A Decentralized Coordination Approach for Dynamic Weighted Task Allocation with Space, Time and Communication Constraints in A Disaster Domain

Abstract

The coordination for dynamic task allocation based on the available resources is a very challenging issue in general disaster domains because of space and time requirements, limited communications between rescue agents, as well as the workload and emergency degree of each task. In addition, it is also very hard or impossible to obtain the global knowledge about the environments, overall resources and tasks on time, due to time, space and communication constraints in many disaster domains and centralized approaches are not practical when allocating tasks in this kind of domains. This paper presents a novel decentralized coordination approach for dynamic task allocation by considering weighted workload of tasks, and space, time and communication constraints in a disaster domain. In the approach, a group formation mechanism is proposed to help agents with limited a communication range to achieve effective task allocation in a group through cooperation. A utility function is designed to help the coordinator (i.e. group leader) choosing the best allocation solution for the group based on a dynamic situation. The overall task allocation is achieved through distributed coordination in each dynamic group without a central control mechanism so it can reflect real life situations in many disaster domains. The experiment result shows that the proposed approach outperforms other centralized or decentralized approaches, in different scales of a disaster domain under space, time, and communication constrains.

1 Introduction

Nowadays, multi-agents technologies have been widely used in computer science, information technology, engineering, as well as other disciplines due to their abilities of autonomous decision making, collaborative problem solving, learning and adaptation abilities under open and distributed environments [Lesser, 1999], [Ren and Sim, 2009], [Vinyals and and, 2010]. In past decade, researchers have done much work on applying the multi-agents technologies to different kinds of domains. Coordination for task allocation is one of these technologies, which can be applied in many domains such as disaster rescue, other planets exploration and CPU calculation etc. The main objective of task allocation is to allocate the most suitable resource (mostly agents) to each task. Except the main objective, more and more new requirements for task allocation appear in recent years.

The main constraints for task allocation in a disaster domain include follows. 1) Space constraint. In a disaster domain, tasks are discovered at different locations within the domain. If an agent wants to work on a task, the first thing is to move to the location of the task. 2) Time constraint In a disaster domain, the objectives of task allocation include saving more survivors in debris, extinguishing fire before it spreads etc. With this consideration, each task should have a hard deadline, which represents that it is only worthy to finish the task before deadline (i.e. when the survivor is alive or the building is not burnt down). 3) Communication constraint. Obviously, in a disaster domain, the quality and range of the communication between agents will be limited due to the local infrastructures destruction. In addition, other features existing in a disaster domain also need to be considered when coordinating the task allocation. Two of main important features are 1) the dynamics of the tasks and agents, i.e. agents are continuously coming in and leaving the domain and tasks are continuously being discovered and finished; and 2) The emergency degree of each task i.e. the most emergent tasks can be finished before deadline, and the least emergent tasks should be discarded by the task allocation approach if resources are not sufficient. In most of current approaches, the researchers only try to finish more tasks before their deadlines [Ramchurn *et al.*, 2010b], [Koes *et al.*, 2005], but omit the emergency difference between tasks. For example, if there are two collapsed buildings, in one building there are 10 survivors and 10 tons of goods are stored in another building. When allocating agents for these two building, it is more reasonable to save 10 survivors and discard the 10 tons of goods. Therefore, the emergency of each task is also a key attribute that needs to be considered.

To handle above issues, various models, mechanisms and approaches have been proposed to assist coordinating task allocation in a disaster domain from different perspectives [Ramamritham *et al.*, 1989], [Koes *et al.*, 2005], [Farinelli *et al.*, 2008], [Chapman *et al.*, 2009], [Ramchurn *et al.*, 2010b], [Ramchurn *et al.*, 2010a].

A number of researchers use centralized approaches to deal with task allocation [Ramamritham *et al.*, 1989], [Koes *et al.*, 2005], [Ramchurn *et al.*, 2010b]. The centralized approaches can ensure to find an optimal allocation solution if the central controller has a global knowledge about overall tasks and agents in the domain. However, in a disaster domain, it is hard for a central controller to have that kind of knowledge due to the dynamic feature and communication constraints.

To overcome the limitations of centralized approaches, some researchers proposed decentralized approaches to deal with coordination for task allocation [Farinelli *et al.*, 2008], [Ramchurn *et al.*, 2010a] based on the max-sum algorithm. However, the max-sum algorithm requires huge amount of messages passing between agents. It is hard to ensure that all of these messages can be passed correctly in a disaster domain with communication constraints.

Furthermore, most of the centralized and decentralized approaches do not consider the emergency difference between tasks when designing coordination approaches for task allocation. In order to meet the challenges for task allocation in the disaster domain, a novel decentralized coordination approach for dynamic task allocation is proposed in this paper. Our approach employs the token passing mechanism of Petri Net to help neighboring agents to communicate with each other under communication constraints and forming groups in a decentralized manner. The contributions of the proposed approach include follows. First, the proposed approach considers the most important constraints, which are space, time and communication to reflect the real situation in a common disaster domain. Second, the weighted workload of each task and the dynamic feature of the domain have been taken into account by the approach in order to meet the requirements of the domain. In addition, a novel group formation mechanism is proposed to help agents to form groups with respect to communication constraints. Finally, comprehensive utility function, in comparing with other proposed approaches, is designed to help coordinator to find its best task allocation of the group.

The rest of paper is organized as follows. In Section 2, the problem is formulated and definitions are given. The basic process of our approach are introduced in Section 3. The experimental setting and analysis are described in Section 4. Some related work about coordination for task allocation are introduced and analyzed in Section 5. The paper is concluded and future work is outlined in Section 6.

2 Problem Description and Definition

This problem can be described as coordination for a weighted task allocation problem with, space, time and communication constraints. A common task allocation problem consists of two sets, which can be described as follows.

A task set can be defined as $TSet = \{T_1, T_2, T_3, ..., T_m\}$, where T_i represents the i^{th} task and $1 \leq i \leq m$.

An agent set can be defined as $ASet = \{A_1, A_2, A_3, ..., A_n\}$, where A_j represents the j^{th} task and $1 \leq j \leq n$.

To describe the tasks and agents in the problem, the formal definitions of the task and agent are given below.

Definition 1: A *task* T_i can be defined as a six-tuple $T_i = < TNo, DLine_i, WLoad_i, Loc_i, Emg_i, TSta_i >$, where TNo is the ID of task T_i, $DLine_i$ is the deadline of T_i (e.g. the time until which the survivor is still alive or build is not burnt down), where $Dline_i \in [0, \infty]$, $WLoad_i$ is the workload of T_i, which represents how many work units is done by agents, T_i can be finished, Loc_i is the location of T_i, Emg_i is the emergency degree of the workload of T_i, where $Emg_i \in [1, 10]$, 1 and 10 represent the lowest and the highest emergency degree respectively and $TSta_i$ is the status of T_i, which can be one of 'available', 'underwork', 'finished' or 'expired'.

Definition 2: An *agent* A_j can be defined as a six-tuple $A_j = < ANo, Uti_j, Loc_j, MSp_j, Comm_j, ASta_j >$, where ANo is the ID of A_j, Uti_j is the working efficiency of A_j, which represents how many work units A_j can perform per time unit. Loc_j is the current location of A_j, MSp_j represents the moving speed of A_j, which represents how many distance A_j can move per time unit, $Comm_j$ is the communication and scan range of A_j and $ASta_j$ is the status of A_j, which can be ether 'available' or 'working'.

Our mechanism defines in a time interval τ agent A_j working on task T_i as $\tau^{A_j \to T_i}$, which represents how many time units agent A_j working on T_i.

$$\tau^{A_j \to T_i} = t_{ij}^{end} - t_{ij}^{begin} (0 \leq t_{ij}^{begin} \leq t_{ij}^{end}) \tag{1}$$

, where t_{ij}^{begin} represents the time point that A_j begins to working on T_i, t_{ij}^{end} represents the time point that A_j stop working on T_i.

A task which is finished means that agents have performed equal or more work to the task than the workload of the task before the deadline can be described as follows.

$$Finish(T_i) = \begin{cases} 1, & \sum_{j=1}^{n} Uti_j \times \tau^{A_j \to T_i} \geq WLoad_i \wedge \\ & Max(t_{ij}^{end}) \leq DLine_i \\ 0, & otherwise \end{cases} \tag{2}$$

, where Uti_j is the work efficiency of A_j, $\tau^{A_j \to T_i}$ is the number of time units that A_j working on T_i, $WLoad_i$ is the workload of T_i, t_{ij}^{end} is the time point that A_i stops working on T_i and $DLine_i$ is the deadline of T_i

The main objective of weighted task allocation is to find an arrangement of agents to tasks, which can finish the maximum sum value of workload. This can be described as follows:

$$argmax \sum_{i=1}^{m} Finish(T_i) \times WLoad_i \times Emg_i \tag{3}$$

Except the main objective, the cost of finishing tasks should be minimized, which includes 1) minimizing the traveling time and distance of agents for finishing tasks and 2) minimizing the working time for agents to finish tasks.

We assume that each agent can offer the workload of a task independently, and no needs to wait for other agent can begin the work.

We employed a token passing mechanism to help agent quickly passing information of tasks and agents under the communication constraints. There are two types of tokens in our approach: task token ($TToken_k$) and agent token ($AToken_j$), both of which are created by agents. The two kinds of tokens are defined as follows:

Definition 3: A *task token* $TToken_k$ represents the k^{th} task token, which can be defined as an six-tuple $TToken_k = < ANo, TNo, DLine_k, WLoad_k, Loc_k, Emg_k >$, where ANo is the ID of the agent which creating $TToken_k$, TNo

is the ID of the task represented by $TToken_k$, $DLine_k$ is the deadline of task represented by $TToken_k$, $WLoad_k$ is the workload of task represented by $TToken_k$, Loc_i is the location of task represented by $TToken_k$ and Emg_k is the emergency degree of the workload of task represented by $TToken_k$.

Definition 4: An *agent token* $AToken_j$ represents the agent token of A_j, which can be defined as a four-tuple $AToken_j =< ANo, Uti_j, Loc_j, MSp_j >$, where ANo is the ID of agent represented by $AToken_j$, Uti_j is the working efficiency of A_j, Loc_j is the current location of A_j and MSp_j represents the moving speed of A_j,.

An agent A_j can be two different roles in our approach, i.e. $Coordinator$ and $Resource\ provider$, which can be defined by following definitions.

Definition 5: A *Coordinator* is an agent, to be in charge of allocating suitable agents to tasks.

Definition 6: A *Resource Provider* is an agent which has the requested resource of for a task and can finish task before deadline .

An agent can be either a coordinator or a resource provider or both.

To make following explanation clear, our approach employed the concept of 'neighbors'. Since communication constraints is common in disaster domains, agents can only discover the information of tasks and communicate with other agents close to its location within a limited communication range.

Definition 7: A *neighbor* of A_j is an agent that can directly communicate with A_j. In other words, a neighbor of A_j is a agent within A_j's communication range ($Comm_j$, see Definition 2).

In our mechanism, the Manhattan distance is employed to represent the distance between two points. By considering the moving speed of agent A_j, the time (represented by $Time(T_i, A_j)$) that A_j uses to travel from its current location to the location of T_i can be calculated by the following equation.

$$Time(T_i, A_j) = \frac{|Loc_i - Loc_j|}{MSp_j} \tag{4}$$

,where Loc_i is the location of T_i, Loc_j is the current location of A_j and Msp_j is the moving speed of A_j.

3 The Basic Processes of Our Approach

Our approach consists of five looping steps, which are 1) Task Token Creation, 2) Group Formation, 3) Token Passing, 4) Task Allocation and 5) Allocation Solution Return. After one loop of five steps, if there are available agents and tasks in the domain, the five steps will be repeated again according to the current location of available tasks and agents. The loop will be repeated until there is no available task in the domain.

3.1 Task Token Creation

In this step, each available agent A_j ($ASta_j$ is 'available' see Definition 2) scans the area within its communication range ($Comm_j$). Once, A_j finds any available task T_i ($TSta_i$ is 'available' see Definition 1), A_j creates task token $TToken_k =< A_j, T_i, DLine_i, WLoad_i, Loc_i, Emg_i >$ (see Definition 3) for T_i. Meanwhile, A_j also create an agent token $AToken_j$ (see Definition 4) for itself.

3.2 Group formation

In this step, agents form groups to coordinate the task allocation for available tasks and agents in the group. Because of communication constraints, agents can only communicate with its neighbors (see Definition 7) in the domain. In order to generate allocation solutions which can be comparable with the centralized solution, the group formation mechanism is designed to connect the maximum number of agents within the domain. In addition, the agent with the maximum number of neighbors in the group is chosen to be the coordinator of the group. The objective of the group formation mechanism is to link agents with its neighbors and form a tree, in which the coordinator of the group is the root node. In the mechanism, agents only compare and update three kinds of information: the Parent Agent (PA), the Coordinator (C), the Number of Neighbors of Coordinator (NNC). To begin with the group formation mechanism, each agent A_j records the number of neighbors it have and choose itself as PA and C of A_j (see lines 2 to 4). Then, the following three steps is repeated by each agent in the domain. First, agent A_j chooses the agent A_u from the neighbors of A_j or A_j, who has the highest NNC as the PA of A_j and the C and NNC of A_j is changed to the C and NNC of A_u (see lines 6 to 8). Third, for each neighbor (A_l) of A_j, if C of A_l is not the C of A_j, the PA of A_l is changed to A_j and the C and NNC of A_l is changed to the C and NNC of A_j (see lines 9 to 11). This steps will be continually repeated by agents until no PA, C and NNC changes in any agent in the domain. The group formation mechanism is described in Algorithm 1:

Algorithm 1: Group Formation

```
1  for each Agent $A_j$ do
2      Get number of neighbors of $A_j$
3      $A_j$.PA← $A_j$
4      $A_j$.C← $A_j$
5      $A_j$.NNC← number of neighbors of $A_j$
6  end
7  for each Agent $A_j$ do
8      Get $A_k$ from neighbors of $A_j$ and $A_j$, Max($A_u$.NNC)
9      $A_j$.PA← $A_u$
10     $A_j$.C← $A_u$.C
11     $A_j$.NNC← $A_u$.NNC
12     for each neighbors of $A_j$:$A_l$ do
13         if $A_l$.C≠ $A_j$.C then
14             $A_l$.PA← $A_j$
15             $A_l$.C← $A_j$.C
16             $A_j$.NNC← $A_j$.NNC
17         end
18     end
19 end
```

3.3 Token Passing

In this step, first, each agent A_j gets task tokens ($TToken_k$ see Definition 3) and agent tokens ($AToken_j$ see Definition 4) (created in the Task Token Creation step) from its child agents. There is a situation that different agents in the same group create task tokens for the same task. For example, A_u and A_{u+1} are child agents of A_j and they create task tokens $TToken_k =< A_u, T_i, DLine_i, WLoad_i, Loc_i, Emg_i >$ and $TToken_{k+1} =< A_{u+1}, T_i, DLine_i, WLoad_i, Loc_i, Emg_i >$ for the same task T_i, respectively. It is can be seen that except creating agent, two task tokens are exactly the same. In this situation, A_j combines the two tokens to together($TToken_{k+1}$ are abandon and only $TToken_k$ is kept). After combination, A_j passes combined task tokens and agent tokens it received and itself's to the parent agent of A_j. Finally, all of task and agent tokens are passed to the coordinator of the group.

3.4 Task allocation

This step is the core of our approach. In this step, coordinator tries to find the best allocation solution for the group according to tasks and agents in the group. In our approach, the allocation solution of a group is represented by an n-tuple $Alloc =< ANo_1 \rightarrow TNo_1, ANo_2 \rightarrow TNo_2, ..., ANo_n \rightarrow TNo_n >$, where, ANo_j is the ID of the j^{th} agent of the group and TNo_j is the ID of the task to which the j^{th} agent of the group is allocated. If the j^{th} agent of the group is not allocated to any task, the TNo_j is $\varnothing$. Since finding the best allocation solution is a NP-hard problem (the proof process can be found Section 4.1 of [Ramchurn *et al.*, 2010b]), the heuristics way is employed in our approach. The heuristics for finding the best allocation solution can be implemented in two sub-steps 1) Useless Allocation Elimination 2) Utility Calculation

Useless Allocation Elimination

Based on the combinatorics, the number of combination for a group with m number of tasks and n number of agents is m^{n+1}. In these allocations, there are many of allocations that the allocated agents cannot ensure to finish the task in time (before deadline). For example, in a group the one of allocation solution is $Alloc_a =< A_1 \rightarrow T_1, A_2 \rightarrow T_2, A_3 \rightarrow T_1 >$. After calculation, if we found that A_1 and A_3 cannot finish T_1 according to the status of A_1, A_2 and T_1. This allocation solution is useless and should be eliminated to make the following calculation more efficient. The process of this elimination can be described as follows. First, the coordinator gets m^{n+1} allocation solutions of the group. Second, for each task T_i exists in allocation solution $Alloc$, calculate the maximum workload of agents which are allocated to T_i can perform before deadline of T_i. If an allocation solution exist task T_i which cannot be finished by allocated agents before deadline, the $Alloc_a$ is eliminated.

Utility Calculation

In this sub-step, the coordinator calculates the utility of each useful allocation solution and choose the allocation solution with the highest utility value as the best allocation solution for the group. To judge an allocation solution is good or not not only depends on how many tasks can be finished, but also depends on the benefits getting from finishing the tasks and the cost spending on finishing the tasks. In our approach, the utility of an allocation solution $Alloc_a$ is affected by two

Algorithm 2: Useless Allocation Elimination

```
1  for each Each $Alloc_a$ of a group do
2      for each Task $T_i$ do
3          if $T_i \in Alloc_a$ then
4              get $MWL_i = \sum_{j=0}^{m}(DLine_i - CurrentTime - Time(T_i, A_j)) \times Uti_j$
5              if $MWL_i < WLoad_i$ then
6                  $Alloc_a$ is eliminated
7              end
8          end
9      end
10 end
```

factors (1) the benefits getting from finishing tasks, and (2) the cost spending on finishing tasks

With the two factors above, the utility of an allocation solution $Alloc_a$ can be calculated by the following formula.

$$Utlity_{i \rightarrow t} = Q_{benefit} - Q_{cost} \tag{5}$$

The benefits getting from finish tasks involves two factors: 1) Emergency degree of workload. 2) Time saved for finishing a task. The benefits of an allocation solution $Alloc$ is the sum benefits getting from all of finished task in $Alloc_a$, which can be described as follows.

$$Q_{benefit} = \sum_{t=0}^{n} Emg_i \cdot (WLoad_i + \frac{(DLine_i - FTime_i) \times WLoad_i}{DLine_i}) \tag{6}$$

,where Emg_i, $WLoad_i$, $DLine_i$ are emergency degree of workload, workload and deadline of T_i in $Alloc_a$, respectively and $FTime_i$ is the predicted finishing time of T_i in $Alloc_a$.

The cost for finishing tasks in our approach includes 1) the traveling time of each agent moving from its current location to the location of the its allocated task and 2) the time of each agent spending on finishing the allocated task. Therefore, the cost of an allocation solution $Alloc_a$ is the sum cost of agents in $Alloc_a$ spending on traveling and finishing tasks, which can be described as follows.

$$Q_{cost} = \sum_{t=0}^{m}((FTime_i - CurTime) \times Uti_j) \tag{7}$$

,where Uti_j is the moving speed of A_j, $FTime_i$ is the predicted finishing time of T_i in $Alloc_a$, which A_j is allocated to and $CurTime$ is the current time when coordinator calculates the utility function of $Alloc_a$.

The $FTime_i$ is the predicted finishing time of T_i in $Alloc_a$, which can be calculated by the following Equation:

$$FTime_i = \frac{WLoad_i + CurTime \times \sum_{j=1}^{n} Uti_j + \sum_{j=1}^{n} Time(T_i, A_j) \times Uti_j}{\sum_{j=1}^{n} Uti_j} \tag{8}$$

,where $Wload_i$ is the workload of T_i in $Alloc_a$, $CurTime$ is the current time point when coordinator calculates the utility function of $Alloc_a$, Uti_j is the j^{th} agent, which is allocated to T_i in $Alloc_a$ and $Time(T_i, A_j)$ is the traveling time from the current location of A_j to the location of T_i.

3.5 Allocation Return

In this step, the coordinator sends the chosen allocation solution to the agents in the group. Once an agent received an allocation, it will do the following things. First, the tokens of allocated tasks (can ensure to be finished) is eliminated from agents token pool. Second, the allocation is passed to the neighbors of the agent. Moreover, the status of the agent is changed from 'available' to 'working'. Finally, the agent moves to the location of the task and starts working on it. When the first agent reaches to the location of a task, it changes the status of the task from 'available' to 'underwork'. After finished the work of the task, all of agents worked on the task change the status of them from 'working' to 'available' and the status of the task is also changed from 'underwork' to 'finished'.

4 Experiments

The experiment is conducted for evaluating the performance of the proposed approach. The benchmark of the experiment is the scheduling heuristics proposed by Ramdchurn et al.[Ramchurn *et al.*, 2010b]. This scheduling heuristics is a centralized task allocation mechanism which is the first mechanism that solves the Coalition Formation with Spatial and Temporal constraints problem (CFSTP).

4.1 Experimental Settings

The basic settings for our experiment are shown in Table 1.

Name	Value	Name	Value
Number of tasks	100	Emergency Degree	$1 \sim 10$
Number of agents	10	Work Efficiency	1
Deadline	$5 \sim 100$	Area size	50×50
Workload	$10 \sim 60$	Communication range	20

Table 1: The Detail Settings in Experiment

This setting is a hard condition for task allocation in a disaster domain, which means that the agents in the domain cannot finished all of tasks. Therefore, the approaches should use their utility function to choose which tasks should be finished and which tasks should be abundant. Furthermore, the Manhattan distance is employed in the experiment. The group formation mechanism is affected by three factors, the number of agents in domain, the communication range of agents and the size of the domain. From the definition of Manhattan distance, in a $m \times m$ size domain, if the communication range of agent A_j is c, the possibility of other agents that is the neighbor of A_j can be calculated as follows.

$$Possibility = \frac{2 \cdot c^2 + 2 \cdot c + 1}{m \cdot m} \tag{9}$$

With the increasing of communication range or agent numbers and the decreasing of domain size, the more agents can find neighbors and form groups. In our experiment, the size of domain is 50×50, the communication range of agents is 20. After calculated by Equation 8, the possibility of two agents being neighbor of each other is 33.6%. This is a relatively good setting for 10 agents, because there is a big possibility for agents to form groups and also exist agents that cannot connect with other agents.

4.2 Experimental Results

The experimental results are shown in Figure 1, 2 and 3 .

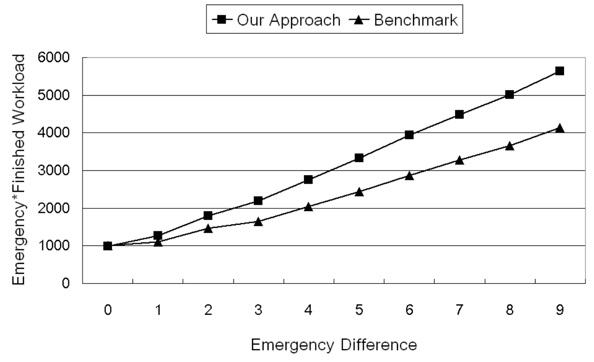

Figure 1: Experimental result of weighted workload

In Figure 1, the X-axis represents the biggest emergency difference between two tasks. In our approach, the emergency degree is in [1, 10] (see, Definition 1), the biggest emergency difference between tasks is 9. The Y-axis indicates the sum of $Emergencydegree \times Workload$ of accomplished tasks (see Equation 3). From Figure 1, it can be seen that when the emergency different is low (e.g.,0), the benchmark approach could offer more weighted workload than our approach. That is because that the benchmark is a centralized approach, where coordinator has the global knowledge about all of tasks and agents in the domain. Hence, a overall optimal result can be achieved. Since the communication constraint is considered, the agents in our approach can only be coordinated in a decentralized manner. Hence, our approach can only provide partially optimal solution. However, when the emergency difference is increasing, our approach considers the emergency degree of finished tasks and try to finish tasks with more emergent first. The benchmark has a relatively easier utility function which only considers the reuse of each agent group and does not take the emergency degree of finished tasks in to account. Therefore, with the increasing of emergency difference, the allocation solution of our approach can finish more weighted workload than the benchmark.

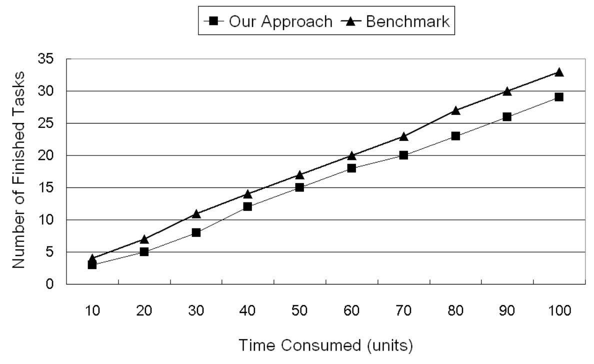

Figure 2: Experimental result optimal of number of finished tasks

In Figure 2, the X-axis represents the time consumed, since the deadline of a task is [5-100], the maximum consumed time should be 100 units. The Y-axis indicates the number of finished tasks. From Figure 2, it can be seen that our approach can finish less tasks than the benchmark in the same time. The reason for this situation is that the benchmark has a globule view about the domain and our approach just has local view and can only reach to partially optimal allocation in an area of the domain.

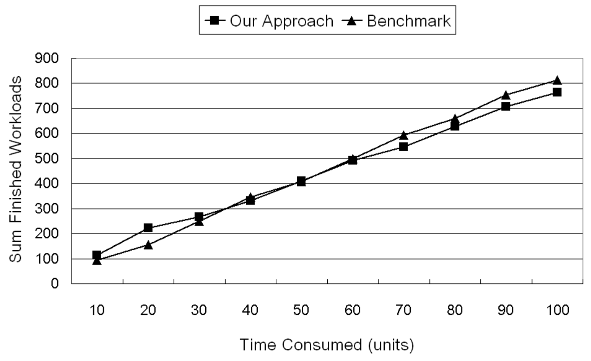

Figure 3: Experiment result of finished workload

In Figure 3, the X-axis also represents the time consumed. The Y-axis indicates the sum finished workload that has been finished. From Figure 3, it can be seen that our approach and the benchmark can finished nearly the same workload in time units.

4.3 Experimental Analysis

From Figure 1, 2 and 3, it can be seen that the benchmark model is based on the assumption that it has a global view about the domain, it can get a relatively optimal allocation solution for the tasks and agents. However, since the utility function of the benchmark is not comprehensive and only considers the reuse of each group, it is hard to get the best solution for the task allocation in a disaster domain. Our approach considers the communication constraint in a disaster domain, where agents only have local view. To reduce the effects of communication constraint on task allocation, a group formation mechanism and a relatively comprehensive utility function are proposed by considering all of constraints and features in a disaster domain. However, the group formation mechanism is greatly affected by the domain size, the communication range and number of agents. The group formation mechanism and the utility function are the reasons that why our approach with only local view can have a similar allocation result as the benchmark with globe view and better performance than the benchmark when consider the emergency degree of each task.

5 Related Work

Ramamritham et al. proposed a distributed coordination way to handle task allocation with deadline and resource requirements in [Ramamritham *et al.*, 1989]. In this paper, the idea for classification for tasks is a practical idea in common task allocation. The utility function in the proposed approach is called bidding and focused addressing which is a simple utility function. However, the main drawback of this approach is that it omits the space and communication constraint in the process of task allocation. Furthermore, in a common disaster domain, most of tasks are critical and seldom to have tasks of other categories. Therefore, the approach proposed by Ramamritham et al. is hard to be employed in a disaster domain.

Some researchers such as Katirya and Jennings use Mixed Integer Linear Programming (MILP) [Gordon *et al.*, 2009] to get optimal allocation solution in a disaster domain in [Ramchurn *et al.*, 2010b], [Koes *et al.*, 2005]. The MILP process can guarantee to find an optimal solution for task allocation, if the coordinator can have global view about the domain. However, the MILP has three main drawbacks. First, in a common disaster domain, it is hard for the coordinator to have this globe view due to the communication constraint. Second, the MILP can only deal with problems with fixed numbers, status of tasks and agents. This feature makes the solution hard to fit the highly dynamic disaster domain. Third, the calculation complex of the MILP is too high.

Some researchers (i.e. Ramchurn et al.) proposed decentralized task allocation approach, which is based on the Max-Sum algorithm named Fast-Max-Sum [Ramchurn *et al.*, 2010a]. The Fast-Max-Sum algorithm borrows the message passing mechanism of Max-Sum algorithm to improve the utility of each individual task. However, the main drawback of the Fast-Max-Sum approach is that it needs huge amount of message passing before tasks and agents can find an optimal allocation solution. In a disaster domain, the communication constraint heavily limits the message passing between agents. Furthermore, the Max-Sum algorithm can only get optimal solution in chain or tree data structure. If the factor graph exists cycle or loop, the Max-Sum algorithm can only get an approximate solution.

6 Conclusion and Future Work

In this paper, a novel decentralized coordination approach for task allocation with space, time and communication constraints is proposed. The approach uses a comprehensive utility function to help coordinator to find the best task allocation of the group by considering weight of workload, workload and traveling time under multiple constraints. A group formation mechanism is set up in order to help agents to form group within limited communication range due to the destroy of infrastructures in a diaster domain. In addition, the weight of workload for each task and dynamic features of the group are covered by our approach to meet the requirements of a common disaster domain. The experiment results show that this approach provides an more effective way to meet the challenges of a common disaster domain than the state-of-arts. In the future, we would like to use prediction way to evaluate the blocks between two locations. Furthermore, we also want to involve the tasks with prerequisite tasks which need to finish in an order.

References

[Chapman *et al.*, 2009] Archie C. Chapman, Rosa Anna Micillo, Ramachandra Kota, and Nicholas R. Jennings. Decentralised dynamic task allocation: a practical game: theoretic approach. In *Proceedings of The 8th International*

Conference on Autonomous Agents and Multiagent Systems (AAMAS '09), volume 2, pages 915–922, Budapest, Hungary, 2009.

[Farinelli *et al.*, 2008] A. Farinelli, A. Rogers, A. Petcu, and N. R. Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems (AAMAS '08)*, volume 2, pages 639–646, Estoril, Portugal, 2008.

[Gordon *et al.*, 2009] Geoffrey J. Gordon, Sue Ann Hong, and Miroslav Dudík. First-order mixed integer linear programming. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, pages 213–222, Arlington, Virginia, United States, 2009. AUAI Press.

[Koes *et al.*, 2005] Mary Koes, Illah Nourbakhsh, and Katia Sycara. Heterogeneous multirobot coordination with spatial and temporal constraints. In *Proceedings of the 20th national conference on Artificial intelligence - Volume 3*, AAAI'05, pages 1292–1297. AAAI Press, 2005.

[Lesser, 1999] V. Lesser. Cooperative multiagent systems: A personal view of the state of the art. *IEEE Transactions on Knowledge and Data Engineering*, 11:133–142, 1999.

[Ramamritham *et al.*, 1989] K. Ramamritham, J.A. Stankovic, and W. Zhao. Distributed scheduling of tasks with deadlines and resource requirements. *Computers, IEEE Transactions on*, 38(8):1110 –1123, aug 1989.

[Ramchurn *et al.*, 2010a] Sarvapali D. Ramchurn, Alessandro Farinelli, Kathryn S. Macarthur, and Nicholas R. Jennings. Decentralized coordination in robocup rescue. *Comput. J.*, 53(9):1447–1461, November 2010.

[Ramchurn *et al.*, 2010b] Sarvapali D. Ramchurn, Maria Polukarov, Alessandro Farinelli, Cuong Truong, and Nicholas R. Jennings. Coalition formation with spatial and temporal constraints. In *The ninth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, pages 1181–1188, 2010.

[Ren and Sim, 2009] F. Ren and K.M. Sim. Adaptive conceding strategies for automated trading agents in dynamic, open markets. *Decision Support Systems*, 48(2):331–341, 2009.

[Vinyals and and, 2010] Meritxell Vinyals and Marc Pujol and. Divide-and-coordinate: Dcops by agreement. In *The ninth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, pages 149–156, 2010.