Improving the Efficiency of a Decentralized Tasking Algorithm for UAV Teams with Asynchronous Communications

Luke B. Johnson*, Sameera S. Ponda†, Han-Lim Choi‡, and Jonathan P. How§
Aerospace Controls Laboratory
Massachusetts Institute of Technology, Cambridge, MA 02139

This work presents a decentralized task allocation algorithm for networked agents communicating through an *asynchronous* channel. The algorithm extends the Consensus-Based Bundle Algorithm (CBBA) to account for more realistic asynchronous communication protocols. Direct implementation of CBBA into such an asynchronous setting requires agents to frequently broadcast their information states, which would cause significant communication overflow. In contrast, the extension proposed in this paper, named Asynchronous CBBA (ACBBA), minimizes communication load while preserving the convergence properties. ACBBA applies a new set of local deconfliction rules that do not require access to the global information state. This new deconfliction protocol also features consistent handling of out-of-order messages and detection of redundant information. A real-time software implementation using multiple PCs communicating through the user datagram protocol (UDP) validates the proposed algorithm.

Nomenclature

N_a	Number of agents
N_t	Number of tasks
L_t	Maximum length of the bundle
$\mathcal{I}$	Index set of agents where $\mathcal{I} \triangleq \{1, \ldots, N_a\}$
$\mathcal{J}$	Index set of tasks where $\mathcal{J} \triangleq \{1, \ldots, N_t\}$
$c_j(\tau)$	Score function for task j as a function of time
$c_{ij}(\mathbf{p}_i)$	Score agent i computes for task j as a function of its path $\mathbf{p}_i$
$\oplus_n$	Operation to "insert" an element in a vector at location n
$\mathbf{b}_i$	Bundle of tasks for agent i
$\mathbf{p}_i$	Path for agent i
$\boldsymbol{\tau}_i$	Vector of task execution times for agent i

*S.M. Candidate, Dept. of Aeronautics and Astronautics, MIT, Cambridge, MA, lbj16@mit.edu
†PhD Candidate, Dept. of Aeronautics and Astronautics, MIT, Cambridge, MA, sponda@mit.edu
‡Assistant Professor, Div. of Aerospace Engineering, KAIST, Daejeon, Korea, hanlimc@kaist.ac.kr
§Richard Cockburn Maclaurin Professor of Aeronautics and Astronautics, MIT, Cambridge, MA, jhow@mit.edu

$\mathbf{s}_i$	Communication timestamps with other agents carried by agent i
$\mathbf{y}_i$	List of winning bids for all tasks carried by agent i
$\mathbf{z}_i$	List of winning agents for all tasks carried by agent i

I. Introduction

The use of autonomous robotic agents for complex missions, such as Unmanned Aerial Vehicles (UAVs) and autonomous ground rovers, has gained increasing popularity in recent years. Teams of heterogeneous unmanned agents are regularly employed in complex missions including intelligence, surveillance and reconnaissance operations[1–3]. Of critical interest for successful mission operations is proper coordination of tasks amongst the fleet of robotic agents. Many different methods have been considered for enabling such agents to distribute tasks amongst themselves from a known mission task list. Centralized planners, which rely on agents communicating their state to a central server that generates a plan for the entire fleet, are useful since they place much of the heavy processing requirements safely on the ground, making robots smaller and cheaper to build[4–10]. Ideally, the communication links between all elements of the system (command station, autonomous vehicles, manned vehicles, etc.) are high bandwidth, low latency, low cost, and highly reliable. However, even the most modern communication infrastructures do not possess all of these characteristics. If the inter-agent communication mechanism has a more favorable combination of these characteristics compared to agent-to-base communication, then a decentralized planning architecture offers performance and robustness advantages. In particular, response times to changes in situational awareness can be significantly faster via decentralized control than those achieved under a purely centralized planner. As a result, decentralized planning methods which eliminate the need for a central server have been explored[11–14]. Many of these methods often assume perfect communication links with infinite bandwidth, to ensure that agents have the same situational awareness before planning. In the presence of inconsistencies in situational awareness, these decentralized tasking algorithms can be augmented with consensus algorithms[15–26] to converge on a consistent state before performing the task allocation. Although consensus algorithms guarantee convergence on information, they may take a significant amount of time and often require transmitting large amounts of data[27].

Other popular task allocation methods involve using decentralized auction algorithms[28–32], which have been shown to efficiently produce sub-optimal solutions. One such algorithm is the Consensus-Based Bundle Algorithm (CBBA)[33,34], a multi-assignment decentralized auction approach with a consensus protocol that guarantees a conflict-free solution despite possible inconsistencies in situational awareness. CBBA is guaranteed to achieve at least 50% optimality[33], although empirically its performance is shown to be within 93% of the optimal solution[35]. The bidding process runs in polynomial time, demonstrating good scalability with increasing numbers of agents and tasks, making it well suited to real-time dynamic environments. Although the CBBA algorithm allows for asynchronous bidding, the consensus phase relies on coordinated communication between all agents, which is achieved by appending agents' bid information to the communication messages being propagated through the network. As the number of agents in the network increases, this

consensus approach may overflow the network bandwidth. Furthermore, a mechanism to ensure synchronized communication between agents (such as a heartbeat) is required, complicating the implementation and incurring unwanted artifical delays. This work extends the current CBBA algorithm by presenting a novel asynchronous communication protocol which produces consistent task assignments using relatively little bandwidth and without requiring artifical time delays. This new algorithm named Asynchronous CBBA (ACBBA), relies on agents communicating with their direct neighbors in an asynchronous fashion, using winning bid and timestamp information to ensure deconflicted task allocation assignments. This paper describes the new consensus approach and demonstrates the performance of ACBBA using a software implementation.

II. Background

II.A. Problem Statement

Given a list of N_t tasks and N_a agents, the goal of the task allocation algorithm is 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 maximum overall number of assignments achievable is given by $N_{\max} \triangleq \min\{N_t, N_a L_t\}$. 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.

This task assignment problem can be written as the following integer (possibly nonlinear) program, with binary decision variables x_{ij} that are used to indicate whether or not task j is assigned to agent i:

$$\begin{aligned} \max \quad & \sum_{i=1}^{N_a} \left(\sum_{j=1}^{N_t} c_{ij}(\tau_{ij}(\mathbf{p}_i(\mathbf{x}_i)))x_{ij} \right) \\ \text{subject to:} \quad & \sum_{j=1}^{N_t} x_{ij} \leq L_t, \ \forall i \in \mathcal{I} \\ & \sum_{i=1}^{N_a} x_{ij} \leq 1, \ \forall j \in \mathcal{J} \\ & x_{ij} \in \{0,1\}, \ \forall (i,j) \in \mathcal{I} \times \mathcal{J} \end{aligned} \tag{1}$$

where $x_{ij} = 1$ if agent i is assigned to task j, and $\mathbf{x}_i \triangleq \{x_{i1}, \ldots, x_{iN_t}\}$ is a vector of assignments for agent i, whose j^{th} element is x_{ij}. The index sets for i and j are defined as $\mathcal{I} \triangleq \{1, \ldots, N_a\}$ and $\mathcal{J} \triangleq \{1, \ldots, N_t\}$ representing the index sets for the agents and tasks respectively. The variable length vector $\mathbf{p}_i \triangleq \{p_{i1}, \ldots, p_{i|\mathbf{p}_i|}\}$ represents the path for agent i, an ordered sequence of tasks where the elements are the task indices, $p_{in} \in \mathcal{J}$ for $n = 1, \ldots, |\mathbf{p}_i|$, 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 $|\mathbf{p}_i|$ and may be no longer than L_t. The summation term in brackets in the objective function above represents the local reward for agent i.

Key assumptions underlying the above problem formulation are:

1. 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).

2. The arrival time τ_{ij} is *uniquely* defined as a function of the path $\mathbf{p}_i$ that agent i takes.

3. 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[4,10,27], in which the sooner an agent arrives at the target, the higher the reward it obtains. A more complex mission 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.

II.B. Consensus-Based Bundle Algorithm (CBBA)

The scoring function in Equation 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, previous combinatorial auction methods[36–39] 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) originally presented in[33,34] 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. The real-time implementation of CBBA has been demonstrated for heterogeneous teams and the algorithm has been extended to account for timing considerations associated with task execution[40–42].

In this section we present a brief review of the current CBBA. In order to perform this decentralized algorithm each agent must carry six vectors of information:

1. A *bundle*, $\mathbf{b}_i \triangleq \{b_{i1}, \ldots, b_{i|\mathbf{b}_i|}\}$, of variable length whose elements are defined by $b_{in} \in \mathcal{J}$ for $n = 1, \ldots, |\mathbf{b}_i|$. The current length of the bundle is denoted by $|\mathbf{b}_i|$, which cannot exceed the maximum length L_t, and an empty bundle is represented by $\mathbf{b}_i = \emptyset$ 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 $b_{i(n+1)}$).

2. A corresponding *path*, $\mathbf{p}_i \triangleq \{p_{i1}, \ldots, p_{i|\mathbf{p}_i|}\}$, whose elements are defined by $p_{in} \in \mathcal{J}$ for $n = 1, \ldots, |\mathbf{p}_i|$. 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; $|\mathbf{p}_i| = |\mathbf{b}_i| \leq L_t$.

3. A vector of times $\boldsymbol{\tau}_i \triangleq \{\tau_{i1}, \ldots, \tau_{i|\boldsymbol{\tau}_i|}\}$, whose elements are defined by $\tau_{in} \in [0, \infty)$ for $n = 1, \ldots, |\boldsymbol{\tau}_i|$. 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.

4. A winning agent list $\mathbf{z}_i \triangleq \{z_{i1}, \ldots, z_{iN_t}\}$, of size N_t, where each element $z_{ij} \in \{\mathcal{I} \cup \emptyset\}$ for $j = 1, \ldots, N_t$ 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.

5. A winning bid list $\mathbf{y}_i \triangleq \{y_{i1}, \ldots, y_{iN_t}\}$, also 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.

6. And finally, a vector of timestamps $\mathbf{s}_i \triangleq \{s_{i1}, \ldots, s_{iN_a}\}$, of size N_a, where each element $s_{ik} \in [0, \infty)$ for $k = 1, \ldots, N_a$ represents the timestamp of the last information update agent i received about agent k, either directly or through a neighboring agent.

The algorithm consists of iterations between two phases: a bundle construction phase and a consensus phase, which are described below.

II.B.1. Phase 1: Bundle Construction

In contrast to the bundle algorithms previously described in the literature[36–39], 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 agent's path (and/or bundle). Selecting the best score for task j can be performed using the following two steps. First, task j is "inserted" 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)[a]. 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, $\tau^\star_{ij}(\mathbf{p}_i \oplus_{n_j} j)$. This can be found by solving the following optimization problem:

$$\begin{aligned} \tau^\star_{ij}(\mathbf{p}_i \oplus_{n_j} j) = \quad & \underset{\tau_{ij} \in [0,\infty)}{\operatorname{argmax}} \, c_j(\tau_{ij}) \\ \text{subject to:} \quad & \tau^\star_{ik}(\mathbf{p}_i \oplus_{n_j} j) = \tau^\star_{ik}(\mathbf{p}_i), \quad \forall k \in \mathbf{p}_i. \end{aligned} \tag{2}$$

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[40]. Note that this is a continuous

[a]The notion of inserting task j into the path at location n_j involves shifting all path elements from n_j onwards by one and changing path element at location n_j to be task j (i.e $p_{i(n+1)} = p_{in}, \forall n \geq n_j$ and $p_{in_j} = j$)

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 $c_j(\tau_{ij}^\star(\mathbf{p}_i \oplus_{n_j} j))$. 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,

$$n_j{}^\star = \arg\max_{n_j} c_j(\tau_{ij}^\star(\mathbf{p}_i \oplus_{n_j} j)) \tag{3}$$

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

Once the scores for all possible tasks are computed ($c_{ij}(\mathbf{p}_i)$ for all $j \notin \mathbf{p}_i$)), the scores need to be checked against the winning bid list, $\mathbf{y}_i$, to see if any other agent has a higher bid for the task. We define the variable $h_{ij} = \mathbb{I}(c_{ij}(\mathbf{p}_i) > y_{ij})$, where $\mathbb{I}(\cdot)$ 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:

$$j^\star = \arg\max_{j \notin \mathbf{p}_i} c_{ij}(\mathbf{p}_i)h_{ij} \tag{4}$$

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

$$\begin{aligned}
\mathbf{b}_i &\leftarrow (\mathbf{b}_i \oplus_{\text{end}} j^\star) \\
\mathbf{p}_i &\leftarrow (\mathbf{p}_i \oplus_{n_j{}^\star} j^\star) \\
\boldsymbol{\tau}_i &\leftarrow (\boldsymbol{\tau}_i \oplus_{n_j{}^\star} \tau_{ij^\star}^\star(\mathbf{p}_i \oplus_{n_j{}^\star} j^\star)) \\
z_{ij} &= i \\
y_{ij} &= c_{ij^\star}(\mathbf{p}_i)
\end{aligned} \tag{5}$$

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 ($h_{ij} = 0$ for all $j \notin \mathbf{p}_i$). Notice that with Equation (5), a path is uniquely defined for a given bundle, while multiple bundles might result in the same path.

II.B.2. Phase 2: Consensus

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 $\mathbf{z}_i$, the winning bids list $\mathbf{y}_i$, and the vector of timestamps $\mathbf{s}_i$ 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,

$$s_{ik} = \begin{cases} \tau_r, & \text{if } g_{ik} = 1 \\ \max\{s_{mk} \mid m \in \mathcal{I}, g_{im} = 1\} & \text{otherwise,} \end{cases} \tag{6}$$

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 $\mathbf{z}_i$, $\mathbf{y}_i$, and $\mathbf{s}_i$ 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:

1. **Update**: $z_{ij} = z_{kj},\ y_{ij} = y_{kj}$
2. **Reset**: $z_{ij} = \emptyset,\ y_{ij} = 0$
3. **Leave**: $z_{ij} = z_{ij},\ y_{ij} = y_{ij}$.

The decision rules for this synchronous communication protocol were presented in[33] and are provided for convenience in Table 2 in Appendix A. 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". In Section III we present a revised communication protocol to handle *asynchronous* communication.

If either of the winning agent or winning bid information vectors ($\mathbf{z}_i$ or $\mathbf{y}_i$) 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 ($\bar{n} = \min\{n \mid z_{i(b_{in})} \neq i\}$ with b_{in} denoting the n^{th} entry of the bundle), then for all bundle locations $n \geq \bar{n}$, with corresponding task indices b_{in}, the following updates are made:

$$\begin{aligned} z_{i(b_{in})} &= \emptyset \\ y_{i(b_{in})} &= 0, \end{aligned} \tag{7}$$

The bundle is then truncated to remove these tasks,

$$\mathbf{b}_i \leftarrow \{b_{i1}, \ldots, b_{i(\bar{n}-1)}\} \tag{8}$$

and the 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.

II.B.3. Scoring Functions

It has previously been shown that if the scoring function satisfies a certain condition, called *diminishing marginal gain* (DMG), CBBA is guaranteed to produce a conflict-free assignment and converge in at most $\max\{N_t, L_t N_a\}D$ iterations, where D is the network diameter (always less than N_a)[33]. The DMG property states that the score for a task cannot increase as other elements are added to the set before it. In other words,

$$c_{ij}(\mathbf{p}_i) \geq c_{ij}(\mathbf{p}_i \oplus_n m) \tag{9}$$

for all $\mathbf{p}_i$, n, m, and j, where $m \neq j$ and $m, j \notin \mathbf{p}_i$.

Many reward functions in search and exploration problems for UAVs satisfy the DMG condition. The present authors have shown in[33,40] that DMG is satisfied for the following two cases: (a) time-discounted rewards, and (b) more generally time-windowed rewards.

Time-Discounted Reward Consider the following time-discounted reward[4,10,27] that has been commonly used for UAV task allocation problems:

$$c_{ij}(\mathbf{p}_i) = \lambda_j^{\tau_{ij}(\mathbf{p}_i)} R_j \tag{10}$$

where $\lambda_j < 1$ is the discount factor for task j, $\tau_{ij}(\mathbf{p}_i)$ is the estimated time agent i will take to arrive at task location j by following path $\mathbf{p}_i$, and R_j is a static reward associated with performing task j. The time-discounted reward can model search scenarios in which uncertainty growth with time causes degradation of the expected reward for visiting a certain location. Equation 10 could also be used to model planning of service routes in which client satisfaction diminishes with time. Since the triangular inequality holds for the actual distance between task locations,

$$\tau_{ij}(\mathbf{p}_i \oplus_n m) \geq \tau_{ij}(\mathbf{p}_i) \tag{11}$$

for all n and all $m \neq j$, $m \notin \mathbf{p}_i$. In other words, if an agent moves along a longer path, it arrives at each of the task locations at a later time than if it had moved along a shorter path, resulting in a further discounted score value. Therefore, assuming the task rewards R_j are nonnegative for all j, the score function in Equation 10 satisfies DMG.

Time-Windowed Reward To incorporating scoring functions with more complicated temporal dependencies we break the score function into two parts:

1. *Time Window, $w_j(\tau)$:* The time window of validity for a task represents the time in which the task is allowed to be started. For task j this window is defined as

$$w_j(\tau) = \begin{cases} 1, & \tau_{j_{start}} \leq \tau \leq \tau_{j_{end}} \\ 0, & \text{otherwise} \end{cases} \tag{12}$$

2. *Score Profile, $s_j(\tau)$:* The score profile $s_j(\tau)$ represents the reward an agent receives from task j when it arrives at the task at time τ. This score is based on the reward for the task, R_j. For example, for the time-discounted case described above this quantity is $s_j(\tau) = \lambda_j^{\Delta\tau} R_j$, where $\Delta\tau = \max\{0, \tau - \tau_{j_{start}}\}$ is the difference between the task start time and the agent arrival time, and $\lambda_j < 1$ is the discount factor to penalize late arrivals. Without time discounting $s_j(\tau) = R_j$.

The score an agent receives for a task is a function of his arrival time at the task location, τ_{ij}, and can be computed as $c_j(\tau_{ij}) = s_j(\tau_{ij}) w_j(\tau_{ij})$. The arrival time, τ_{ij}, is in turn a function of the path the agent has taken before reaching task j, as described in the previous sections, and can be optimized as in Equation 2. Using time windows for tasks provides a framework to penalize early arrivals as well as late arrivals and accelerates the computation of the optimal task execution time by restricting the range of values that the arrival time can take.

To verify that the time-windows framework satisfies the DMG property we want to ensure that for all $j \notin \mathbf{p}_i$,

$$c_{ij}(\mathbf{p}_i) \geq c_{ij}(\mathbf{p}_i'),$$

where $\mathbf{p}_i' = \{\mathbf{p}_i \oplus_{n_m^\star} m\}$ such that $n_m^\star$ is the optimal location in the path for task $m \notin \mathbf{p}_i$, $m \neq j$, with a corresponding optimal time of $\tau_{im}^\star$. Note that the constraint set $\tau_{ik}^\star(\mathbf{p}_i') = \tau_{ik}^\star(\mathbf{p}_i), \forall k \in \mathbf{p}_i$ is assumed to be satisfied during the addition of task m as described in Equation 2. For a new task $j \notin \mathbf{p}_i'$, the problem of computing $\tau_{ij}^\star$ given the new path $\mathbf{p}_i'$ becomes,

$$\begin{aligned} \tau_{ij}^\star(\mathbf{p}_i' \oplus_{n_j} j) = \quad & \underset{\tau_{ij} \in [0,\infty)}{\operatorname{argmax}} \, c_j(\tau_{ij}) \\ \text{subject to:} \quad & \tau_{ik}^\star(\mathbf{p}_i' \oplus_{n_j} j) = \tau_{ik}^\star(\mathbf{p}_i'), \quad \forall k \in \mathbf{p}_i'. \end{aligned} \tag{13}$$

for each insertion location n_j. The constraint set for this optimization can be rewritten as the following set of constraints,

$$\tau_{ik}((\mathbf{p}_i \oplus_{n_m^\star} m) \oplus_{n_j} j) = \tau_{ik}^\star(\mathbf{p}_i \oplus_{n_m^\star} m) = \tau_{ik}^\star(\mathbf{p}_i), \quad \forall k \in \mathbf{p}_i \tag{14}$$

$$\tau_{im}((\mathbf{p}_i \oplus_{n_m^\star} m) \oplus_{n_j} j) = \tau_{im}^\star(\mathbf{p}_i \oplus_{n_m^\star} m) \tag{15}$$

Note that the second equality represents an additional constraint to the original set corresponding to inserting task j in path $\mathbf{p}_i$. In fact, with each new task that is inserted into the path one additional constraint must be satisfied. Since at each iteration of the bundle building process we are solving a more constrained problem than we would have with a shorter bundle, the optimal score for tasks can only decrease, thus satisfying DMG.

III. Asynchronous CBBA (ACBBA)

III.A. Direct Implementation of CBBA and Associated Issues

As pointed out by Choi et al.[33], the bundle construction phase of CBBA can be executed independently and asynchronously because agents only require local information about the world. However,

implementing the nominal CBBA consensus phase in an asynchronous setting causes unfortunate issues that impact the system's convergence. The main problem is that the decision rules provided by the nominal CBBA deconfliction table require that every agent know the most recent information from each of its neighboring agents at each communication iteration. We could foresee this being implemented using a global heartbeat, where every agent shares its current state and then waits to receive messages from all of its neighbors (or waits a suitable amount of time), before moving onto the next algorithmic phase. For real-time systems this type of forced algorithmic delay is impractical and severely limits the performance and convergence rate of the algorithm, thus motivating the development of *asynchronous* deconfliction rules. In addition to these new rules, we created a lighter weight framework that in addition to handling asynchronous messages, also reduces the overall message bandwidth.

The main challenge associated with asynchronous communication protocols is that messages can arrive out of order, i.e., a message created earlier than another message could potentially arrive at a later time. Due to this, the agent timestamp s_{ik} that is updated using Equation 6 based on the message reception time might not correctly represent how up-to-date the information about agent k really is. As such, implementation of an asynchronous CBBA would require either of the following modifications: (a) delay the broadcast of new messages to ensure that the earlier messages have reached the target agents as mentioned above, or (b) change the message timestamps to represent the actual times that the winning agents placed their bids rather than the times that agents received messages about each other.

Considering option (b), some issues may still occur when messages are received out of order, as illustrated by the example in Figure 1. Here, Agent 1 sends a message at time t_1, with a bid of y_1. Before this message reaches Agent 2, Agent 2 bids on the same task at time t_2, with an associated bid, y_2; where $t_2 > t_1$ and $y_1 > y_2$. Agent 1's message will arrive at Agent 3's location first and Agent 3 will update its winners list to include this data. But just after this, Agent 2's message will arrive. This bid has a lower score but a later time. Agent 3 does not know if Agent 2's bid was made with knowledge of Agent 1's bid or not. Thus, simply updating the agent timestamp with the message creation time is not enough to correctly and safely implement CBBA in an asynchronous setting. One way to deal with this issue it to let agents keep broadcasting their information states with high frequency to increase likelihood that agents' most up-to-date information is shared across the network. However, the downside of this approach is clearly the excessive communication burden on the network, which could even be disastrous for low bandwidth systems.

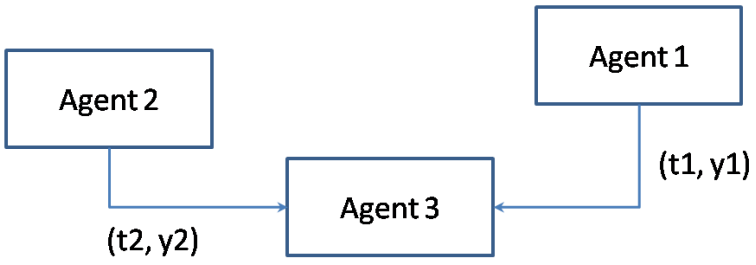

Figure 1. Example of Networked Agents with Asynchronous Communication

Therefore, to facilitate rapid information propagation and decision making while minimizing the communication load on the network, this paper proposes a new conflict resolution protocol for the implementation of CBBA in an asynchronous communication environment. This new protocol does not record when a message was received, it instead looks at the times at which bids were created, and on reception carefully deconflicts potentially ambiguous information.

III.B. Asynchronous Local Decision Rules

The set of local deconfliction rules for the new Asynchronous CBBA (ACBBA) algorithm are summarized in Table 1. The table describes what actions agent i should take after receiving a message from agent k. As before, the term z_{kj} refers to agent k's belief of who won task j and y_{kj} represents the associated winning bid. The new term t_{kj} refers to the timestamp of when this winning bid was made.

One key change in the ACBBA deconfliction protocol is that, not only does it specify how the receiver should update its winning agents and winning bids lists, but it also specifies what messages to rebroadcast. The main reasons for having this new rebroadcast option are: (a) to reduce the communication load over the network by preventing the rebroadcast of redundant information, and (b) to enable additional decision options to deal with the resultant ambiguity of the asynchronous system's timestamps. There are five possible actions the receiver can choose from:

1. **Update & Rebroadcast:** The receiver i updates its winning agent z_{ij}, winning bid y_{ij}, and winning time t_{ij} with the received information from the sender k. It subsequently rebroadcasts this update.

2. **Leave & Rebroadcast:** The receiver does not change its information state, but rebroadcast's its local copy of the winning agent's information because either, it believes its information is more correct than the sender's, or its unsure and its looking for confirmation from another agent.

3. **Leave & No-Rebroadcast:** The receiver neither changes its information state nor rebroadcasts it. This action is applied when it is clear that the received message is identical to the existing information.

4. **Reset & Rebroadcast:** The receiver resets its information state: $z_{ij} = \emptyset$ and $y_{ij} = 0$, and rebroadcasts the original *received* message so that the confusion can be resolved by other agents.

5. **Update Time & Rebroadcast:** The receiver is the task winner and receives a possibly confusing message. The receiver updates the timestamp on his bid to reflect the current time, confirming that he still thinks he is the winner. This helps to deconflict situations that often arise as illustrated in Figure 1.

Table 1. ACBBA decision rules for agent i based on communication with agent k regarding task j (z_{uj}: winning agent for task j from agent u's perspective; y_{uj}: winning bid on task j from agent u's perspective; t_{uj}: timestamp of the message agent u received associated with the current z_{uj} and y_{uj})

	Agent k (sender) thinks z_{kj} is	Agent i (receiver) thinks z_{ij} is	Receiver's Action (default: leave & rebroadcast)
1	k	i	if $y_{kj} > y_{ij} \rightarrow$ update & rebroadcast
2			if $y_{kj} = y_{ij}$ and $z_{kj} < z_{ij} \rightarrow$ update & rebroadcast
3			if $y_{kj} < y_{ij} \rightarrow$ update time & rebroadcast
4		k	if $t_{kj} > t_{ij} \rightarrow$ update & no-rebroadcast
5			$\lvert t_{kj} - t_{ij} \rvert < \epsilon_t \rightarrow$ leave & no-broadcast
6			if $t_{kj} < t_{ij} \rightarrow$ leave & no-rebroadcast
7		$m \notin \{i, k\}$	if $y_{kj} > y_{ij}$ and $t_{kj} \geq t_{ij} \rightarrow$ update & rebroadcast
8			if $y_{kj} < y_{ij}$ and $t_{kj} \leq t_{ij} \rightarrow$ leave & rebroadcast
9			if $y_{kj} = y_{ij} \rightarrow$ leave & rebroadcast
10			if $y_{kj} < y_{ij}$ and $t_{kj} > t_{ij} \rightarrow$ reset & rebroadcast
11			if $y_{kj} > y_{ij}$ and $t_{kj} < t_{ij} \rightarrow$ reset & rebroadcast
12		none	update & rebroadcast
13	i	i	if $\lvert t_{kj} - t_{ij} \rvert < \epsilon_t \rightarrow$ leave & no-rebroadcast
14		k	reset & rebroadcast
15		$m \notin \{i, k\}$	leave & rebroadcast
16		none	leave & rebroadcast
17	$m \notin \{i, k\}$	i	if $y_{kj} > y_{ij} \rightarrow$ update & rebroadcast
18			if $y_{kj} = y_{ij}$ and $z_{kj} < z_{ij} \rightarrow$ update & rebroadcast
19			if $y_{kj} < y_{ij} \rightarrow$ update time & rebroadcast
20		k	if $t_{kj} \geq t_{ij} \rightarrow$ update & rebroadcast
21			if $t_{kj} < t_{ij} \rightarrow$ reset & rebroadcast
22		m	if $t_{kj} > t_{ij} \rightarrow$ update & no-rebroadcast
23			$\lvert t_{kj} - t_{ij} \rvert < \epsilon_t \rightarrow$ leave & no-rebroadcast
24			if $t_{kj} < t_{ij} \rightarrow$ leave & no-rebroadcast
25		$n \notin \{i, k, m\}$	if $y_{kj} > y_{ij}$ and $t_{kj} \geq t_{ij} \rightarrow$ update & rebroadcast
26			if $y_{kj} < y_{ij}$ and $t_{kj} \leq t_{ij} \rightarrow$ leave & rebroadcast
27			if $y_{kj} = y_{ij} \rightarrow$ leave & rebroadcast
28			if $y_{kj} < y_{ij}$ and $t_{kj} > t_{ij} \rightarrow$ reset & rebroadcast
29			if $y_{kj} > y_{ij}$ and $t_{kj} < t_{ij} \rightarrow$ reset & rebroadcast
30		none	update & rebroadcast
31	none	i	leave & rebroadcast
32		k	update & rebroadcast
33		$m \notin \{i, k\}$	if $t_{kj} > t_{ij} \rightarrow$ update & rebroadcast
34		none	leave & no-rebroadcast
	NOTE:	rebroadcast	With "leave," broadcast own information With "update," broadcast sender's information With "reset," broadcast sender's information With "update time," broadcast own information

III.B.1. Remarks on Key Decision Rules

- **Redundancy Detection (Lines 5, 13, and 23):** For these entries the sender's information is consistent with the receiver's information; also, the timestamps for both pieces of information are the same (within some small ϵ_t). This means that both pieces of information are based on the identical message originating from the same agent $z_{kj}(= z_{ij})$ at time $t_{kj}(= t_{ij})$. In these cases, the receiver can safely judge that the two messages are identical to each other, and does not need to rebroadcast the message (presumably because the agent broadcast this message the first time it received the information). This is necessary to ensure that this message is not bounced back and forth between agents unnecessarily forever. Moreover, as a result of this redundancy detection, when the network goes silent, we know that that the system has converged to its final solution. Note that the original CBBA does not support this redundancy detection functionality.

- **Tie-Breaking (Lines 2 and 18):** Ties are broken on the basis of smallest-index agent ID. The receiver is only allowed to break ties when it thinks it is one of the winning agents. This is because otherwise the receiver's information about z_{kj} might not be up-to-date and could therefore break ties using old information (such as in lines 9 and 27). Ties in lines 9 and 27 will eventually be broken by the prospective winning agents z_{kj} or z_{ij} if the network is connected. The original CBBA assumes consistent tie-breaking without explicitly including it in the local decision rules.

- **Dropping Tasks Based on Relayed Information (Line 17):** This is the baseline case when the receiver gets outbid for a task by another agent. This process is especially crucial for conflict resolution in a non-fully connected network, where deciding upon the actual winner is often non-trivial considering the receiver could have no direct contact with the other prospective winning agents.

  In the original CBBA, the decision rule was: if $y_{kj} > y_{ij}$ and $s_{km} > s_{im}$, then "update." In other words, if the receiver has older information about the prospective winning agent, i.e., z_{kj}, then the receiver drops its assignment and updates with the newer information. However, in the asynchronous setting, s_{km} and s_{im}, i.e., the last time k and i heard about m, are not available; only the message timestamps t_{kj} and t_{ij} are available. Note that $t_{kj} > t_{ij}$ does not mean that the sender's information about $m = z_{kj}$ is more reliable than the receiver's information about the same agent m. Moreover, because $z_{ij} = i$ and agents tend to have the most up-to-date information about themselves, it is likely that $t_{kj} < t_{ij}$. Therefore, replacing the condition for s_{km} and s_{im} by a similar condition for t_{kj} and t_{ij} will lead to agents not dropping their tasks, causing the whole distributed negotiation process to stall.

  For this reason, this work suggests only making decisions based on the winning bids, as shown in the particular case given on line 17. With these revised rules, agents more readily lose their assignments than in the original CBBA, because there is more doubt about message validity in this asynchronous environment. However, the CBBA process is iterative, and rules in lines 15, 17, 20, and 21 applied for communications with z_{kj} being a sender, will eventually create

updated information about z_{kj} that will propagate through the network. At that point, agent i can apply rules as the receiver in lines 4 through 11 to reflect the new information about z_{kj}, eventually leading to a globally consistent decision across the network.

- **Update to Propagate Information of Winning Agent (Lines 7 through 12):** This case is one of the most common cases, and is a key step to propagate the information from the winning agent. In the original CBBA, the decision rule was: if $s_{km} > s_{im}$ or $y_{kj} > y_{ij}$, then update. Namely, if the sender, claiming that it is the winning agent, has more up-to-date information about the competing winning agent m from the receiver's perspective, the receiver takes the sender's information because at that point the sender has already found that it outbid m.

  However, for the asynchronous setting, s_{km} and s_{im} are not available, and $t_{kj} \geq t_{ij}$ does not mean that the sender k has a better idea about the winning agent $z_{ij} = m$ from the receiver's perspective. Thus, the receiver needs to be more restrictive on incorporating the sender's information. In particular, as in lines 10 and 11, if the timestamp on the arguably-winning bid is old, the receiver does not make a strong decision about who is the winner and just forwards the message from the sender, clearing its information state. In a connected network, this forwarded message will eventually go to m or some other agents that have more up-to-date information; at that point, a clearer decision on task j will be made. This might incur some latency in the overall decision process, but will also avoid cycling between conflicted assignments in the network.

  Note that compared to the original CBBA, agents more readily update their assignments (case $z_{kj} = m$ and $z_{ij} = i$), but are more restrictive on propagating the information from the agent claiming to be a winning agent (case $z_{kj} = k$ and $z_{ij} = m$). This balances maintains the speed of information propagation while preserving safety in the decision making.

It should be pointed out that with these changes in the decision rules, ACBBA might not produce exactly the same solution as CBBA (applied to a synchronized setting). However, empirical studies have demonstrated that ACBBA produces conflict-free solutions for all the cases that the original CBBA does, with similar achieved scores. Results comparing the performance of CBBA and ACBBA are presented in the next section.

IV. Software Implementation & Results

IV.A. Decentralized Software Architecture

The asynchronous CBBA algorithm implementation allows each agent to separate the build bundle operations from the message passing and receiving operations. This ensures that each agent in the network can run its bundle building operation on its own schedule, making the algorithm robust to large communication delays and drop outs. The specific approach was implemented using two separate modules running in different threads, a `Listener` and a `BundleBuilder`. The `Listener` is the piece that receives and deconflicts all of the information sent by other agents. The

`BundleBuilder` then takes this information, builds a local bundle, and rebroadcasts the results. The overall system architecture is depicted in Figure 2.

The `Listener` operates in its own thread and receives messages sent by other agents. This communication takes the form of either a `TaskInfo` message or a `TaskBidInfo` message. `TaskInfo` messages consist of all the information needed for the `BundlBuilder` to calculate a bid. These messages do not change in time. `TaskBidInfo` messages are much smaller and only contain the information necessary for task assignment consensus as specified by Table 1. The elements of this message are the sender ID, the task ID, the winning agent ID, the winning agent score, and the time at which the bid was last updated.

The `Listener` continuously fills one of two buffers, and possibly one of two rebroadcast buffers with these messages. A flag set in the `BundleBuilder` thread switches between the two buffers in order to prevent overwriting active data. These two buffers have a special feature in that they only store the last message entering for a given task ID. This ensures that the other parts of the code receive only the most recent information about a task. Since this is run in a separate thread from the main CBBA bundle building routine, the `Listener` thread does not hang during bundle building, allowing for low incoming message queue sizes.

The `BundleBuilder` thread obtains information from the buffer mentioned above, which is populated by the `Listener`. This thread then proceeds to build a bundle based on the latest information. All changes made to the bundle are logged and sent to the rebroadcast buffer. At the end of the `BundleBuilder` thread, the rebroadcast function is called. In this rebroadcast buffer, there is usually a mixture of data from the `Listener` and from the `BundleBuilder`. The information in the rebroadcast buffer that was sent from the `Listener` is exactly the same information that the `BundleBuilder` thread received before it built its most recent bundle. Packing the rebroadcast information in this way allows updates made in the `BundleBuilder` to prune outdated information in the outgoing queue. This guarantees that the outgoing messages accurately represent the state of the current bundle build, saving computation resources and message bandwidth at the expense of a longer rebroadcast period (with more messages at each iteration).

IV.B. Implementation and Results

The implementation of ACBBA is completely decentralized in that each individual agent runs on a separate computer. All of these computers are connected to an internal lab network and communicate through the User Datagram Protocol (UDP). This means that there is no shared memory between agents and the only communication between agents is through messages sent via UDP.

To compare the performance of ACBBA versus the original synchronous CBBA, a Monte Carlo experiment was implemented. The test script randomly generated tasks over the theater of operation with random start and end times. Specifically the Monte Carlo runs consisted of 54 trials run at every even number of tasks between 2 and 40, inclusive. All 1080 Monte Carlo runs were performed with 9 agents, and the bundle size for each agent was capped at 5 tasks.

We used a MATLAB implementation of the synchronous CBBA algorithm to compare with

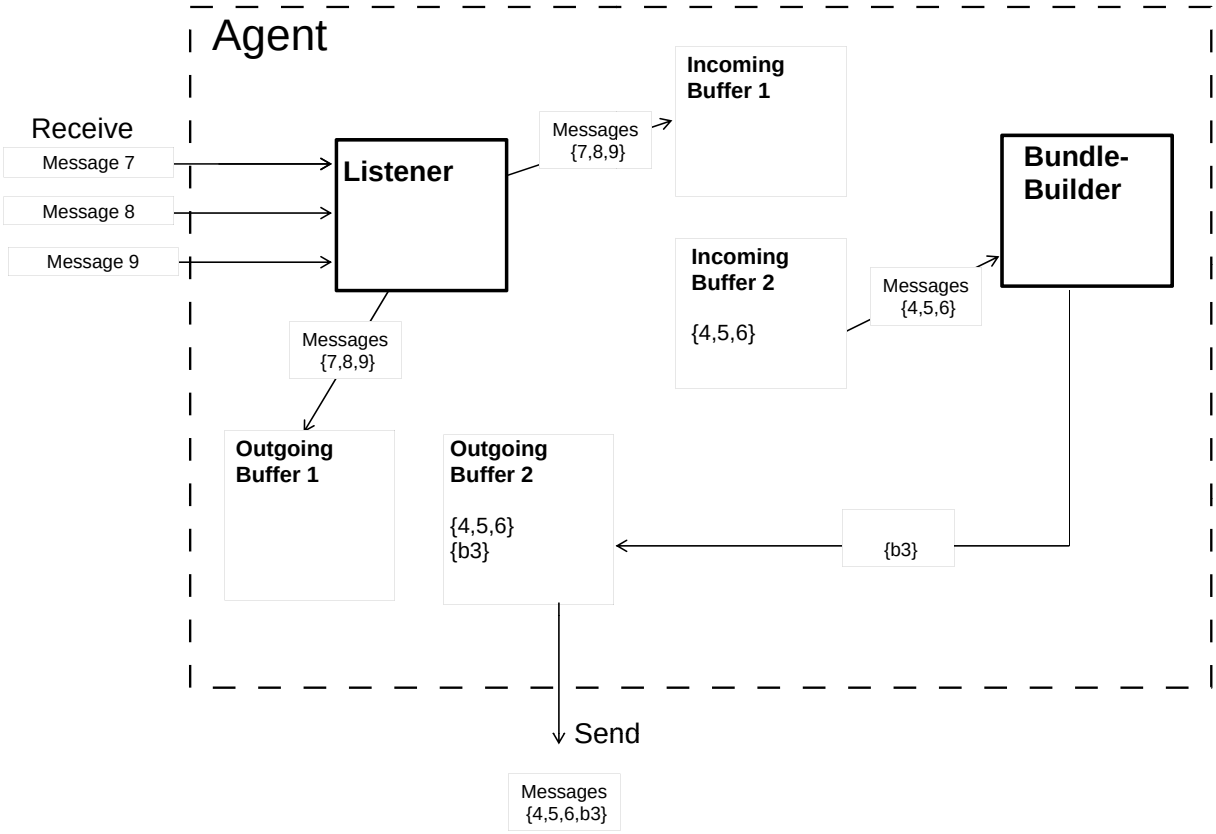

Figure 2. Agent decision architecture: Each agent runs two main threads, Listener and BundleBuilder; these two threads interface via the Incoming Buffers – there are two buffers to avoid unexpected message overriding; outgoing buffers manage rebroadcasting of information, triggered by the completion of the BundleBuilder thread.

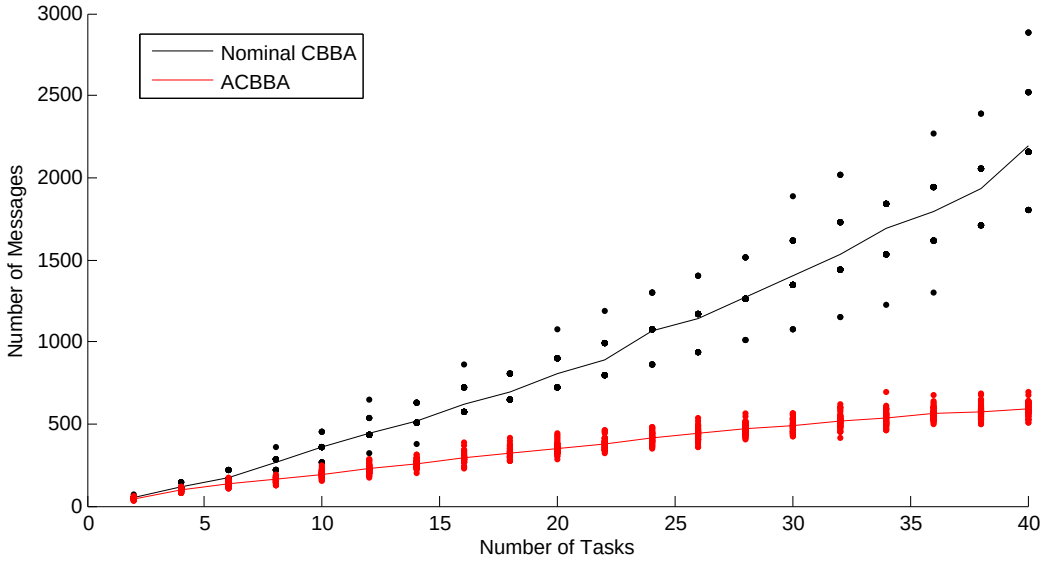

Figure 3. Comparison of the number of messages sent between asynchronous CBBA (ACBBA) and synchronous CBBA in a fully connected network.

the C++ implementation of ACBBA. The results in Figure 3 show a comparison of the number of TaskBidInfo messages sent in a fully connected network in order to converge to a plan. It is worth noting that both algorithms usually produced the exact same plan and score. Trials in which the scores differed occurred rarely and only at high task numbers due to the differences in synchroneity. There was no trend as to which algorithm would have a higher score in these cases and the differences were usually within a percent. Another note about the figure is that even though it looks like there are only a few data points, all 54 trials are recorded for each task number. In the synchronous CBBA, the number of messages is discretized with the number of iterations the algorithm takes to converge, therefore for each number of tasks, there were only 3-4 corresponding discrete values for number of messages sent.

Figure 4 below shows similar results for another scenario where the network topology is now a line network instead of being fully connected. We set up the line network such that Agents 1-9 could only talk to their numeric neighbors (i.e. Agent 3 can only communicate with Agents 2 and 4), with agents at the ends of the line only communicating with one other agent.

The results demonstrate that ACBBA is able to converge to a solution using significantly fewer messages, which are almost an order of magnitude less for cases nearing full bundles. Although, other network configurations were not tested we expect these results to be characteristic of the algorithm's performance. As the network topology becomes more sparse, we expect ACBBA to outperform the original synchronous CBBA significantly in terms of message bandwidth and time to convergence, while achieving a similar score.

V. Conclusions

This work extends the Consensus-Based Bundle Algorithm (CBBA), a decentralized task allocation algorithm for heterogeneous networked teams, to explicitly handle communication through

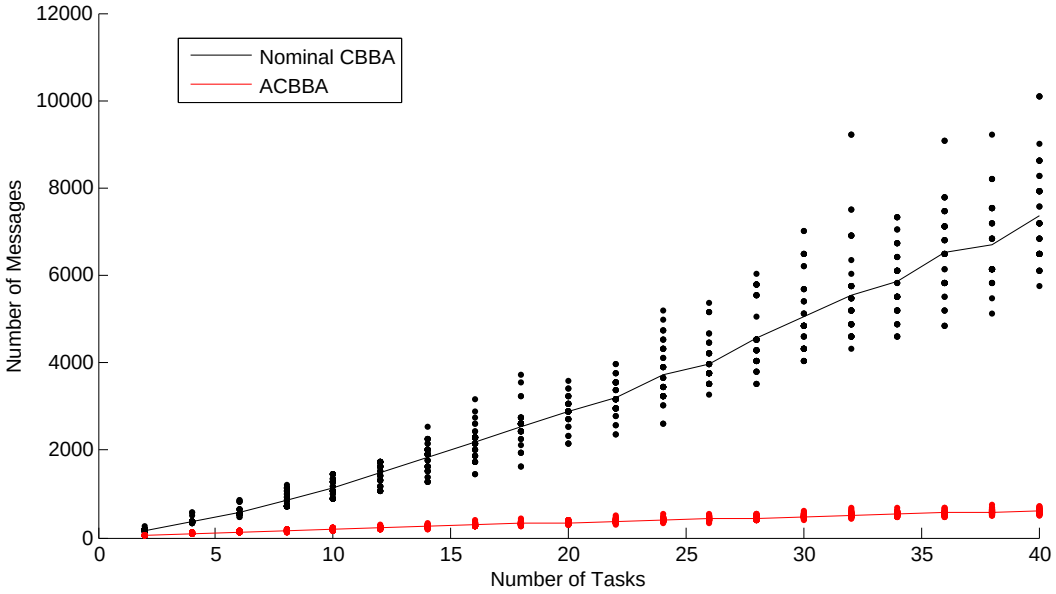

Figure 4. Comparison of the number of messages sent between asynchronous CBBA (ACBBA) and synchronous CBBA in a line network.

asynchronous channels. Direct implementation of CBBA in an asynchronous setting would require synchronization through a heartbeat and frequent broadcasting of agents' information states, incurring time delays and excessive bandwidth usage. In contrast, ACBBA uses a new set of local deconfliction rules that do not require access to the global information state, thus minimizing communication load and time delays while preserving the convergence properties of the original algorithm. This new deconfliction protocol also features consistent handling of out-of-order messages and detection of redundant information. A real-time software implementation using multiple PCs communicating through UDP was used to validate the proposed algorithm. The results show that ACBBA achieves similar mission performance to the original CBBA with significantly faster convergence and fewer overall messages passed between agents.

Acknowledgments

This research was supported by AFOSR (FA9550-08-1-0086) and by Boeing Research & Technology, Seattle, WA.

References

[1] "Unmanned Aircraft Systems Roadmap, 2005-2030," Tech. rep., Office of the Secretary of Defense, August 2005.

[2] United States Air Force Scientific Advisory Board, "Air Force Operations in Urban Environments – Volume 1: Executive Summary and Annotated Brief," Tech. Rep. SAB-TR-05-01, United States Air Force Scientific Advisory Board, `http://www.au.af.mil/au/awc/awcgate/`

sab/af_urban_ops_2005.pdf, August 2005.

[3] Headquarters, U. S. A. F., "United States Air Force Unmanned Aircraft Systems Flight Plan 2009-2047," Tech. rep., USAF, Washington DC, http://www.govexec.com/pdfs/072309kp1.pdf, 2009.

[4] Bellingham, J., Tillerson, M., Richards, A., and How, J., "Multi-Task Allocation and Path Planning for Cooperating UAVs," *Proceedings of Conference of Cooperative Control and Optimization*, Nov. 2001.

[5] Schumacher, C., Chandler, P. R., and Rasmussen, S. R., "Task allocation for wide area search munitions," *American Control Conference, 2002. Proceedings of the 2002*, Vol. 3, 2002, pp. 1917–1922 vol.3.

[6] Cassandras, C. and Li, W., "A Receding Horizon Approach for Solving Some Cooperative Control Problems," *Proceedings of the IEEE Conference on Decision and Control*, 2002.

[7] Jin, Y., Minai, A., and Polycarpou, M., "Cooperative Real-Time Search and Task Allocation in UAV Teams," *Proceedings of the IEEE Conference on Decision and Control*, 2003.

[8] Xu, L. and Ozguner, U., "Battle management for unmanned aerial vehicles," *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, Vol. 4, 9-12 Dec. 2003, pp. 3585–3590 vol.4.

[9] Turra, D., Pollini, L., and Innocenti, M., "Fast Unmanned Vehicles Task Allocation with Moving Targets," *Proceedings of the IEEE Conference on Decision and Control*, Dec 2004, pp. 4280 – 4285.

[10] Alighanbari, M., *Task assignment algorithms for teams of UAVs in dynamic environments*, Master's thesis, Massachusetts Institute of Technology, 2004.

[11] McLain, T. W. and Beard, R. W., "Coordination Variables, Coordination Functions, and Cooperative-Timing Missions," *Journal of Guidance, Control, and Dynamics*, Vol. 28(1), 2005, pp. 150–161.

[12] Castanon, D. A. and Wu, C., "Distributed algorithms for dynamic reassignment," *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, Vol. 1, 9-12 Dec. 2003, pp. 13–18 Vol.1.

[13] Curtis, J. and Murphey, R., "Simultaneous Area Search and Task Assignment for a Team of Cooperative Agents," *Proceedings of AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2003.

[14] Shima, T., Rasmussen, S. J., and Chandler, P., "UAV team decision and control using efficient collaborative estimation," *Proceedings of the American Control Conference*, 8-10 June 2005, pp. 4107–4112 vol. 6.

[15] Ren, W., Beard, R. W., and Kingston, D. B., "Multi-agent Kalman consensus with relative uncertainty," *Proceedings of the American Control Conference*, 8-10 June 2005, pp. 1865–1870 vol. 3.

[16] Ren, W. and Beard, R., "Consensus seeking in multiagent systems under dynamically changing interaction topologies," Vol. 50, No. 5, May 2005, pp. 655–661.

[17] Olfati-Saber, R. and Murray, R. M., "Consensus Problems in Networks of Agents with Switching Topology and Time-Delays," *IEEE Transactions on Automatic Control*, Vol. 49(9), 2004,

pp. 1520–1533.

[18] Alighanbari, M. and How, J. P., "Unbiased Kalman Consensus Algorithm," *Journal of Aerospace Computing Information and Control*, Vol. 5, No. 9, 2008, pp. 209–311.

[19] Moallemi, C. C. and Roy, B. V., "Consensus Propagation," *IEEE Transactions on Information Theory*, Vol. 52(11), 2006, pp. 4753–4766.

[20] Olshevsky, A. and Tsitsiklis, J. N., "Convergence Speed in Distributed Consensus and Averaging," *Proceedings of the 45th IEEE Conference on Decision and Control*, Dec 2006, pp. 3387 – 3392.

[21] Ren, W., Beard, R. W., and Atkins, E. M., "Information consensus in multivehicle control," *IEEE Control Systems Magazine*, Vol. 27(2), 2007, pp. 71–82.

[22] Hatano, Y. and Mesbahi, M., "Agreement over Random Networks," *IEEE Transactions on Automatic Control*, Vol. 50, No. 11, Nov 2005, pp. 1867–1872.

[23] Wu, C. W., "Synchronization and Convergence of Linear Dynamics in Random Directed Networks," *IEEE Transactions on Automatic Control*, Vol. 51(7), 2006.

[24] Tahbaz-Salehi, A. and Jadbabaie, A., "On Consensus Over Random Networks," *44th Annual Allerton Conference*, 2006.

[25] Olfati-Saber, R., Fax, J., and Murray, R., "Consensus and cooperation in networked multi-agent systems," *Proceedings of the IEEE*, Vol. 95, No. 1, 2007, pp. 215–233.

[26] Ren, W. and Beard, R., *Distributed consensus in multi-vehicle cooperative control: theory and applications*, Springer Verlag, 2008.

[27] Alighanbari, M. and How, J., "Decentralized Task Assignment for Unmanned Aerial Vehicles," *Proc. of the European Control Conference and the 44th IEEE Conference on Decision and Control (CDC-ECC '05)*, 2005, pp. 5668–5673.

[28] Sariel, S. and Balch, T., "Real Time Auction Based Allocation of Tasks for Multi-Robot Exploration Problem in Dynamic Environments," *Proceedings of the AIAA Workshop on "Integrating Planning Into Scheduling"*, 2005.

[29] Ahmed, A., Patel, A., Brown, T., Ham, M., Jang, M., and Agha, G., "Task assignment for a physical agent team via a dynamic forward/reverse auction mechanism," *International Conference on Integration of Knowledge Intensive Multi-Agent Systems*, 2005.

[30] Atkinson, M. L., "Results Analysis of Using Free Market Auctions to Distribute Control of UAVs," *AIAA 3rd "Unmanned Unlimited" Technical Conference, Workshop and Exhibit*, 2004.

[31] Lemaire, T., Alami, R., and Lacroix, S., "A Distributed Task Allocation Scheme in Multi-UAV Context," *Proceedings of the IEEE International Conference on Robotics and Automation*, 2004.

[32] Walsh, W. and Wellman, M., "A market protocol for decentralized task allocation," *Proceedings of International Conference on Multi Agent Systems*, 1998.

[33] Choi, H.-L., Brunet, L., and How, J. P., "Consensus-Based Decentralized Auctions for Robust Task Allocation," *IEEE Trans. on Robotics*, Vol. 25 (4), 2009, pp. 912 – 926.

[34] Brunet, L., *Consensus-Based Auctions for Decentralized Task Assignments*, Master's thesis, Massachusetts Institute of Technology, 2008.

[35] Bertuccelli, L., Choi, H., Cho, P., and How, J., "Real-time Multi-UAV Task Assignment in

Dynamic and Uncertain Environments," *AIAA Guidance, Navigation, and Control Conference*, (AIAA 2009-5776) 2009.

[36] Parkes, D. C. and Ungar, L. H., "Iterative Combinatorial Auctions:Theory and Practice," *Proceedings of the 17th National Conference on Artificial Intelligence*, 2000.

[37] Berhault, M., Huang, H., Keskinocak, P., Koenig, S., Elmaghraby, W., Griffin, P., and Kleywegt, A., "Robot Exploration with Combinatorial Auctions," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003.

[38] Andersson, A., Tenhunen, M., and Ygge, F., "Integer programming for combinatorial auction winner determination," *Proceedings of the Fourth International Conference on MultiAgent Systems*, 2000.

[39] de Vries, S. and Vohra, R., "Combinatorial auctions: A survey," *INFORMS Journal of Computing*, Vol. 15(3), 2003, pp. 284–309.

[40] Ponda, S., Redding, J., Choi, H. L., How, J., Vavrina, M. A., and Vian, J., "Decentralized Planning for Complex Missions with Dynamic Communication Constraints," *American Control Conference*, June/July 2010.

[41] Ponda, S., Choi, H.-L., and How, J. P., "Predictive planning for heterogeneous human-robot teams," *AIAA Infotech@Aerospace*, April 2010.

[42] Choi, H.-L., Whitten, A. K., and How, J. P., "Decentralized Task Allocation for Heterogeneous Teams with Cooperation Constraints," July 2010.

A. Original Decision Rules for CBBA

Table 2. CBBA Action rules for agent i based on communication with agent k regarding task j

Agent k (sender) thinks z_{kj} is	**Agent i (receiver) thinks z_{ij} is**	**Receiver's Action (default: leave)**
k	i	if $y_{kj} > y_{ij} \rightarrow$ update
	k	update
	$m \notin \{i, k\}$	if $s_{km} > s_{im}$ or $y_{kj} > y_{ij} \rightarrow$ update
	none	update
i	i	leave
	k	reset
	$m \notin \{i, k\}$	if $s_{km} > s_{im} \rightarrow$ reset
	none	leave
$m \notin \{i, k\}$	i	if $s_{km} > s_{im}$ and $y_{kj} > y_{ij} \rightarrow$ update
	k	if $s_{km} > s_{im} \rightarrow$ update else $\rightarrow$ reset
	m	$s_{km} > s_{im} \rightarrow$ update
	$n \notin \{i, k, m\}$	if $s_{km} > s_{im}$ and $s_{kn} > s_{in} \rightarrow$ update if $s_{km} > s_{im}$ and $y_{kj} > y_{ij} \rightarrow$ update if $s_{kn} > s_{in}$ and $s_{im} > s_{km} \rightarrow$ reset
	none	if $s_{km} > s_{im} \rightarrow$ update
none	i	leave
	k	update
	$m \notin \{i, k\}$	if $s_{km} > s_{im} \rightarrow$ update
	none	leave