

*Control & Guidance of
Multiple Air-Vehicle Systems (MAS)*

Research Program POD0914267

Performed for
DRTech, Ministry of Defence
by:

Temasek Laboratories @NUS
with
Department of Electrical & Computer Engineering, NUS
School of Electrical & Electronics Engineering, NTU



Technical Report No. TL/CG/2013/04

Progress Report for Deliverable D7

April 3, 2013

Part IV

Distributed Tasking for Autonomous Multi-Agent Teams

S.M. Dymkou

Aeronautical Sciences Division, Control Science Group,
Temasek Laboratories, National University of Singapore,
T-Lab Building 5A, Engineering Drive 1,
Singapore 117411

Contents

IV	Distributed Tasking for Autonomous Multi-Agent Teams	0
1	Contract Net Protocol	1
1.1	Introduction	1
1.2	Contract Net Stages	2
1.3	Definition of UAVs and targets	5
1.4	Agent information	5
1.5	Manager UAVs Logic	7
1.6	Potential Contractor UAVs Logic	9
2	Consensus-Based Bundle Algorithm	12
2.1	Problem formulation	13
2.2	Key assumptions	14
2.3	Vectors of agent information	15
2.4	Bundle construction phase	16
2.5	Consensus phase	18
2.6	Algorithm summary	21
2.7	Simulation result	22

Chapter 1

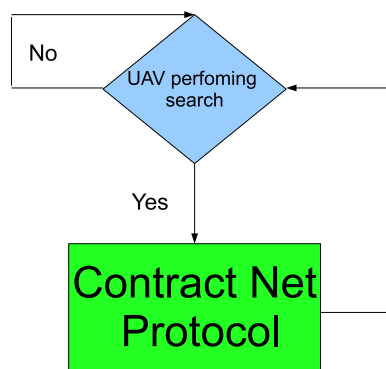
Contract Net Protocol

1.1 Introduction

The Contract Net Protocol (CNP) is a widely used task allocation protocol in Multi-Agent Systems. It is fast, flexible and has low communication costs. However it is limited in some issues and has shortcomings if the setting for task assignment is more complicated. Coordination theory is defined as "a body of principles about how activities can be coordinated, and how agents can work together". The task assignment process forms an important part of the coordination and assigning tasks in a multi agent system (MAS) architecture is not simple. When agents have different capabilities and different subproblems require different capabilities, a protocol is needed in order to find the right agent for the right task in an efficient way. The assignment of tasks to agents and the (re-) allocation of tasks in a MAS is one of the key features of automated negotiation systems. One possibility of solving this task assignment problem is using a centralized approach, where all initiators (agents with the tasks) and participants (agents that compete for acquiring tasks) send their information to a central decision maker. This decision maker will decide which participant will perform which task and then notifies it accordingly. This centralized scheme has lower coordination costs, but is very vulnerable to agent failures. Contract Net (CNet) is a well known high-level protocol that uses a decentralized scheme. In CNet, there is no central decision maker. Initiators (also known as managers) are responsible for monitoring the execution of a task and processing the results of its execution. Participants (also known as contractors) are responsible for the actual execution of the tasks. Agents in a MAS are not designated a priori as managers or contractors. These are only roles, and any node can take on either role dynamically during the course of problem solving. The negotiation process of CNet begins by initiators announcing available tasks. Participants evaluate these tasks and submit bids on those for which they are suited. The initiators then evaluate the

bids and award contracts to the nodes they determine to be most appropriate. Note that a participant may further partition a task and award contracts to other nodes, by acting as the initiator of that task. Control of the whole task assignment is distributed, because processing and communication are not focused at particular nodes.

1.2 Contract Net Stages

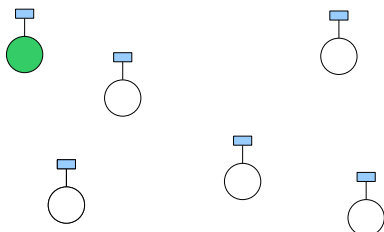


- Recognition;
- Announcement ;
- Bidding;
- Awarding;
- Expediting.

1.2.1 Recognition

In this stage, an agent recognises it has a problem it wants help with.

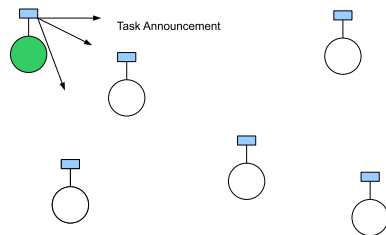
Agent has a goal, and either



- realises it cannot achieve the goal in isolation - does not have capability;
- realises it would prefer not to achieve the goal in isolation (typically because of solution quality, deadline, etc)

As a result, it needs to involve other agents.

1.2.2 Announcement



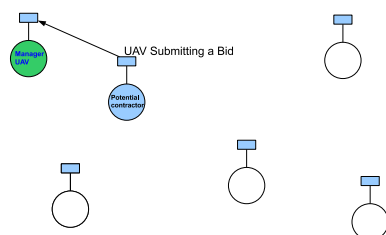
In this stage, the agent with the task sends out an announcement of the task which includes a specification of the task to be achieved.

Specification must encode:

- description of task itself (maybe executable);
- any constraints (e.g., deadlines, quality constraints).
- meta-task information (e.g., bids must be submitted by...)

The announcement is then broadcast.

1.2.3 Bidding



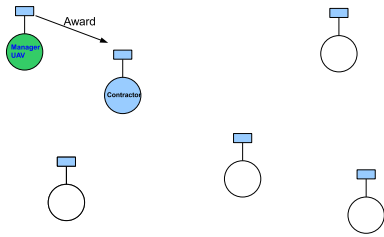
UAVs that receive the announcement decide for themselves whether they wish to bid for the task.

Factors:

- agent must decide whether it is capable of expediting task;
- agent must determine quality constraints and price information (if relevant).

If they do choose to bid, then they submit a tender.

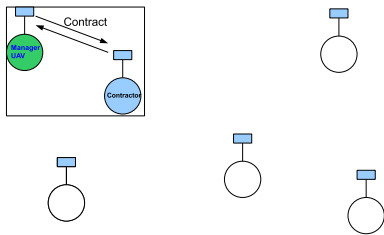
1.2.4 Awarding



Agent that sent task announcement must choose between bids and decide who to "award the contract" to.

The result of this process is communicated to agents that submitted a bid.

1.2.5 Expediting

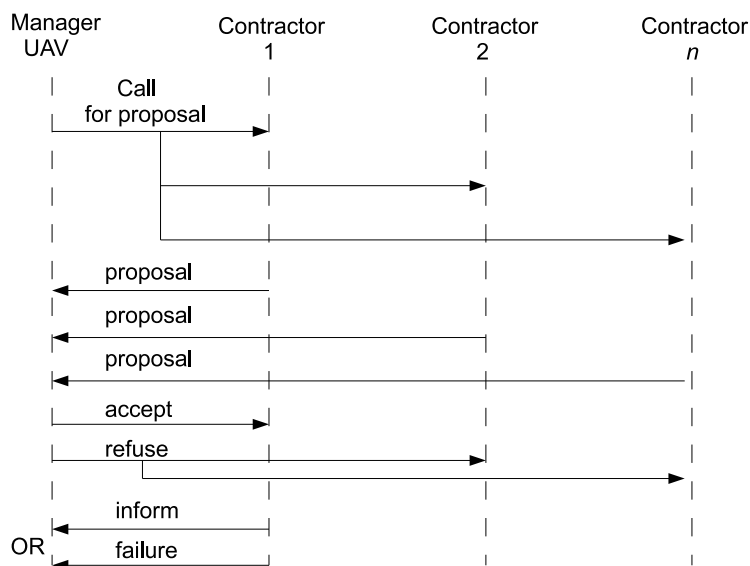


The successful contractor then expedites the task.

May involve generating further manager-contractor relationships: sub-contracting.

Also may involve another contract net protocol.

1.2.6 Procedure Diagram



Issues for Implementing CNP:

- How to specify tasks?
- How to specify quality of service?
- How to decide how to bid?
- How to select between competing offers?
- How to differentiate between offers based on multiple criteria?

1.3 Definition of UAVs and targets

Possible target (task) fields:

- id - task id;
- type -task type;
- value -task reward;
- start-task start time (sec);
- end - task expiry time (sec);
- duration -task default duration (sec);
- x- task position (meters);
- y-task position (meters);
- z-task position (meters).

Possible UAVs fields:

- id- agent id;
- type- agent type;
- avail- agent availability (expected time in sec);
- x- agent position (meters);
- y- agent position (meters);
- z- agent position (meters);
- velocity - agent cruise velocity (m/s);
- fuel-(agent fuel per meter)).

1.4 Agent information

Each agent must carry the following vectors of information in order to be able to perform decentralized algorithm:

N_a Number of agents

N_t - Number of tasks

L_t - Maximum length of the bundle, i.e. each agent can be assigned a maximum L_t tasks

\mathcal{I} - Index set of agents where $\mathcal{I} \doteq \{1, \dots, N_a\}$

\mathcal{J} - Index set of tasks where $\mathcal{J} \doteq \{1, \dots, N_t\}$

- A bundle, $\mathbf{b}_i \doteq \{b_{i1}, \dots, b_{i|\mathbf{b}_i|}\}$

of variable length whose elements are defined by $b_{in} \in \mathcal{J}$ for $n = 1, \dots, |\mathbf{b}_i|$. The current length of the bundle is denoted by b_i , which cannot exceed the maximum length L_t , and an empty bundle is represented by $b_i = \emptyset$ and $|\mathbf{b}_i| = 0$. The bundle represents the tasks that agent i has selected to do, and is ordered chronologically with respect to when the tasks were added (i.e. task b_{in} was added before task $b_{i(n+1)}$).

- A corresponding path, $\mathbf{p}_i \doteq \{p_{i1}, \dots, p_{i|\mathbf{p}_i|}\}$

whose elements are defined by $p_i \doteq \{p_{i1}, \dots, p_{i|\mathbf{p}_i|}\}$ for $n = 1, \dots, |\mathbf{b}_i|$. The path contains the same tasks as the bundle, and is used to represent the order in which agent i will execute the tasks in its bundle. The path is therefore the same length as the bundle, and is not permitted to be longer than L_t ; $|\mathbf{p}_i| = |\mathbf{b}_i| \leq L_t$.

- A vector of times $\tau_i \doteq \{\tau_{i1}, \dots, \tau_{i|\tau_i|}\}$

whose elements are defined by τ_{in} for $n = 1, \dots, |\tau_i|$. The times vector represents the corresponding times at which agent i will execute the tasks in its path, and is necessarily the same length as the path.

- A winning agent list $\mathbf{z}_i \doteq \{z_{i1}, \dots, z_{iN_t}\}$ of size N_t

where each element $z_{ij} \in \{\mathcal{I} \cup \emptyset\}$ for $j = 1, \dots, N_t$ indicates who agent i believes is the current winner for task j . Specifically, the value in element z_{ij} is the index of the agent who is currently winning task j according to agent i , and is $z_{ij} = \emptyset$; if agent i believes that there is no current winner.

- A winning bid list $\mathbf{y}_i \doteq \{y_{i1}, \dots, y_{iN_t}\}$ of size N_t

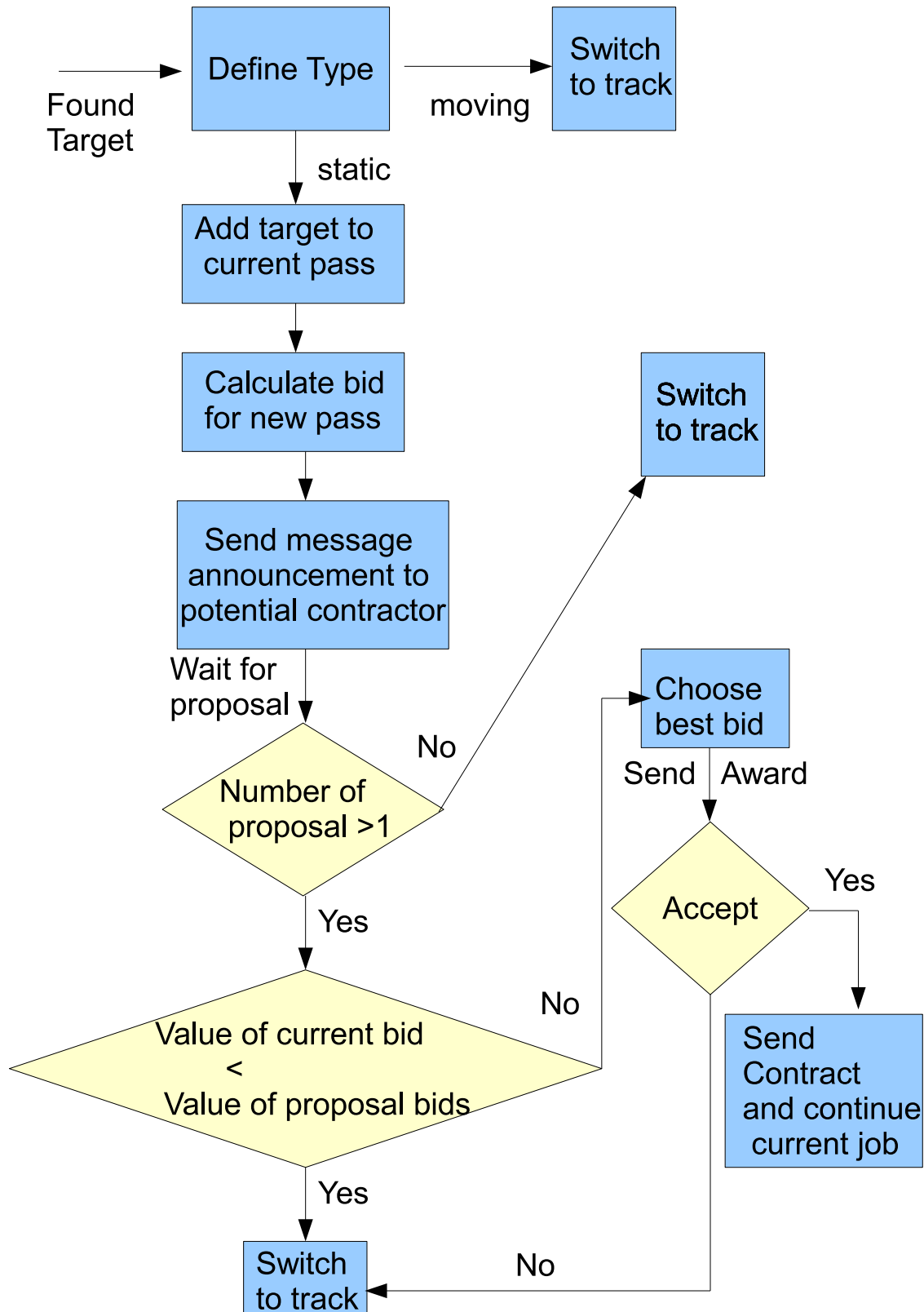
where the elements $y_{ij} \in [0, \infty)$ represent the corresponding winners bids and take the value of 0 if there is no winner for the task.

Manager UAV_i					Values
<i>Bundle</i>					$b_i = []$
<i>Path</i>					$p_i = []$
<i>Time</i>					$\tau_i = []$


Performing Search

Manager UAV_i					Values
<i>Winning Agent</i>					$z_i = []$
<i>WinningBids</i>					$y_i = []$

1.5 Manager UAVs Logic



For example i-th UAV found a static target;

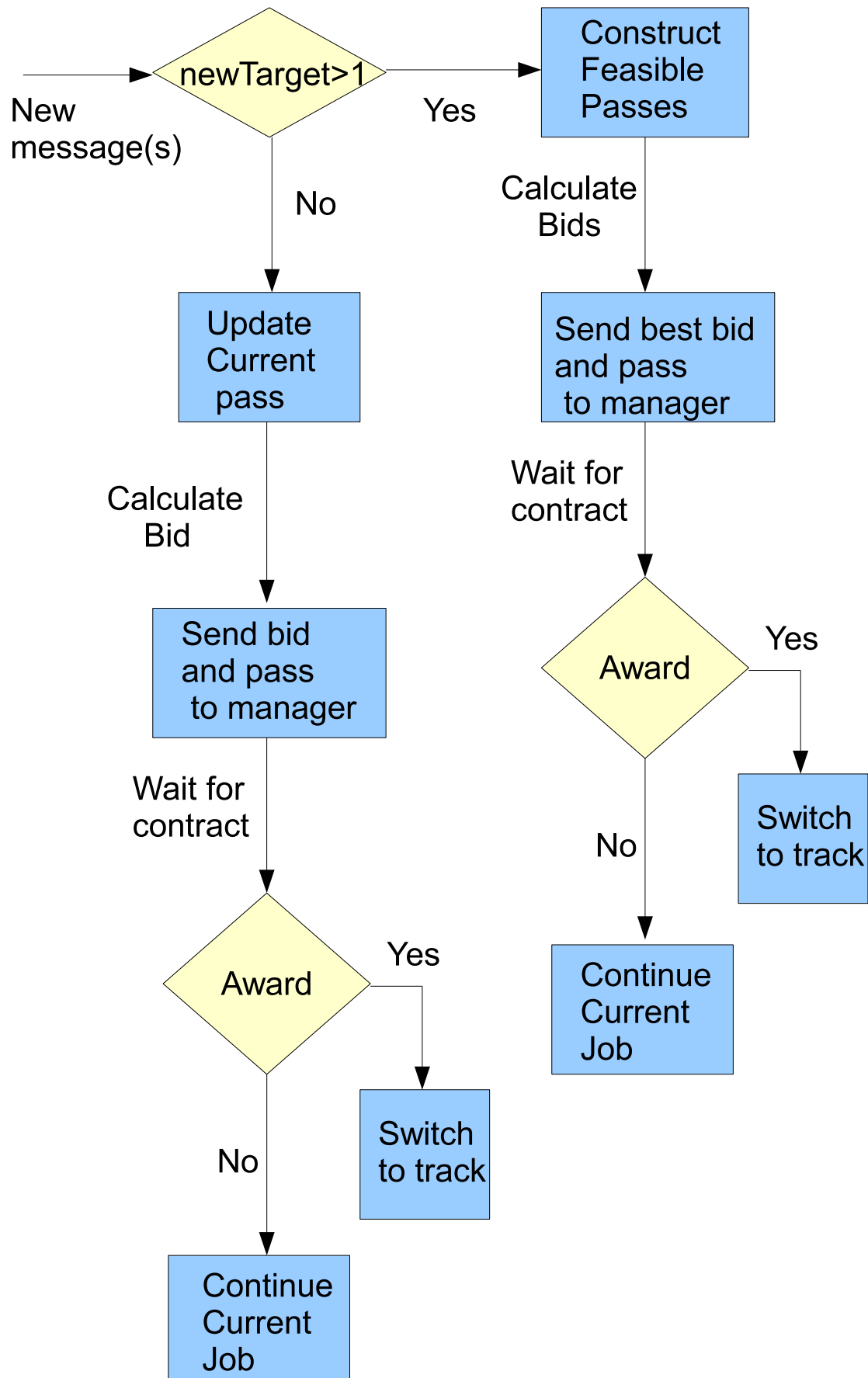
Manager UAV_i	$Target_1$				Values
<i>Bundle</i>					$b_i = [b_{i1}]$
<i>Path</i>	p_{i1}				$p_i = [p_{i1}]$
<i>Time</i>	τ_{i1}				$\tau_i = [\tau_{i1}]$

Then this UAV should calculate arrival time $\tau_{i1}(p)$ and corresponding bid y_{i1}

Manager UAV_i	$Target_1$				Values
<i>Winning Agent</i>	i				$z_i = [z_{i1}]$
<i>WinningBids</i>	y_{i1}				$y_i = [y_{i1}]$

And disseminate this information to other UAVs (Potential contractors UAVs), and waiting for their reply.

1.6 Potential Contractor UAVs Logic



For example k-th UAV receive a message about new target;

UAV_k	$Target_1$	$Target_2$	$Target_k$		Values
<i>Bundle</i>	✓	✓	✓		$\mathbf{b}_k \leftarrow (\mathbf{b}_k \oplus_{end} 1)$
<i>Path</i>	p_{k1}	p_{k2}	p_{kk}		$\mathbf{p}_k \leftarrow (\mathbf{p}_k \oplus_{n_1^*} 1)$
<i>Time</i>	τ_{k1}	τ_{k2}	τ_{kk}		$\tau_k \leftarrow (\tau_k \oplus_{n_1^*} \tau_{k1} (\mathbf{p}_k \oplus_{n_1^*} 1))$

Update current bundle of targets

Update current pass and arrival time

\Rightarrow And optimal location n_1^* is then given by $n_1^* = \max_{n_1} c_1(\tau_{k1}^*(\mathbf{p}_k \oplus_{n_1} 1))$

\Rightarrow Then the final score for new task j (which is include $|b_k|$ targets) is

$$c_{kj}(\mathbf{p}_i) = c_j(\tau_{kj}^*(\mathbf{p}_k \oplus_{n_j^*} j))$$

Then according to the logic above the manager UAVs should compare a bids received from potential contractors UAVs, for example the manager UAV $i = 3$ we can have the following comparison tables:

For case, when bundle of manager UAV3 was not empty $|b_3| \neq \emptyset$

	<i>Proposal1</i>	<i>Proposal2</i>	<i>Proposal3</i>
<i>UAV1</i>	c_{11}	-	-
<i>UAV2</i>	-	c_{22}	-
<i>UAV3</i>	-	-	c_{33}

For case, when bundle of manager UAV3 was empty $|b_3| = \emptyset$

	<i>Proposal1</i>	<i>Proposal2</i>	<i>Proposal3</i>
<i>UAV1</i>	c_{11}	-	-
<i>UAV2</i>	-	c_{22}	-
<i>UAV3</i>	c_{31}	c_{32}	c_{33}

Using Contract Net can be advantageous when compared with other coordination strategies as outlined below.

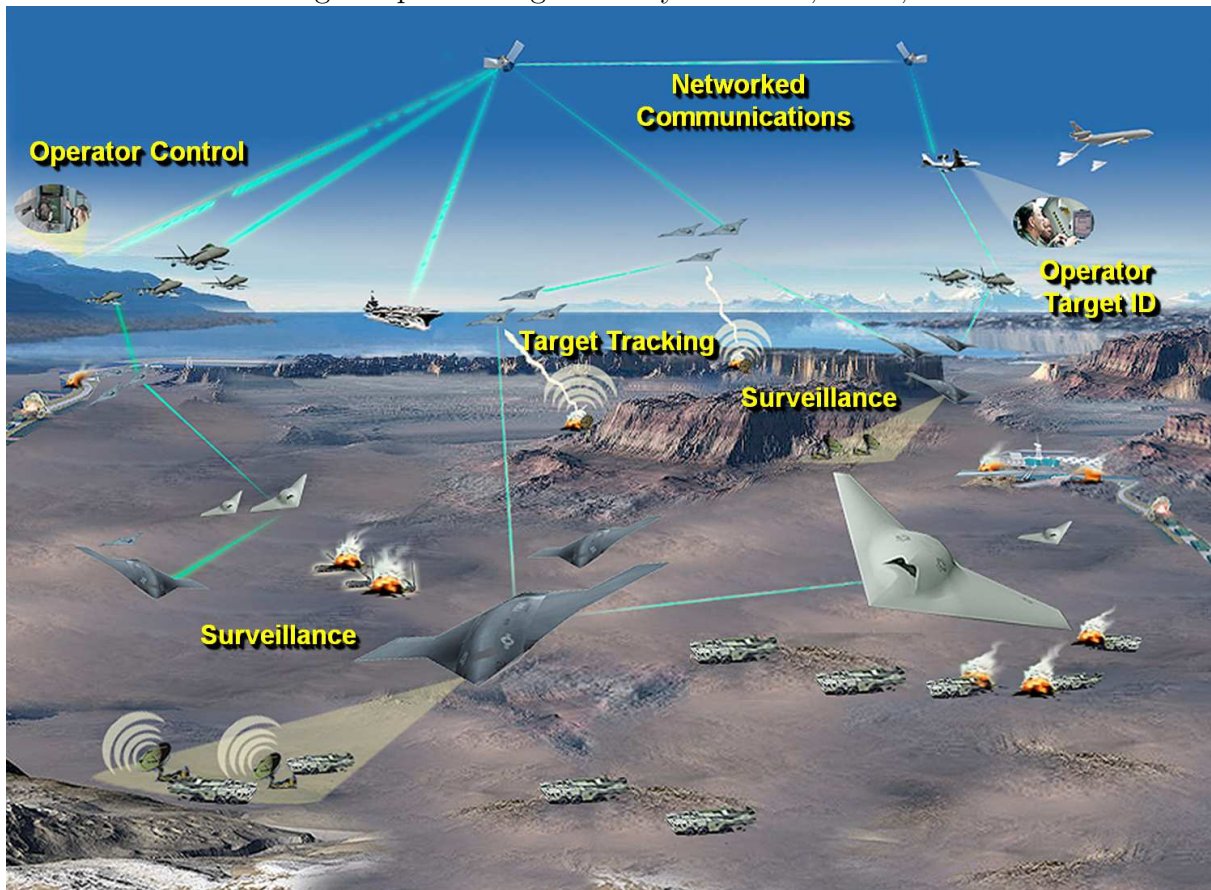
- Tasks are assigned (contracts awarded) dynamically, resulting in the better deals for the parties (agents) involved.
- Agents can enter and leave the system at will.
- The tasks will be naturally balanced among all the agents since agents that already have contract(s) do not have to bid on new ones. If an agent is already using all its resources, it will be unable to bid on new contracts until the current ones are completed.
- A reliable strategy for distributed applications with agents that can recover from failures (to be discussed more in the following paragraphs).

Unfortunately, CNP also have some disadvantages. For example CNP assumes that all agents are friendly and benevolent. As a result, there is no mechanism to detect conflicts and more importantly solve them. By these reason we will introduce the extension of the CNP so-called Consensus-Based Bundle Algorithm (CBBA).

Chapter 2

Consensus-Based Bundle Algorithm

Network centric operations involve large teams of agents, with heterogeneous capabilities, interacting together to perform missions. These missions involve executing several different tasks such as conducting reconnaissance, surveillance, target classification, and rescue operations. Within the heterogeneous team, some specialized agents are better suited to handle certain types of tasks than others. For example, UAVs equipped with video can be used to perform search, surveillance and reconnaissance operations, human operators can be used for classification tasks, ground teams can be deployed to perform rescue operations, etc. Figure below illustrates an example of such complex mission scenario involving numerous networked agents performing a variety of search, track, and surveillance tasks.



Ensuring proper coordination and collaboration between agents in the team is crucial to efficient and successful mission execution, motivating the development of autonomous task allocation methods to improve mission coordination.

Consensus-Based Bundle Algorithm (CBBA) is a decentralized market-based protocol that provides provably good approximate solutions for multi-agent multi-task allocation problems over networks of heterogeneous agents. CBBA consists of iterations between two phases: a bundle building phase where each vehicle greedily generates an ordered bundle of tasks, and a consensus phase where conflicting assignments are identified and resolved through local communication between neighboring agents. There are several core features of CBBA that can be exploited to develop an efficient planning mechanism for heterogeneous teams. First, CBBA is a decentralized decision architecture, which is a necessity for planning over large teams due to the increasing communication and computation overhead required for centralized planning with a large number of agents. Second, CBBA is a polynomial-time algorithm leading to a framework that scales well with the size of the network and/or the number of tasks (or equivalently, the length of the planning horizon). Third, CBBA is capable of handling various design objectives, nonlinear agent models, and constraints, and under a few assumptions on the scoring structure, a provably good feasible solution is guaranteed.

2.1 Problem formulation

The goal of task assignment problem is, given a list of N_t tasks and N_a agents, to find a conflict-free matching of tasks to agents that maximizes some global reward. An assignment is said to be free of conflicts if each task is assigned to no more than one agent. Each agent can be assigned a maximum of L_t tasks, and the assignment is said to be completed once $N_{max} \doteq \min\{N_t, N_a, L_t\}$ tasks have been assigned. The global objective function is assumed to be a sum of local reward values, while each local reward is determined as a function of the tasks assigned to each agent. The task assignment problem described above can be written as the following integer (possibly nonlinear) program with binary decision variables x_{ij} that indicate whether or not task j is assigned to agent i :

$$\sum_{i=1}^{N_a} \left(\sum_{j=1}^{N_t} c_{ij}(\tau_{ij}(\mathbf{p}_i(\mathbf{x}_i))) x_{ij} \right) \rightarrow \max (2.1)$$

subject to :

$$\sum_{j=1}^{N_t} x_{ij} \leq L_t, \quad \forall i \in \mathcal{I}$$

$$\sum_{i=1}^{N_a} x_{ij} \leq 1, \quad \forall j \in \mathcal{J}$$

$$\sum_{i=1}^{N_a} \sum_{j=1}^{N_t} x_{ij} = N_{max}$$

$$x_{ij} \in \{0, 1\}, \forall (i, j) \in \mathcal{I} \times \mathcal{J}$$

where $x_{ij} = 1$ if agent i is assigned to task j , and $\mathbf{x}_i \doteq \{x_{i1}, \dots, x_{iN_t}\}$ is a vector of assignments for agent i , whose j -th element is x_{ij} .

The summation term in brackets in the objective function represents the local reward for agent i .

N_a Number of agents

N_t - Number of tasks

L_t - Maximum length of the bundle, i.e. each agent can be assigned a maximum L_t tasks

\mathcal{I} - Index set of agents where $\mathcal{I} \doteq \{1, \dots, N_a\}$

\mathcal{J} - Index set of tasks where $\mathcal{J} \doteq \{1, \dots, N_t\}$

$\mathbf{p}_i \doteq \{p_{i1}, \dots, p_{i|\mathbf{p}_i|}\}$ - The variable length vector represent the path for agent i , an ordered sequence of tasks where the elements are the task indices, $p_{in} \in \mathcal{J}$ for $n = 1, \dots, |\mathbf{p}_i|$, i.e. its n -th element is $j \in \mathcal{J}$ if agent i conducts task j at the n -th point along the path. The current length of the path is denoted by $|\mathbf{p}_i| \leq L_t$.

Also note that score function $c_{ij}(\mathbf{x}_i, \mathbf{p}_i)$ is assumed to be nonnegative. The score function can be any nonnegative function of either assignment \mathbf{x}_i or path \mathbf{p}_i (usually not a function of both); in the context of task assignment for unmanned vehicles with mobility, it often represents a path-dependent reward such as the path length, the mission completion time, and the time-discounted value of target.

2.2 Key assumptions

- The score c_{ij} that agent i obtains by performing task j is defined as a function of the arrival time τ_{ij} at which the agent executes the task (or possibly the expected arrival time in a probabilistic setting).
- The arrival time τ_{ij} is uniquely defined as a function of the path \mathbf{p}_i that agent i takes.

- The path \mathbf{p}_i is uniquely defined by the assignment vector of agent i , \mathbf{x}_i .

Several design objectives commonly used for multi-agent decision making problems feature scoring functions that satisfy the above set of assumptions. An example is the problem involving time-discounted values of targets, in which the sooner an agent arrives at the target, the higher the reward it obtains. Or for scenario involves re-visit tasks, where previously observed targets must be revisited at some scheduled time. In this case the score function would have its maximum at the desired re-visiting time and lower values at other re-visit times.

2.3 Vectors of agent information

The scoring function in (2.1) depends on the assignment vector \mathbf{x}_i and on the path \mathbf{p}_i , which makes this integer programming problem significantly complex for $L_t > 1$. To address these dependencies, the classical combinatorial auction methods explored simplifications of the problem by treating each assignment combination or "bundle" of tasks as a single item for bidding. These approaches led to complicated winner selection methods as well as bad scalability due to the increase in computation associated with enumerating all possible bundle combinations. In contrast, the Consensus-Based Bundle Algorithm (CBBA) consists of an auction process which is performed at the task level rather than at the bundle level, where agents build their bundles in a sequential greedy fashion.

And in order to perform this decentralized algorithm each agent must carry the following vectors of information:

- A bundle, $\mathbf{b}_i \doteq \{b_{i1}, \dots, b_{i|\mathbf{b}_i|}\}$

of variable length whose elements are defined by $b_{in} \in \mathcal{J}$ for $n = 1, \dots, |\mathbf{b}_i|$. The current length of the bundle is denoted by b_i , which cannot exceed the maximum length L_t , and an empty bundle is represented by $b_i = \emptyset$ and $|\mathbf{b}_i| = 0$. The bundle represents the tasks that agent i has selected to do, and is ordered chronologically with respect to when the tasks were added (i.e. task b_{in} was added before task $b_{i(n+1)}$).

- A corresponding path, $\mathbf{p}_i \doteq \{p_{i1}, \dots, p_{i|\mathbf{p}_i|}\}$

whose elements are defined by $p_i \doteq \{p_{i1}, \dots, p_{i|\mathbf{p}_i|}\}$ for $n = 1, \dots, |\mathbf{p}_i|$. The path contains the same tasks as the bundle, and is used to represent the order in which agent i will execute the tasks in its bundle. The path is therefore the same length as the bundle, and is not permitted to be longer than L_t ; $|\mathbf{p}_i| = |\mathbf{b}_i| \leq L_t$.

- A vector of times $\tau_i \doteq \{\tau_{i1}, \dots, \tau_{i|\tau_i|}\}$

whose elements are defined by τ_{in} for $n = 1, \dots, |\tau_i|$. The times vector represents the corresponding times at which agent i will execute the tasks in its path, and is necessarily the same length as the path.

- A winning agent list $\mathbf{z}_i \doteq \{z_{i1}, \dots, z_{iN_t}\}$ of size N_t

where each element $z_{ij} \in \{\mathcal{I} \cup \emptyset\}$ for $j = 1, \dots, N_t$ indicates who agent i believes is the current winner for task j . Specifically, the value in element z_{ij} is the index of the agent who is currently winning task j according to agent i , and is $z_{ij} = \emptyset$; if agent i believes that there is no current winner.

- A winning bid list $\mathbf{y}_i \doteq \{y_{i1}, \dots, y_{iN_t}\}$ of size N_t

where the elements $y_{ij} \in [0, \infty)$ represent the corresponding winners bids and take the value of 0 if there is no winner for the task.

- Vector of timestamps $\mathbf{s}_i \doteq \{s_{i1}, \dots, s_{iN_a}\}$, of size N_a

where each element $s_{ik} \in [0, \infty)$ for $k = 1, \dots, N_a$ represents the timestamp of the last information update agent i received about agent k , either directly or through a neighboring agent.



Each agent must carry these vectors of information in order to be able to perform decentralized algorithm which consists of iterations between two phases:

a bundle building phase where each vehicle greedily generates an ordered bundle of tasks, and a

consensus phase where conflicting assignments are identified and resolved through local communication between neighboring agents.

Algorithm will iterates between these two phases until no changes to the information vectors occur anymore.

2.4 Bundle construction phase

In contrast to the combinatorial algorithms, which enumerate all possible bundles for bidding, in CBBA each agent creates just its single bundle which is updated as the assignment process progresses. During this phase of the algorithm, each agent continuously adds tasks to its bundle in a sequential greedy fashion until it is incapable of adding any others. Tasks

in the bundle are ordered based on which ones were added first in sequence, while those in the path are ordered based on their predicted execution times.

The bundle construction process is as follows: for each available task not currently in the bundle, or equivalently not in the path ($j \notin \mathbf{p}_i$), the agent computes a score for the task, $c_{ij}(\mathbf{p}_i)$. The score is checked against the current winning bids, and is kept if it is greater. Out of the remaining ones, the agent selects the task with the highest score and adds that task to its bundle.

Computing the score for a task is a complex process which is dependent on the tasks already in the agents path (and/or bundle). Selecting the best score for task j can be performed using the following two steps. First, task j is "inserted" (\oplus_{n_j}) in the path at some location n_j (the new path becomes $(\mathbf{p}_i \oplus_{n_j} j)$, where \oplus_n signifies inserting the task at location n). The score for each task $c_j(\tau)$ is dependent on the time at which it is executed, motivating the second step, which consists of finding the optimal execution time given the new path, $\tau_{ij}^*(\mathbf{p}_i \oplus_{n_j} j)$. This can be found by solving the following optimization problem:

$$\begin{aligned} \tau_{ij}^*(\mathbf{p}_i \oplus_{n_j} j) &= \max_{\tau_{ij} \in [0, \infty)} c_j(\tau_{ij}) \\ &\text{subject to :} \\ \tau_{ik}^*(\mathbf{p}_i \oplus_{n_j} j) &= \tau_{ik}^*, \forall k \in \mathbf{p}_i \end{aligned} \quad (2.2)$$

The constraints state that the insertion of the new task j into path \mathbf{p}_i cannot impact the current times (and corresponding scores) for the tasks already in the path. Note that this is a continuous time optimization, which, for the general case, involves a significant amount of computation. The optimal score associated with inserting the task at location n_j is then given by $c_j(\tau_{ij}^*(\mathbf{p}_i \oplus_{n_j} j))$. This process is repeated for all n_j by inserting task j at every possible location in the path. The optimal location is then given by

$$n_j^* = \max_{n_j} c_j(\tau_{ik}^*(\mathbf{p}_i \oplus_{n_j} j)) \quad (2.3)$$

and the final score for task j is $c_{ij}(\mathbf{p}_i) = c_j(\tau_{ij}^*(\mathbf{p}_i \oplus_{n_j^*} j))$.

Once the scores for all possible tasks are computed ($c_{ij}(\mathbf{p}_i) \forall j \notin \mathbf{p}_i$), the scores need to be checked against the winning bid list, y_i , to see if any other agent has a higher bid for the task. We define the variable $h_{ij} = I(c_{ij}(\mathbf{p}_i) > y_{ij})$, where $I(\cdot)$ denotes the indicator function that equals unity if the argument is true and zero if it is false, so that $c_{ij}(\mathbf{p}_i)h_{ij}$ will be nonzero only for viable bids. The final step is to select the highest scoring task to add to the bundle:

$$j^* = \max_{j \notin \mathbf{p}_i} c_{ij}(\mathbf{p}_i) h_{ij} \quad (2.4)$$

The bundle, path, times, winning agents and winning bids vectors are then updated to include the new task:

$$\begin{aligned} \mathbf{b}_i &\leftarrow (\mathbf{b}_i \oplus_{end} j^*) \\ \mathbf{p}_i &\leftarrow (\mathbf{p}_i \oplus_{n_{j^*}} j^*) \\ \tau_i &\leftarrow (\mathbf{tau}_i \oplus_{n_{j^*}} \tau_{ij^*}^*(\mathbf{p}_i \oplus_{n_{j^*}} j^*)) \\ z_{ij} &= i \\ y_{ij} &= c_{ij^*}(\mathbf{p}_i) \end{aligned} \quad (2.5)$$

The bundle building recursion continues until either the bundle is full (the limit L_t is reached), or no tasks can be added for which the agent is not outbid by some other agent ($h_{ij} = 0 \forall j \notin \mathbf{p}_i$). Notice that with equation (2.5), a path is uniquely defined for a given bundle, while multiple bundles might result in the same path.

2.5 Consensus phase

Once agents have built their bundles of desired tasks they need to communicate with each other to resolve conflicting assignments amongst the team. After receiving information from neighboring agents about the winning agents and corresponding winning bids, each agent can determine if it has been outbid for any task in its bundle. Since the bundle building recursion, described in the previous section, depends at each iteration upon the tasks in the bundle up to that point, if an agent is outbid for a task, it must release it and all subsequent tasks from its bundle. If the subsequent tasks are not released, then the current best scores computed for those tasks would be overly conservative, possibly leading to a degradation in performance. It is better, therefore, to release all tasks after the outbid task and redo the bundle building recursion process to add these tasks (or possibly better ones) back into the bundle.

This consensus phase assumes that each pair of neighboring agents synchronously shares the following information vectors: the winning agent list z_i , the winning bids list y_i , and the vector of timestamps s_i representing the time stamps of the last information updates received about all the other agents. The timestamp vector for any agent i is updated using the following equation,

$$s_{ik} = \begin{cases} \tau_r(\text{i.e. message reception time}), & \text{if } g_{ik} = 1; \\ \max\{s_{mk} | m \in \mathcal{I}, g_{im} = 1\}, & \text{otherwise} \end{cases} \quad (2.6)$$

which states that the timestamp s_{ik} that agent i has about agent k is equal to the message reception time τ_r if there is a direct link between agents i and k (i.e. $g_{ik} = 1$ in the network graph), and is otherwise determined by taking the latest timestamp about agent k from the set of agent i 's neighboring agents. For each message that is passed between a sender k and a receiver i , a set of actions is executed by agent i to update its information vectors using the received information. These actions involve comparing its vectors z_i , y_i , and s_i to those of agent k to determine which agent's information is the most up-to-date for each task. There are three possible actions that agent i can take for each task j :

i, (receiver)	$Task_1$	$Task_2$	$Task_k$	$Task_{N_t}$	<i>Values</i>
<i>Winning Agent</i>					$z_i = [z_{i1}, z_{i2}, \dots]$
<i>WinningBids</i>					$y_i = [y_{i1}, y_{i2}, \dots]$
<i>Timestamps</i>					$s_i = [s_{i1}, s_{i2}, \dots]$

Update : $z_{ij} = z_{kj}, \quad y_{ij} = y_{kj}$

Reset : $z_{ij} = \emptyset, \quad y_{ij} = 0$

Leave : $z_{ij} = z_{ij}, \quad y_{ij} = y_{ij}$

k, (sender)	$Task_1$	$Task_2$	$Task_k$	$Task_{N_t}$	<i>Values</i>
<i>Winning Agent</i>					$z_k = [z_{k1}, z_{k2}, \dots]$
<i>WinningBids</i>					$y_k = [y_{k1}, y_{k2}, \dots]$
<i>Timestamps</i>					$s_k = [s_{k1}, s_{k2}, \dots]$

Next we will give the decision rules for this communication protocol.

2.5.1 Decision Rules

Agent k thinks z_{kj} is	Agent i thinks z_{ij} is	Receiver Action
k	i	if $y_{kj} > y_{ij} \rightarrow \text{update}$
k	k	update
k	$m \notin \{i, k\}$	if $s_{km} > s_{im}$ or $y_{kj} > y_{ij} \rightarrow \text{update}$
k	none	update

$$s_{ik} = \begin{cases} \tau_r(\text{i.e. message reception time}), & \text{if } g_{ik} = 1; \\ \max\{s_{mk} | m \in \mathcal{I}, g_{im} = 1\}, & \text{otherwise} \end{cases}$$

Agent k thinks z_{kj} is	Agent i thinks z_{ij} is	Receiver Action
i	i	leave
i	k	reset
i	$m \notin \{i, k\}$	if $s_{km} > s_{im} \rightarrow \text{reset}$
i	none	leave

The first two columns of the table indicate the agent that each of the sender k and receiver i believes to be the current winner for a given task; the third column indicates the action that the receiver should take, where the default action is "Leave".

Agent k thinks z_{kj} is	Agent i thinks z_{ij} is	Receiver Action
$m \notin \{i, k\}$	i	if $s_{km} > s_{im}$ and $y_{kj} > y_{ij} \rightarrow \text{update}$
$m \notin \{i, k\}$	k	if $s_{km} > s_{im} \rightarrow \text{update}$ else $\rightarrow \text{reset}$
$m \notin \{i, k\}$	m	$s_{km} > s_{im} \rightarrow \text{update}$
$m \notin \{i, k\}$	$n \notin \{i, k, m\}$	if $s_{km} > s_{im}$ and $s_{kn} > s_{in} \rightarrow \text{update}$ if $s_{km} > s_{im}$ and $y_{kj} > y_{ij} \rightarrow \text{update}$ if $s_{kn} > s_{in}$ and $s_{im} > s_{km} \rightarrow \text{reset}$
$m \notin \{i, k\}$	none	if $s_{km} > s_{im} \rightarrow \text{update}$

$$s_{ik} = \begin{cases} \tau_r, & \text{if } g_{ik} = 1; \\ \max\{s_{mk} | m \in \mathcal{I}, g_{im} = 1\}, & \text{otherwise} \end{cases}$$

Agent k thinks z_{kj} is	Agent i thinks z_{ij} is	Receiver Action
none	i	leave
none	k	update
none	$m \notin \{i, k\}$	if $s_{km} > s_{im} \rightarrow \text{update}$
none	none	leave

If either of the winning agent or winning bid information vectors (z_i or y_i) are changed as an outcome of the communication, the agent must check if any of the updated or reset tasks were in its bundle. If so, those tasks, along with all others added to the bundle after them, are released. Thus if \bar{n} is the location of the first outbid task in the bundle ($\bar{n} = \min\{n | z_{i(b_{in})} \neq i\}$ with b_{in} denoting the n -th entry of the bundle), then for all bundle locations $n \geq \bar{n}$, with corresponding task indices bin , the following updates are made:

$$\begin{aligned} z_{i(b_{in})} &= \\ y_{i(b_{in})} &= 0 \end{aligned} \tag{2.7}$$

The bundle is then truncated to remove these tasks,

$$b_i \leftarrow \{b_{i1}, \dots, b_{i(\bar{n}-1)}\} \tag{2.8}$$

and corresponding entries are removed from the path and times vectors as well. From here, the algorithm returns to the first phase where new tasks can be added to the bundle. CBBA iterates between these two phases until no changes to the information vectors occur anymore.

2.6 Algorithm summary

- Calculate marginal score for all tasks

$$c_{ij}(\mathbf{p}_i) = \begin{cases} 0, & \text{if } j \in \mathbf{p}_i; \\ \max_{n \leq l_b} S_{path}(\mathbf{p}_i \oplus_n j) - S_{path}(\mathbf{p}_i), & \text{otherwise} \end{cases}$$

- Determine which tasks are winnable

$$h_{ij} = \mathbf{I}(c_{ij}(\mathbf{p}_i) > y_{ij}), \forall j \in \mathcal{J}$$

- Select the index of the best eligible task, j^* , and select best location in the plan to insert the task, n_j^*

$$j^* = \max_{j \in \mathcal{J}} c_{ij} h_{ij}$$

$$n_j^* = \max_{n \in \{0, \dots, l_b\}} S_{path}(\mathbf{p}_i \oplus_n j^*)$$

- If $c_{ij^*} \leq 0$, then **return**. otherwise, continue

- Update agent information

$$\mathbf{b}_i \leftarrow (\mathbf{b}_i \oplus_{l_b} j^*)$$

$$\mathbf{p}_i \leftarrow (\mathbf{p}_i \oplus_{n_j^*} j^*)$$

- Update shared information vectors

$$y_{i(j^*)} = c_{i(j^*)}$$

$$z_{i(j^*)} = i$$

- if $l_b = L_t$, then **return**, otherwise, go to 1.

2.7 Simulation result

The current version of CBBA supports tasks with time windows of validity, homogenous agent-task compatibility requirements, and score functions that balance task reward and fuel costs. Namely to the given local information the algorithm allows the UAVs manage themselves automatically as a team in a distributed manner. The centralized agency is no longer involved in receiving requests and allocating UAVs to the requests. It only needs to ensure that there are sufficient UAVs in the pool to service the requests.

The general system configuration for simulation research of distributed tasking problem are as follows

Configuration:

- UAV Airspace;
- Range for air-to-air communications between the UAVs;
- Homogenous Service UAVs;
- Local information about targets;

- Static and moving targets.

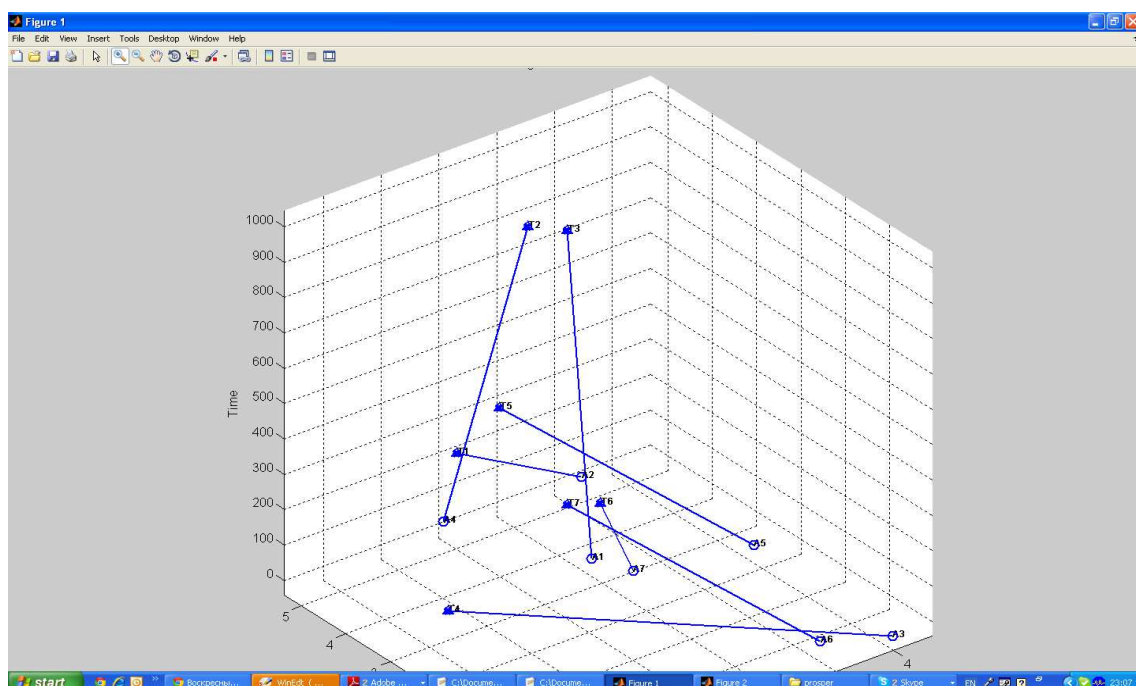
Requirements:

- Manage UAVs themselves automatically as a team in a hierarchical and / or distributed manner;
- Provide automatic self-allocation and self-deployment.;
- Maximize the local reward for each Service-UAVs.

Considerations:

- Limited UAV sensing range and communication range;
- Breakdown of UAVs and variations of communication topologies;
- Distributed information fusion between UAVs;
- Distributed motion planning for UAVs;

For example the so called Tethered UAVs Self-Assignment Problem is a particular case of the problem described above. And the solution of this particular problem presented in a figure below, namely, we find the logic that enabled the tethered UAVs (7 UAVs) to self-deploy one UAV to each specified location (7 locations).



Bibliography

- [1] L. F. Bertuccelli, W.M. Beckers, M.L Cummings: Developing operator models for UAVs search scheduling, AIAA Guidance, Navigation, and Control Conference 2-5 August, 2010, Toronto, Ontario, Canada
- [2] Gabasov R., Kirillova F. M. and Prischepova S. V.: Optimal Feedback Control, 1995, Springer-Verlag Lecture Notes in Control and Information Sciences Series, vol. 207.
- [3] R. Gabasov, F.M. Kirillova, O.I. Kostyukova, Constructive methods of optimization, Part 3(Network problems) University Press, Minsk 1986 (in Russian)
- [4] Danzig G.B, Ford L.R, Fulkerson D.R. A primal-dual algorithm for linear programming. In: Kuhn HW, Tucker AW, editors. Linear inequalities and related systems. New Jersey: Princeton University Press; 1956, p. 171-81.
- [5] S. Dymkou, Graph and 2-D Optimization Theory and their application for discrete simulation of gas transportation networks and industrial processes with repetitive operations. PhD Thesis, RWTH, Aachen, Germany 2006.
- [6] Adam N. Elmachtoub, Charles F. van Loan, :From Random Polygon to Ellipse: An Eigenanalysis, SIAM Review, 2010, Vol.52, No. 1, pp.151-170
- [7] Kostyukova O. I.: Parametric optimal control problem with variable index, 2003, Computational Mathematics and Mathematical Physics, 43(1), pp. 26–41 (translated from Journal Vychislitelnoi Matematiki i Matematicheskoi Fiziki, 2003, v. 43, No. 1, pp. 26–41.
- [8] A. Martin and M. Moeller. Mixed integer models for the stationary case of gas network optimization. Technical report. TU Darmstadt. 2004.
- [9] Ford L.R, Fulkerson D.R. Flows in networks. New Jersey: Princeton University Press; 1962.
- [10] Wei Ren, Yongcan Cao. Distributed Coordination of Multi-agent Networks. Springer-Verlag London Limited, 2011