, it is natural for the agent to disperse the information to all its negotiation threads. A negotiation thread does not know the status of a coalition (e.g., whether the coalition is failing or succeeding); only the agent knows that. With that knowledge, an agent can decide whether to scale back on some of its negotiations, or make other modifications. This way, the chain of command is direct and less confusing, and certainly less computationally intensive. Third, with each negotiation thread having its own dedicated awareness link, the information or data passed through the parent agent and that particular thread does not interfere with the other threads. This way, each negotiation thread can concentrate on the instructions specifically directed to it from the parent agent.

There are several issues one needs to address—the frequency of the updates and checks and the number of items to be updated and checked at each awareness link. In a real-time system, an agent or a processing thread cannot afford constantly polling or checking for possible information before moving to the next step. A parent agent should only update the awareness link when it has some significant changes to a particular negotiation thread and should only access the awareness (to check upon the status of the negotiation) only when it needs to make a decision based on that piece of information. This is straightforward since both the updates and accesses are driven by events. On the other hand, the negotiation thread needs to check its awareness link to see if its parent agent has something for the negotiation, and that is an asynchronous matter. This frequency of checks depends on the degree of *anytime*-ness that the problem calls for. If the design requires the negotiation thread to check at every step so that it can bail out as soon as possible to conserve, say, the communication channel usage, then the thread has to check the awareness link at every iteration. The thread also may need to report back to its parent agent on its progress: Has the negotiation been going as planned? How far has it fallen behind? And so on. The frequency of this type of updates depends on the resolution of the decision points of the parent agent. The higher the resolution, the parent agent can have more complete data for its options while incurring more computational costs and delays.

In our model, the parent agent checks the negotiation status of its negotiation threads within a framework of tasks. It checks its messages, its sensors for events, tasks, and the negotiations, and then repeats. This lifecycle varies in its duration, as the environment is dynamic and uncertain. For the negotiation thread, we propose a gradated scheme based on percentage of time elapsed. For example, a thread checks and updates its status (1) less frequently in the beginning, (2) more frequently towards the end, (3) less frequently when it is progressing according to plan, and (3) more frequently when it is failing. This is because we assume that the event status is still relatively constant in the beginning of the negotiation and only changes after a certain time period has passed. We also assume that when a negotiation is progressing well and succeeding, that negotiation thread should carry on and complete the negotiation unless some significant event occurs and calls it off. In this manner, the agent does not lose the utility of such a negotiation and learns to be efficient. We also assume that when a negotiation is not doing well, after reporting it to the parent agent, the negotiation thread can expect further instructions from the parent agent, hence the increase in its access of the awareness link.

3.4.2. Relaxation and Termination

Each agent is responsible for the coordination among its negotiation threads as the negotiation threads do not talk to each other directly. The agent monitors the status of the negotiations and makes decisions. Two of the decisions it can make are relaxation and termination. From the initiating agent standpoint, this relaxation results in a smaller demand; from the responding agent standpoint, this relaxation results in a more yielding stance. Since this paper's focus is in coalition formation, we will discuss relaxation and termination from the viewpoint of an initiating agent.

As previously mentioned in Section 3.3, in a task-driven or function-driven approach, a parent agent may intentionally negotiate with additional neighbors as an insurance policy. Similarly, in a resource-driven approach, a parent agent may intentionally increase its request for a certain resource two-fold to increase the chance of getting what it actually needs. As the individual threads start to report in one by one, the agent may realize that it no longer needs those insurance policies anymore. Similarly, the event that the agent has planned its negotiations for may have changed for the better, or the agent itself has completed other computational intensive tasks and just freed up some much-needed resources. These changes also cause the agent to relax or terminate its active negotiations.

Here we have several heuristics on the relaxation. Suppose in a 1-to-1 task allocation problem, we have $\left|\Lambda_{approached}\left(a_i, e_j\right)\right| > \left|F_\tau\right|$. That is, the number of candidates that the agent a_i approaches is greater than the number of tasks required to respond to the event e_j. At time *t*, a_i activates all its negotiation threads, each with a partial-assignment $\rho = \left\langle \alpha_\rho, f_\rho \right\rangle$. At time $t + \Delta$, some changes have been detected (in the agent status, in the event, or in the negotiation status), such that F_τ is now F_τ'. If $F_\tau' \not\subset F_\tau$, then the agent needs to (1) immediately terminate all negotiation threads with $\rho = \left\langle \alpha_\rho, f_\rho \right\rangle$ where $f_\rho \notin F_\tau'$, and (2) label this change as a new event and proceed from there accordingly. If $F_\tau' \subset F_\tau$, then the agent uses the following algorithm:

Algorithm 1-to-1 Relaxation and Termination: (1) for all ongoing negotiations with $\rho = \langle \alpha_\rho, f_\rho \rangle$ where $f_\rho \notin F'_\tau$ do: (1.1) check to see whether an additional agent performing the same task or an unnecessary task can improve the quality of the solution or benefit the environment and denote this as $benefit_{a_i}(\alpha_\rho, f_\rho, t)$, (1.2) check the current status of the negotiation and denote this as $progress_{\beta_x}(\alpha_\rho, f_\rho, t)$ for negotiation thread β_x, (1.3) compute the expected utility of continuing with this negotiation:

$$EU_{\beta_x}(\alpha_\rho, f_\rho, t) = \frac{PU_{\alpha_\rho, f_\rho, a_i} + progress_{\beta_x}(\alpha_\rho, f_\rho, t) \cdot benefit_{a_i}(\alpha_\rho, f_\rho, t)}{2},$$

(1.4) if $EU_{\beta_x}(\alpha_\rho, f_\rho, t)$ is greater than what the agent can afford to spend in resources, then the agent continues in the negotiation; otherwise, it terminates the negotiation.

In a many-to-1 task allocation problem, we have a higher level of flexibility. Suppose that the negotiation thread in question has $\rho = \langle \alpha_\rho, \{f_{\rho_1}, f_{\rho_2}\} \rangle$. Then, the agent can compute $benefit_{a_i}(\alpha_\rho, f_{\rho_1}, t)$ and $benefit_{a_i}(\alpha_\rho, f_{\rho_2}, t)$ (assuming disjoint tasks). It then can decide to drop f_{ρ_1} or f_{ρ_2} from its original demand or keep both. The algorithm becomes:

Algorithm Many-to-1 Relaxation and Termination: (1) for all ongoing negotiations with $\rho = \langle \alpha_\rho, f_\rho \rangle$ where $f_{\rho_s} \in f_\rho$ and $f_{\rho_s} \notin F'_\tau$ do: (1.1) compute $benefit_{a_i}(\alpha_\rho, f_{\rho_s}, t)$ for all $f_{\rho_s} \in f_\rho$ and $f_{\rho_s} \notin F'_\tau$, (1.2) check the current status of the negotiation and denote this as $progress_{\beta_x}(\alpha_\rho, f_\rho, t)$ for negotiation thread β_x, (1.3) compute the expected utility of continuing with this negotiation for all $f_{\rho_s} \in f_\rho$ and $f_{\rho_s} \notin F'_\tau$:

$$EU_{\beta_x}(\alpha_\rho, f_{\rho_s}, t) = \frac{PU_{\alpha_\rho, f_{\rho_s}, a_i} + progress_{\beta_x}(\alpha_\rho, f_\rho, t) \cdot benefit_{a_i}(\alpha_\rho, f_{\rho_s}, t)}{2},$$

(1.4) if $EU_{\beta_x}(\alpha_\rho, f_{\rho_s}, t)$ is greater than what the agent can afford to spend in resources, then the agent retains f_{ρ_s} in the negotiation; otherwise, it drops f_{ρ_s} from the negotiation.

The expected utility is basically the potential utility of the coalition member to the original event response plus the utility of continuing with the negotiation. The latter utility says that if the negotiation is progressing well and the eventual outcome will benefit the environment, then the agent should not waste the ongoing effort and should continue with the negotiation. However, in step (1.4) above, the agent is also rational: it continues only when it is affordable to do so. If the agent has other tasks to do, has to divert its computational resources to other areas, or has to come up with free negotiation threads for other events, then the continuation becomes costly and counter-productive.

Similarly, in a resource-driven approach, we have R_τ, R'_τ, $\rho = \langle \alpha_\rho, r_\rho \rangle$, and $PU_{\alpha_k, a_i} = \{PU_{\alpha_k, r_1, a_i}, PU_{\alpha_k, r_2, a_i}, \cdots, PU_{\alpha_k, r_N, a_i}\}$. Hence, we may use the same algorithms for relaxation and termination of negotiations in a resource-allocation problem.

The relaxation and termination behavior is both rational and altruistic. An agent should not be conducting negotiations that use its resources when those negotiations become useless. Neither should an agent *impose* or *transfer* that cost to its responding agent by insisting on useless negotiations. The responding agent would have to entertain the negotiation without knowing that the negotiation has become useless, and would have to perform the request had the negotiation reached a successful conclusion. Also, this behavior enables the initiating agent to be bold and aggressive in its coalition initialization (Section 3.3). Without this relaxation and termination capability, the initiating agent would have to be more careful in its initialization since it has less room for errors. That would mean for the initiating agent to collect more information in its rationalization which would in turn decrease the autonomy and robustness of the multiagent system. So, the coupling of the initialization and relaxation/termination is very important in our dynamic, negotiation-based coalition formation model.

Finally, at the end of all negotiations, we have the final coalition $\Lambda_{final}(a_i, e_j)$ and $\Lambda_{final}(a_i, e_j) \subseteq \Lambda_{approached}(a_i, e_j) \subseteq \Lambda_{ini}(a_i, e_j)$.

3.5. Chain Reaction

In our model, the multiagent system can also help guarantee a successful group of coalitions. Since each agent is autonomous, a partially successful coalition from an agent to its neighbor can lead to the neighbor detecting an event and forming another coalition to help respond to the event. Moreover, the propagation of constraints, commitments, and perceptions from an agent to another is possible via negotiations. As a result, the first initiating agent can create a chain reaction throughout the multiagent system across various neighborhoods resulting in multiple overlapping coalitions. This self-organizing behavior allows the coalition formation process to be more reactive and less rational, which is advantageous in a multiagent system of incomplete information.

To alleviate the impact of a coalition failure, our agent behavior design resembles a natural mitigation approach. Our agent constantly monitors its environments. So, when a coalition fails, the agent continues to monitor and if it finds the same problem still present, it can start another round of coalition formation. Since our agents are reflective and situation-aware, this new coalition will have a different problem profile, neighborhood profile, and agent profile. This in effect allows the agent to look at the problem from a slightly different viewpoint, which may eventually lead to a successful coalition being formed. The rate of such a recovery (the number of coalition failures before success) is critically dependent on the dynamism of the problem.

This mitigation approach belies the principles of our coalition formation approach: the agents are willing to fail many times before getting it right because of the dynamism of the system that does not allow the agents to rationalize optimally an does not allow the agents to guarantee the successful formation of an optimal coalition.

3.6. Multiple Coalitions and Inter-Coalition Behavior

If an agent a_i detects E simultaneous events, then it will have E possible initial coalitions, $\vec{\Lambda}_{ini}(a_i, \vec{E}) = \{\Lambda_{ini}(a_i, e_1), \Lambda_{ini}(a_i, e_2), \cdots, \Lambda_{ini}(a_i, e_E)\}$. Since negotiation is time consuming and resource taxing, the agent a_i cannot afford finalizing every coalition to find out the utilities of the finalized coalitions. Thus, we use the potential utilities: PU_{α_k, a_i, e_j}. This is also where domain-specific heuristics may come into play. For example, if each event type is prioritized, then we will want the initiating agent to handle the top-priority event and thus its corresponding coalition. In our model, since an agent cannot possibly handle all coalition formations at once due to limited resources, it invokes coalition formation in a first-come first-serve manner. Since the agent monitors its environment and its own status, if the event that has lost out still persists, then the agent can launch another coalition formation process at a later time. This is how our model handles a series of events.

In our model, an agent can have multiple working coalitions concurrently as long as parts that the agent play in the multiple coalitions are not conflicting in goals and do not use the same resources at the same time. Since multiple coalitions can concurrently occur, they may affect each other's immediate environment. Thus there are intra-coalition behaviors via the actuation of the agents that affect the environment and the sensing of the changes. In addition, suppose an agent has one working coalition, and is now forming a second coalition. Since being a part of a working coalition indicates a set of commitments, constraints, and perceptions, the agent invariably considers those when it is trying to form the second coalition. For example, if the agent is executing a task that requires a large portion of its allocated CPU resource, then it will have less resource for the coalition formation, and it will require more help from its coalition candidates for its second coalition. This is how a coalition interacts with another.

4. An Implementation and Discussions

The driving application for our system is multisensor target tracking, a distributed resource allocation and constraint satisfaction problem. The objective is to track as many targets as possible and as accurately as possible using a network of sensors. Each sensor has a set of consumable resources, such as beam-seconds (the amount of time a sensor is active), battery power, and communication channels, which each sensor desires to utilize efficiently. Each sensor is at a fixed physical location and, as a target passes through its coverage area, it has to collaborate with neighboring sensors to triangulate their measurements to obtain an accurate estimate of the position and velocity of the target. As more targets appear in the environment, the sensors need to decide which ones to track, when to track them, and when not to track them, always being aware of the status and usage of sensor resources.

The problem is further complicated by the real-time constraints of the environment and the fact that agents have to share physical resources such as communication channels and disk storage. For example, for a target moving at 0.5 foot per second, accurate tracking requires one measurement each from at least three different sensors within a time interval of less than 2 seconds. The real-time constraints force our agents to deal with issues such as CPU allocation (since speed of execution depends on it), disk space allocation, communication latency, and processing times. Finally, the environment is noisy and subject to uncertainty and error: messages may be lost, a sensor may fail to operate, or a communication channel could be jammed. Thus, in addition to improving autonomy, one is required to promote noise-resistance in agent reasoning, sensor control, and communications.

The sensors are 9.35 GHz Doppler MTI radars that communicate using a 900 MHz wireless, radio-frequency (RF) transmitter with a total of eight available channels. Each sensor can at any time scan one of three sectors, each covering a 120-degree swath. Sensors are connected to a network of CPU platforms on which the agents controlling each sensor reside. The agents (and sensors) must communicate over the eight-channel RF link, leading to potential channel jamming and lost messages. Finally, there is software (the "tracker") that, given a set of radar measurements, produces a possible location and velocity for a target; the accuracy of the location and velocity estimates depend on the quality and frequency of the radar measurements: as we mentioned, the target must be sensed by at least three radars within a two second interval for accurate tracking.

Our agent architecture is as follows. Each agent has 3+*B* threads. It has a core main thread that does the decision making, manages the tasks, performs coalition formation, and oversees the negotiations. It has a communication thread that interacts with the message send/receive system of the radar (or the simulated software) to poll for incoming messages and to physically send out messages. It has an execution thread that actuates the physical sensor: calibration, search-and-detect for a target, turn on/off a sensing sector, change the orientation of the sensor, and measure a target's return signals. It then has *B* negotiation threads. Each thread is dormant until activated. When it is activated, it downloads pertinent information form the parent agent and proceeds with its negotiation—either as an initiating agent or a responding agent, depending on the information from the parent agent through its dedicated awareness link.

We have implemented part of our dynamic, negotiation-based coalition formation model in our multisensor target tracking system and plan to implement the entire model as we include more complicated tasks and events into our system. Currently, all agents are homogeneous and autonomous. Each is capable of sensing its environment, reacting to it, making decisions on coalitions, negotiating with its neighbors, and performing tasks that affect the environment. Currently, we have implemented two types of events: an incoming target and a CPU shortage crisis.

4.1. Multisensor Target Tracking

When an agent detects a target in its sensing sector, it first obtains its estimated velocity and position from a tracker software module. Equipped with these estimates, it is able to generate $\Psi_{F_\tau, a_i}(t)$ based on a geometric model of the orientations of the neighbors' sensors and their locations. Given this list, the agent is able to obtain $\Lambda_{ini}(a_i, e_j) = \Psi_{F_\tau, a_i}(t) \cap \eta_{a_i}$. Subsequently, the agent ranks the candidates based on their potential utility following evaluation scheme outlined in Section 3.3.2. Since the standard response to target tracking is to turn on a specific sensing sector and measure, $|F_\tau| = 1$. So, we use the priority 1 and use the *Algorithm Priority-Based 1-To-1 Bounded* to allocate the tasks and use the number of available negotiation threads as $|\Lambda_{approached}(a_i, e_j)|$. The initiating agent then activates its negotiation threads with the corresponding assignments. The negotiation threads conduct their negotiations. Since we have only one target in the environment, the initiating agent does not have the opportunity to perform relaxation and termination. As a result, each negotiation thread currently only monitors its own progress and if it is running out of time, it counter-offers to speed up the negotiation, and if it has run out of time, it aborts the negotiation and reports back to its parent agent. The parent agent then downloads the information from the completed negotiation thread and carries out the deal

reached—scheduling the deal in its job queue, allocating CPU resource in anticipation of the task, and performing the agreed task. One of the domain-specific criteria used in determining $ability_{a_i}(\alpha_k, e_j, t)$ for the candidates is either the time of arrival of the target into the sensing sector of the candidate or the time of departure of the target from the sensing sector of the candidate. Candidates that have less time will have a higher ability as they need to be approached soon before the target leaves those sensing coverage areas.

We have tested our multiagent system in a physical hardware setup and in a software simulation. Here are some experimental setup parameters. There are four agents. Each agent controls a radar. The radars are fixated at four corners of a 20x20 feet square. A target moving at 0.5 ft/sec moves in a route (rectangular, diagonal, or circular). Figure 1 shows a graphical representation of our simulation.

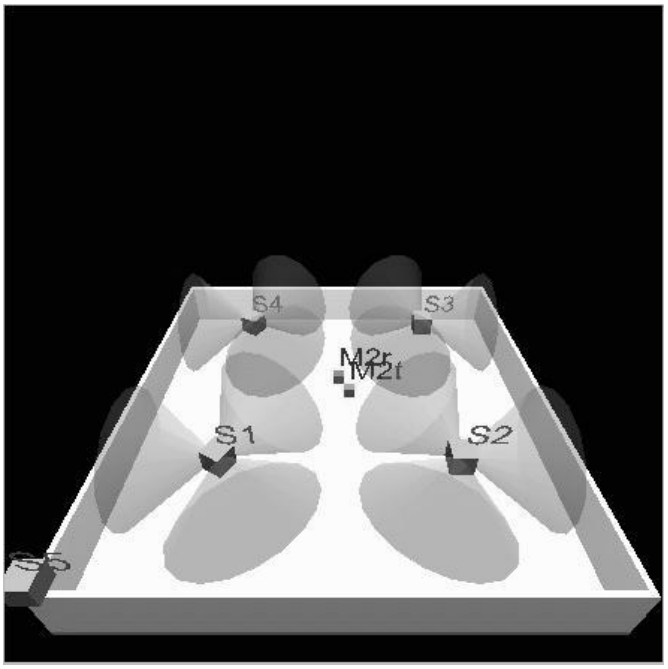

Figure 1 The graphical representation of the environment that our agents operate. S1-S4 are four sensors, each controlled by an agent. M2r is the ground truth of the target. M2t is the estimated target position. Each sensor has three sensing sectors.

Table 1 shows the best mean square error (MSE) in feet of our tracking in a noiseless environment.

Error	**Dx (feet)**	**Dy (feet)**	**MSE (feet)**
Rectangular Route	1.76	0.80	2.00
Diagonal Route	1.32	1.50	2.29
Circular Route	1.22	1.51	2.22

Table 1 The best tracking errors of the target travelling in different routes.

Figures 2-4 show some of the better track results for the rectangular, diagonal, and circular routes, respectively.

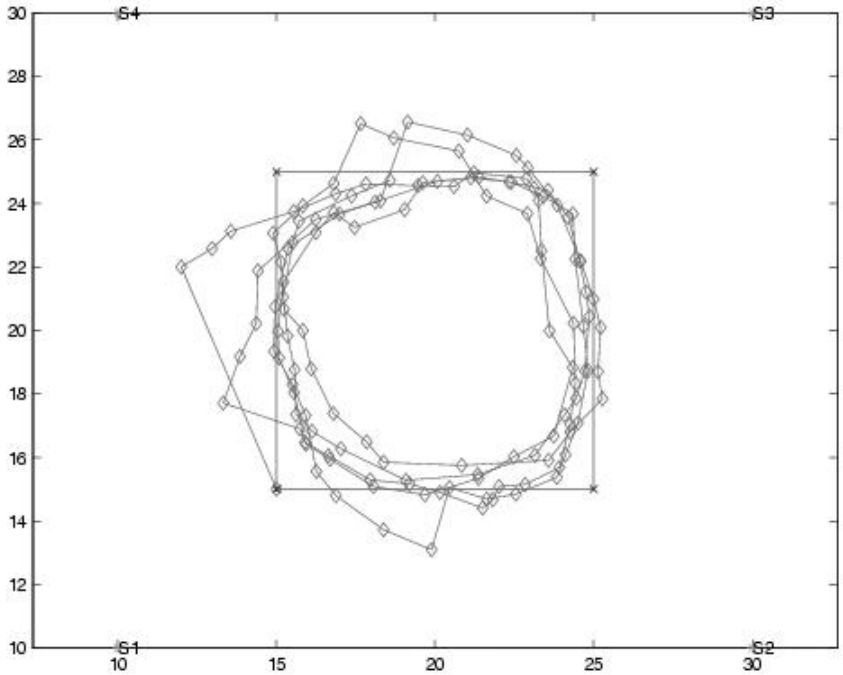

Figure 2 Rectangular route: Dx = 1.76, Dy = 0.80, MSE = 2.00. The blue track is the ground truth and the magenta points are the estimated points by the multiagent system.

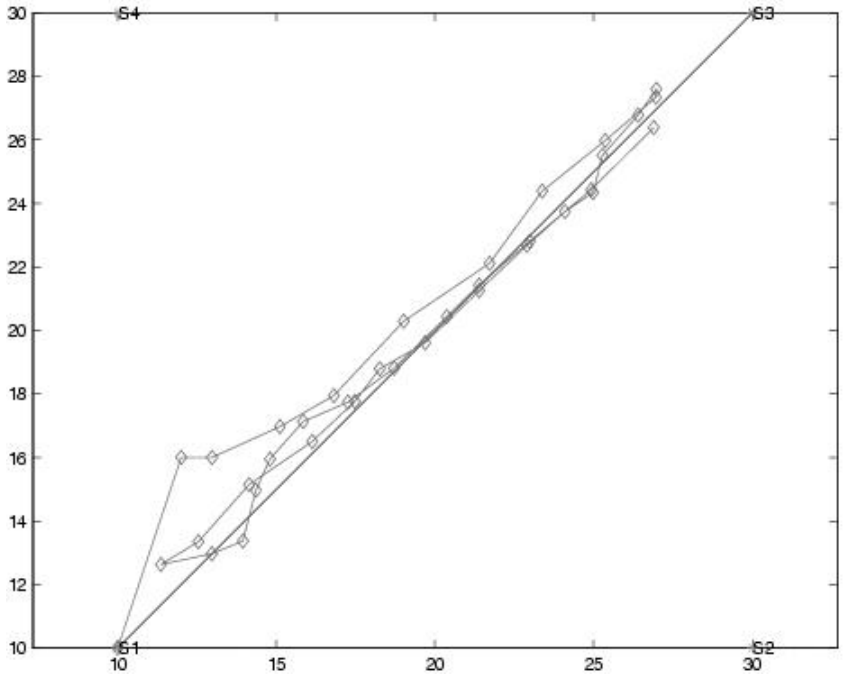

Figure 3 Diagonal route: Dx = 2.36, Dy = 1.94, and MSE = 3.12. The blue track is the ground truth and the magenta points are the estimated points by the multiagent system.

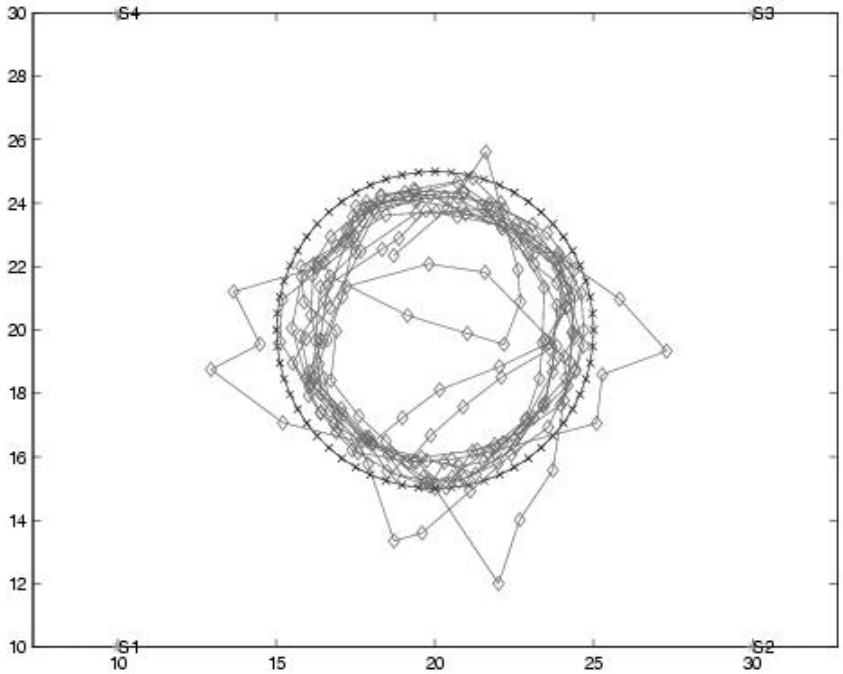

Figure 4 Circular route: Dx = 1.73, Dy = 1.62, and MSE = 2.59. Excluding the first two and the last nine bad points, we have: Dx = 1.65, Dy = 1.58, and MSE = 2.49. The blue track is the ground truth and the magenta points are the estimated points by the multiagent system.

4.2. CPU Resource Allocation

We have implemented a Real-Time Scheduling Service (RTSS) in 'C', on top of the KU Real-Time system (KURT) (Srinivasan *et al.* 1998) that adds real-time functionality to Linux. First, the RTSS provides an interface between the agents and the system timers, allowing agents to: (1) query the operating system about the current time; (2) ask the RTSS to notify them after the passage of certain length of time; and (3) ask the RTSS to ping them at fixed time intervals. This allows agents to know when to, for example, conclude a negotiation process or turn on a radar sector. Second, the agents may ask the RTSS to notify them when certain system-level events occur, such as process threads being activated, or communication messages going out or coming into the system. Third, the agents can ask the RTSS to allocate them a percentage of the CPU for each one of their threads (such as the ones controlling the radar and tracking or the ones used in negotiations) and to schedule this allocation within an interval of time. This RTSS allows an agent to monitor the progress of its own negotiations and the usage status of its allocated CPU resource.

Currently, a CPU shortage is detected whenever an agent is using 90% of its allocated CPU. When this happens, it first requests for more CPU allocation from the RTSS. If the RTSS has the CPU available, it will grant it. If the RTSS can only grant partially or grant none of the request, then the agent faces a crisis and declares a new CPU shortage event. When this occurs, it retrieves $\Psi_{r_\tau, a_i}(t)$ from the RTSS, where r is CPU allocation, and τ is CPU shortage. This initiating agent then evaluates potential utility, PU_{α_k, a_i} of each candidate and then determines

the amount of resource using $|r|_{\alpha_k} = 2 \cdot \lfloor r \rfloor_{e_j, a_i} \cdot \frac{PU_{\alpha_k, r, a_i}}{\sum_{\alpha_m \in \Lambda_{ini}(a_i, e_j)} PU_{\alpha_m, r, a_i}}$ where r is simply CPU allocation, $\lfloor r \rfloor_{e_j, a_i}$ is the additional CPU allocation that the agent wants, and the number 2 is the factor used as an insurance policy. After the resource allocation and assignment, the initiating agent uses the *Algorithm Greedy Priority-Based 1-To-1 Bounded* (Section 3.3.3) to determine $\Lambda_{approached}(a_i, e_j)$. It then negotiates with the candidates in $\Lambda_{approached}(a_i, e_j)$ to form the final coalition. If there is an agreement, the initiating agent shoulders the task of resource allocation with the RTSS. It informs the RTSS that its particular neighbor has agreed to give up a certain amount of CPU allocation to the agent. That particular neighbor obtains its updated CPU allocation from the RTSS and subsequently re-organizes its CPU distribution among its processing threads.

Currently, we are running experiments to collect data on this agent-driven CPU resource allocation to measure its effectiveness and how it cooperates with an operating-system-level scheduling service (i.e., the RTSS) to achieve a high-level job queuing of low-level tasks.

4.3. Case-Based Learning

There is another feature that we have not touched upon concerning our negotiation protocol. Our negotiation strategy is derived through case-based reasoning (CBR). So, our agents actually learn good cases of negotiations and store them in their casebases. As an agent evolves, it becomes more adaptive to its environments and learns more useful negotiation strategies. These strategies will help generate more efficient and effective negotiation behavior, which in turn improve the overall coalition formation process. Interested readers are referred to (Soh and Tsatsoulis 2001) for a detailed discussion of our case-based reasoning mechanism.

Briefly, when an event occurs, the initiating agent collects its current status and the profile of the event to build a problem space. With this problem space, the initiating agent searches the case base to retrieve the most similar case. From that case, the initiating agent obtains a parametric negotiation strategy that has been used before as a solution to a problem similar to the current one. The initiating agent then adapts the strategy according to the differences between the current problem and the old one. The new strategy is then dynamically stored at each awareness link for its associated negotiation thread to use.

After a negotiation completes, the parent agent collects the information and obtains the outcome to generate a new case with the problem space, the adapted negotiation strategy as the solution space and the negotiation outcome as the outcome space. It then computes the difference between this new case and the casebase. If the inclusion of this new case increases the diversity in either the problem or the solution space, then the case is learned. In this way, the casebase evolves to cover wide problem and solution spaces. In the end, this will help agents negotiate more confidently and successfully, ultimately resulting in a better and faster coalition formation process.

4.4. Learning Results

We ran two basic experiments: In the first one we investigated the effect of potential-utility-based evaluation in the quality of the resulting coalition, and in the second one we looked into the quality of the negotiation as a function of learning new cases of negotiation strategies. The

quality was based in the number of successful negotiations, since when the agents reach a negotiated deal to jointly track a target, the overall system utility increases. Initial results indicate that the agents form coalitions with partners who are more willing to accommodate them in negotiation, and that the cases learned are being used in future negotiations. Agents that use learning of negotiation cases have between 40% and 20% fewer failed negotiations. Agents that use the utility-driven approach (Section 3.3.2) to determine future coalition partners tend to prefer neighbors who are more conceding. Note that the utility-driven approach is a form of reinforcement learning.

In the second experiments, we investigated four versions of learning: (1) both case-based learning and reinforcement learning (CBLRL), (2) only reinforcement learning (NoCBL), (3) only case-based learning (NoRL), and (4) no learning at all (NoCBLRL). Figure 5 shows the result in terms of the success rates for negotiations and coalition formations. As can be observed from the graph, the agent design with both case-based learning and reinforcement learning outperformed others in both its negotiation success rate and coalition formation success rate. That means with learning, the agents were able to negotiate more effectively (and perhaps more efficiently as well) that led to more coalitions formed. Without either case-based learning or reinforcement learning (but not both), the negotiation success rates remained about the same but the coalition formation rate tended to deteriorate. This indicates that without one of the learning mechanisms, the agents were still able to negotiate effectively, but may be not efficiently (resulting in less processing time for the initiating agent to post-process an agreement). Without both learning mechanisms, there was significant drop in the negotiation success rate. Therefore, we conclude that our utility-driven ranking of coalition candidates improves our coalition formation process.

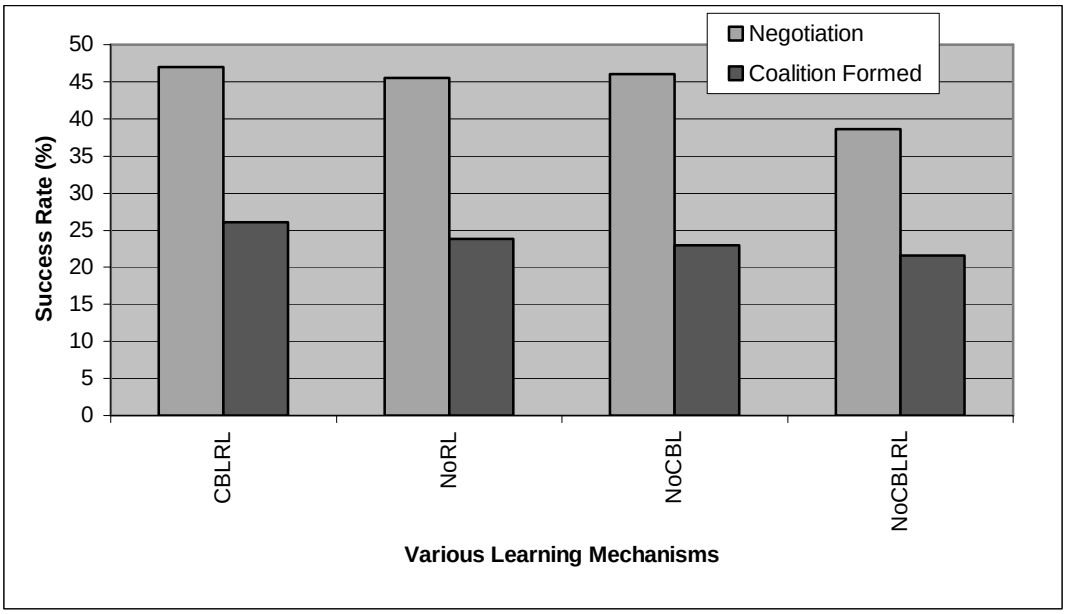

Figure 5 Success rates of negotiations and coalition formations for different learning mechanisms.

4.5. System-Related Experiments

Here we report on some preliminary experiment results for the behavior analsyis of our multiagent system, specifically for multisensor target tracking. We consider here an exemplary

run that we used to adjust our system parameters. In this run, the total number of attempts to form a coalition was 150. The total number of coalitions successfully formed (after coalition finalization) was 30, or 20%. The total number of coalitions confirmed by all three coalition members was 26, or 86.7% of all successfully formed coalitions. Finally, the total number of coalitions executed on time was 18, or 61.5% out of all successfully confirmed coalitions.

First, the percentage of successfully formed coalitions was only 20.0%. Out of the 120 failed attempts, 86 (71.7%) of them were caused by one of the coalition members outright refusing to negotiate, 17 (14.2%) were caused by the communication channels being jammed, and 17 (14.2%) were caused by busy negotiation threads. When an initiating agent initiates a negotiation request to a candidate and that candidate immediately refuses to entertain the negotiation, it can be due to (1) the responding agent does not have idle negotiation threads, or (2) the responding agent cannot project the requested task into its job queue. Thus, we expect this failure rate to decrease once we increase the number of negotiation threads allocated per agent. When an agent fails to send a message to another agent, or fails to receive an expected message, we label this as a communication "channel-jammed" problem. When an initiating agent fails to approach at least two candidates, it immediately aborts the other negotiation process that it has invoked for the same coalition. This causes the coalition to fail.

Second, the probability of a successfully formed coalition getting confirmed completely was 86.7%. For each coalition successfully formed, three confirmations were required. Out of 30 coalitions, 4 coalitions were confirmed only by two of the members. The causes were (1) the acknowledgment message sent out by the initiating agent was never received by the responding agent expecting a confirmation, and (2) the agreed task had been removed from the job queue before the confirmation arrived. The first cause happened since communication channels could be jammed. The second cause happened because of a contention for a slot in the job queue by two tasks. For example, suppose agent A receives a request from agent B to track a target starting at 8:00 a.m. Agent A responds to the request and starts a negotiation. Then later on, agent A receives a request from agent C to track a target also starting at 8:00 a.m., but using a different sensing sector (each sensor has three difference sensing sectors). Agent A checks its job queue and sees that it is free at that time and thus agrees to negotiate. Note that a task is inserted into the job queue only after the agent agrees to perform it. Now, suppose that both negotiations are successful. The negotiation between A and B ends first and then that between A and C. When the first negotiation ends, agent A adds the task requested by B to the job queue. Immediately after, when the second negotiation also ends successfully, agent A adds the second task, requested by C to the job queue, and this causes the second task to replace the first task. This is a problem with over-commitment.

Third, the probability of a confirmed coalition getting executed was 61.5%. Out of 26 coalitions confirmed, only 16 of them were executed completely. Of the 10 failures, there were two cases where none of the members executed its planned task; one case where only one of the members executed; and seven cases where only two members executed. The cause for the failure to execute was that the agreed task had been removed from the job queue before the execution took place.

The above experiments allow us to test our coalition formation model in a real-time dynamic environment and devise problem-specific heuristics to improve the coalition formation performance.

5. Related Work

A definition from the rational coalition theory outlined in (Kahan and Rapoport 1984) states that a coalition game with transferable utility in normal characteristic form is (N, v) where $N = \{1, 2, \ldots, n\}$ set of agents, and $v: 2^N \rightarrow \Re$. For each coalition that is a subset of agents $S \subseteq N$, $v(S)$ is the value of the coalition *S*, which is the total utility that the members of *S* can achieve by coordinating and acting together. However, in our problem domain, the agents do not have the information they need to compute the value of the coalition. Furthermore, our problem domain is not superadditive in which a merged coalition of any pair of sub-coalitions is better than any pair of sub-coalitions operating individually. Superadditivity does not apply when coalition formation costs (such as communication and computation costs) are considered. On the other hand, our model does not adhere to subadditive games either. In a subadditive game, the agents are best off by operating alone since any coalition is worse than its individual members in terms of utility. As previously mentioned in Section 2, in resource allocation, task distribution, or collaboration problem domains that our model is designed to address, subadditivity does not apply. Most research work in coalition formation studies characteristic function games (CFGs) (Tohmé and Sandholm 1999) in which the value of each coalition is independent of nonmembers' actions. In our model, the value of a coalition is not independent of nonmembers' actions since dynamic activities—for example, shared resource usage, conflicting goals, modifications of the environment, etc.—exerted by agents in the environment are monitored and factored into the coalition design process. However, our evaluation or scoring of a coalition does not explicitly consider nonmembers' actions and that is akin to the characteristic function game theory.

Sandholm and Lesser (1995) introduce a bounded rationality in which agents are guided by performance profiles and computation costs in their coalition formation process. In traditional coalition formation, a rational agent can solve the combinatorial problem optimally without paying a penalty for deliberation. The authors' bounded rationality model requires each agent to pay for computational resources that it uses for deliberation. Each agent knows the value (computation cost) of each potential coalition *S* upfront as shared deterministic performance profiles are known. In our model, an agent holds performance profiles (including time and CPU costs) for its execution. It devises its coalition design based on the performance profiles not for optimization but for framing the coalition—identifying neighboring agents that can help. Our model is also affected by the current status of an agent, such as the availability of negotiation threads, and the environment, such as the availability of the communication channel. These factors are included in our model when computing the expected utility of a neighbor but do not guarantee success and are definitely not deterministic and not shared.

Zlotkin and Rosenschein (1994) describe a coalition driven by task-oriented utilities. In a task-oriented domain (TOD), a coalition can coordinate by redistributing their tasks among themselves. In a subadditive TOD, the way to minimize total cost is to aggregate as many tasks as possible into one execution batch (since the cost of the union of tasks is always less than the sum of the costs). Therefore, the maximum utility that a group can derive in a subadditive TOD is the difference between the sum of stand-alone costs and the cost of the overall union of tasks. This difference is then defined to be the value of the coalition. In a superadditive TOD, on the other hand, two disjoint coalitions can derive a utility at least the sum of their separate utilities. To facilitate the coalition formation process, agents are guided with the following rules of interaction. First, assuming the coalition game is superadditive, the sum of utilities of the agents

should be equal to $v(N)$ where N is the total number of agents in the coalition. This motivates the agents to be efficient such that there should be no wasted utility when an agreement is reached. Second, there are three levels of stability (or rationality) conditions: individual, group, and coalition. Individual rationality means that that no agent would like to opt out of the full coalition; i.e., utility of each in a coalition is at least the utility value of the agent by itself (forming a group). Group rationality (or pareto-optimality) means that the group as a whole would not prefer any other payoff vector over the vector of individual utilities $\vec{u}$; i.e., $\sum_{i=1}^{N} u_i = v(N)$. Coalition rationality means that no group of agents should have an incentive to deviate from the full coalition and create a sub-coalition, i.e., for each subset of agents $S \subseteq N$, $\sum_{i \in S} u_i \geq v(S)$. Third, two agents are symmetric when they contribute the same value to all possible coalitions and should be assigned the same utility by the evaluation mechanism. In our dynamic negotiation-based model, an agent can be of two or more coalitions simultaneously as each agent is autonomous and capable of reacting to separate events. Some of these events, even though do not occur concurrently, call for responses that overlap in time. On the other hand, the agents are motivated by a global directive (to cooperate to achieve global goals while optimizing local resources) and this allows each agent to achieve individual rationality and event-dependent group and coalition rationality.

Shehory *et al.* (1997) relax some of the restrictive assumptions of theoretical coalition formation algorithms for a real-world system. In their model, each agent has a vector of real non-negative capabilities. Each capability is a property of an agent that quantifies its ability to perform a specific type of action and is associated with an evaluation function. A vector of capabilities satisfying a task is computed to obtain the benefits gained from performing the task, and the benefits are measured from the system viewpoint and that motivates the design and selection of a coalition. The model assumes that the agents are group-rational and the agent population does not change during the coalition formation. There are several differences between this model and ours. First, the authors' model assumes that all agents know about all of the tasks and the other agents. In our model, an initiating agent knows only the agents in its neighborhood and knows about partially the updated status of a selective subset of the neighbors after negotiation. Second, the details of intra-coalitional activity are not necessary for agents outside of the coalition in the authors' model. On the contrary, in our model, an agent can and does belong to multiple coalitions concurrently. When an agent belongs to a coalition, it performs a task contributing to that coalition and this execution of task is reflected in the agent's commitments, constraints, and perceptions—for example, its usage of resources is affected, it has constraints on the type of services it can provide, and it has an updated viewpoint of the world including its neighbors. These factors are significant in the agent's handling of the next coalition. In a way, therefore, intra-coalitional activities are indirectly felt among agents of different coalitions in our model. Third, there is no clock synchronization among the agents in the authors' model. Since our problem domain is time-critical and our agents negotiate with time, there is a facility in our design that provides a synchronized clock across different computing platforms. Fourth, in the authors' model, all possible coalitions are calculated in a distributed manner. The distribution of the calculations is done by having each agent approaching the members of the coalition in the list of coalitions (preliminarily derived) and committing to the calculation of the values of coalitions in which they are both members. In our model, a coalition is dynamically designed and evaluated by the initiating agent. The responding

agents in the coalition are implicitly motivated by a global directive to help and explicitly driven by their objectives to optimize their local utilities.

Shehory and Kraus (1998) further extend their work to incorporating negotiations, computational and communication costs. The initial state of a coalition consists of a set of agents. The agent then begin negotiating and gradually form the coalition. Agents that join the coalition quit the coalition-formation process and only the remaining agents continue to negotiate. The reduction in the number of agents that continue negotiating reduces the computational and communication costs. This model is similar to ours in its reduction of computational and communication costs. However, our model allows an agent to conduct 1-to-*N* coalition candidates and adjust its negotiation strategies to re-design its coalition as its commitments, constraints, and perceptions change. Hence, the resulting coalition can be very different from the initial coalition.

Tohmé and Sandholm (1999) studies coalition formation among self-interested agents that cannot make sidepayments—reward each other with payments for agreement to join some coalition, making the evaluation of a coalition solely on its utility. The authors also consider deliberation and communication costs. When explicitly modeling these costs, the authors use a greedy algorithm that stepwise maximizes the expected payoff of an action given its beliefs. Subsequently, the beliefs such as conditional probabilities of the costs and actions will be revised before the next action and this allows an agent to determine the expected payoff of a sequence of actions within a coalition, giving the agents in the member a metric to converge to. To rationalize about the utility of a coalition, the agents within the coalition assume that other non-member agents pick strategies that are worst for the coalition following the *α-assumption*. The resulting *α-core* solution is Pareto-optimal, i.e., an individual utility cannot be improved without diminishing the utility of another agent, in a perfect information scenario. The authors further show that when agents agree to a process that is in the *α-core* solution, the greedy algorithm leads to convergence of the agents' beliefs in a finite number of steps. The authors finally show that the outcome of any communication/deliberation process that leads to a stable coalition structure is Pareto-optimal for the original game that does not incorporate communication or deliberation. Conversely, any Pareto-optimal outcome can be supported by a communication/deliberation process that leads to a stable coalition structure. Our model is very similar to what the authors describe. During a negotiation between, the two agents exchange information on constraints, commitments, and perceptions, allowing each other to update their beliefs about each other. This allows a compromised deal to be reached (or where beliefs converge) if possible. However, in our model, we do not guarantee a convergence or a stable solution to our negotiations. There are external factors such as the dynamic events and noisy communication channels that may thwart the successful completion of a negotiation, rendering a negotiation outcome unpredictable. In addition, in our coalition formation model, the responsibility for organizing a coalition lies with the initiating agent. The other candidates of the coalition simply decide whether it is *rational* to join the coalition when recruited. A candidate makes its decision based on the arguments provided by the initiating agent, its current status, and its previous experience in the matter. Moreover, the initiating agent does not assume non-member agents pick strategies that are worst for its coalition. Without perfect information, each initiating agent tries to build the best coalition given time and resources. When the next event comes in, another initiating agent tries to build the best coalition it can given time and resources that the coalition has after a partial set of its members having committed to previous coalitions.

Sen and Dutta (2000) propose an order-based genetic algorithm (OBGA) as a stochastic search process to identify the optimal coalition structure. Though the OBGA has no performance guarantees (neither does our model), it is found to dominate the deterministic algorithm in a significant number of problem settings. It is also scaleable and applicable to problems where performance of a coalition depends on other coalitions in the environment. The design of OBGA has a distinct advantage over traditional coalition formation algorithms on the combinatorial problems. Traditional algorithms exhaustively and exponentially search the space of all possible coalition structures while the OBGA is linear to the number of agents in the system. A significant difference between the authors' work and our model is the scope of coalition formation. The authors' algorithm is for searching for an optimal coalition structure, which consists of all the agents in the environment grouped into one or more coalitions. Our model, however, focuses on the formation of a single coalition for a particular event while allowing multiple coalitions to be formed concurrently. Our model also allows intra-coalition issues to be factored into the coalition formation process, but does not take into account optimizing coalitions in a coalition structure concurrently.

Ketchpel (1994) presents a coalition formation method for rational agents that have different expectations of coalition values and have perfect information but lack the ability to deduce perfectly. Klusch and Shehory (1996) present an approach for cooperation and coalition formation among information agents for heterogeneous databases. They propose a special decentralized coalition formation mechanism to address the required association autonomy of these information agents. The coalition formation in this case requires finding partitions of the agents with respect to their utilities. These utilities are calculated and result from the execution of either the agents' own search tasks or tasks that they receive from other information agents. If the agents are rational, then such partitions, or coalitions, will form if and only if each member of a coalition will gain more if its joins the coalition than it could gain by itself previously. This is akin to the group rationality or pareto-optimality. Sandholm *et al.* (1999) focus on establishing a worst-case bound on the quality of the coalition structure when an agent has only time allowed to search a small fraction of the possible coalition structures. This is motivated by the fact that finding the optimal coalition structure is NP-complete. The authors also show that, if additional time remains, their anytime algorithm searches the coalition lattice further and establishes a monotonically lower tight bound. There are other theoretical works in rationality for coalition formation (Kahan and Rapoport 1984; Ketchpel 1994; Raiffa 1982; van der Linden and Verbeek 1985; Zlotkin and Rosenschein 1994).

Finally, note that our coalition formation activities differ from that presented in (Sandholm *et al.* 1999) which defines coalition formation as three interacting activities:

(1) Coalition structure generation where agents within each coalition coordinate their activities but do not coordinate between coalitions. This means partitioning the set of agents into exhaustive and *disjoint* coalitions and the partition is called a coalition structure. In our model, an agent forms a coalition from its neighborhood. Some neighbors may be part of the coalition, may be part of other coalitions, or may be simply idle. Coalitions in our model may also overlap.

(2) Solving the combinatorial optimization problem of each coalition whose objective is to maximize the utility of the coalition. This means pooling the tasks and resources of the agents in the coalition, and solving their joint problem. In our model, the initiating agent shoulders this optimization using imperfect information to increase the chance for a successful coalition.

(3) Dividing the value of the generated solution among agents. There is no such explicit value distribution in our model. Our agents are altruistic and directed to help if possible to achieve global goals, and thus do not require additional motivation such as rewards or values. However, our agents do want to manage their own resources efficiently and this motivates negotiations and the task allocation among agents.

6. Conclusions

We have described a coalition formation strategy that is dynamic and negotiation-based for a cooperative multiagent system. Rational optimality in our problem domain is infeasible because the agents do not have complete information of other agents in the neighborhood, the environment is dynamic and events change, the environment is uncertain and noisy such that communication is not always perfect, agents do not have enough time to collect enough data to rationalize optimally and finally agents have limited computational resources to support combinatorial computations. Our model has two stages: coalition initialization and coalition finalization. The goal of the initialization is to extract a set of coalition candidates from an agent's neighborhood, as a response to a detected event. These initial candidates are scored for their potential utilities and ranked. Then the initialization process allocates and assigns tasks or resources to the candidates. We have introduced *prioritized*, *bounded*, *greedy* and *worried* algorithms for 1-to-1 or many-to-1 assignments. Our initialization approach is for the agent to come up with a sub-optimal coalition that has the best chance to succeed given the incomplete information and dynamic resource profile that the agent has at the time of the formation. To avoid over-demanding on a particular negotiation, the agent uses utility-based caps. Since the surviving coalition is bound to be imperfect, the agent uses greedy algorithms to rely on a successful negotiation to propagate its mission further through a chain reaction behavior. As insurance policies, the agent also uses worried algorithms to ask for more than it needs in its negotiations. After the initialization, the initiating agent invokes negotiations to finalize the coalition.

During the coalition finalization phase, a parent agent and its negotiation threads interact to provide situated awareness so that a parent agent may generate new instructions for its negotiations to modify the coalition. In a negotiation, information pertinent to individual constraints, commitments, and perceptions is exchanged. The initiating agent monitors the progress of each of its negotiation threads and each negotiation thread is aware of the current status of its parent agent. These threads are almost-autonomous, relieving the parent agent from detailed, local decision-making of iterations of negotiations. The initiating agent oversees the negotiations, and carries out its other tasks and updates its additional instructions to the negotiation threads through dedicated awareness links. In effect, an initiating agent conducts 1-to-many negotiations when it is forming a coalition. The parent agent also makes the decisions on relaxation and termination. Relaxation allows an initiating agent to be less demanding in its requests. Termination releases the responding agent from further involvement in useless negotiations. These two awareness-based procedures allow the agent to be *reactive* in its coalition initialization since they help correct sub-optimality in the coalition at a later stage. In that way, an initiating agent can continue with a new, refined coalition. In conclusion, with this model, an agent is able to form a coalition that can give a "good-enough, soon-enough" solution and response to an event. Our model allows the agents to be autonomous and that increases the robustness of the multiagent system. It also allows the agents to self-organize into multiple,

overlapping, concurrent coalitions that can interact indirectly via actuation-perception links or internal agent-based rationalization, which in turn facilitates better coalition formation for the entire system over time. Moreover, we have provided algorithms in resource allocation and task distribution.

Finally, we have also presented an implementation that incorporates the coalition formation model and discussed some preliminary results. Our multiagent system is able to detect and respond to two types of events. One is for task distribution to address multiagent target tracking, and the other is for resource allocation to address CPU shortage problems. An agent is capable of handling both events simultaneously as parts of two separate working coalitions. Our preliminary results are promising.

7. Acknowledgements

The authors would like to thank Kelly Corn, Will Dinkel, Jim Emery, Arun Gautam, Douglas Niehaus, Pete Prasad, and Huseyin Sevay for their work on the ANTS Project at the University of Kansas. The work described in this paper is sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-99-2-0502. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), the Air Force Research Laboratory, or the U.S. Government.

8. References

Brazier, F., & Treur, J. (1996). Compositional modelling of reflective agents, in *Proceedings of the 10th Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW'96)*.

Doran, J. E., S. Franklin, N. R. Jennings, and T. J. Norman (1997). On Cooperation in Multi-agent Systems, *Knowledge Engineering Review*, **12**(3):309-314.

Durfee, E. H. and V. R. Lesser (1991). Partial Global Planning: a Coordination Framework for Distributed Hypothesis Formation, *IEEE Transactions on Systems, Man, and Cybernetics*, **21**(5):1167-1183.

Faratin, P., C. Sierra, and N. R. Jennings (1998). Negotiation Decision Functions for Autonomous Agents, *International Journal of Robotics and Autonomous Systems*, **24**(3-4):159-182.

Galliers, J. R. (1998). A strategic framework for multi-agent cooperative dialogue, in *Proceedings of the Eighth European Conference on Artificial Intelligence (ECAI'88)*, August, Munich, Germany, 415-420.

Kahan, J. P. and A Rapoport (1984). *Theories of Coalition Formation*, Lawrence Erlbaum Associates Publishers.

Ketchpel, S. (1994). Forming coalitions in the face of uncertain rewards, in *Proceedings of the National Conference on Artificial Intelligence*, July, Seattle, WA, 414-419.

Klusch, M. and O. Shehory (1996). Coalition Formation among Rational Information Agents, in *Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'96)*, Eindhoven, Netherlands, 22-25.

Kraus, S. (1997). Beliefs, Time, and Incomplete information in Multiple Encounter Negotiations among Autonomous Agents, *Annals of Mathematics and Artificial Intelligence*, **20**(1-4):111-159.

Kraus, S., K. Sycara, and A. Evenchik (1998). Reaching Agreements through Argumentation: a Logical Model and Implementation, *Artificial Intelligence*, **104**(1-2):1-69.

Lâasri, B., H. Lâasri, S. Lander, and V. Lesser (1992). A Generic Model for Intelligent Negotiating Agents, *International Journal of Intelligent and Cooperative Information Systems*, **1**(2):291-317.

Parsons, S., C. Sierra, and N. R. Jennings (1998). Agents that Reason and Negotiate by Arguing, *Journal of Logic and Computation*, **8**(3):261-292.

Raiffa, H. (1982). *The Art and Science of Negotiation*, Cambridge, MA: Harvard Univ. Press.

Sandholm, T. W., K. Larson, M. Andersson, O. Shehory, and F. Tohmé (1999). Coalition Structure Generation with Worst Case Guarantees, *Artificial Intelligence*, **111**(1-2):209-238.

Sandholm, T. W. and V. R. Lesser (1995) Coalition Formation Amongst Bounded Rational Agents, in *Proceedings of IJCAI 1995*, Montreal, Canada, 662-669.

Sen, S. and P. S. Dutta (2000). Searching for Optimal Coalition Structures, in *Proceedings of the Fourth International Conference on Multiagent Systems (ICMAS'2000)*, July 7-12, Boston, MA, 286-292.

Shehory, O. and S. Kraus (1996). Formation of Overlapping Coalitions for Precedence-Ordered Task-Execution Among Autonomous Agents, in Proceedings of International Conference on Multiagent Systems (ICMAS'96), Kyoto, Japan, 330-337.

Shehory, O. and S. Kraus (1998). Methods for Task Allocation via Agent Coalition Formation, *Artificial Intelligence*, **101**(1-2):165-200.

Shehory, O. M., K. Sycara, and S. Jha (1997). Multi-agent Coordination through Coalition Formation, in *Proceedings of the 4th International Workshop on Agent Theories, Architectures, and Languages (ATAL'97)*, Providence, RI.

Soh, L.-K. and C. Tsatsoulis (2001). Reflective Negotiating Agents for Real-Time Multisensor Target Tracking, in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'01)*, August 4-10, Seattle, WA.

Srinivasan, B., S. Pather, R. Hill, F. Ansari, and D. Niehaus (1998). A Firm Real-Time System Implementation Using Commercial Off-The Shelf Hardware and Free Software, in: *Proceedings of RTAS-98*, June, Denver, CO, 112-119.

Tohmé, F. and T. Sandholm (1999). Coalition Formation Processes with Belief Revision among Bounded-Rational Self-Interested Agents, *Journal of Logic and Computation*, **9**(6):793-815.

van der Linden, W. J. and A. Verbeek (1985). Coalition formation: a game-theoretic approach, in H. A. M. Wilke (Ed.) *Coalition Formation*, volume 24 of Advances in Psychology, North Holland.

Wooldridge, M. and N. Jennings (1995). Intelligent Agents: Theory and Practice, *Knowledge Engineering Review*, **10**(2):114-152.

Zlotkin, G. and J. S. Rosenschein (1994). Coalition, cryptography and stability: mechanisms for coalition formation in task oriented domains, in *Proceedings of the National Conference on Artificial Intelligence*, July, Seattle, WA, 432-437.

Zlotkin, G. and J. S. Rosenschein (1996). Compromise in Negotiation: Exploiting Worth Functions over States, *Artificial Intelligence*, **84**(1-2):151-176.