Asynchronous Decentralized Task Allocation for Dynamic Environments

Luke B. Johnson[*], Sameera S. Ponda[†], Han-Lim Choi[‡], Jonathan P. How[§]

This work builds on a decentralized task allocation algorithm for networked agents communicating through an *asynchronous* channel. The algorithm extends the Asynchronous Consensus-Based Bundle Algorithm (ACBBA) to account for more real time implementation issues resulting from a decentralized planner. This work utilizes a new implementation that allows further insight into the consensus and message passing structures of ACBBA. This paper specifically talks to the comparisons between global and local convergence in asynchronous consensus algorithms. Also a feature called asychronous replan is introduced to ACBBA's functionality that enables efficient updates to large changes in local situational awareness. A real-time software implementation using multiple agents communicating through the user datagram protocol (UDP) validates the proposed algorithm.

I. Introduction

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 a fleet of robotic agents. Many different methods have been considered for enabling 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, 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 distributed planning architecture offers performance and robustness advantages. In particular, response times to changes in situational awareness can be made significantly faster via distributed control than those achieved under a purely centralized planner. As a result, distributed 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 in order to ensure that agents have the same situational awareness before planning. In the presence of inconsistencies in situational awareness, these distributed tasking algorithms can be augmented with consensus algorithms[15–24] 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.[25]

[*]Dept. of Aeronautics and Astronautics,MIT, Cambridge, MA, `lbj16@mit.edu`
[†]Dept. of Aeronautics and Astronautics, MIT, Cambridge, MA, `sponda@mit.edu`
[‡]Div. of Aerospace Engineering, KAIST, Daejeon, Korea, `hanlimc@kaist.ac.kr`
[§]R. C. Maclaurin Professor of Aeronautics and Astronautics, MIT and Associate Fellow of AIAA `jhow@mit.edu`

Other popular task allocation methods involve using distributed auction algorithms,[26–29] which have been shown to efficiently produce sub-optimal solutions. One such algorithm is the Consensus-Based Bundle Algorithm (CBBA),[30,31] a multi-assignment distributed 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,[30] although empirically its performance is shown to be within 93% of the optimal solution.[32] 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 at each iteration of the algorithm, the consensus phase relies on synchronized communication between all agents. In order to operate in a decentralized *synchronous* environment, artificial delays and extra communication must be built into algorithms to ensure agents remain in sync. These delays and extra messages reduce mission performance and may even be unrealizable in physical systems.

This work extends CBBA for networked agents communicating through an *asynchronous* channel. The algorithmic extensions proposed in this paper allow the Asynchronous Consensus-Based Bundle Algorithm (ACBBA) to operate in real-time dynamic tasking environments. As previous work has shown,[33] ACBBA can provide the task space convergence properties of nominal CBBA,[30] while maintaining a relatively low message bandwidth. The power of asynchronous algorithms emerges when distributed methods are implemented in real time. ACBBA applies a set of local deconfliction rules that do not require access to the global information state, consistently handle of out-of-order messages and detect redundant information.

II. Background

II.A. Problem Statement

Given a list of N_t tasks and N_a agents, the goal of a 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:

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

where $x_{ij} = 1$ if agent i is assigned to task j, and $\mathbf{x}_i \triangleq \{x_{i1}, \dots, x_{iN_t}\}$ 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 $\mathcal{I} \triangleq \{1, \dots, N_a\}$ and $\mathcal{J} \triangleq \{1, \dots, N_t\}$, representing the index sets for the agents and tasks respectively. The variable length vector $\mathbf{p}_i \triangleq \{p_{i1}, \dots, p_{i|\mathbf{p}_i|}\}$ represents the path for agent i, an ordered sequence of tasks where the elements are the task indices, $p_{in} \in \mathcal{J}$ for $n = 1, \dots, |\mathbf{p}_i|$, 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 $|\mathbf{p}_i|$ 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, 25] 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[34–37] 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 [30, 31] 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.[38–40]

In this section we present a brief review of 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,[34–37] 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.

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. A more detailed discussion of the interaction between these two phases can be found in [33] .

III. Synchronous CBBA

CBBA was originally designed to be decentralized but was tightly synchronized. Many possible solutions were considered for implementing this synchronization. Figure 1 highlights the main reasons why these attempts were unsuccessful. The first idea was to impose a global heartbeat on the entire network that would work to resynchronize every consensus step in the same way that the distributed version of CBBA was implemented. As can be seen in the left 2 blocks of Figure 1, if the consensus phase iteration is too long then time is wasted by forcing all agents to idle during the end of the consensus phase. If the consensus phase iteration time is too short then some agents are not admitted into certain rounds of consensus. This effectively means that the algorithm is creating an artificial dynamic network, which is detrimental to convergence properties. Even more likely would be that a combination of the previous two problems would happen every single iteration, further complicating and slowing consensus. The third idea involved having all agents wait for all other agents to return with assignments, then only advance once they had heard from everyone. This solution was determined to be non-robust for our planning applications because it was unclear how to scale this method to handle dynamic networks with possible dropped connectivity.

Given these and other reasons explained in further detail in [33], an asynchronous version of the algorithm was developed.

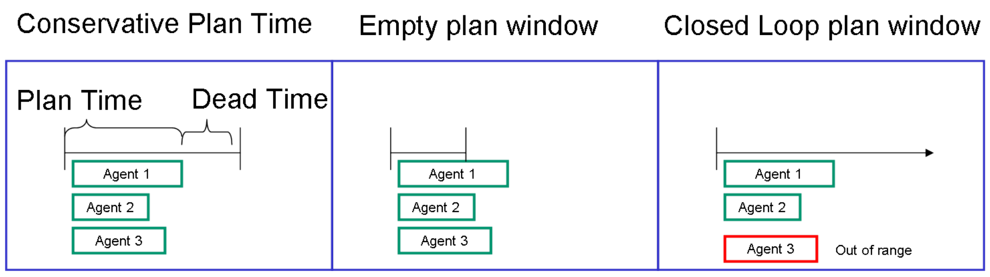


Figure 1. Possible Implementations of a Synchronization in a decentralized system

IV. Asynchronous CBBA (ACBBA)

To ensure the greatest degree and flexibility in decentralized environments, the Asynchronous Consensus Based Bundle Algorithm (ACBBA) was created. The main feature enabled with ACBBA is the ability for each agent to build bundles and perform consensus on their own schedule. This requires breaking the synchronous assumptions of CBBA and allowing the algorithm to run both the Bundle Building Phase and the Consensus Phase simultaneously and asynchronously. This is accomplished by carefully understanding information states on the agent level in combination with a new consensus task deconfliction table to account for more complicated information interactions. One of the other useful properties that emerge when the information state of the agent is carefully controlled include the introduction of a rebroadcast decision. Each agent then only broadcasts new or relevant information, reducing the overall message bandwidth requirements of a decentralized system. Given that ACBBA assumes worst case networking scenarios, it has robustness built into the deconfliction table that handles dynamic network topologies naturally. This is accomplished by ensuring that the task consensus process can be run using only local information. As a result, agents may enter and leave the network and tasks may be created or deleted, all in a completely decentralized manner.

IV.A. Local Deconfliction Rules of ACBBA

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 then propagates this new information.

2. **Leave & Rebroadcast:** The receiver does not change its information state, but rebroadcasts its local copy of the winning agent's information because either it believes its information is more correct than the sender's, or the agent is unsure and it's 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 the information is either not new or is outdated and should have been corrected already.

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 & 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 $t_{kj} > t_{ij} \rightarrow$ update & rebroadcast
11			if $y_{kj} > y_{ij}$ and $t_{kj} < t_{ij} \rightarrow$ update & 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	update & rebroadcast
22		m	if $t_{kj} > t_{ij} \rightarrow$ update & rebroadcast
23			$\lvert t_{kj} - t_{ij} \rvert < \epsilon_t \rightarrow$ leave & no-rebroadcast
24			if $t_{kj} < t_{ij} \rightarrow$ leave & 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
28			if $y_{kj} < y_{ij}$ and $t_{kj} > t_{ij} \rightarrow$ update & rebroadcast
29			if $y_{kj} > y_{ij}$ and $t_{kj} < t_{ij} \rightarrow$ leave & 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
		rebroadcast*	empty bid with current time

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:** This case happens when the receiver is the task winner and observes a possibly confusing message. The receiver updates the timestamp on his bid to reflect the current time, confirming that the bid is still active at the current time.

More remarks on the key decision rules, and the more detailed implementation details for ACBBA can be viewed in [33].

V. Convergence

As a feature of the algorithm, ACBBA runs the Bundle Building and Consensus Phases simultaneously and continuously in separate threads. This added flexibility changes the way that we must think of convergence. In this continuously running system there is never a point in which CBBA ceases to execute. This means that other criteria need to be constructed to recognize that the system has entered a stable, conflict-free assignment. Furthermore, this criterion is also required to be a local measure, such that each agent can decide individually.

In a static environment ACBBA is guaranteed to converge. The convergence problem shown in Figure 2 illustrates how we think about full network convergence in the ACBBA algorithm. The scenario in this figure is for 8 agents, with a maximum bundle size of 5, bidding on 35 tasks. As we see in the figure, there is a large spike within the first 10 ms of the tasks being released. This heavy message passing regime continues for about 0.1 s, then, after some clean up messages, settles down to 0 messages being sent. Once the network goes quiet in this configuration we say that the network has converged. For the rest of the results in this paper, we define 1 message being sent as an agent broadcasting some piece of information to every vehicle that can hear it, no matter how many vehicles hear this one message being broadcast, we count it as 1 message. Also, for reference, throughout this entire process 956 messages were sent.

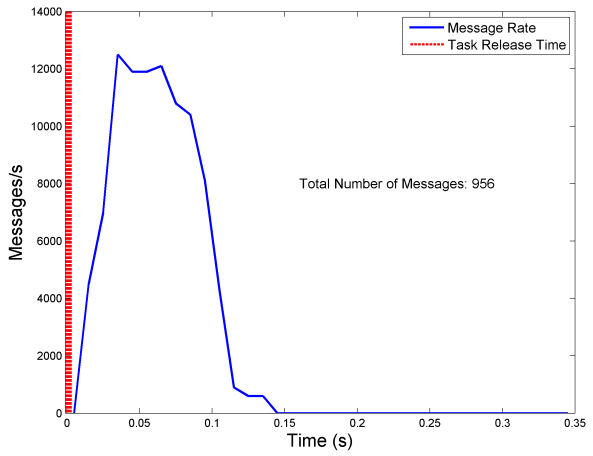


Figure 2. Message profile for a convergence iteration where all the tasks are released at the start of the program

The case shown in Figure 2 is interesting, but it ignores one of the big regimes in which ACBBA was designed to operate. In Figure 3 we show an example of what the network looks like under the presence of pop-up tasks. The parameters for this test are identical the above scenario, except, instead of releasing all 35 tasks at once, an initial allocation of 10 tasks starts off the ACBBA convergence, then every 0.2 seconds after, a single pop-up task is added. In this scenario we can see that the initial convergence takes a similar form to what was shown in Figure 2. At 0.2 seconds we see that the first pop-up task is released. The algorithm has a small spike of messages (about 50) for a little less that 0.1 seconds. These few messages are used to assign the pop-up task to an agent quickly, then the global message state returns to idle. Part a) of Figure 3 shows a close up of the convergence for the first second of the demonstration, while part b) shows the general trend of the steady state behavior. For reference, the number of messages sent in the timespan shown in a) of Figure 3 was 620.

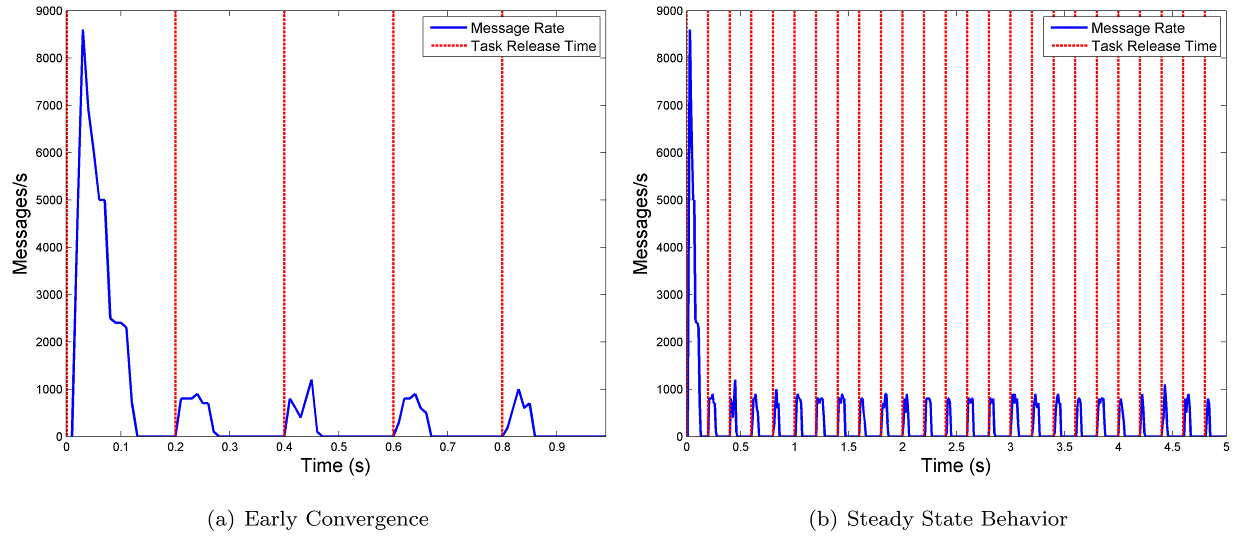


Figure 3. Message profile for a convergence sequence when 10 messages seed the environment and pop-up tasks are released every 0.2s after

Given the results in Figure 3, we recognize that there may be a problem with defining agent convergence in terms of the global convergence if tasks are arriving at a rate faster than the algorithm requires to converge to a 0-message state. This case is shown in Figure 4, and is identical to the one outlined in Figure 3, except that tasks come every 50ms instead of 0.2s. This means that on the global scale, the network may never converge. This is an unavoidable side effect of allowing the network to handle pop-up tasks. As can be seen in part a) of Figure 4, only briefly does the global network ever converge. This can be seen even more compellingly in part b) of this figure: there are never large gaps, where the network has become completely quiet.

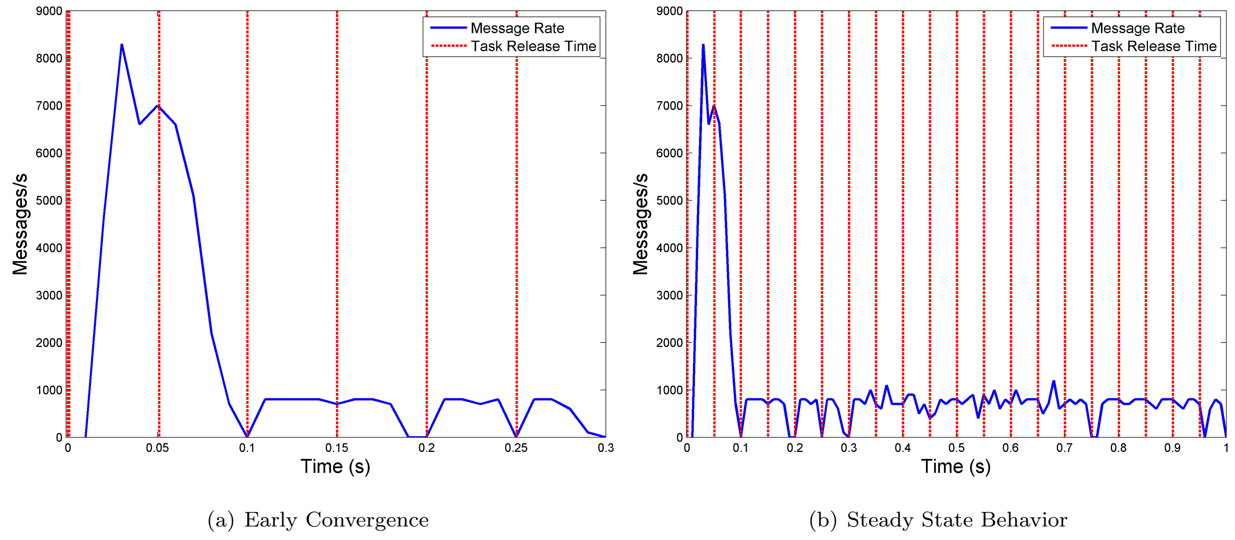


Figure 4. Message profile for a convergence sequence when 10 messages seed the environment and pop-up tasks are released every 0.05s after

This shows that in general, it does not make sense to define the idea of global convergence for ACBBA. Despite this we can say that given pop-up tasks, the network does not diverge. We can see the 10 tasks that are released at the beginning add a large number of messages to the network, but this spike is quickly handled and the message load stabilizes to the load of handling 1 new task at a time.

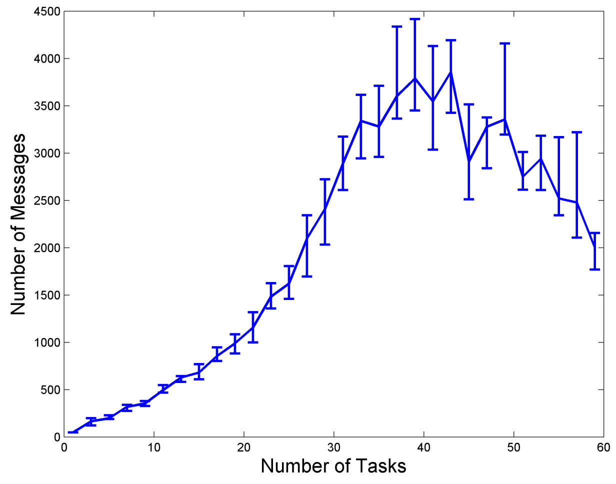
The above example highlights that in certain dynamic environments ACBBA convergence can only be thought of in terms of local convergence. This requires that each agent decide when their own bundles are conflict-free and ready for execution. Given the decentralized nature of ACBBA, the agents are free to make this decision independently of one another. In the example in Figure 4, the messages in the network during the steady-state phase are only due to a couple of agents deciding who wins the pop-up task. In general, 9 of the other 10 agents will not change their bundles based on the presence of this task, so when it is clear that they will not win it, they can decide that they have reached self convergence, even though there is no global convergence. In this way, certain agents can be in a converged state even though others are still actively engaging in a consensus phase.

The parameter that ACBBA uses to define the notion of local convergence is called separation time. This number refers to the maximum amount of time in between the arrival of any two instances of relevant information. By relevant information, we mean any information that affects the local agents' ACBBA state. This means that any time gap longer than a predefined upper bound that an agent hasn't received any relevant information, the agent can assume with high probability that it has reached local consensus. This separation time is computed in CBBA every time it checks to see if new information is available to update its bundle. A property of this parameter that can be seen in Figure 5 is that it is nearly independent of other relevant parameters. To explore this further we see a plot in part a) of Figure 5 of the number of messages versus number of tasks for a 10 agent simulation. Comparing part a) with part b), we can see that there is almost a direct mapping between convergence time and number of messages. However, for our convergence problem, the value for separation time is virtually independent to changes in number of messages, convergence time, or number of tasks in the network. In part c) of this figure we can see that it is also independent of the number of agents in the network. This value may not always be the same value given different communication constraints. However, since we observe it to be relatively independent of other parameters during CBBA operation, we can learn the value of this parameter and use it as a reliable measure of when each individual agent has converged.

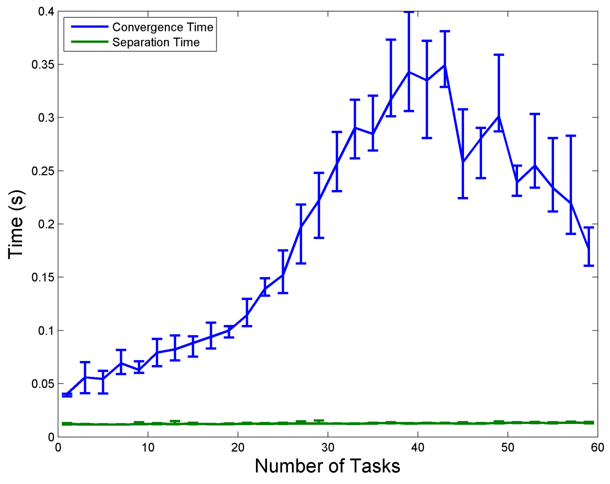
VI. Asynchronous Replan

Another key feature that is enabled through ACCBA is the ability to execute what we call an asynchronous replan. Due to the nature of ACBBA's bundle building process, tasks in an agents' bundle can only be dropped if another agent outbids him for a task, or if a replan is called. A desired feature for a decentralized algorithm is for an agent to be able to react quickly to large changes in personal situational awareness. With the introduction of an asynchronous replan, an individual agent can decide to drop its entire bundle and rebuild it from scratch, so that it can re-optimize for these new local changes in situational awareness.

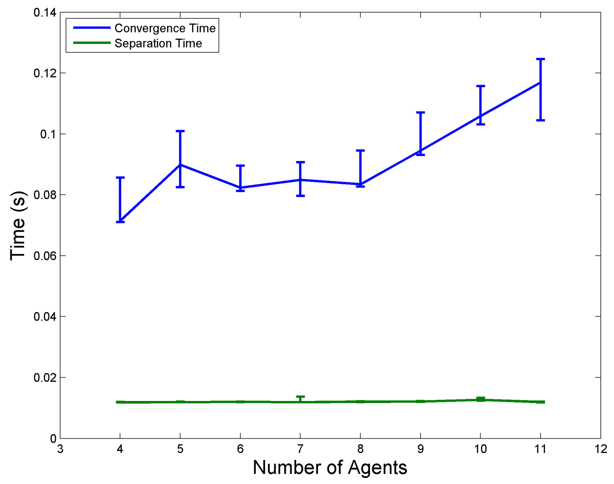
The notion of an asynchronous replan is especially attractive when the change to a decentralized agent's situational awareness will change that agent's bundle severely, but keep all the other agents' bundles unchanged. This is a typical scenario that might happen if a pop-up task is discovered near a remote agent. In some circumstances that agent will want to drop everything it is doing to service this task, but within the old CBBA framework, this would require a full global replan. In this situation, as well as many others in decentralized operations, events happen locally and only a few agents in a potentially large fleet care about. This asynchronous re-planning infrastructure gives these agents the flexibility to react dynamically, without tying the entire network down in a costly computational replan. These asynchronous replans utilize the same



(a) Number of messages



(b) Convergence time varying number of tasks



(c) Convergence time varying number of agents

Figure 5. Monte Carlo simulation results showing comparisons between number of messages, and convergence times

type of properties of ACBBA's convergence that normal convergence utilizes, such as the independence of separation time and the stability of the global network to consensus disturbances.

VII. Conclusion

In this paper and previous work by the authors [33] we have evolved CBBA from a distributed synchronous planning algorithm to a full decentralized asynchronous planning algorithm. A new implementation was developed for this paper that allows for greater insight into the consensus, as well as the message passing process inside of ACBBA. With this new implementation, our ability to track and analyze key metrics of the algorithm in real-time has vastly improved. The key algorithmic steps introduced in this paper were identifying a new metric of convergence for ACCBA, introducing the asynchronous replan capability and implementing and testing these new features of ACBBA in simulation and hardware.

Acknowledgments

This work was sponsored (in part) by the AFOSR and USAF under grant FA9550-08-1-0086 and by Boeing Research & Technology, Seattle, WA. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research or the U.S. Government.

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., and Rasmussen, S., "Task Allocation For Wide Area Search Munitions," *Proceedings of the American Control Conference*, 2002.

[6] Casal, A., *Reconfiguration Planning for Modular Self-Reconfigurable Robots*, Ph.D. thesis, Stanford University, Stanford, CA, 2002.

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

[8] Xu, L. and Ozguner, U., "Battle management for unmanned aerial vehicles," *IEEE Conference on Decision and Control*, 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.

[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," *IEEE Conference on Decision and Control*, 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," *IEEE Transactions on Automatic Control*, 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*, 2006.

[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," *43rd IEEE Conference on Decision and Control*, 2004.

[23] Wu, C. W., "Synchronization and Convergence of Linear Dynamics in Random Directed Networks," *IEEE Transactions on Automatic Control*, Vol. 51, No. 7, 2006, pp. 1207?–1210.

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

[25] Alighanbari, M. and How, J. P., "Decentralized Task Assignment for Unmanned Aerial Vehicles," *Proceedings of the 44th*

IEEE Conference on Decision and Control, and the European Control Conference, 2005.

[26] 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.

[27] 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.

[28] 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.

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

[30] 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.

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

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

[33] Johnson, L. B., Ponda, S., Choi, H.-L., and How, J. P., "Improving the Efficiency of a Decentralized Tasking Algorithm for UAV Teams with Asynchronous Communications," *AIAA Guidance, Navigation, and Control Conference (GNC)*, August 2010.

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

[35] 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.

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

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

[38] Ponda, S., Redding, J., Choi, H.-L., How, J. P., Vavrina, M. A., and Vian, J., "Decentralized Planning for Complex Missions with Dynamic Communication Constraints," *American Control Conference (ACC)*, July 2010.

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

[40] Choi, H.-L., Whitten, A. K., and How, J. P., "Decentralized Task Allocation for Heterogeneous Teams with Cooperation Constraints," *American Control Conference (ACC)*, July 2010, pp. 3057–3062.