Organization-Based Coalition Formation

Sherief Abdallah and Victor Lesser
University of Massachusetts, Amherst
MAS Laboratory
{shario,lesser}@cs.umass.edu
UMASS Computer Science Technical Report 2004-04

February 8, 2004

Abstract

The coalition formation problem has received a considerable amount of attention in recent years. In this work we present a novel distributed algorithm that returns a solution in polynomial time and the quality of the returned solution increases as agents gain more experience. Our solution utilizes an underlying organization to guide the coalition formation process. We use reinforcement learning techniques to optimize decisions made locally by agents in the organization. Experimental results are presented, showing the potential of our approach.

1 Introduction

Agents can benefit by cooperating to solve a common problem [2, 10]. For example, several robots may cooperate to move a heavy object, sweep a specific area in short time, etc. In general, this cooperation is achieved by coordination and negotiation among all agents. However, as the number of agents increases, having all agents involved in this detailed coordination/negotiation process will limit the scalability of the system. It is better to first *form a coalition* of agents that has enough resources to undertake the common problem. Then only the agents in this coalition coordinate and negotiate among themselves.

This situation is common in domains where a task requires more than one agent and there are more than one task competing for resources. Computational grids and distributed sensor networks are examples of such domains. In computational grids a large number of computing systems are connected via a high-speed network. The goal of the grid is to meet the demands of new applications (tasks) that require large amounts of resources and reasonable responsiveness. Such requirements cannot be met by an individual computing system. Only subset of the available computing systems (aka a coalition) has enough resources to accomplish an incoming task.

The work in [7] defined the coalition formation problem as follows (a formal definition is given in Section 2). The input is a set of agents, each controlling some amount of resources, and a set of tasks, each requiring some amount of resources and each worth some utility. The solution assigns a coalition of agents to each task, such that each task's requirements are satisfied and total utility is maximized.

In this paper we propose a novel approach for solving the coalition formation problem approximately using an *underlying organization* to guide the coalition formation process. The intuition here is to exploit whatever knowledge is known a priori in order to make the coalition formation process more efficient. For instance, in many domains, agents' capabilities remain the same throughout the lifetime of the system. Additionally, incoming tasks may follow some statistical pattern. Can we *organize* agents to exploit this knowledge (of their capabilities and task arrival patterns) to make the search for *future* coalitions more efficient? If so, will all organizations yield the same coalition formation performance, or do some organizations perform better than others? In the remainder of this paper we try to provide answers to these questions. The main contributions of this work are:

- an organization-based distributed algorithm for approximately solving the coalition formation problem
- the use of reinforcement learning to optimize the local allocation decisions made by agents in the underlying organization

The paper is organized as follows. In Section 2 we define the problem formally, laying out the framework we use throughout the paper. In Section 3 we present our approach to solving the problem. Section 4 describes our experimental results. We compare our approach to similar work in Section 5. We conclude and discuss future work in Section 6.

2 Problem definition

Let $T = \{T_1, T_2, ..., T_q\}$ be the set of tasks over which we are evaluating the system. Each task T_i is defined by the tuple $\langle u_i, rr_{i,1}, rr_{i,2}, ..., rr_{i,m}\rangle$, where u_i is the utility gained if task T_i is accomplished; and $rr_{i,k}$ is the amount of resource k required by task T_i. Let $I = \{I_1, I_2, ..., I_n\}$ be the set of individual agents in the system. Each agent I_i is defined by the tuple $\langle cr_{i,1}, cr_{i,2}, ..., cr_{i,m}\rangle$, where $cr_{i,k}$ is the amount of resource k controlled by agent I_i.

The coalition formation problem is finding a subset of tasks $S \subseteq T$ that maximizes utility while satisfying the coalition constraints, i.e.:

- $\sum_{i|T_i \in S} u_i$ is maximized
- there exists a set of coalitions $C = \{C_1, ..., C_{|S|}\}$, where $C_i \subseteq I$ is the coalition assigned to task T_i, such that $\forall T_i \in S, \forall k : \sum_{I_j \in C_i} cr_{j,k} \geq rr_{i,k}$, and $\forall i \neq j : C_i \cap C_j = \emptyset$

In other words, each task is assigned a coalition capable of accomplishing it and any agent can join at most one coalition. This means if the resources controlled (collectively) by a coalition exceed the amount of resources required by the assigned task, the excess resources are wasted.[1]

[1]Having more than one type of resource means that there will be trade-offs, where decreasing the excess

2.1 Complexity

In this section we prove that the Coalition Formation Problem (CFP), as we formulated it, is NP-hard. We do so by reducing the multidimensional knapsack problem, which is known to be NP-hard, to CFP.

2.1.1 The Multi-dimensional Knapsack Problem

The input of this problem consists of a set of constraints $C = \{c_1, c_2, ..., c_m\}$ and a set of objects $O = \{o_1, o_2, ..., o_q\}$, where each object is defined by the tuple $o_i =< u_i, w_{i,1}, w_{i,2}, ..., w_{i,m} >$, where u_i is its value and $w_{i,j}$ is its weight for dimension j. The goal is to find a subset of objects $S \subset O$, s.t. $\sum_{o_i \in S} u_i$ is maximized, while $\forall c_j \in C, \sum_{o_i \in S} w_{i,j} \leq c_j$

Theorem 1 *Coalition Formation Problem, CFP, is NP-hard*

Proof: This is proved by reducing an MDKP instance to a CFP instance. This is done as follows. The decision version of the MDKP problem is: given a set of objects O and a set of constraints C, is there a valid subset of objects that satisfy the constraints and has total utility of k or more?

For each object $o_i =< u_i, w_{i,1}, ..., w_{i,m} >$ in MDKP, we define an agent $a_i =< w_{i,1}, ..., w_{i,m} >$ and a task $T_i =< u_i, w_{i,1}, ..., w_{i,m} >$. We also add task $T' =< U, W_1, ..., W_m >$, where $U = \sum_{o_i \in O} u_i$ and $W_j = (\sum_{o_i \in O} w_{i,j}) - C_j$ (this amount can be viewed as the gap between the demand of a resource and its supply). And the CFP decision problem then becomes: given the set of tasks T and the set of agents A, is there a solution that results in $U + k$ utility or more? Note that T' encodes the constraints of the MDKP instance such that the coalition assigned ti this task corresponds to the set of objects left **outside** the knapsack.

■

2.2 Assumptions and Limitations

- Cooperation: Agents are cooperative.
- Closed system: No agents enter or exit the system.
- Stable environment: Incoming tasks follow a statistical pattern. Also the resources controlled by agents are constant over time.
- Non-consumable resources: A task does not permanently consume a resource, but temporarily decreases the available amount of it. Once a task finishes, it releases the resources it acquired previously. Examples of non-consumable resources are network bandwidth and processing power.
- Disjoint coalitions: In future work we plan to tackle the generic case, where an agent can multiplex its resources over more than one task. Such multiplexing may incur an overhead cost, which we also need to take into account.
- Episodes: We assume time is divided into episodes and that each task is completed within an episode. At the beginning of each episode, a set of tasks arrives

of one resource type may increase the excess of another resource type.

to the system that need to be assigned coalitions for this episode. We made this assumption to avoid adding scheduling to our problem, which is a very hard problem by itself. In future we plan to integrate scheduling in our framework.

- Linear resources: The amount of resources controlled by a coalition is the summation of the resources controlled by each member of the coalition. Many resources either follow this assumption or can be considered approximately linear (e.g., processing power). We plan to tackle other types of resources/capabilities (e.g., speed) in the future.

3 Proposed Solution

Because the coalition formation problem is NP-hard, an optimal algorithm will need exponential time in the worst case (unless NP = P). We need an approximation algorithm that can exploit information about the problem. If the environment (in terms of incoming task classes and patterns) does not follow any statistical model, and agents continually and rapidly enter and exit the system, there is little information to be exploited. Luckily, in many real applications the environment does follow a model, and the system can be assumed closed.

In such cases, it is intuitive to take advantage of this stability and *organize* the agents in order to guide the search for future coalitions. A possible organization may consist of a centralized manager keeping track of which agents have which resources. When a task arrives, this centralized manager only contacts those agents who have the necessary resources to undertake the incoming task. This approach is not scalable and is inefficient in domains that are distributed by nature (e.g., distributed sensor networks). We chose to organize agents in a *hierarchy*, which is both distributed and scalable.

Figure 1 shows a sample hierarchical organization.[2] An *individual* (the leaves in Figure 1) represents the resources controlled by a single agent. A *manager* (shown as a circle in Figure 1) is a computational role, which can be executed on any individual agent, or on dedicated computing systems. A manager *represents* agents beneath it when it comes to interaction with other parts of the organization.

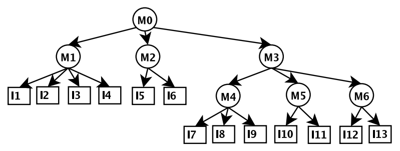


Figure 1: An Organization Hierarchy

Each manager M has a set of children, $children(M)$, which is the set of nodes directly linked below it. So for instance, in the organization shown in Figure 1, $children(M6) = \{I12, I13\}$, while $children(M3) = \{M4, M5, M6\}$. Conversely, each child C has a

[2]Note that the example in Figure 1 shows a strict tree organization. In general, an organization may be represented by a directed acyclic graph, where the same agent may have more than one manager.

set of managers $managers(C)$. For example, $managers(M4) = \{M3\}$. For completeness, children of an individual are the empty set, and so are the managers of a root node.

Each agent A (either a manager or an individual) controls, either directly or indirectly, a set of individuals, $cluster(A)$ (i.e., the leaves reachable from agent A). In the example above, $cluster(M6) = \{I12, I13\}$, $cluster(M3) = \{I7, I8, I9, I10, I11, I12, I13\}$, and $cluster(I6) = \{I6\}$.

Also for each agent A, we define $organization(A)$ to be the set of all agents reachable from A. In the above example, $organization(M3) = \{M3, M4, M5, M6, I7, I8, I9, I10, I11, I12, I13\}$.

3.1 Local Decision

Algorithm 1 describes the decision process made by each manager in the organization. The algorithm is executed every time a task T is received by a manager M (either from the environment or from another agent). LOC is the list of coalitions allocated for previous tasks. The algorithm works as follows. M evaluates its current state s (Section 3.2.1). M then selects an action a based on its policy (Section 3.2.3). This action a can either be to stop forming the coalition or to select a child $M_i \in children(M)$. If a child is selected, a subtask T_i of T is dynamically created based on M_i's state (Section 3.2.2). M then asks M_i to form a subcoalition capable of accomplishing T_i. (The notion $M_i.allocateCoalition(T_i)$ means that the function $allocateCoalition$ is called remotely on agent M_i). M_i forms a subcoalition C_{T_i} and sends a commitment back to M. M updates C_T and learns about this action. M updates its state, including the amount of resources to be allocated (UR) and the corresponding utility to be gained (uu).

M selects the next best action and the process continues as long as all the following conditions hold (step 3): T requires more resources than currently allocated, M still controls some unallocated resources that are required by T, and the stop action has not been selected. At the end M adds the formed coalition C_T to its list of commitments LOC and returns C_T.

Note that manager M executes Algorithm 1 if and only if $organization(M)$ has enough resources to accomplish T. Otherwise, M passes T up the organization hierarchy until it reaches a capable manager (T is rejected if even the root manager does not have enough resources). Also to simplify handling of multiple tasks, we do not allow coalition formation of a task to be interrupted. This means that if a new task T_{new} arrives at manager M while M is still forming a coalition for an older task T_{old}, then M will finish forming the coalition for T_{old} before considering T_{new}.

3.2 Example

Figure 2 shows how a group of agents, organized in a hierarchy, can cooperate to form a coalition. A task $T = \langle u = 100, rr_1 = 50, rr_2 = 150\rangle$ is discovered by agent $M6$. Knowing that $organization(M6)$ does not have enough resources to accomplish T, $M6$ sends task T to its manager $M3$. Since $organization(M3)$ has enough resources to achieve T, $M3$ uses its local policy to chose the best child to contribute in achieving T, which is $M5$. $M3$ decomposes T into subtask $T_5 = \langle u_5 = 50, rr_{5,1} = 0, rr_{5,2} =$

Algorithm 1 allocateCoalition(T)

```
INPUT: task $T = \langle u, rr_1, ..., rr_m \rangle$
OUTPUT: coalition $C_T = \{I_1, ..., I_{|C_T|}\}$
1:  let $C_T = \{\}$, $uu \leftarrow u$, $UR \leftarrow \langle rr_1, ..., rr_m \rangle$, $stop \leftarrow$ false, $AR \leftarrow$
    the amount of available resources controlled by $M = availableResources()$
    $= totalResources(M) - \sum_{C \in LOC} totalResources(C)$
2:  $s \leftarrow$ encodeState( $uu, UR$)
3:  **while** $UR > \overline{0}$ AND $UR.AR > 0$ AND $stop =$ false **do**
4:    $a \leftarrow$ selectAction( s )
5:    **if** $a$ is the **stop** action **then**
6:      $stop \leftarrow$ true
7:    **else**
8:      let $M_i$ be the child corresponding to $a$.
9:      $T_i \leftarrow$ decomposeTask( $T$ , $M_i$ )
10:     $C_{T_i} \leftarrow M_i$.allocateCoalition( $T_i$ )
11:     $C_T \leftarrow C_T \cup C_{T_i}$
12:     $UR \leftarrow UR - totalResources(C_{T_i})$, $uu \leftarrow u_{T_i}$, and $AR \leftarrow AR -$
        $totalResources(C_{T_i})$
13:     $r \leftarrow$ time and communication costs of forming $C_{T_i}$
14:     **if** $UR = \overline{0}$ /* $T$ does not need more resources */ **then**
15:       $r \leftarrow r + u$
16:     **end if**
17:     $s' \leftarrow$ encodeState( $uu, UR$) /* the next state */
18:     learn( $s, a, r, s'$ )
19:     $s \leftarrow s'$
20:   **end if**
21: **end while**
22: $LOC \leftarrow LOC \cup C_T$ /* to exclude agents in $C_T$ from next allocations */
23: return $C_T$
```

$100\rangle$, and asks $M5$ to allocate a coalition for it. $M5$ returns a committed coalition $C_{T_5} = \{I10, I11\}$. The process continues until the whole task T is allocated. Finally, $M3$ integrates all subcoalitions into C_T and sends it back to $M6$.

3.2.1 State Abstraction

For a manager M, the function *encodeState* collects the abstract states of each child $M_i \in children(M)$ and encodes this information along with the vector of resources to be allocated, UR, and the utility to be gained, uu, to produce the current state of $organization(M)$. (This encoding is then fed to neural nets to get action values, as we discuss in Section 3.2.3.)

Since managers control exponentially more individuals as we ascend in the organization, abstraction of state information is necessary to achieve scalability (otherwise we are effectively centralizing the problem). In our solution, each manager M abstracts the state of its organization, $organization(M)$. The price of this abstraction is loss of information (a manager higher in the hierarchy "sees" fewer details about its organization). This leads to uncertainty in the manager state, and hence makes the local decision process more difficult to optimize.

Additionally, due to the large state space, we use a factored state. That is, a state is defined by a set of features. To abstract a feature at a manager M, we need to define it recursively in terms of features abstracted at M's children. For example, we defined the feature vector $totalResources(M) = \langle tr_1, ..., tr_m \rangle$ as the total amount of resources controlled by manager M (where m is the number of different resource types). It can be defined recursively as follows: $totalResources(M) = \sum_{c \in children(M)} totalResources(c)$. That is, the total resources controlled by a manager is the sum of the total resources controlled by its children. For an individual I_i, $totalResources(I_i) = I_i$.

Some features cannot be abstracted directly, but can still be inferred from other abstracted features. For example, $averageResources(M)$ is a feature vector of the average amount of resources controlled by any individual in $organization(M)$. To compute this feature, we need another abstract feature: the total number of individuals in an organization, $size(M) = \sum_{c \in children(M)} size(c)$. Then we have $averageReources(M) = totalResources(M)/size(M)$.

The above features are assumed constant throughout the system lifetime. For each constant feature we define a corresponding dynamic feature, preceded by $avail$, to indicate the current value of the feature. For example, the number of individuals not allocated to tasks $= availSize(M) = size(M) - \sum_{C \in LOC(M)} size(C)$, and their aggregated resources $= availTotalResources(M) = totalResources(M) - \sum_{C \in LOC(M)} totalResources(C)$.

3.2.2 Task Decomposition

When a manager M selects a child M_i to ask for contribution regarding task T, M decomposes T heuristically to T_i as shown in Algorithm 2.

As we described in Section 3.2.1, a manager M only sees abstract features of its child M_i. Using this information, M needs to find T_i that is more suitable to $Organization(M_i)$. The heuristic we use is to try to ask each child a multiple, α, of the average available resources it controls; i.e., $\alpha \times \frac{availTotalResources(M_i)}{availSize(M_i)}$. We want to chose α such that the *expected* excess of resources is minimized.[3]

The intuition behind the heuristic is as follows. If all individuals controlled by M_i are identical, the heuristic is the only choice to avoid wasting resources. As individuals become more diverse, the multiple of average available resources remain the most likely to succeed without wasting any resources. Because agents can not participate in more than one coalition, the minimum of the ratio l_j over all resource types is selected and used for all other resource types. Also to ensure progress, α is at least 1. Finally, the utility of the decomposed task is proportional to the total of the decomposed resources.

[3]When M decomposes T into T_i, it does not know what coalition M_i would return, which makes it difficult to minimize the wasted excess of resources.

For example, let $T = \langle u = 100, rr_1 = 50, rr_2 = 150\rangle$, $availTotalResources(M_4) = \langle ar_{4,1} = 100, ar_{4,2} = 100\rangle$, and $availSize(M_4) = 10$. Using the algorithm below we get $\alpha = 5$ and hence $T_4 = \langle u_4 = 50, rr_{4,1} = 50, rr_{4,2} = 50\rangle$. Note that asking M_4 for as much as possible will result in wasted resources. For example, the decomposed task $T'_4 = \langle u'_4 = 50, rr'_{4,1} = 50, rr'_{4,2} = 100\rangle$ can only be satisfied if all individuals controlled by M_4 are allocated, resulting in 50 units of resource type 1 being wasted.

Finally, because our decomposition is not optimal, [4] we allow each manager to select the same child more than once to fine tune the decomposition at the expense of more communication and time cost.

Algorithm 2 decomposeTask(T, M_i)

```
  INPUT: task $T = \langle u, rr_1, ..., rr_m\rangle$ AND manager $M_i$
  OUTPUT: task $T_i = \langle u_i, rr_{i,1}, ..., rr_{i,m}\rangle$
1: $AR_i \leftarrow availTotalResources(M_i) = \langle ar_{i,1}, ..., ar_{i,m}\rangle$
2: $z_i \leftarrow availSize(M_i)$
3: $\forall j : l_j \leftarrow \lfloor z_i \times \frac{rr_j}{ar_j} \rfloor$
4: $\alpha \leftarrow min_j(l_j)$
5: $\alpha \leftarrow max(\alpha, 1)$
6: $rr_{i,j} \leftarrow min(\alpha \times \frac{ar_{i,j}}{z_i}, rr_j)$
7: $u_i \leftarrow u \times \frac{\sum_j rr_{i,j}}{\sum_j rr_j}$
8: return $T_i$
```

3.2.3 Learning

A key factor in the performance of our system is how a manager selects its actions (function *selectAction* in Algorithm 1). In particular, in what order should a manager ask each child for its contribution?[5] We considered three possible policies: random, greedy, and learning. The random policy just picks a child at random. The greedy policy selects the child M_i with the highest preference value $p_i = \sum_{k=1}^{m} min(cr_{i,k}, rr_k)$, which measures how much resources M_i can contribute to the incoming task. For example, let the incoming task $T = \langle u = 100, rr_1 = 50, rr_2 = 150\rangle$ and let manager M has two possible children M_1 and M_2 where $availTotalResources(M_1) = \langle cr_{1,1} = 200, cr_{1,2} = 0\rangle$ and $availTotalResources(M_2) = \langle cr_{2,1} = 0, cr_{2,2} = 200\rangle$. Then $p_1 = 50$ and $p_2 = 150$, hence M will select M_2.

In the learning approach, we used the Q-learning algorithm [9] with neural nets to approximate action values. Unlike value or policy iteration, Q-learning is a model-free algorithm that does not require an environment model. Q-learning also learns in an incremental manner; as an agent gains more experience, its performance improves.

[4] In the previous example, if the whole organization only has 150 units of resource type 2 available, then the decomposed task T'_4 may be better than T_4.

[5] A more sophisticated decision process would consider parallelism. Here we focus on strictly serialized orderings only.

This is important in domains containing huge number of states, many of which will not be visited.

We used a decaying exploration rate to select actions so that agents explore less as they gain more experience. We also used a separate neural net for each action. This uses more memory space, but provides better approximation.

In reinforcement learning, rewards determine what an agent learns. From Algorithm 1, intermediate rewards are small negative rewards to reflect the communication and the processing costs of each additional step spent forming the coalition. Once a manager M successfully allocates a coalition to task T, it gains a reward equal to T's utility.[6] Note that even if T is a subtask of another task T', the rewards received by M are independent of whether the coalition formation for T' will succeed or not. This *recursive optimality* speeds up learning, while not affecting the quality of the formed coalitions.

We explored several techniques to speed up learning further. One technique involved minimizing the input fed to each neural net. The key observation is that the value of choosing a child M_i depends mainly on M_i's state, and to a lesser extent on the other children's states. We also tried using eligibility tracing, but the learning algorithm often diverged so this approach was dropped.

3.2.4 Organization Structure

If we view the underlying organization as a search tree, our distributed algorithm searches the same search tree several times for each task and for each episode. Each time, the search has a different start state (where and when the task is discovered) and different goal state (the set of individuals — leaves — that form the coalition.)

To optimize performance, not only do we need to find a good *search mechanism*, but we also need to find an *organization* that for a specific environment model and agent population yields the best performance. In other words, we are modifying the search tree so that the search mechanism can perform better. The closest analogue in classical AI is the use of macro operators, which adds edges to the search tree to speedup the search. In our case we have more flexibility, as we can modify the search tree in whatever way we see appropriate. In our experiments we verify this by testing different organization structures of the same agent population and same tasks distribution, as we describe in Section 4.

4 Experiments and Results

4.1 Setup

In our experiments, we wanted to know if using an underlying organization improved the system's performance. To do so, we compared our approach to centralized (a single manager controlling all individuals) random policy (CRP) and to centralized greedy

[6]We can implicitly indicate our preferences by modifying the reward function. For example, in [7] the author prefers coalitions of smaller size. This can be achieved by adjusting the reward function accordingly (e.g., dividing the utility gained by the size of the coalition formed).

policy (CGP). In CRP, an agent is picked at random and added to the coalition. In CGP, the agent that may contribute the most resources to the coalition is selected.

We also investigated the effect of learning in an organization by comparing three local policies: distributed learned policy (DLP), distributed random policy (DRP), and distributed greedy policy (DGP). Finally to measure the effect of the organization structure on system performance, we collected results using different organizations, all constructed from the same population of individual agents. In the system we tested, agents control two types of resources, and the fall into 6 types of agents:

Type A controls $\langle cr_{A,1} = 2, cr_{A,2} = 2\rangle$ resources
Type B controls $\langle cr_{B,1} = 10, cr_{B,2} = 10\rangle$ resources
Type C controls $\langle cr_{C,1} = 0, cr_{C,2} = 30\rangle$ resources
Type D controls $\langle cr_{D,1} = 1, cr_{D,2} = 10\rangle$ resources
Type E controls $\langle cr_{E,1} = 20, cr_{E,2} = 2\rangle$ resources
Type F controls $\langle cr_{F,1} = 8, cr_{F,2} = 0\rangle$ resources

In these classes, we tried to represent different specializations among agents. We studied four different organization structures shown in Figure 3. Organization H is homogeneous. Agents of each type are clustered together, then similar types (e.g., A and B) are clustered together. Organization SE is semi-homogeneous. Each couple of agents of similar types are clustered together, then similar clusters are clustered together. Organization SH is similar to H, but one organization level is omitted. Finally, organization RS has the same "structure" of SH, but individual agents are assigned randomly to each cluster.

Results for every organization/technique combination were computed over 10 simulation runs. Each simulation run consisted of 30,000 episodes. Seven tasks arrive at every episode and are randomly picked from a bag of tasks (to simulate a stable environment). Tasks in the bag are generated randomly such that each task requires between 4 and 10 agents to be accomplished. At any episode, the resources required by arriving tasks exceed the resources available to the system.

Our experiments focused only on 40 individuals and 10 managers so we can easily hand code different organization structures and study their effect. However, to verify the scalability of our approach, we tested it in a population of 90 agents and 13 managers. Agents were organized in a way similar to organization H and were randomly generated (using 9 distributions to represent 9 different classes of agents). Tasks were also randomly generated (from two different distributions). We plan to study even larger populations and use clustering techniques to automatically generate different organizations.

4.2 Results

Figure 4 shows the average utility for different organizations and policies. As expected, CRP performed worst. DRP performed better than CRP.[7] CGP is better than both. Our approach, DLP, outperformed all other policies for all organization structures,

[7]We believe this is due to the goal decomposition component of the organization, which encodes part of the domain knowledge.

except when using a random organization structure. Figure 5 illustrates how the performance of our system improves as agents gain more experience (i.e., witness more episodes). Interestingly, DGP (not shown in the figures), performed worse than DRP and DLP in all organizations except RS, where it performed better than both.

This reinforces our belief that organization structure does affect performance. Learning the local policy lessens this effect, but state abstraction and task decomposition remain sensitive to the structure of the organization. For the abstraction and task decomposition algorithms that we used, if agents are randomly organized, little can be gained by learning.

In our experiments with larger agent population (90 agents), DLP was better than other policies, achieving 35% more utility than CRP and at least 20% better than DRP and DGP.

More importantly, DLP is more stable than other approaches as Figure 6 shows. The standard deviation (of achieved utility) using CGP is 70% worse than DLP with SE organization. CRP is 30% worse than DLP. Also DGP was the worst for all organizations except RS. We had similar results with the larger agent population. DLP had the least standard deviation, which was around one third that of DGP.

Figure 7 compares the average number of exchanged messages per task. As expected, centralized approaches exchange fewer messages. Still, learning the local decision reduces the number of exchanged messages. Finally, Figure 8 shows the average resources wasted. CGP wasted 20% more resources than DLP, while CRP wasted 40% more. We got similar results for the larger agent population.

5 Related work

In [7], the authors presented a distributed algorithm for solving the coalition formation problem. The algorithm is optimal, complete and requires exponential time. Our algorithm is an approximation algorithm that returns a solution in polynomial time.

The work in [5] introduced an anytime coalition structure generation algorithm (the term *coalition structure* refers to the solution of the coalition formation problem). As in [7], the work did not use any organization for guiding the coalition formation search and assumed a black box function that given a feasible solution returns the *value* of such solution, while we evaluate the solution based on the total utility of the allocated tasks.

The work in [8] used a multi-leveled learning scheme to form coalitions. Both reinforcement learning and case based reasoning were used. Unlike our work, they do not use an underlying organization, which limits the scalability of their approach (their experiments were limited to 4 agents).

Though some extensions to the original contract net protocol [11] proposed the use of an underlying organization, none of these extensions (to our knowledge) provided a formal model of such an organization, nor evaluated the performance for different organizations, unlike our work here.

The coalition formation problem can be mapped to a multi-unit combinatorial auction[8]. Each task, to be assigned a coalition, is mapped to an auction, where the required

[8]In a multi-unit combinatorial auction, there might be more than one identical items, i.e. multiple units

resources are the items to be bidded on. Each individual agent is represented by a bid that contains the resources this agent controls and willing to contribute to any coalition it joins.

However, none of the algorithms developed for combinatorial auctions [6] make use of *stable knowledge*, which remains relatively unchanged throughout the system lifetime. This includes agents' capabilities (e.g., same bids repeat for consecutive auctions) and task patterns (e.g., consecutive auctions follow some statistical model). We on the other hand try to exploit this knowledge implicitly using an underlying organization.

The work in [3] tried to provide a unified framework for coordination in MAS. In this framework each agent follows a set of behaviors that differ in their level of abstraction. As behaviors become more and more abstract, an (implicit) underlying organization becomes more and more apparent. The goal of such an organization is to optimize the immediate individual goals. In our work, the goal of the organization is to optimize the coalition formation process, which *indirectly* optimizes the performance of the MAS as a whole.

In [4], the authors proposed and analyzed a simplified and restricted model of an organization, which takes only processing and communication costs into account. While they tried also to analyze the performance of different organizations, unlike our work there was no notion of resources, individual capabilities, coalition capabilities, task requirements, and coalition formation.

In our approach a group of agents co-learn to work together in an organization. This can be viewed as distributed learning of a hierarchical policy that targets recursive optimality [1]. However, none of the work in hierarchical learning area introduced the concepts of quantitative/dynamic state *abstraction* and task *decomposition*. We defined these concepts to decouple agents' local decision problems while minimizing communication, and hence achieve scalability. Our work also quantitatively evaluates how different action hierarchies affect the learning performance.

6 Conclusions and Future work

In this work we defined a generic problem solving framework using an underlying organization, and applied it to the coalition formation problem. We provided an algorithm for the local decision to be made by each agent, given state abstractions from other agents and its decomposed task. We used Q-learning with neural nets as functional approximators to improve the local decision. Our initial results show that our approach outperformed both random and greedy policies for most of the organizations we studied. It achieved higher utility and more stability with a smaller percentage of wasted resources and fewer exchanged messages. The results also verify the scalability of our approach as it still outperforms the other approaches we studied for larger systems.

In future, we aim to study a wider variety of organizations for different types of environments. We will also investigate further our abstraction and decomposition

of the same item may exist.

schemes, as we believe better schemes can considerably improve the learned policy performance. We also plan to study the optimization of the underlying organization and how this interacts with optimizing the hierarchical policy.

References

[1] A. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. In *Discrete Event Systems journal*, volume 13, pages 41–77, 2003.

[2] K. Decker and V. Lesser. Designing a family of coordination algorithms. In *1st International Conference on Multi-Agent Systems*, 1995.

[3] E. Durfee and T. Montgomery. Coordination as distributed search in a hierarchical behavior space. *IEEE Trans. on Systems, Man, and Cybernetics*, 21:1363–1378, 1991.

[4] Y. pa So and E. Durfee. Designing tree-structured organizations for computational agents. *Computational and Mathematical Organization Theory*, 2(3):219–246, 1996.

[5] T. Sandholm and et al. Coalition structure generation with worst case guarantee. *Proceedings of the 3rd Internation Conference on Autonomous Agents*, 1999.

[6] T. Sandholm and et al. Winner determination in combinatorial auction generalizations. *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems*, 2002.

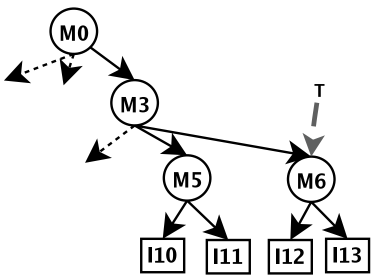
[7] O. Shehory. Methods for task allocation via agent coalition formation. *Artificial Intelligence Journal*, 101(1–2):165–200, 1998.

[8] K. Soh and X. Li. An integrated multilevel learning approach to multiagent coalition formation. *International Joint Conference on Artificial Intelligence*, pages 619–624, August 2003.

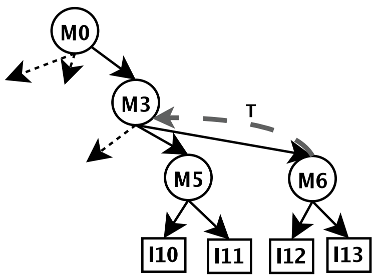
[9] R. Sutton and A. Barto. *Reinforcment Learning: An Introduction.* MIT Press, 1999.

[10] M. Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.

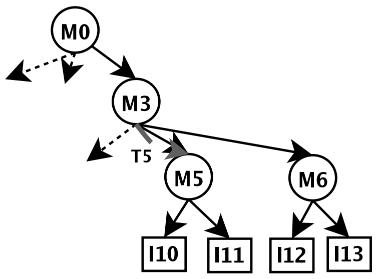
[11] D. N. W. Shen. An agent-based approach for dynamic manufacturing scheduling. *Proceedings of the 3rd International Conference on the Practical Applications of Agents and Multi-Agent Systems*, 1998.



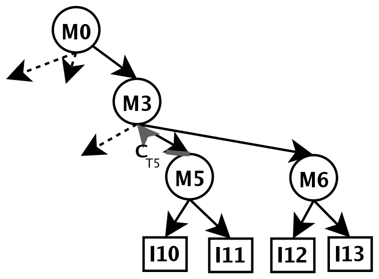
[a] Agent M6 discovers a new task, T



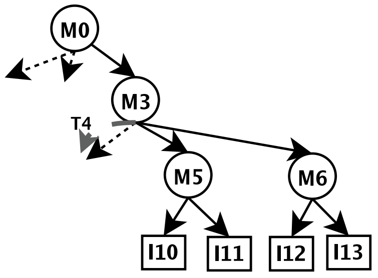
[b] Task T is beyond M6's capabilities, so it hands T to a higher manager, M3



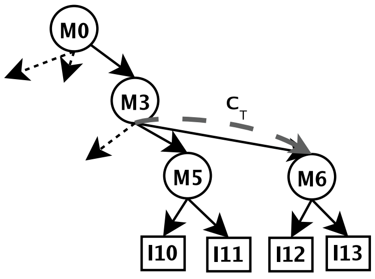
[c] Having more global view of the organzation, M3 decides to decompose T into subtask T5 and ask M5 to allocate a coalition for it.



[d] M5 successfully allocate T5, and sends committed coalition back to M3



[e] M3 decomposes the rest of T into subgoal T4 and ask another agent to allocate it. The process continues until all T is allocated



[f] At the end, Task T is achieved. M3 integrates all subcoalition into C_T and sends it to the originating agent, M6.

Figure 2: An example of organization-based coalition formation.

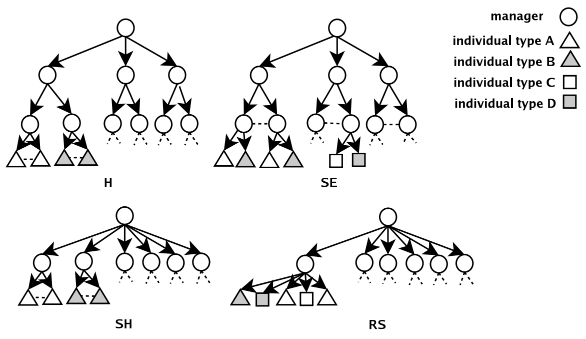


Figure 3: An Organization Hierarchy

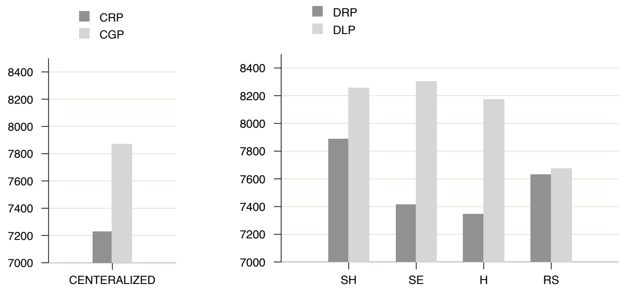


Figure 4: Utility average.

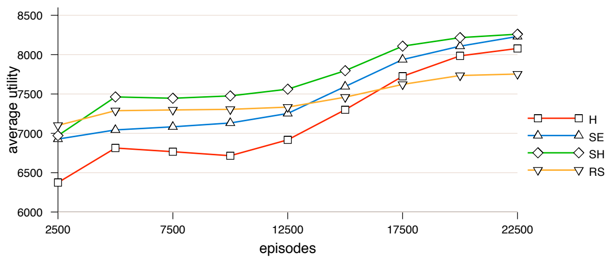


Figure 5: Learning curve.

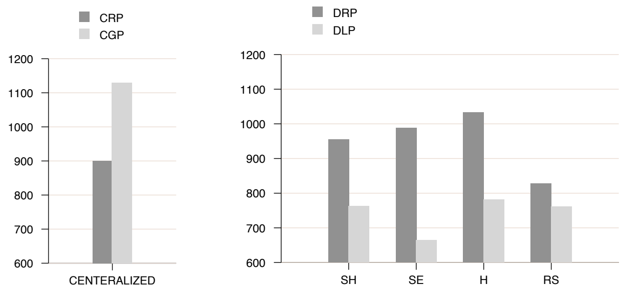


Figure 6: Utility standard deviation.

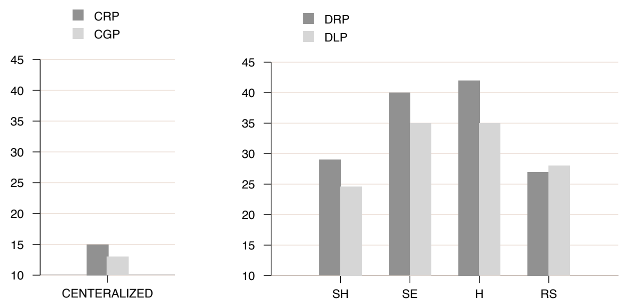


Figure 7: Messages average.

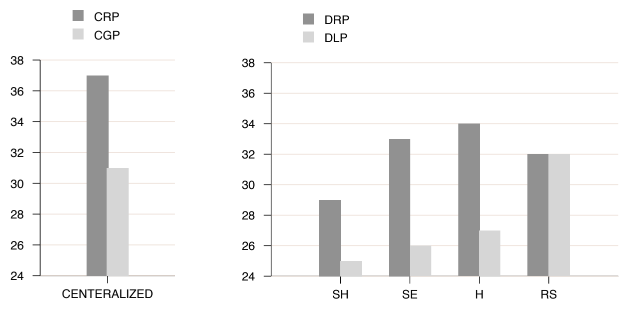


Figure 8: Average percentage of wasted resources.