29th Congress of the International Council of the Aeronautical Sciences
St. Petersburg, Russia; September 7-12, 2014

MULTILE TASK ASSIGNMENT FOR COOPERATIVE TRANSPORT SYSTEM WITH GROUP BUYING APPROACH

Gun Hee Moon*, Dong Wan Yoo*, Byung Yoon Lee*, Hae In Lee* and Min Jea Tahk*
***Korea Advanced Institute of Science and Technology**

Keywords: *slung load system, multiple cooperative task assignment, group buying, UAV*

Abstract

This paper addresses the task assignment algorithm for the multiple cooperative load transportation system. The centralized group buying algorithm is a heuristic security strategy algorithm based on the group buying market. Through the simulation, the algorithm the sub optimality and applicability is shown.

1 Introduction

The slung load type load transportation is recently being researched [1-2]. The helicopter or the quadrotor has a possibility to be used as slung load system. As the quad rotor system is useful in the quick package delivery service, the amazon.com recently announced a plan for the quad rotor 30 minutes home delivery service.

To utilize these kind of service efficiently, the high level task scheduling is necessary. As the system size is getting bigger, the need for the task assignment increases more. Choi, and K. Whitten has introduced the CBBA and the CCBBA algorithm [3-5]. It is a consensus based auction algorithm, that assigns the task to the agents sequentially. The CBBA guarantees 50% optimality and the fast convergence.

This author have wrote a couple of papers related to the group buying algorithm. [6-7] The group buying algorithm inspired from the collective buying market. In the following section, this paper, to solve the task assignment of the load transportation problem, formulated the problem as the multiple cooperative task assignment problem, and introduces the group buying algorithm.

2 Problem Statement

2.1 Cooperative Transport Mission

The cooperative transport mission by swarm UAVs is one of the promising usage of the quadrotor robot. As the technology of the UAV system improves, the cost of UAV system is reduced, and the reliability is increased. In a close future, the artificial intelligence and the advanced control theories would help the UAV to operate on these kind of complex mission. In the cooperative transport operation, a group of quadrotors or any kinds of unmanned rotary aerial vehicle cooperates simultaneously to transport a package to somewhere.

In many area area, cooperative transport mission can be utilized for, such as, the immediate munition deployment, the package delivery service, and so on. In Fig. 2.1, the cargo is supported to each UAV through a wire, and thus the swarm robots can carry a heavy package that exceeds the payload capacity of each UAV. As the group of a single type agents can carry various weights of the payload, the agent can be standardized, and thus the manufacturing and maintenance cost of the entire system also can be reduced.

In the actual application, there would be multiple delivering request, and multiple agents which can perform the requested service. If at least n_{min} agents are needed to shift a cargo, the agents should gather to initiate the mission, that is, an early arrival agent ought to wait the other n_{min}-1 agent near the task. Namely, *the mission*

is constrained by the number of agent to initiate it. This constraints causes the forbidding of *the cross deployment*, which is discussed later. Generally the n_{min} is related to the payload capacity of each agent, as following,

$$n_{\min} = \left\lceil \frac{W}{w_{\max}} \right\rceil \quad (1)$$

where W is the weight of the payload, and w_{max} is the payload limits of an agent. Here all the agents are assumed to be same as. The operator $\lceil\ \rceil$ is the round up operator.

The cooperative transport mission is a non-synergetic mission. If the agents more than n_{min} are assigned to a task, there would be no advantage, compared to the case that only n_{min} agents involves in. Those agents are actually redundant agents, it might help the service to be more reliable, but unnecessary. Therefore if there exists other tasks, assigning the redundant agents to the other task is more profitable. Thus the mission planning of these system is required for the autonomous service system in the future and it can be formulated into the multiple, cooperative task assignment (MCTA) problem, which is discussed in the following section.

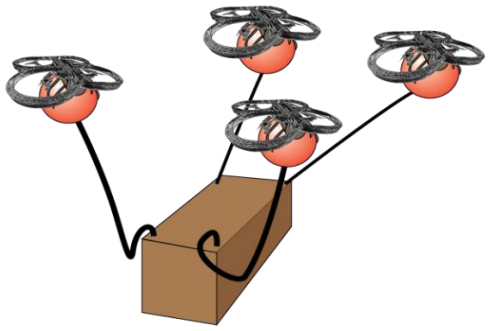

Figure 2.1 Quadrotor Cooperative Transport Mission concept, carrying a cargo in a slung load type.

2.2 Multiple Cooperative Task Assignment

In this section, this paper formulate the MCTA problem mathematically. The MCTA is an operation scheduling problem that assigns N_u agents to N_t tasks as many as possible. Here the satisfaction of the society is expressed by the utility function, and the purpose of the task assignment is maximizing the summation of the utility, *global utility*, then MCTA problem for the cooperative transportation mission can be expressed as,

$$\arg_{\mathbf{P}} \max \sum_{i=1}^{N_u} \sum_{j=1}^{N_t} u_{ij}(p_i) x_{ij} \quad (2)$$

Subject to

$$\Sigma_i^{N_u} x_{ij} = n_{\min,j} \text{ or } = 0 \quad \forall j \in \{1,2,\ldots N_t\} \quad (3)$$

$$\Sigma_j^{N_t} x_{ij} \le L_t, \qquad \forall i \in \{1,2,\ldots N_u\} \quad (4)$$

where x_{ij} is the decision variable which indicates agent i is assigned to task j, when agent i is allocated to task j, the corresponding decision variable x_{ij} is '1,' and if not allocated, it is '0.' u_{ij} is the utility value or score that is earned by for agent i performing task j along with the path list of agent i. Thus the meaning of Eq. (2) is finding the best path strategy of the fleet that can maximize the summation of the utility of whole society. In Eq.(3), the left hand side of the equation is j-th column sum of decision variable. It means the number of agent that involves in task j is only $n_{min,j}$ or zero. It implies that the cooperative transport mission has the non-synergetic property, so that, constrains the number of the agent assigned for the task to be no more than or less than $n_{min,j}$ when the mission is taken, and to be zero when mission is abandoned. In Eq.(4), the left hand side of equation is row sum of the decision variable and it means the number of task assigned to agent i. Thus each agent can't involve in more than L_t tasks in a sortie. This constraint is a simple modeling of agent fuel capacity.

In order to get the best result from the assignment, one have to fill in '1' as many as possible. The task assignment is said to be ideally completed, when there is no better agent-task pairs than now. Define the N_{min} as Eq.(5).

$$\Sigma_{i=1}^{N_u} \Sigma_{j=1}^{N_t} x_{ij} \triangleq N_{\min} \quad (5)$$

where N_{min} is the total number of the assigned agent-task pair. When all agents to perform all task is assigned, all agents are exhausted, or no more task-agent pair is feasible even though there left task and available agent both, then the assignment said to be completed. It is mathematically expressed as Eq.(6),.

$$N_{\min} = \min\{\sum_{\forall j} n_{\min j}, \sum_{\forall i} L_i, N_{poor}\} \quad (6)$$

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

In Eq.(6), the first value, the sum of whole $n_{min,j}$ is the number of all agent to perform all task. The second one, sum of all L_t of entire fleet, is the number of task-agent pair that the fleet can support. The third element, N_{poor} indicates *poor situation*. where N_{poor} is defined as Eq. (7), that is, the biggest sum of $n_{min,j}$ which is less than or equal to the sum of the L_t. In this third case there exist both remaining tasks and redundant agents, which are not enough to perform the remaining task. In Eq.(7), if the equality holds, the poor case is not in the problem setup. For instance, if there are three agents, which can take participate in three task respectively, and five tasks, which require two agents to be initiated, and the agents evenly allocated to the five tasks, then there always left one task and one agent. Thus the possible maximum task-agent pair is eight, not nine.

The utility function for cooperative transportation mission has an exponential discounted scheme along the time of the task completion.

$$u_{ij} = \bar{u}_j \lambda_j^{TOC_j} \quad (8)$$

The utility, u_{ij}, gained by agent i performing task j is proportional to the static gain, $\bar{u}_j$, and discounted by the time of completion TOC_j within the rate of the discounting factor λ_j. This time discounted utility model makes the agents to complete the tasks as soon as possible to get more utility. Additionally, this utility function results in the monotonically increasing function along the task number. However for the MCTA, it is hard to say that it holds the sub modularity. Namely the marginal utility function might not monotonically decrease as it does in the multiple assignment problem of Choi [4].

3 Group Buying Algorithm

3.1 Group Buying

The group buying is also known as collective buying or group purchasing. It is originated from the Chinese "Tuángòu", which gives a special discount on an item, when a group of purchasers order to buy the same item. The group buying is beneficial to both the retailer and the client. As the retailer sales goods in discounted price, the clients get some incentive on it. Simultaneously, the retailer can extend the market area, thus it is profitable for them too, although they sales it in cheap price.

The group buying market has the *curse of winner* problem, which is a natural born phenomenon that the earlier mover or the customer of appetence should wait the others until the minimum required number of agents is ready. In the sense of the utility of the given problem, the utility of those urgent customers are discounted along the time, and they have to take the loss of the utility.

This paper shortly discusses about the optimal solution of the MCTA problem in the following chapter. In consequently, the only ways to find the optimal solution is the complete enumeration, which enumerates all the possible solution and test all the cases to find best input. Unfortunately, this complete enumeration takes the exponential time to find proper solution. In a practical manner, thus this paper are going to introduce a heuristic suboptimal algorithm.

The centralized group buying algorithm (CGBA) finds a feasible solution for the MCTA problem. The CGBA is inspired from the property of the group buying market. It decides the schedule of each agent sequentially, and the agents has to take the loss of the utility caused by the winner's curse. But the proposed algorithm finds as best one as possible, within the safety strategy.

3.2 Centralized Group Buying Algorithm

Similar to the sequential greedy algorithm of Choi [4], the centralized group buying algorithm decides the task to perform sequentially. Algorithm 3.1 describes the CGBA process. Each agent initialize the task bundle, $\mathbf{b}_i$, path list $\mathbf{p}_i$ and the best marginal utility $\hat{u}_{ij}$ through line 1 to 5. $n_{\min,j}$ is a dummy variable

that contains the number of agents needed for each task. In line 3, the process calculates the number of task-agent pair, which is equal to Eq. (7). For the number of task-agent pair to be closer to the optimal solution, the global utility of the whole fleet need to hold the monotonically increasing function scheme. Literally, it means that eating more pies is always better than eating fewer, and it makes sense for the greedy individual, or the homoeconomicus.

Through line 6 to line 36, each task is matched with a group of agents of the entire fleet. It goes through this loop for N_{min} times. The CGBA estimates the best marginal utility $\hat{u}_{ij}$ which is an expected maximum marginal utility for agent i to be earned by performing task j between its path list, with the assumptions that new assignment doesn't affect the waiting time on the other task, and newly assigned task can be initiated just in time. Therefore, the marginal utility calculated here is a pseudo value.

$$\tilde{U}^{\mathbf{p}_i} = \sum_{j}^{n(\mathbf{p}_i)} \bar{u}_{p_i(j)} \lambda_{p_i(j)}^{TOC_{p_i(j)} - t_o} \tag{9}$$

$$\hat{u}_{ij}[\mathbf{b}_i] = \max_{n \le |\mathbf{p}_i|+1} \tilde{U}^{\mathbf{p}_i \oplus_n \{j\}} - \tilde{U}^{\mathbf{p}_i} \tag{10}$$

where the pseudo global utility $\tilde{U}^{\mathbf{p}_i}$ is the summation of the utility rewards from all of the individual tasks, and it is function of the path of the agent i. Then the pseudo marginal utility is obtained as Eq. (10). Here the operator $\oplus_n$ means inserting following list into the n-th position of the preceding list. Thus it is an expectation of agent i to be earned by performing task j on its best chance.

In line 9, these marginal utility is ranked for each task. Additionally algorithm arranges the utility in the descending order, and point out first to $n_{min,j}$-th higher scoring agents $\mathbf{i}_j$ and corresponding utility values $\mathbf{s}_j$. Especially the $n_{min,j}$-th agent is called as the critical agent. The critical agent are expected to arrive the task at last so as to it results in the $n_{min,j}$-th highest utility. As the other agents has to wait the critical agent, *the winner's curse*, even those are arrived the task early, the utility rewards is determined by the critical agent.

To avoid the cross deployment issue, which is discussed on the following section, the algorithm check the cross deployment with the fleet's original path and the path of agents $\mathbf{i}_j$. If newly updated path is cross-deployed case, the marginal utility is updated, taking the path of $n_{min,j}$-th agent of agents list $\mathbf{i}_j$ as the pivot points of the path candidate. That is, without changing the order of the path of the agent, only changes the path order of the other agents, it calculates the marginal utility again. Then it goes to line 9 again.

After this refining loops, one have a tables of the pseudo marginal utility. Then it finds the argument j_n^* which is the task of the maximum of the minimum of the $n_{min,j}$-th best marginal utility between each task j. This is a kind of safety strategy to reduce the impact of the curse of the winner, by assigning tasks mainly based on the critical agent. Then the $n'_{\min j_n^*}$-th agent of agents $\mathbf{i}_n^*$ get assigned task j_n^*, as line 21. Thus agent $\mathbf{i}_n^*$ is assigned to the task j_n^* by inserting task j_n^* to the bundle list and the path list of the agent. As one agent is allocated to the task, the required number of agent, $n'_{\min j_n^*}$, decreases by one, and the number of task assigned to the agent, $\eta_{i_n^*}$, increase by one. When the agent is fully exhausted, it is removed from the agent pool, I_n, as line 22 to 24. Likewise, when the task is completely assigned, the task is removed from the task pool, J_n, and the global utility and the marginal utility is updated with the new path list of the fleet. As the utility is the function of the complete time, the *TOC* of fleet, τ_s, is obtained, when the global utility is calculated. Actual the marginal utility of the fleet is possible to calculate, only after the path of entire fleet is fixed.

In the multiple group buying market, each retailers willing to attract more customer. In order to that, there should be more incentives on the task. In the CGBA, to avoid the agents to be assigned to the non-cooperative task first, which leads to redundant agent-task pair, the incentive strategy might be useful. By giving incentives on the static utility of task j, which requires $n_{min,j}$ agents, as following,

$$\bar{u}_j \propto n_{\min,j} \tag{11}$$

it leads to the cooperative task is more attractive than other, and avoids the redundant pairs.

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

Algorithm 3.1 Centralized Group Buying Algorithm.

3.2 Cross Deployment

If the path lists of some agents, which are mutually dependent on performing a task, has conflicts in the order of the tasks then it is defined as the *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. The conflict can appear indirectly. Consider that the case agent 1 has path {1 2}, agent 2 has path {2 3}, and agent 3 has {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 1, the agent has to do task 3 first, thus a conflict rises.

3.2.1 Definition of Cross Deployment

Mathematically the cross deployment can be defined as following. Definition 6.1 and 6.2 define direct cross deployment (D. C. D.) and indirect cross deployment (I. C. D), respectively.

Definition 6.1, direct cross deployment
$D.C.D. \Leftrightarrow$

$$\exists i_q \ s.t.\ \mathbf{p}_{i_p,m} = \mathbf{p}_{i_q,g} = j_A \text{ and } \mathbf{p}_{i_p,n} = \mathbf{p}_{i_q,f} = j_B \quad (12)$$
$$\text{for } m < n \text{ and } f < g$$

Definition 6.2 indirect cross deployment
$I.C.D. \Leftrightarrow$

$$\mathbf{p}_{i_p,m} = j_A \text{ and } \mathbf{p}_{i_p,n} = \mathbf{p}_{i_q,x} = j_B, \mathbf{p}_{i_q,y} = j_C$$
$$\text{for } m < n,\ x < y \text{ and } i_q \in \mathbf{i} \setminus \{i_p\} \quad (13)$$
$$\exists i_r \ s.t.\ \mathbf{p}_{i_r,f} = j_C \text{ and } \mathbf{p}_{i_r,g} = j_A \text{ for } f < g$$

It is direct cross deployment if there exist agent i_q such that, when agent i_p performs path m earlier than path n and agent i_q performs path f earlier than path g, the m-th path list element of the agent $\mathbf{p}_{i_p,m}$ and the g-th path list element of agent i_q $\mathbf{p}_{i_q,g}$ are same as task j_A, and the n-th element of the path list of the agent i_p $\mathbf{p}_{i_p,n}$ and the f-th element of the path list of the agent i_q $\mathbf{p}_{i_q,f}$ are same as task j_B. In a similar manner, it is indirect cross deployment, if agent i_r exists such that the agent has task j_C and task j_A in the path list of f-th and g-th element, here path f is ahead the path g, when agent i_p has task j_A in the m-th element of the path, task j_B is the n-th element of the path of agent i_p and the x-th element of the path of agent i_q simultaneously, and task j_C is the y-th element of the path of agent i_q. Here path m proceeds path n, path x proceeds path y.

Fig. 3.1 shows two examples of the DCD case and ICD case. In Fig. 3.1 (a) the two agents

has mutually dependent task list. Agent 1 should perform task 1 2 and 3, and agent 2 show task in the order of 2, 4, and 1. Then the two agent cannot proceed the task forever, unless the task order is changed. The two agents are directly dependent each other as so it is DCD case. In Fig. 3.1 (b), agent 1, 2 and 3 are involved in the mission. Agent 1 should perform task 1 and 2, agent 2 should perform tasks in the order of task 2, 5 and 3, and the final agent should perform tasks in the order of 3, 4, and 1. In this case, none of the tasks are directly dependent each other, but the task 1 is directly depends on task 3, and task 3 is indirectly depends on task 1 through the task 2, thus it is ICD.

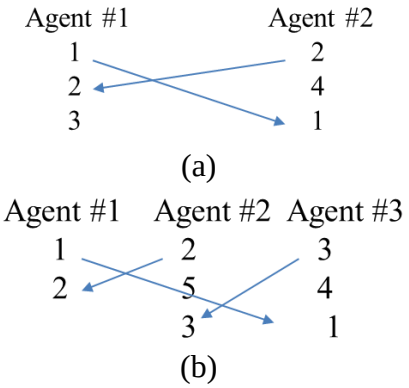

Figure 3.1 D.C.D ex (a), I.C.D. ex (b)

3.2.2 Cross Deployment Discriminant

As the cross deployment causes the logical error on the mission schedule, it should be avoided. In order to find out whether the given task schedule of the fleet is cross deployed or not, this paper introduces a discriminant for the cross deployment of the MCTA. The algorithm 3.2 defines the preceding task list (PTL), **B**, and the following task list (FTL), **A**, for all of the tasks in the path list of the agents. In the PTL, the algorithm collects the preceding task, and in the FTL it collects the following task of each task.

The algorithm build a PTL and a FTL for all task j that is assigned to any agent, and if PTL and FTL of the tasks are already build up, it is given as input of the algorithm. For each agent, from the first task to the last task in the list, it figure out the temporal preceding task list, T_b, and temporal following task list, T_a, as line 5 to 6. If any task in T_a exists in PTL of the z-th task of the agent i, j_{this}, or any task in T_b exists in FTL of task j_{this}, it is determined as cross deployed, as line 7 to 8. With this determinant, any case of the DCD or ICD can be detected.

If the condition isn't satisfied, the T_a and the T_b is updated to the PTL and the FTL respectively. For the T_b, when the element of T_b is out of the PTL of task j_{this}, the element is added to the PTL of task j_{this}. And simultaneously, the task of the T_b is updated to the PTL of the tasks that already follows the task j_{this}. It is logical that any preceding task to task j_{this}, also precedes the following tasks of task j_{this}. In a similar manner, the elements of T_a is updated to the FTL.

1: **function**
Input $\mathbf{p}_i \ \forall i \in I$, paths of all agent.
Input $\mathbf{B}\{j\} \ \forall j \in J$, (P.T.L.)
Input $\mathbf{A}\{j\} \ \forall j \in J$, (F.T.L.)
2: **for** $i = 1$ to $n(I)$
3: **for** $z = 1$ to $n(\mathbf{p}_i)$
4: $j_{this} = \mathbf{p}_{iz}$
5: $T_b = \mathbf{p}_{i\,(1:z-1)}$, $p = n(T_b)$
6: $T_a = \mathbf{p}_{i\,(z+1:n(\mathbf{p}_i))}$, $q = n(T_a)$
7: **if**
$\exists q' \ s.t. T_{aq'} \in \mathbf{B}\{j_{this}\}$ for $q' \le q$ or
$\exists p' \ s.t. T_{bp'} \in \mathbf{A}\{j_{this}\}$ for $p' \le p$
8: **return true.**
9: **end if**
10: **if** $T_{bp'} \notin \mathbf{B}\{j_{this}\}$ for $p' \le p$
11: $\mathbf{B}\{j_{this}\} \oplus_{\text{end}} T_{bp'}$
12: **if** $T_{bp'} \notin \mathbf{B}\{a\}$ for $\forall a \in \mathbf{A}\{j_{this}\}$
13: $\mathbf{B}\{a\} \oplus_{\text{end}} T_{bp'}$
14: **end if**
15: **end if**
16: **if** $T_{aq'} \notin \mathbf{A}\{j_{this}\}$ for $q' \le q$
17: $\mathbf{A}\{j_{this}\} \oplus_{\text{end}} T_{aq'}$
18: **if** $T_{aq'} \notin \mathbf{A}\{b\}$ for $\forall b \in \mathbf{B}\{j_{this}\}$
19: $\mathbf{A}\{b\} \oplus_{\text{end}} T_{aq'}$
20: **end if**
21: **end if**
22: **end for**
23: **end for**

Algorithm 3.2 Checking Cross Deployment

3.2.3 Cross Deployment Discriminant Algorithm Complexity Study.

In order to analyze the complexity of the cross deployment discriminant, worst case analysis is done. To be sure that the CGBA takes the polynomial time, it is a necessary condition that the cross deployment checking algorithm takes the polynomial time at worst case.

In this paper the comparison operator "==" is assumed as a basic operational load element. Before one starts the analysis, here comes some assumptions for the worst case analysis.

1. There are 'n' numbers of distinguishable tasks in the path lists of the agents. Therefore, $n \le N_t$ holds.
2. n is dominant to the number of agent, N_u, or the capacity of agent, L_i.
3. There are no ambiguously ordered task, that is, all tasks are ordered from task 1 to task n sequentially, so that task j always precede task $j + 1$.
4. All elements in the path list of an agent shouldn't be identical each other.
5. For simplicity, the PTL and FTL is already built at first.

For agent i, the length of the path of the agent is L_i. Thus the z is one of the value between 1 and L_i. Then the numbers of the elements in each temporarily preceding task list and following task list is

$$n(T_b) = z - 1 \tag{14}$$

$$n(T_a) = L_i - z \tag{15}$$

If the task of the z-th element in the path list of agent i, is task j_z, the algorithm inspects if the tasks in T_b exists in the FTL and the other opponent case also. Thus the comparison occurs the operational load as much as,

$$O(T_b \in \mathbf{A}) = (z-1)(n - j_z) \tag{16}$$

$$O(T_a \in \mathbf{B}) = (L_i - z)(j_z - 1) \tag{17}$$

As one of them in Eq. (16) and (17) is true, the procedure terminated, here assume that the checking algorithm is not terminated. Then the T_a and T_b should be updated to the PTL and the FTL. For the update, it has to check whether the element in T_b is already in PTL, and the other case also. So it costs as following,

$$O(T_b \in \mathbf{B}) = (z-1)(j_z - 1) \tag{18}$$

$$O(T_a \in \mathbf{A}) = (L_i - z)(n - j_z) \tag{19}$$

As the PTL and FTL already fully built from the beginning, the cost related to the sub-sequential updates is zero. That is, above two equations are always true, so the loop doesn't go deeper. Then the total complexity of the cross checking algorithm would be,

$$\begin{aligned} O(n) &= \Sigma_{N_u} \Sigma_z {}^{\{O(T_b \in \mathbf{A}) + O(T_a \in \mathbf{B}) \\ &\quad + O(T_b \in \mathbf{B}) + O(T_a \in \mathbf{A})\}} \\ &= \Sigma_{N_u} \Sigma_z (L_i - 1)(n-1) \end{aligned} \tag{20}$$

When L_i of all agent is equal to L, the operational load is,

$$O(n) = N_u L(L-1)(n-1) \tag{21}$$

Therefore the algorithm complexity is first order polynomial to the number of tasks.

3.2 Optimal Solution to the MCTA.

The task assignment problem for the multiple assignment is able to be stated in the form of the multidimensional multiple-choice knapsack problem (MMKP), by transforming the decision variable from the task-agent pair to the path candidates of each agent. When it is turned into MMKP, it have a chance to solve by using MILP solver like CPLEX. However, as the utility of the task of the MCTA is determined by the critical agent, this is mixed integer nonlinear programming problem. By the reason, the only way to the optimal solution of the MCTA is the complete enumeration.

The complete enumeration searches every solution field. Therefore, in the worst case, it has to visit many solution space as much as the number of Eq. (22) to find the optimal solution.

$$\sum_{k=1}^{2^{N_t \cdot N_u}} \left\{ \prod_{i=1}^{N_u} [(\sum_{j=1}^{N_t} x_{ij}(k))!] \right\} \tag{22}$$

However, the most of the solution in the space is infeasible one, due to the constraints, such as the number of agents required to a task, n_{min}, the capacity of an agent, L_i, or the cross deployment. Thus if one can detect the

feasibility of a candidate soon, the speed of complete enumeration could be enhanced. In the [2], there introduced a modified complete enumeration procedure for the MCTA. This paper compare the solution of the CGBA with the optimal solution from the complete enumeration procedure.

5 Simulation and Results

5.1 Load Transportation Simulation

To simulate the load transportation simulation, following 2D particle dynamics are used,

$$\dot{x} = V\cos\psi \quad \dot{y} = V\sin\psi$$
$$\dot{\psi} = \begin{cases} a/V \\ a/10 \quad \text{if } V = 0 \end{cases} \tag{23}$$

For the case 1, the agents are moving in 1 m/s and the problem set as Fig. 5.1.

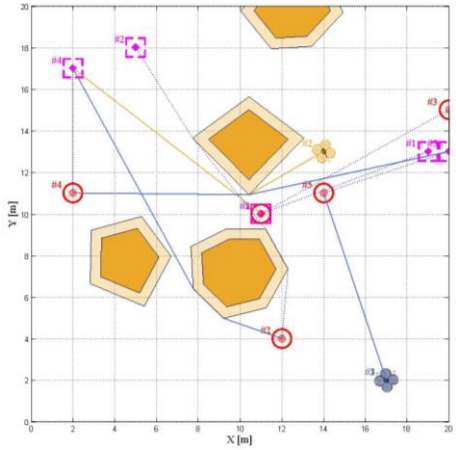

Figure 5.1 Load Transportation Simulation Problem Setup CASE 1

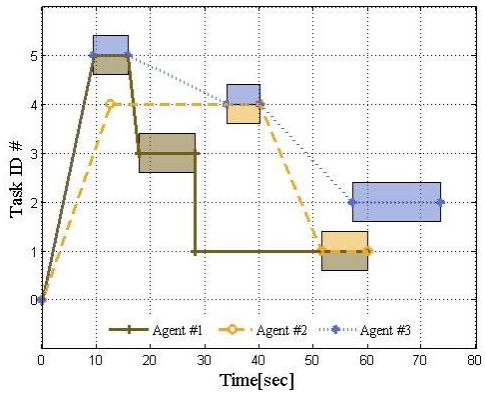

Figure 5.2 Load Transportation Simulation Time Schedule CASE 1

There are 3 agent and 5 tasks, each task requires two agents. In Fig. 5.2, the tasks is evenly allocated well. Here the cost is discounted as much as 5% / hour. The straight lines along the time axis means the waiting of the agent. And the oblique line means the moving of the agent.

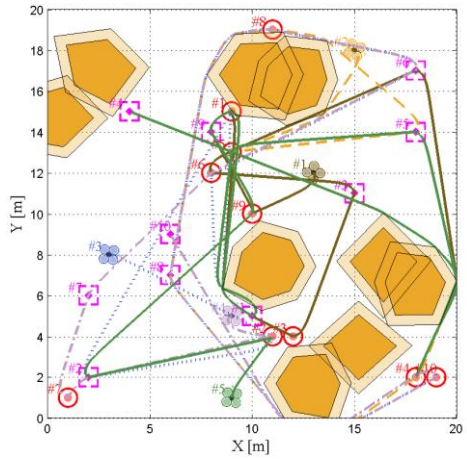

Figure 5.2 Load Transportation Simulation Problem Setup CASE 2

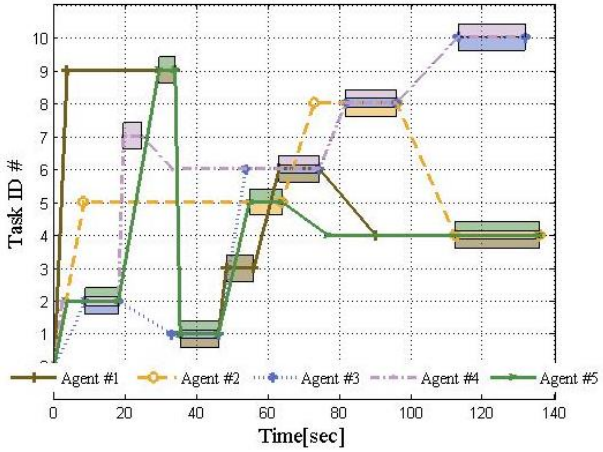

Figure 5.4 Load Transportation Simulation Time Schedule CASE 2

In case 2 setup, there are 5 agents and 10 tasks. The tasks requires the number of agents from one to three. The agent can involves in 5 tasks each. For this large problem, the CGBA shows a feasible solution.

5.2 Sub Optimality of the CGBA

To compare the sub optimality of the CGBA algorithm, a Monte Carlo simulation has been taken. The agent number is fixed to three, and it moves 50 m/s. The position of agent is

randomly chosen. Each agent's load capacity is three. Tasks and obstacles position is randomly chosen with in the map 20 by 20 km, and time discounting factor leads to 5%/hour discounting.

Figure 5.5 and 5.6 shows the total utility rewards of the Monte Carlo simulation of the complete enumeration and the CGBA, respectively. The green area means 3 sigma variance area, and black dots are each simulation result. The red line is expectation of the total utility.

In Fig 5.7, the expectations of the optimal solution and the CGBA is compared. In Figure 5.8, the optimality of the CGBA is shown. The optimality of the total utility reward is nearly over than 90 %.

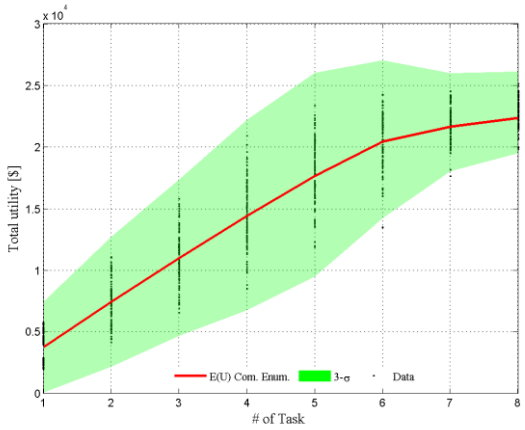

Figure 5.5 The Utility Reward of Monte Carlo Simulation of the Complete Enumeration

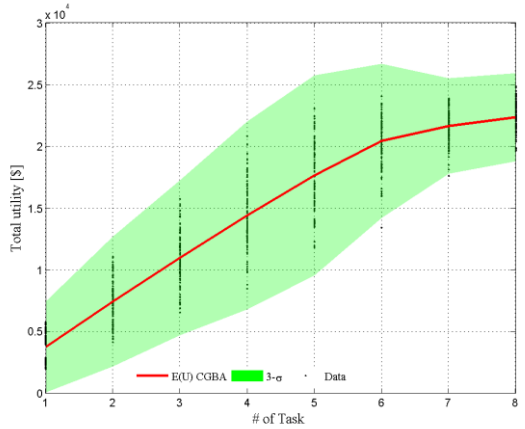

Figure 5.6 The Utility Reward of Monte Carlo Simulation of the CGBA.

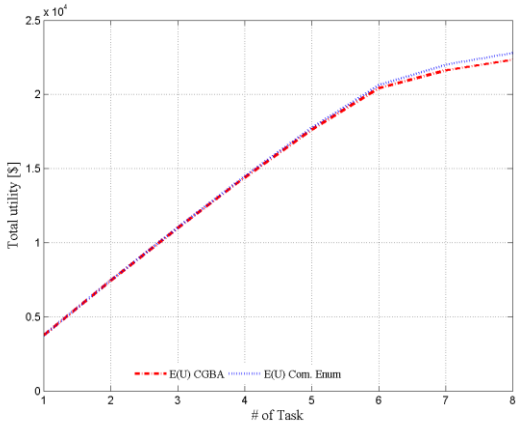

Figure 5.7 The Utility Reward Expectation Comparision

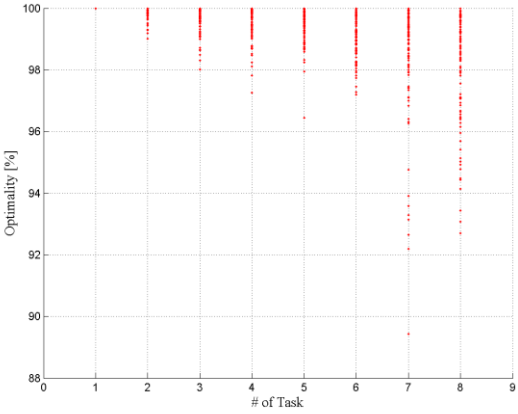

Figure 5.8 The Optimality of the CGBA

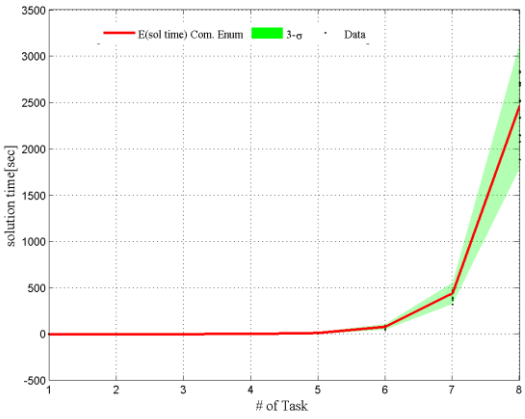

Figure 5.9 The Elapsed Time of Monte Carlo Simualtion of the complete Enumeration

Fig. 5.9 and 5.10 shows the elapsed time of the Monte Carlo simulation of the complete enumeration and the CGBA, respectively. And it is compared together in Fig 5.11, in the log scale. In Fig. 5.11, the elapsed time for the optimal solution is increasing exponentially, but that of the CGBA is increasing linearly as the problem size increase.

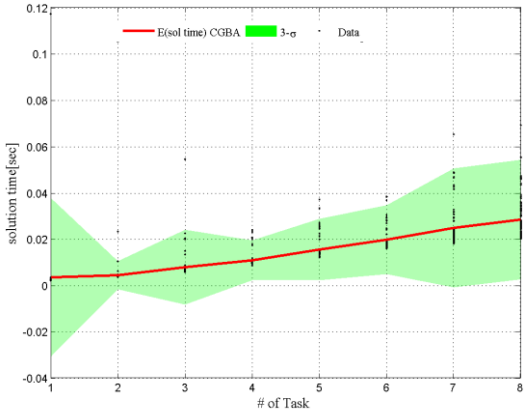

Figure 5.10 The Elapsed Time of Monte Carlo Simualtion of the complete Enumeration

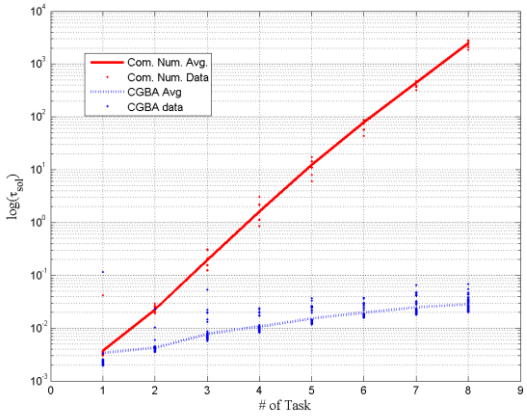

Figure 5.11 The Expectation of the Elapsed Time Comparison.

6 Conclusion

This paper introduces the MCTA problem for the multi agent's slung load type transport system. The CGBA is a heuristic security strategy algorithm, inspired from the group buying market. It gives a feasible solution with in a finite time, and show over 90% optimality in practical manner. The algorithm applicability were shown by the case study and the Monte Carlo simulation.

Acknowledgment

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

References

[1] M. Bisgaard, Modeling, Estimation and Control of Helicopter Slung Load System, Ph.D. Thesis, Aalborg University, Fredrik Bajers Vej 7, Denmark.

[2] B. Y. Lee, S. M. Hong, D. W. Yoo, H. I. Lee, G. H. Moon, and M. J. Tahk, Design of a Neural Network Controller for a Slung-Load System Lifted by 1 Quad-Rotor, 2014 The 2nd International Conference on Intelligent and Automation Systems, Hanoi, Vietnam, Feb 2014, (published on JOACE vol. 3, no. 1, pp. 9-14, Feb 2015)

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

[4] H. L. Choi, A. K. Whitten, J. P. How, Decentralized task allocation for Heterogeneous Teams with cooperation constraints, American Control Conference, Marriott Waterfront, Baltimore, MD. USA, 2010.

[5] 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.

[6] G. H. Moon, D. W. Yoo, B. Y. Lee, H. I. Lee and M. J. Tahk, Centralized Group Buying Approach for Multiple Cooperative Task Allocation, 2014 The 2nd International Conference on Intelligent and Automation Systems, Vietnam, Hanoi, (published on JOACE vol. 3, no. 2, pp. 92-97, April 2015)

[7] G. H. Moon, *Group Buying Algorithm for Task Assignment of Multi-Agent Load Transport System*, M.S. thesis, Korea Advanced Institute of Science and Technology, Daejeon, Korea, 2014

7 Contact Author Email Address

mailto:mjtahk@fdcl.kaist.ac.kr

Copyright Statement