Workshop on Teamwork and Coalition Formation (held during the *1st Int. Conf. On Autonomous Agents and Multiagent Systems*, 2002.

Allocation Algorithms in Dynamic Negotiation-Based Coalition Formation

Leen-Kiat Soh
Computer Science and Engineering
University of Nebraska
115 Ferguson Hall
Lincoln, NE
(402) 472-6738
lksoh@cse.unl.edu

Costas Tsatsoulis
Dept. of Electrical Engineering and Computer Science
Information & Telecommunication Technology Center
University of Kansas
Lawrence, KS 66044
(785) 864-7749
tsatsoul@ittc.ukans.edu

ABSTRACT

In this paper, we present a set of allocation algorithms for a dynamic, negotiation-based coalition formation model. The model is 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. Thus, the allocation algorithms have to take these constraints into account.

Keywords

Allocation algorithms, dynamic coalition formation, negotiation, incomplete information

1. INTRODUCTION

In this paper we present a set of allocation algorithms for a dynamic, negotiation-based coalition formation model. The goals for the design of our allocation algorithms include improving (1) the chance of a coalition formation on time, (2) the chance of a coalition formation with incomplete information, (3) the robustness of a coalition formation, and (4) the flexibility of a coalition formation changing dynamically. Our coalition formation model deals with cooperative agents in a dynamic and uncertain world with incomplete information and time constraints. A coalition is a group of agents that collaborate to perform a coordinated set of tasks that a single agent cannot accomplish by itself, and 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. Ideally, an 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 or 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.

Our model applies to an environment where each agent has incomplete information about its world. Incomplete information may be due to polling and updating costs, constrained resources, and 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. As a result, when an agent needs to rationalize based on its profile of other agents, it can only do so based on partial or outdated information. 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 and a set of allocation algorithms that increase the chance of a successful coalition formation.

An optimal rationalization for coalition formation may not be possible due to noise and uncertainty in the environment, or 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 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 that the environment has changed making the action moot.

In our approach 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 propose using negotiations to refine a coalition. The motivation of a negotiation is for the initiating agent to persuade a potential coalition partner to agree to help. 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*.

Unlike traditional coalition formation techniques that assume that 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 allocation algorithms that are *greedy* or *worried*.

In the following, we first present briefly our dynamic, negotiation-based coalition formation model. Then we describe the allocation algorithms in Section 3. Subsequently, we describe our current work using the model and the allocation algorithms in a multiagent sensor tracking problem domain and discuss some experimental results. Finally, we conclude.

2. COALITION FORMATION MODEL

Briefly, our coalition formation model works as follows. In a multiagent system, when one of the agents initiates the coalition formation process in hope of organizing a group of cooperative agents to perform tasks in response to some event, this initiating agent (also known as the "computing agent" [1]) 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. First, during the *coalition initialization*, the initiating agent creates 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 [2]. 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.

2.1. Neighborhood

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 $a_j \in \eta_{a_i}$.

2.2. 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.

2.3. 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.

2.4. 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 it 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. 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.

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 $ability_{a_i}(\alpha_k, e_j, t)$[1]:

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

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 $F_\tau = \{f_1, f_2, \cdots, f_N\}$.

[1] 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. When we talk about multiple coalitions, we will use PU_{α_k,a_i,e_j}.

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

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

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. The details of handling multiple coalitions are in [4].

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. Note also 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 [3].

3. ALLOCATION ALGORITHMS

In this section, we propose and examine several allocation algorithms: priority-based, flexibility-bounded, greedy, and worried. The goals for the design of our allocation algorithms include improving the chance of a coalition formation on time, the chance of a coalition formation with incomplete information, the robustness of a coalition formation, and the flexibility of a coalition formation changing dynamically.

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*.

3.1. Priority-Based

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 function, *f*. In this case, the initiating agent simply negotiates with each candidate to perform their respective tasks—i.e., executing *f* in 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_{α_k,f_n,a_i} values for $n = 1 \dots N$. 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.

3.2. Flexibility-Bounded

The number of coalition members that an initiating agent can approach is bounded by its available resources. 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 some negotiation thread of another agent. Then, the number of coalition members to be approached is determined by the number of negotiation threads that are currently available, and 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. 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 reorganize $assign_{a_i}(e_j,t)$, (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 reorganized, the task-candidate pair that has been absorbed is removed from the list, and the new task-candidate pair has become $\rho_i = \left\langle \alpha_{\rho_i}, \left\{ f_{\rho_i}, f_{\rho_j} \right\} \right\rangle$. 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. Absorption is feasible only if the two tasks do not compete for the same resources 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.

3.3. Imperfect Coalition and Greedy Algorithms

As already mentioned, when an initiating agent has more negotiations to perform than it has available resources to conduct negotiations with its coalition members, it either quits or continues with as many negotiations as possible to recruit as many coalition members as possible. 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.

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:

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

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.

3.4. 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 $\left\lceil \Lambda_{approached}(a_i, e_j) \right\rceil$.

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).

3.5. 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 $\left\lceil f_\rho \right\rceil_{\alpha_k}$, then $\left\lceil f_\rho \right\rceil_{\alpha_k} \propto rel_{past,a_i}(\alpha_k, t)$, $\left\lceil f_\rho \right\rceil_{\alpha_k} \propto rel_{now,a_i}(\alpha_k, t)$, and $\left\lceil f_\rho \right\rceil_{\alpha_k} \propto 1/ability_{a_i}(\alpha_k, e_j, t)$. And these caps can be inserted into all the algorithms above to prevent too many assignments to a single agent

4. IMPLEMENTATION AND RESULTS

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, controlled by an agent, 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 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.

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: (1) a core main thread that does the decision making, manages the tasks, performs coalition formation, and oversees the negotiations, (2) 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, (3) 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, and (4) *B* negotiation threads. Each negotiation thread is dormant until activated. When it is activated, it downloads pertinent information form the parent agent and proceeds with its negotiation.

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, 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. 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 $\left|\Lambda_{approached}(a_i, e_j)\right|$. 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). Table 1 shows the best mean square error (MSE) in feet of our tracking in a noiseless environment.

Error	**Dx (ft)**	**Dy (ft)**	**MSE (ft)**
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 in different routes.

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) [5] 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 $|r|_{\alpha_k} = 2 \cdot \lfloor r \rfloor_{e_j, a_i} \cdot \dfrac{PU_{\alpha_k, r, a_i}}{\sum_{\alpha_m \in \Lambda_{ini}(a_i, e_j)} PU_{\alpha_m, r, a_i}}$ 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 greedy algorithm (Section 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

5. CONCLUSIONS

We have described a set of allocation algorithms for a coalition formation model that is dynamic and negotiation-based in 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. We have implemented some of the algorithms and preliminary results are promising. As our ongoing and future work, we plan to install all algorithms into our systems and study the agents' behavior in different settings and measure their efficiency and effectiveness in coalition formation.

6. ACKNOWLEDGEMENTS

The work described in this paper is sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL), 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 DARPA, the AFRL, or the U.S. Government.

7. REFERENCES

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

[2] Soh, L.-K. and Tsatsoulis, C. 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.

[3] Soh, L.-K. and Tsatsoulis, C. 2002. Learning to Form Negotiation Coalitions in a Multiagent System, *Working Notes of AAAI Spring Symposium on Collaborative Learning Agents*, Stanford, CA, March 25-27, pp. 106-112.

[4] Soh, L.-K. and Tsatsoulis, C. 2002. Time-Bounded, Negotiated Coalition Formation in Multiagent Systems, in preparation.

[5] Srinivasan, B., Pather, S., Hill, R., Ansari, F., and Niehaus, D. 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.