Centralized Group Buying Approach for Multiple Cooperative Task Allocation

Gun-Hee Moon, Dong-Wan Yoo, Byung-Yoon Lee, Hae-In Lee, and Min-Jae Tahk
Korea Advanced Institute of Science and Technology, Aerospace Engineering, Daejeon, Korea
Email: {ghmoon, dwyoo, bylee, hilee, mjtahk}@fdcl.kaist.ac.kr

***Abstract*—This paper is about a centralized group buying task assignment algorithm (CGBA) for cooperative multiple task assignment, which are non-synergetic and hardly constrained on the minimum required number of agents to perform it. CGBA utilize greedy market strategy for each agent's path decision, and gives an incentive to the task with a larger required number of agents. In the multi-cooperative task allocation, there is a distinguish problem that would cause infinite loop in mission, which is defined as cross deployment. To resolve this, a checking algorithm using before & after task list is also proposed. Numerical experiment confirms that the algorithm gives a suboptimal solution in polynomial time.**

***Index Terms*—group buying, multi-agents, swarm robots, UAV task assignment**

I. Introduction

A. Mult-Allocation Cooperative Task

In the future, it is expected that swarm robots would be employed for a variety of missions that are incapable of being performed by a single agent. For instance, multiple UAVs are capable of transporting a heavy load in slung load style. The slung load is a freight delivery style, linking vehicle with cargo by external cables. One UAV cannot shift a heavy cargo, that exceed its' thrust capacity, alone. However a couple of UAVs, which linked with a package, may get a chance to lift the object.

In order to efficiently utilize the multi-agents system, several researches related to task assignment have been studied. Byeon used the consensus based bundle algorithm (CBBA) for the forest fire suppression mission [1]. The author utilized CBBA on the multiple air to air weapon assignment, and suppression of enemy air defense mission [2], [3]. Whitten extended the CBBA to handle coupled-constraints [4]. Choi [5] called such a tasks needed to be done by dual agent as duo task (DT), which is distinguished from single task (ST). He categorized the duo task into required duo task (RDT) and preferred duo task (PDT).

This paper deals with a class of task similar to RDT. However, this research is not only about the dual agent problem, but also about a task of hardly constrained on minimum required number of agent. Until all of the agents gather, which are assigned to the same task, they cannot start the mission. Because of it, there can be a cross deployment problem in multiple cooperative task assignment, and it will be explained more detail in the next chapter.

B. Background

Group buying (GB) is also called as *group purchasing* or *collective buying*. It is known to be originated from China, where "Tuángòu" was executed to get a discount price from retailer when a group of purchasers were willing to buy same item. As individual customers cannot freely make a deal with local retailers, a group of customers with same needs are organized to fulfill the minimum number of orders.

By the GB, individual customers as well as retailers get satisfied. In the customer's point of view, as commission charges are vanished, the retailer offers product in a discounted price. In the retailer's point of view, as the needs on a product are increased as much as the customer wants, their total incomes increase.

This article is inspired from the benefits of the GB, utilizing it for task assignment for cooperative task that multiple allocation process is required. Centralized group buying algorithm (CGBA) is also based on sequential greedy algorithm (SGA) by Choi [6]. The SGA sequentially finds out a feasible suboptimal solution in greedy way, however, it is restricted only for multi-task problem. Thus CGBA extends SGA to solve given multiple cooperative task allocation problem.

Rest of the paper is organized as follows. In chapter II the formal cooperative multi-task assignment problem is defined. The details of CGBA are explained on chapter III. In chapter IV, this work verifies the CGBA's applicability with simulations, and finishes with conclusion in chapter V.

II. Multiple Cooperative Task Assignment

A. Problem Statement

A cooperative task assignment is decision making problem that matches N_u-agents with N_t-tasks as many as possible. By allocating agents to proper tasks, society can gain utilities which measure its degree of satisfaction. Mathematically, the problem is formulated as an optimal decision problem that maximize utility summation of the entire fleet as,

Manuscript received December 15, 2013; revised February 11, 2014

$$\arg\max_{p} J = \sum_{\forall j}\sum_{\forall i} u_{ij}(x_{ij}, p_i) x_{ij} \tag{1}$$

subject to,

$$n_{\min_j} \le \sum_{\forall i} x_{ij} \le n_{\max_j} \quad \text{for } \forall j \qquad \text{or} \quad \sum_{\forall i} x_{ij} = 0 \tag{2}$$

$$\sum_{\forall j} x_{ij} \le L_i \qquad \text{for } \forall i \tag{3}$$

$$i \in I = \{1, ..., N_u\}, \quad j \in J = \{1, ..., N_t\} \tag{4}$$

where u_{ij} is the utility earned by of agent-*i* performing task-*j* along its' path p_i , and x_{ij} is a decision variable indicating agent-*i*'s assignment to task-*j*. If agent-*i* is allocated to task-*j*, x_{ij} is one, otherwise it is zero. The index *i* represents the ID number of each agent, and index *j* is the ID of the task.

The purpose of this optimization is finding fleet's path plan P that maximizes utility summation. The utility summation of whole agent represents the happiness of whole society. Therefore, maximizing utility summation of all agents is a reasonable decision making process in the sense of social benefits.

Equation (2) indicates the number of agents assigned to each task. The number of agents assigned to given task-*j* is represented by *j*-th column sum of decision variable matrix x . A cooperative task is hardly constrained, so that, to perform it, at least $n_{\min j}$ agents required at the same time, and the number of agents for a task cannot exceed $n_{\max j}$. In case of the planner assigning none of agent task-*j*, the column sum of decision matrix becomes zero. If there is no synergetic effect on utility function, $n_{\min j}$ would be absolutely more preferred to $n_{\max j}$. This paper concentrates on non-synergetic case only, so (2) can be shorten as,

$$\sum_{\forall i} x_{ij} = n_{\min_j} \ \text{ or } \ 0 \qquad \text{for } \forall j \tag{5}$$

Each agent is expected to perform L_i tasks at most for one sortie, as (3). It is a rough assumption to simplify the assignment problem. In real field, an agent's capability is correlated with fuel amount, fuel consumption rate, flight distance and so on.

The assignment is said to be ideally completed, if there is no more agents left to assign, or no more task needs to be allocated. Unfortunately, there could be some unideal secneraio from the beginning; Even though there are usable agents, sometimes it is not enough to perform leftover tasks. Thus a complete condition for the assignment is expressed as,

$$\sum_{\forall j}\sum_{\forall i} x_{ij} = N_{\min} \ \text{A}\min\{\sum_{\forall j} n_{\min j}, \sum_{\forall i} L_i, N_{poor}\} \tag{6}$$

$$N_{poor} = \max(\textstyle\sum_q n_q) \le \sum_{\forall i} L_i \quad \text{for} \quad n_q \in \{n_{\min, j}\} \tag{7}$$

where $\sum_{\forall j} n_{\min j}$ is the required number of agents to perform the whole tasks, $\sum_{\forall i} L_i$ is the number of available assignments of all agents, and N_{poor} is the number of possible assignments for the unideal case. It is the maximum sum of $n_{\min j}$, while being less or equals to the second candidate of (6).

B. Utility Function Scheme

A utility is function, mapping consumption of product or service to the happiness of the consuming subject. The utility theorem says that the utility is a monotonically increasing positive valued function as,

$$U : X \to R_{+} \qquad \Delta U / \Delta X > 0 \tag{8}$$

where X is the product or service to be consumed, and U is the utility function .

As the utility is positive real valued function, there is no trivial action that causes unhappiness while consuming it. Besides the monotonically increasing property of the utility says that one would never get satisfied until it consumes every resource, which is nature of a homo-economicus.

The expected utility from consuming one more resource is called as the marginal utility which is mathematically defined as,

$$u_{ij}[b_i] = \max_{n \le |p_i|+1} U^{p_i \oplus_n \{j\}} - U^{p_i} \tag{9}$$

where u_{ij} is the maximum marginal utility, b_i is agent-*i*'s task bundle list, p_i is agent-*i*'s ordered path list, U^{p_i} is total utility expected from given paths and $\oplus_n$ is an operator that inserts the second list right after the *n*-th element of the first list.

Provided that the marginal utility is monotonically increasing and all agents show the same utilities on a resource, then an agent is unwilling to share the resource with the others. It is because that the more resource it obtains, the more marginal gain is increased, like a gamble addict. For task allocation, such selfish agents will cause serious task concentration, which is unfamiliar with common sense. Therefore, to fairly protect the task assignment equilibrium, *diminishing marginal gain* (DMG) property is needed.

DMG is also similar to the *sub-modularity* set function, which is defined for a set function that the sum of outcome of given function from intersection and union of set X and set Y is always smaller than the sum of outcome of set X and set Y , for given arbitrary sets. The difference between DMG and the sub-modularity is that DMG is defined for ordered list, path, but the sub-modularity is for unordered set. As long as the score scheme satisfies the DMG property, the agents will fairly share the tasks. Mathematically, DMG is expressed as follows,

$$u_{ij}[b_i] \ge u_{ij}[b_i \oplus_{end} b] \tag{10}$$

Equation (10) means that the marginal utility gained by inserting task-*j* on bundle b_i is always bigger than the marginal utility gained by inserting task-*j* on bundle $b_i \oplus_{\mathrm{end}} b$.

Ref. [6] showed that next *time discounting scheme* insures DMG property of scoring scheme, by triangle inequality of given path. The time discounting utility scheme is,

$$U^p = \sum_j^J \bar{u}_j \lambda_j^{(TOC_j(p)-t_0)} \tag{11}$$

$$\begin{aligned} TOC_j(p) &= TOE_j(p) + \tau_{dj}(p) \\ &= T_{sj}(p) + \tau_{wj}(p) + \tau_{dj}(p) \end{aligned} \tag{12}$$

where $\bar{u}_j$ is the static utility, λ is real number between zero and one which exponentially discounts the utility, *TOC* is the time of completion, *p* is path list of the fleet, and t_0 is the time origin of task. The *TOC* is the sum of time of execution, *TOE*, and duration time τ_d of task-*j*, and the *TOE* of p-th target task of agent-*i* is sum of reaching time, T_s, on that task and waiting time interval τ_w, for the other agents. All time variables are determined by the path of entire fleet, so that, the total utility regarding to task-*j* is function of the path of fleet. Each task is weighted by static utility along the priority of it. The static utility should be a nonnegative value, and the utility is discounted exponentially as the *TOC* increases.

III. Group Buying Algorithm

A. *Centralized Group Buying Algorithm (CGBA)*

The cooperative multiple task assignment problem with hardly constrained order is harder problem than a multiple task assignment problem that is handled by other researches, and until now a complete enumeration survey is the only way to find the global optimal solution, However, its calculation time increases exponentially depending on the problem size. It is because that it is impossible to transform a target problem into a multi-dimensional multiple knapsack choice problem (MMKP), as like in RHTA or petal algorithm did in [6]-[8]. Hence a suboptimal algorithm is practically useful.

This article proposes such a greedy heuristic algorithm, named as group buying algorithm (GBA). Group buying is also called as collective buying or group purchasing. It is originated from Chinese tu ángòu, and now many online group-buying web servers exist. A centralized group buying algorithm (CGBA) is depicted in the algorithm 1. In the CGBA, each agent sequentially collects a task in greedy. It is similar to sequential greedy algorithm (SGA) of Choi [8]. Along the algorithm, it results in N_{min} task-agent pairs. When the ranking flag, $\mathrm{F}_{\mathrm{rank}}$, is true, it sorts a fake marginal utility $\hat{u}_{ij}$ for each task-*j* in descending order, to find out the $n_{min,j}$-th best task-agent pair. From here, this paper will just say the fake marginal utility as the marginal utility, and it is defined as,

$$\hat{u}_{ij}[b_i] = \max_{n \le |p_i|+1} \tilde{U}^{p_i \oplus_n \{j\}} - \tilde{U}^{p_i} \tag{13}$$

$$\tilde{U}^{p_i} = \sum_{p_i}^{n(p_i)} \bar{u}_{p_i} \lambda_{p_i}^{TOC_{p_i} - t_0} \tag{14}$$

where $\tilde{U}^{p_i}$ is the expected utility by agent-*i* , when it follows its path, p_i. The agents calculate the marginal utility $\hat{u}_{ij}$ as if it can perform the task-*j* right after it reaches the task, without waiting the other agents. That is, the agent considers itself as a *critical agent* for the task-*j*. On the other hand, for the other tasks in its path list, it just calculates the expected utility with the previous assignment's waiting time. For these reasons, this marginal utility is not the actual marginal utility, but it is the best way of prediction, as the actual utility is function of the path of all fleet, not only a single agent. For instance, Consider that there are an agent with a path list of {1 3 4}. Although, the marginal utility of the agent for task-*j* is expected when the task is performed right after task 3, namely, when the agent has the path list of {1 3 *j* 4}, the task-*j* might not be assigned by others, and the expected utility for task 4 might changes because of task *j*. Thus, here at first, the algorithm gives the fake marginal utility, which could be gained when it is actually critical agent, and the new assignment doesn't affect the later tasks in path list. To amend this rough expectation, the waiting time is recalculated with the actual assignment, later.

By setting the static utility of any task-*j* as proportional to $n_{min,j}$, it gives an incentive on cooperative tasks which is suitable to common sense; If several men works on a cooperative job, the job should be more profitable than any non-cooperative jobs in option. In addition to that, giving an incentive on a task with big $n_{min,j}$, results in a higher priority on the task, so that, the algorithm can avoid a local best strategy. For example, consider that there are 4 tasks and 3 agents, which are capable of three tasks each. The tasks require $n_{min,j}$ agents, respectively, where it is {2 2 2 3}. Therefore, the optimal number of task-agent pair is 9, when all agents involve in task 4, and share task 1 to 3 with proper combination. However, If the algorithm sequentially assign two agents to the task 1, 2 and 3, the remaining agent cannot proceed the task 4 alone, as the task require two agents to do. Thus, giving incentive to the task requiring many agents can hinder the tasks from being assigned to improper agents.

If regarding path of the $n_{min,j}$ agents, which has larger marginal utility than the other, are expected to suffer from cross deployment, their marginal utility is recalculated with the new path constraints, $\tilde{p}$, on the line 11. The cross deployment will be handled on next section.

When all cross deployment paths are resolved, this procedure finds the task, j_n^*, that gives the $n_{min,j}$-th best marginal utility among the whole task set, and corresponding agents are i_n^*. During the next $n_{min,j}$ loops, i_n^* agents get assigned to the task j_n^*, by adding this task to their bundle. If an agent is exhausted, the agent is removed from agent list I. If a task is perfectly assigned, the algorithm gets rid of it from task list J. Then, it updates waiting time, τ_w, of all agents on each tasks in their path lists, and the marginal utility is updated as if the agent-*i* is critical agent to the task-*j*, same as line 4.

Algorithm 1. Centralized Group Buying Algorithm

```
1:  $b_i = \{\varnothing\},\ p_i = \{\varnothing\},\ \eta_i = 0,\ \forall i \in I$
2:  $n'_{\min j} = n_{\min j}\qquad \forall j \in J$
3:  $N_{\min} = \min\{\sum_{i=1}^{N_u} L_i, \sum_{j=1}^{N_t} n_{\min j}, N_{poor}\}$
4:  $\hat{u}_{ij}^{(1)} = \hat{u}_{ij}(\{\varnothing\})\ \ \forall (i,j) \in I \times J$
5:  F_rank = true
6:  for n = 1 to $N_{\min}$ do
7:      if F_rank = true
8:          for $j \in J_n$
9:              $(i_j, s_j) = \mathrm{rank}_j^{n_{\min,j}}(\hat{u}_{ij}^n)$
10:             if $\tilde{p}_c = p^{(n)} \cup p_{i_j}$ is cross deployed
11:                 $u_{ij}^n = \tilde{u}_{ij}(b_i^{(n)}, \tilde{p}_c)\ \ \forall i \in I_{n+1}$
12:                 go to 8
13:             end if
13:         end for
14:         $j_n^* = \arg_j \max_j \min_i s_j \quad \forall j \in J_n\ \forall i \in i_j$
15:         $i_n^* = i_{j_n^*}$
16:         F_rank = false
17:     end if
18:     $i_n^* = i_n^*(n'_{\min j_n^*})$
19:     $n'_{\min j_n^*} = n'_{\min j_n^*} - 1$
20:     $\eta_{i_n^*} = \eta_{i_n^*} + 1$
21:     $b_{i_n^*} = b_{i_n^*} \oplus_{\text{end}} \{j_n^*\}$
22:     end if
23:     if $\eta_{i_n^*} = L_i$ then
24:         $I_{n+1} = I_n \setminus \{i_n^*\}$
25:         $u_{i_n^*, j}^{(n+1)} = 0 \quad \forall j \in J$
26:     else
27:         $I_{n+1} = I_n$
28:     end if
29:     if $n'_{\min j_n^*} = 0$ then
30:         $J_{n+1} = J_n \setminus \{j_n^*\}$
31:         $\tau_w(i,j) = U_T(p)\ \ \forall (i,j) \in I_{n+1} \times J_{n+1}$
32:         $\hat{u}_{ij}^{(n+1)} = \hat{u}_{ij}[b_i^{(n)}]\ \ \forall (i,j) \in I_{n+1} \times J_{n+1}$
33:         F_rank = true
34:     else
35:         $J_{n+1} = J_n$
36:     end if
37: end for
```

B. Cross Deployment Resolving.

If path lists of some agents that involve in mission are mutually dependent in order, then it is defined as *cross deployment*. For example, consider that agent #1 has path list {1 2 3}, and agent #2 has path list {3 2}. To do task 3, agent #1 should perform task 2 first. On the other hand, in the point of view of agent #2 to perform task 2, it has to perform task 3 first. This conflict can appear indirectly. Consider that the case agent #1 has path {1 2}, agent #2 has path {2 3}, and that of agent #3 is {3 1}. In this case, to perform task 3, agent 1 and agent 2 should perform task 1 and task 2. But for agent 3 to do task 3, it has to do task 3 first.

To check whether cross deployment exists or not, this work introduces a checking process in algorithm 2, which uses before & after task list. Along the path list of every agent, it builds and updates the before and after list of each task. When it updates some task A to before list of task B, it also updates task A to the before list of all tasks that are in the after list of task B. It is in the same manner for updating task A to after list of task B. If a task exists in both of before & after lists of a task, it means that there is a cross deployed assignment.

Algorithm 2. Checking Cross Deployment

```
1:  function
        Input $p_i\ \forall i \in I$, paths of all agent.
        Input $B\{j\}\ \forall j \in J$, before task lists of all tasks
        Input $A\{j\}\ \forall j \in J$, After task lists of all tasks
2:  for i = 1 to $n(I)$
3:      for z = 1 to $n(p_i)$
4:          $j_{this} = p_i(z)$
5:          $T_b = p_i(1:z-1),\ p = n(T_b)$
6:          $T_a = p_i(z+1:n(p_i)),\ q = n(T_a)$
7:          if $\exists q'\ s.t. T_a(q') \in B\{j_{this}\}$ for $q' < q$
            or $\exists p'\ s.t. T_b(p') \in A\{j_{this}\}$ for $p' < p$
8:              return true.
9:          end if
10:         if $T_b(p) \notin B\{j_{this}\}$
11:             $B\{j_{this}\} \oplus_{\text{end}} T_b(p)$
12:             if $T_b\{p\} \notin B\{a\}$ for $\forall a \in A\{j_{this}\}$
13:                 $B\{a\} \oplus_{\text{end}} T_b(p)$
14:             end if
15:         end if
16:         same manner for after list (line 10 to 15)
17:     end for
18: end for
```

On the line 10 of algorithm 1, this checking algorithm is used to check the cross deployment. Also on the line 11 of algorithm 1, the function $\tilde{u}_{ij}(b_i^{(n)}, \tilde{p}_c)$ calculate the marginal utility, that doesn't violate the cross deployment constraints. In the line 11 of algorithm 1, the path candidate $\tilde{p}_c$ is path with the previous assignment and the candidate of critical agent of this assignment. As CGBA searches the $n_{\min j}$ th best marginal utility, this recalculation changes the marginal utility and corresponding path of the agents from the best to $n_{\min j}-1$ - th best candidate for the task-*j*.

TABLE I. SIMULATION SCENARIO

CASE 1
AGENT 5 – TASK 4. Required agents of TASK to operate {2 3 2 2}. Each agent can take 3 tasks, respectively, throughout CASE 1 and 2.
CASE 2
100 times Monte-Carlo simulations AGENT 3 – TASK 1 to 100 varying Positions of tasks and agent, static utility are random. Required number are random between 1 to 3.
CASE 3
Massive assignment scenarios, AGENT 10 - TASK 20 Each agent can take 5 tasks, respectively. Other specific numbers are omitted.

IV. Simulations

A. Simulation Scenarios

To show the applicability and the performance of proposed CGBA, authors simulated the task assignment scenarios in Table I. The simulation is done on a dual core 2.1 GHz CPU with Ubuntu 12.04, and MATLAB 2011.

B. Simulation Results

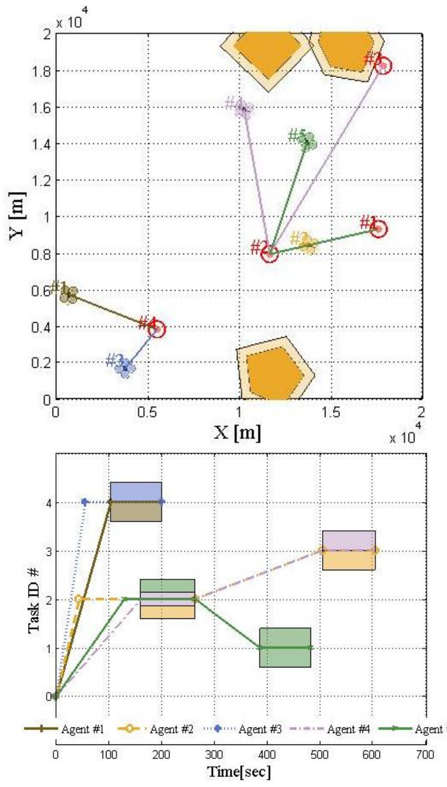


Figure 1. Case 1 (up) agent path (down) agent assignment time widow

The case 1 simulation is moderately small size problem. As the five agents can take 3 tasks, respectively, and the four tasks need 9 task-agent pairs, all the tasks are performable with this fleet. In Fig. 1 (up), it shows the paths of all agents. The orange polygons are obstacles. In Fig. 1 (down), the assignment result is depicted over the time window, where y-axis is for Task ID. The boxes in the graph mean that the agents are doing the assigned tasks, and the horizontal lines between boxes represent waiting time of each agent.

In the case 2, a Monte-Carlo simulation is done, with randomly chosen 3 agents and tasks of varying number. In Fig. 2, as the number of task increases, elapsed time increases linearly, and the performance seemed to be saturated. As the multiple cooperative task assignment is NP-hard class problem, it will suffer the curse of dimension. However, CGBA gives a feasible decision in polynomial time. The performance saturation is a general phenomenon, because the agent number is fixed, while the number of task is increased. Thus at a glance, the service provider have to balance the agents number with respect to the number of tasks, to get an efficient mission result.

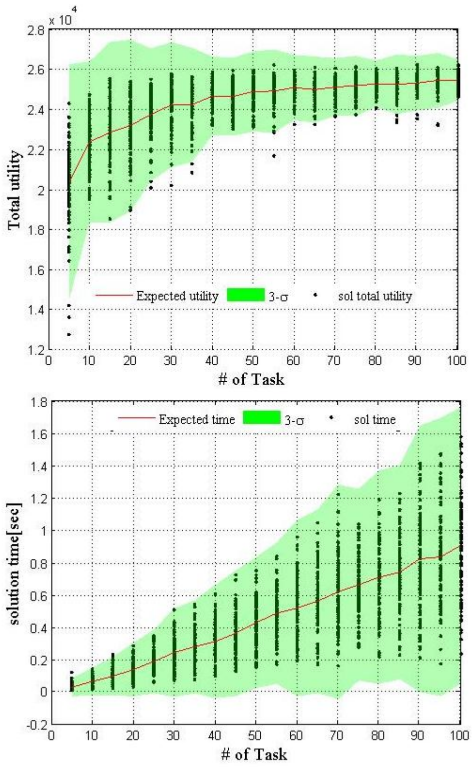


Figure 2. Case 2, Monte Carlo Simulation, (up) performance saturation, (down) polynomial time solution.

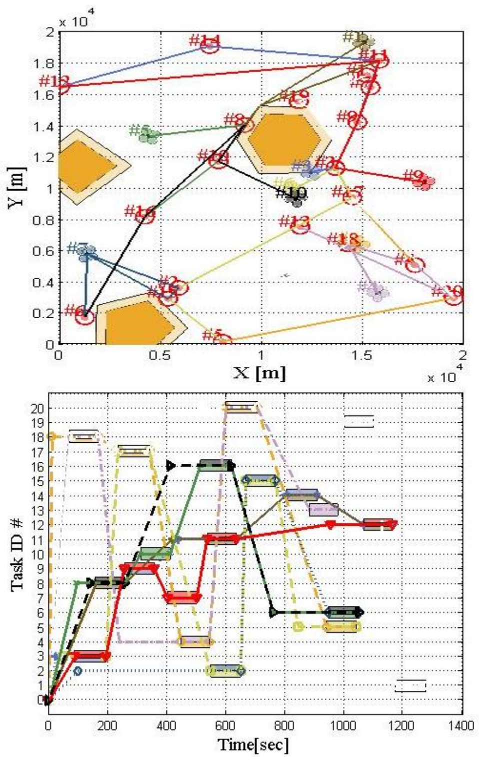


Figure 3. Case 1 (up) agent path (down) agent assignment time widow.

The case 3 is a massive task assignment problem. It contains 20 tasks, and 10 agents. The assignment result is shown in Fig. 3. This scenario is solved within 3.2 second, although the problem size is such big. As the CGBA gives solution within polynomial time, it is very useful for the practical situations as well.

V. CONCLUSION

This paper presented a centralized suboptimal task assignment algorithm for multiple- cooperative task assignment problem. The suggested algorithm resolves the cross deployment issues with before and after task lists, and successively allocates the agents to a proper task. To be more precise in calculating the marginal utility, it updates the waiting time for a given task in each loop. This algorithm can be used for other applications; robotic slung-load transport, UAV air to air combat algorithm, and so on. The task assignment simulation is done to check the elapsed time and utility performance of CGBA.

ACKNOWLEDGMENT

This research was supported by Defense Acquisition Program Administration and Agency for Defense Development under the contract UE124026JD.

REFERENCES

[1] S. Y. Byeon, W. S. Lee, and H. C. Bang, "Task assignment for forest fire suppression by multiple UAVs," *Journal of Mechanics Engineering and Automation*, pp. 65-70, March 2013

[2] G. H. Moon, C. H. Lee, D. W. Yoo, and M. J. Tahk, "Con-sensus based bundle algorithm for air to air weapon assignment," presented at the JSASS Western Branch Conference 2012, Fukuka, Japan, Nov 2012.

[3] G. H. Moon, C. H. Lee, D. W. Yoo and M. J. Tahk, "Decetraized cooperative control algorithm for JSEAD mission," presented at the 2012 Asia-Pacific International Symposium on Aerospace Technology, Jeju, Korea, Nov 2012.

[4] A. K. Whitten, "Decentralized planning for autonomous agents cooperating in complex missions," M.S. Thesis, Massachusetts Institute of Technology, Massachusetts, United States of America, 2010.

[5] H. L. Choi, A. K. Whitten, and J. P. How, "Decentralized task allocation for heterogeneous teams with cooperation constraints," presented at the 2010 American Control Conference, Marriott Waterfront, Baltimore, MD, USA, 2010.

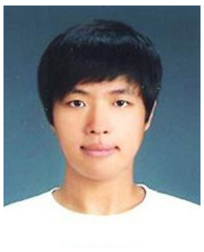
[6] H. L. Choi, L. Brunet, and J. P. How, "Consensus-based decentralized auction for robust task allocation," *IEEE Trans. on Robotics*, vol. 25, no. 4, pp. 912-925, 2009.

[7] J. Bellingham, M. Tillerson, A. Richard, and J. P. How, "Multi-task allocation and path planning for cooperative UAVs," in *Cooperative Control: Models, Applications and Algorithms,* Boston, MA: Kluwer, 2003, pp. 23-41.

[8] M. Alighanbari, "Robust and decentralized task assignment algorithm for UAVs," Ph.D. dissertation, Massachusetts Institute of Technology, Massachusetts, United State of America. 2007.

Gun-Hee Moon was born in Seoul, Korea, in 1990. He received the B.S. degree in aerospace engineering from the Inha University, Incheon, Korea, in 2012. He is currently pursuing the M.S. degree in aerospace engineering at Korea Advanced Institute of Science and Technology, Daejeon, Korea. His current research interests include decision making of multi agents for cooperative tasks and, guidance and control of unmanned system.

Dong-Wan Yoo was born in the Seoul, Korea in 1984 and received B.S. degree from Pennsylvania State University, University Park PA, USA and the M.S. degree from Korea Advanced Institute of Science and Technology, Daejeon, Korea, all in aerospace engineering. He is currently a Ph.D Student of Aerospace Engineering at Korea Advanced Institute of Science and Technology, Daejeon, Korea. His research interests includes flight dynamics and control.



Byung-Yoon Lee was born in the Seoul, Korea in 1984 and received B.S. degree in electronic and computer engineering from Kyunghee University, Gyonggi-do, Korea, in 2010. He is presently a Ph.D. Student of Aerospace Engineering at Korea Advanced Institute of Science and Technology, Daejeon, Korea. His research interests include flight control system design and cooperative control system design of multiple UAVs.

Hae-In Lee was born in Busan, Korea in 1991 and received B.S degree in aerospace engineering from Korea Advanced Institute of Science and Technology, Daejeon, Korea, in 2013. She is presently a Master's Student of aerospace engineering at Korea Advanced Institute of Science and Technology, Daejeon, Korea. Her research interests include flight control design and guidance.

Min-Jea Tahk was born in South Korea in 1954 and received the Ph. D. degrees in aerospace engineering from the University of Texas at Austin in 1986. He is currently a Professor of the department of aerospace engineering, Korea Advanced Institute of Science and Technology, Daejeon, Korea. His research areas include flight control, parameter optimization, modeling and simulation.