2011 American Control Conference
on O'Farrell Street, San Francisco, CA, USA
June 29 - July 01, 2011

Decentralized Task Allocation with Coupled Constraints in Complex Missions

Andrew K. Whitten, Han-Lim Choi, Luke B. Johnson, and Jonathan P. How

Abstract— **This paper presents a decentralized algorithm that creates feasible assignments for a network of autonomous agents in the presence of coupled constraints. The coupled constraints address complex mission characteristics that include assignment relationships, where the value of a task is conditioned on whether or not another task has been assigned, and temporal relationships, where the value of a task is conditioned on when it is performed relative to other tasks. The new algorithm is developed as an extension to the Consensus-Based Bundle Algorithm (CBBA), introducing the notion of pessimistic or optimistic bidding strategies and the relative timing constraints between tasks. This extension, called Coupled-Constraint CBBA (CCBBA), is compared to the baseline in a complex mission simulation and is found to outperform the baseline, particularly for task-rich scenarios.**

I. Introduction

Decentralized task planning for a team of networked agents (e.g., unmanned aerial vehicles (UAVs)) has been investigated extensively [1]–[13]. Ideally, the communication link between all elements of the autonomous decision system (command station, autonomous vehicles, manned vehicles, etc.) is high bandwidth, low latency, low cost, and highly reliable. However, even the most modern communication infrastructures do not posses all of these characteristics. If the inter-agent communication mechanisms have 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. In addition, decentralized planning schemes are well-suited for situations where the information needed for decision making is local with respect to the network diameter. This is particularly noticeable in the task assignment consensus problem where agents near each other will require the most communication to resolve task assignment conflicts, whereas agents that are spatially separated are less likely to choose the same tasks. Decentralized algorithms with strong inter-agent communication should support this scenario more efficiently than centralized approaches.

One decentralized tasking approach the authors have recently developed is the Consensus-Based Bundle Algorithm (CBBA) [12], which is a market-based distributed agreement protocol upon the winning agents and associated winning scores. By enforcing agreement upon the solution rather than the information set, CBBA was shown to converge despite inconsistent information among the networked agents, guaranteeing a conflict-free assignment for all the agents in the network. CBBA is a polynomial-time algorithm that scales well with the size of the network and/or the number of tasks (or equivalently, the length of the planning horizon). From the design perspective, various design objectives, agent models, and constraints can be incorporated by defining appropriate scoring functions. If the resulting scoring scheme satisfies a property called *diminishing marginal gain* (DMG), a provably good feasible solution is guaranteed (50% minimum performance bound; over 90% in simulations). Furthermore, recent extensions to CBBA [13] have enabled incorporation of heterogeneity in agent capabilities and time windows of validity for tasks, which significantly enriches the mission characteristics that can be handled.

However, this recent progress is still not enough to address task planning for complex missions where tasks are coupled both spatially and temporally. The main source of such coupling is the set of rules of engagement, which can often be described as logical constraints and temporal precedence relationships. This paper particularly focuses on embedding these coupled constraints within the CBBA-based decentralized tasking framework. While a preliminary approach to achieve this functionality was presented in [14] for specific types of constraints described by cooperation requirements and preference, this paper provides a more systematic methodology that can handle a much broader class of coupled constraints, including mutual and unilateral task dependency constraints, mutual exclusions, and temporal constraints. Numerical experiments on cooperative sensing & neutralization missions for unmanned aerial vehicles (UAVs) verify the performance advantages of the proposed extension as compared to the baseline CBBA approach.

II. Task Allocation with Constraints

Consider a bounded environment containing a network of N_u autonomous agents, and a set of N_t tasks. The set of all agent indices is denoted $\mathcal{I} \triangleq \{1, \ldots, N_u\}$, while the set of all task indices is denoted $\mathcal{J} \triangleq \{1, \ldots, N_t\}$.

Each task, $j \in \mathcal{J}$, is given a score function, $S_j(a_j, t_j) \in \mathbb{R}_+$ which represents the value that task adds to the mission as a function of $a_j \in \mathcal{I} \cup \{\emptyset\}$, the index of the agent assigned to task j, and $t_j \in \mathbb{R}_{\geq 0} \cup \{\emptyset\}$ the time the agent plans to arrive at the j-th task location. The goal of the task planner is to assign agents to tasks in such a way that the cumulative score over all tasks in the task set is maximized. The task

A. K. Whitten, L.B. Johnson, and J. P. How was/are with the Dept. of Aeronautics and Astronautics, MIT, Cambridge, MA. {awhitten, lbj, jhow}@mit.edu.

H.-L. Choi is with the Div. of Aerospace Engineering, KAIST, Korea. hanlimc@kaist.ac.kr

978-1-4577-0079-8/11/$26.00 ©2011 AACC

assignment problem at a given mission time has an objective function expressed as

$$\max \sum_{j=1}^{N_t} S_j(a_j, t_j) \tag{1}$$

Note that in this formulation, there exists no requirement to assign an agent to every task, and in fact doing so may be infeasible. However, a task only earns a positive score if an agent is assigned to it, $S_j(a_j, t_j) > 0$ only if $a_j \neq \emptyset$. Therefore, the task planner must determine an appropriate subset of tasks to assign, and select the agent, and visit time for each assigned task. The assignment is thus completely determined by selecting a_j and t_j for each task $j \in \mathcal{J}$.

The problem is subject to two types of constraints: 1) agent-specific constraints, and 2) coupled constraints. Agent-specific constraints affect the options available to a given agent i independently of decisions made with respect to all other agents, and include:

1) *Capability constraints*: Each task has a requirement, and each agent has a set of capabilities. Compatibility matrix, $\mathcal{M}$ captures each agent's fitness for performing the task. $\mathcal{M}_{ij} > 0$ if agent i is allowed to perform task j, and zero otherwise.
2) *Time window of validity*: Each task j, has a time window of validity given by $[\tau_j^{\text{start}}, \tau_j^{\text{end}}]$. The task must be started on the interval of time specified by the time window. If a task is not time sensitive, it is given a time window of $[0, \infty)$.
3) *Vehicle dynamics*: Each agent has a set of dynamics that impose a set of constraints on that agent. Agent i has a maximum speed, $v_i^{\max}$, a maximum acceleration, $\dot{v}_i^{\max}$, and a minimum turn radius $r_i^{\min}$. These parameters determine the minimum amount of time required between each task the agent performs.
4) *Fuel constraints*: At a given time t, agent i has remaining fuel mass, $m_{\text{fuel}}^i(t)$, and has nominal fuel consumption rate, $\dot{m}_{\text{fuel}}^i$. These parameters determine the maximum remaining time aloft for agent i.

Coupled constraints include any situation where the decisions regarding one task or agent affect the options available regarding other tasks or agents. Often, in a complex mission, there are a variety of such constraints due to rules of engagement or simply the nature of the tasks involved. Coupled constraints include:

• *Assignment constraints* deal with the relationship between tasks with respect to which combination of assignments are permitted. Examples include:

1) *Conflict-freeness:* Each task may have at most one agent assigned to it. (Note that this is a simplifying assumption: mission activities requiring or benefiting from multiple agents may be represented by multiple instances of the same task.)
2) *Unilateral Dependency:* Task A is dependent on another task B, but task B is not dependent on A. In other words, task A cannot be assigned unless B is also assigned, but B may be assigned independently of task A.
3) *Mutual Dependency:* Task A is dependent on another task B, and task B is dependent on task A. Task A and task B must be assigned together or not at all.
4) *Mutually Exclusive:* Task A cannot be assigned if another task B is assigned, and task B cannot be assigned if task A is assigned.

• *Temporal constraints* include any specified relationship between the chosen visit times for a subset of tasks. Common temporal constraints include, but are not limited to:

1) *Simultaneous:* Task A and B must begin at the same time.
2) *Before:* Task A must end before task B begins.
3) *After:* Task A must begin after task B ends.
4) *During:* Task A must begin while task B is in progress.
5) *Not during:* Task A must either end before task B begins, or begin after task B ends.
6) *Between:* Task A must begin after task B ends and end before task C begins.

III. CBBA: The Baseline Tasking Algorithm

This section summarizes CBBA, which was first developed in [12] and recently extended in [13].

A. Phase I: Task Selection

In this first phase of CBBA each agent iteratively adds tasks to its bundle. The selection process is *sequentially greedy*; given an initial bundle, $\mathbf{b}_i$, agent i adds the task that will result in the largest marginal increase in score, and repeats the process until the bundle is full, or until no more tasks can be added. Agent i is only allowed to add a task if the computed marginal score is higher than the current highest bid for that task. This score is computed locally for each agent and depends on individual capabilities and constraints. Each time agent i adds a task j to its bundle, it enters its own index into the winning agent list, $z_{ij} \leftarrow i$, and its corresponding bid into the winning bid list, $y_{ij} \leftarrow S_j(i, t_{ij})$.

B. Phase II: Consensus

After all agents complete a round of the task selection phase, agents communicate with each other to resolve conflicting assignments within 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 (Section III-A) 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. For each message (consisting of bid information for a task) 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 winning agent list $\mathbf{z}_i$, its winning bid list $\mathbf{y}_i$, and the timestamp vector $\mathbf{s}_i$ to those of agent k to determine which agent's information is the most up-to-date for each task.

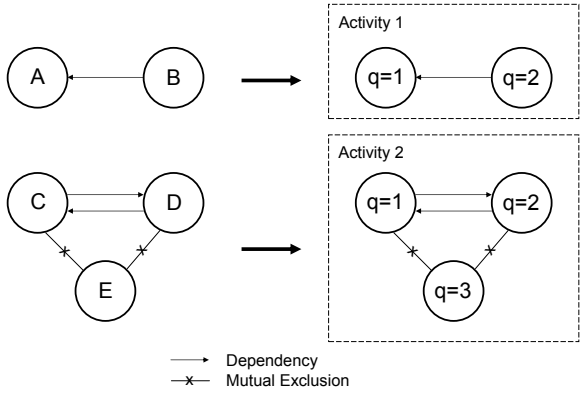

Fig. 1. Example of task set partitioning

C. Properties

The baseline CBBA is capable of handling non-coupled constraints including capability constraints, time windows, and basic vehicle dynamics such as maximum speed. It has also been proven that the baseline algorithm is capable of guaranteeing conflict-free assignments, assuming a strongly connected network, which is one type of coupled constraint.

However, it is not able to account for the other types of coupling which are common in complex missions. This paper describes the necessary machinery for enforcing the following types of coupled constraints within the CBBA framework: 1) unilateral dependency constraints, 2) mutual dependency constraints, 3) mutual exclusions, and 4) temporal constraints.

IV. Coupled-constraint CBBA (CCBBA)

A. Task Set Partitioning

For book-keeping purposes, the task set is partitioned into sub-groups of tasks that share coupled constraints. Each of these sub-groups is called an *activity*. For notational convenience, each task in an activity is referred to as an *element* of that activity, and is given an index, $q \in \mathbb{Z}_+$. Each task in the task set can be uniquely identified by its activity number and element index, and the notation k_q is used to indicate the task associated with the q^{th} element of activity k. The set of all tasks is still denoted $\mathcal{J}$. The set of all activities is denoted $\mathcal{A}$, while the set of all elements in an activity k is denoted $\mathcal{A}_k$. Each pair of elements in an activity may have a coupled constraint between them but is not required to. However, by construction, coupled constraints do not exist between tasks belonging to different activities with the exception of the conflict-freeness constraint.

Examples are shown in Fig. 1. Task B may not be assigned unless task A is assigned, but task A may be assigned independently of task B. Task C may not be assigned unless task D is assigned, and task D may not be assigned unless task C is assigned. The task pair (C, E) may not both be assigned, and the task pair (D, E) may not both be assigned. This particular task set can be grouped into two activities, with two and three elements respectively. The task set $\mathcal{J} = \{A, B, C, D, E\}$ is now written as $\mathcal{J} = \{1_1, 1_2, 2_1, 2_2, 2_3\}$.

For each activity $k \in \mathcal{A}$, the constraint structure can be compactly written in the form of a *Dependency Matrix*, $\mathcal{D}^k$ where the row column entry, (q, u), denoted as $\mathcal{D}^k_{qu}$, describes the relationship between the q^{th} element of activity k and the u^{th} element of activity k. The notation for encoding the constraints is provided in Table I. The diagonal entries of $\mathcal{D}^k$ are defined to be 0 for all activities.

TABLE I

Code for Dependency Matrix Entry $\mathcal{D}^k_{qu}$

1	u depends on q
0	u may be assigned independently of q
-1	q and u are mutually exclusive
$a > 1$	u requires either q or another element with the same code, a. Entries are used sequentially: 3 is not used unless 2 is used, etc.

B. Bidding Strategies in Phase I

Each task in the task set is given a *bidding strategy* which represents how the bidding protocol is handled; the bidding strategy for each task $k_q \in \mathcal{J}$ is determined from the dependency matrix as

$$\text{bidStrat}_{k_q} = \begin{cases} \text{optimistic} & \text{if } \{u : \mathcal{D}^k_{uq} \geq 1 \wedge \mathcal{D}^k_{qu} = 1\} \neq \emptyset \\ \text{pessimistic} & \text{otherwise} \end{cases}$$

The two types of bidding strategies, pessimistic and optimistic, are described below.

1) Pessimistic Bidding Strategy: In the baseline CBBA, at the beginning of the task selection phase, an agent i calculates the marginal score for each of the tasks in the task set that are not in its bundle. In CCBBA, a step is added prior to this where the agent determines which tasks it is allowed to bid on.

Task k_q is given a pessimistic bidding strategy (PBS) if it is only dependent upon tasks which are not also dependent on k_q. The PBS specifies that agent i is allowed to bid on task k_q only if all of the dependency constraints for task k_q are satisfied. An indicator $\text{canBid}_i(k_q)$ is used to keep track of this permission:

$$\text{canBid}_i(k_q) = \begin{cases} \text{true} & \text{if } n^{\text{sat}}(k_q|\mathbf{z}_i) = N_{\text{req}}(k_q) \\ \text{false} & \text{otherwise} \end{cases}$$

where $N_{\text{req}}(k_q)$ and $n^{\text{sat}}(k_q|\mathbf{z}_i)$ denote the total number of constraints required to be satisfied, and the number of satisfied constraints according to the current knowledge, respectively. For the case where $\mathcal{D}^k_{uq} \in \{-1, 0, 1\}, \forall u, q$, these quantities are given by

$$\begin{aligned} N_{\text{req}}(k_q) &= \textstyle\sum_{u=1}^{|\mathcal{A}_k|} \mathbb{I}\left(\mathcal{D}^k_{uq} = 1\right) \\ n^{\text{sat}}(k_q|\mathbf{z}_i) &= \textstyle\sum_{u=1}^{|\mathcal{A}_k|} \mathbb{I}\left(\left(z_{i(k_u)} \neq \emptyset\right) \wedge \left(\mathcal{D}^k_{uq} = 1\right)\right), \end{aligned}$$

while their expressions for more sophisticated cases can be found in [15, Section 3.3]. If $\text{canBid}_i(k_q)$ is true, then the marginal score for k_q is calculated to determine if this task should be added to agent i's bundle. If $\text{canBid}_i(k_q)$ is false, then the marginal score for k_q is set to 0.

2) Optimistic Bidding Strategy: Tasks which are mutually dependent on at least one other task are given an optimistic bidding strategy. Consider a task k_q with an optimistic bidding strategy. An agent who can benefit from adding task k_q to its bundle may do so, even if the number of satisfied constraints is smaller than the number required.

The agent then keeps track of the number of iterations that pass where at least one of the tasks which k_q depends on remains unassigned. If too many iterations pass, the agent releases task k_q from its bundle. The task is released with the assumption that the other agents in the network are not interested in selecting the tasks k_q depends on, because other tasks are more valuable to these agents. To prevent an agent from repeatedly bidding on a task only to release it once it discovers no other agents are interested in performing the tasks which it depends on, the number of attempts is limited. Once an agent has run out of attempts on a particular task, they are no longer permitted to bid on that task unless all of the required constraints are satisfied. Thus, the agent has transitioned to a pessimistic bidding strategy for this task. To formalize the technique, the following definitions are introduced:

a) Number of Iterations in Constraint Violation: $\boldsymbol{\nu}_i \triangleq \{\nu_{i1}, \ldots, \nu_{iN_t}\}$ is the list which keeps track of the number of CBBA iterations which have passed with agent i violating a constraint. Element notation $\nu_{i(k_q)}$ is used to indicate the number of iterations agent i has been winning element q of activity k while at least one other element in k is unassigned.

b) Permission to Bid Solo: $\mathbf{w}_i^{\text{solo}} \triangleq \{w_{i1}^{\text{solo}}, \ldots, w_{iN_t}^{\text{solo}}\}$ indicates which elements agent i is allowed to bid on as the first agent. The list is initialized to contain positive integers representing the number of attempts an agent is given to win the particular element. If $w_{i(k_q)}^{\text{solo}} > 0$, agent i is permitted to bid on task k_q even if no other elements of k have a winning agent. If a task k_q is released due to a timeout, or a timing constraint violation, $w_{i(k_q)}^{\text{solo}}$ is decremented by one.

c) Permission to Bid Any: $\mathbf{w}_i^{\text{any}} \triangleq \{w_{i1}^{\text{any}}, \ldots, w_{iN_t}^{\text{any}}\}$ indicates which tasks agent i is allowed to bid on given that at least one of the dependency constraints is satisfied for that task. The array is initialized to contain positive integers in a similar manner to $\mathbf{w}_i^{\text{solo}}$, except the initial values are typically larger in magnitude. If $w_{i(k_q)}^{\text{any}} > 0$, and any other element of k has a winner, agent i is permitted to bid on task k_q. If an element k_q is released due to a timeout, or a timing constraint violation, $w_{i(k_q)}^{\text{any}}$ is decremented by one.

d) Timeout Parameter: For each element q of an activity k, a *timeout* parameter, o_{k_q} is defined. At each iteration, agent i increments $\nu_{i(k_q)}$ for each task k_q for which agent i is the winner, but the other elements belonging to activity k are not filled. If at any time, $\nu_{i(k_q)}$ exceeds o_{k_q}, the task k_q must be released from $\mathbf{b}_i$, and the values $w_{i(k_q)}^{\text{solo}}$ and $w_{i(k_q)}^{\text{any}}$, are each decremented by 1.

Whether or not an agent i is allowed to bid on task k_q is determined by:

$$\begin{aligned}\text{canBid}_i(k_q) = &\left(w_{i(k_q)}^{\text{any}} > 0 \wedge n^{\text{sat}}(k_q|\mathbf{z}_i) > 0\right) \\ &\vee \left(w_{i(k_q)}^{\text{solo}} > 0\right) \vee \left(n^{\text{sat}}(k_q|\mathbf{z}_i) = N_{\text{req}}(k_q)\right).\end{aligned}$$

If $w_{i(k_q)}^{\text{solo}} > 0$, then the agent has not yet exhausted its attempts to acquire the proper number of partnering agents for task k_q, and so it is allowed to bid even if 0 dependency constraints are satisfied. If $w_{i(k_q)}^{\text{solo}} = 0$, but $w_{i(k_q)}^{\text{any}} > 0$, then the agent has not yet exhausted its attempts to acquire the proper number of partnering agents, but at least one dependency constraint must be met in order for the agent to bid on task k_q. If both $w_{i(k_q)}^{\text{solo}} = 0$ and $w_{i(k_q)}^{\text{any}} = 0$, then agent i has exhausted its attempts to bid optimistically on task k_q, and may only bid if all of the dependency constraints for task k_q are satisfied.

Note that appropriate selection of o_{k_q} and appropriate initialization of $\mathbf{w}_i^{\text{solo}}$ and $\mathbf{w}_i^{\text{any}}$ are needed to achieve good performance and convergence properties. Selecting o_{k_q} too small may lead to performance degradation, discouraging agents to bid on coupled tasks, while choosing it too large may lead to longer convergence times. Selecting a higher initial value for $w_{i(k_q)}^{\text{solo}}$ gives agent i more attempts to bid on task k_q, however may adversely affect the runtime of the algorithm. Selecting too small of a value may reduce performance due to conservatism in bidding on coupled tasks.

C. Mutual Exclusions in Phase I

In the baseline CBBA, an agent must be able to outbid the current winner of a task in order to be eligible to bid on that task. In Coupled-Constraint CBBA, to bid on task k_q, the marginal score for k_q must be greater than all of the tasks which are mutexed with task k_q. Therefore, task k_q is eligible to be bid on by agent i, if

$$\left(c_{i(k_q)} > y_{i(k_u)} \vee \mathcal{D}_{uq}^k \neq -1\right) = \texttt{true}, \ \forall u \in \mathcal{A}_k.$$

D. Temporal Constraints in Phase I

Often in complex missions, there exist timing constraints between some of the tasks in the task set. CCBBA allows a timing relationship to be specified between any pair of tasks belonging to the same activity. The pairwise timing constraints can be written compactly in the form of a *temporal constraint matrix*, $\mathcal{T}^k \in \mathbb{R}^{|\mathcal{A}_k| \times |\mathcal{A}_k|}$. The (q, u) entry of $\mathcal{T}^k$, denoted as $\mathcal{T}_{qu}^k$, specifies the maximum amount of time task k_q can begin after task k_u begins. Thus, task k_u must begin at least $\mathcal{T}_{qu}^k$ before task k_q begins and at most $\mathcal{T}_{uq}^k$ after task k_q begins, which can be interpreted as the *relative time window* imposed on task k_u by task k_q. There exists no sign restriction on the entries of the temporal constraint matrix, which means that the relative time window a task k_q imposes on another task k_u does not have to contain the start time for task k_q. This makes it possible to specify constraints like *before*, or *after* mentioned in Section II. If no timing constraint exists between task k_q and k_u, then $\mathcal{T}_{qu}^k = \mathcal{T}_{uq}^k = \infty$. The diagonal entries of matrix $\mathcal{T}^k$ are 0 by definition.

To satisfy coupled timing constraints, it is necessary that each agent be aware of the scheduled times of arrival of each of the tasks which have winners. Therefore, a new information list is introduced, $\zeta_i \triangleq \{\zeta_{i1}, \ldots, \zeta_{i(N_t)}\}$. Entries are denoted $\zeta_{i(k_q)}$ for $k_q \in \mathcal{J}$, and the list keeps track of the arrival times for each task in $\mathbf{z}_i$ that has a winner. $\zeta_{i(k_q)} = \emptyset$ if task k_q has no winner according to agent i.

Agent i must calculate the time interval that is valid for each task under consideration to determine if that task is feasible given their current path, and to determine the

marginal score for that task. Agent i considering task k_q calculates the permissible start time by intersecting the original time window for task k_q with each of the coupled timing constraints imposed by the agents winning each of the tasks upon which task k_q depends, according to its knowledge, i.e., $\mathbf{z}_i$ and ζ_i.

Agent i needs to compute the interval $[\tau_{k_q}^{\min}, \tau_{k_q}^{\max}]$ where $\tau_{k_q}^{\min}$ is the minimum allowed start time for task k_q given the current knowledge of agent i and $\tau_{k_q}^{\max}$ is the maximum allowed start time for task k_q given the current knowledge of agent i. Recall that the original time window is given by $[\tau_{k_q}^{\text{start}}, \tau_{k_q}^{\text{end}}]$.

The minimum admissible arrival time, $\tau_{k_q}^{\min}$ is calculated by

$$\tau_{k_q}^{\min} = \max\left(\tau_{k_q}^{\text{start}}, \max_{u\in\{1,\ldots,|\mathcal{A}_k|\}\setminus\{q\}} t_{\text{const}}^{\min}(k_u, k_q|\mathbf{z}_i),\right)$$

where

$$t_{\text{const}}^{\min}(k_u, k_q|\mathbf{z}_i) = \begin{cases} \zeta_{iu} - \mathcal{T}_{uq}^k & \text{if } z_{i(k_u)} \neq \emptyset \wedge \mathcal{D}_{uq}^k > 0 \\ -\infty & \text{otherwise.} \end{cases}$$

is the constraint imposed on the start time of task k_q by the agent winning task k_u. Notice that the constraint is only active if agent i believes there is an agent winning task k_u, and k_q is dependent on k_u. $\tau_{k_q}^{\min}$ is taken to be the tightest active constraint including the original time window. Similarly, $\tau_{k_q}^{\max}$ is given by

$$\tau_{k_q}^{\max} = \min\left(\tau_{k_q}^{\text{end}}, \min_{u\in\{1,\ldots,|\mathcal{A}_k|\}\setminus\{q\}} t_{\text{const}}^{\max}(k_u, k_q|\mathbf{z}_i)\right)$$

with

$$t_{\text{const}}^{\max}(k_u, k_q|\mathbf{z}_i) = \begin{cases} \zeta_{iu} + \mathcal{T}_{qu}^k & \text{if } z_{i(k_u)} \neq \emptyset \wedge \mathcal{D}_{uq}^k > 0 \\ \infty & \text{otherwise.} \end{cases}$$

E. Enforcing Constraints in Phase II

The consensus phase is augmented to enforce the additional coupled constraints: unilateral dependency constraints, mutual dependency constraints, mutual exclusions, and temporal constraints.

a) Tasks with Pessimistic Bidding Strategy: During the consensus process, an agent i may find that another agent in the network outbid it for some task k_q in its bundle. When this happens, k_q as well as all tasks after k_q in bundle $\mathbf{b}_i$ are released. The tasks which are released may be depended on by other agents to satisfy the constraints associated with the tasks in their bundles. It is therefore necessary for each agent i to verify at each iteration that the tasks in its bundle $\mathbf{b}_i$ which have a pessimistic bidding strategy have all their dependency constraints satisfied. If an agent finds that it is winning a task k_q which depends on some task which is no longer assigned, it must release task k_q as well as all tasks in its bundle after k_q.

b) Tasks with Optimistic Bidding Strategy: The protocol for optimistic bidding requires that each agent keep track of the number of iterations they have been winning a task with at least one violated dependency constraint. At each iteration, agent i counts the number of satisfied constraints for each task in its bundle, and compares it to the required number of satisfied constraints for that task. If it is less, $\nu_{i(k_q)}$ is incremented for each appropriate k_q.

Agent i also checks $\nu_{i(k_q)}$ for each task in its bundle. If $\nu_{i(k_q)} = o_{k_q}$ for any task, then k_q is released from the agent's bundle and $w_{i(k_q)}^{\text{solo}}$ and $w_{i(k_q)}^{\text{any}}$ are each decremented by 1. At this point, agent i has waited for o_{k_q} iterations of CBBA in hopes of acquiring all of the partnering agents that task k_q requires. The agent infers that if the constraints are not satisfied by this time, then the other agents in the network are either unable to select the tasks k_q depends on, or choose not to select them because other tasks result in a greater score. Task k_q is worth nothing to agent i unless all dependency constraints are met, and so it releases that task so it can select other tasks. Agent i also releases any task appearing in its bundle after k_q, but does not decrement $w_{i(k_q)}^{\text{solo}}$ or $w_{i(k_q)}^{\text{any}}$ for those tasks.

c) Enforcing Mutexes: An agent i is allowed to bid on a mutexed task k_q as long as it can place a bid larger than the current winning bid of all tasks that are mutexed with k_q, and larger than the winning bid for task k_q itself. The agent placing the bid on task k_q assumes that the other agents will release the tasks mutexed with task k_q once they are informed that they were "outbid." Because of this protocol, it is possible for an agent to discover that it is currently winning a task k_u which is mutexed with another task k_q which it also believes to be assigned. Therefore it is necessary for each agent to evaluate whether they are winning a task which is mutexed with another assigned task at every iteration. Agent i is required to release task k_q from its bundle if it discovers another task k_u where $\mathcal{D}_{uq}^k = -1$ and $y_{i(k_u)} \geq y_{i(k_q)}$; it is permitted to keep task k_q at a given iteration only if

$$\left(y_{i(k_q)} > y_{i(k_u)} \vee \mathcal{D}_{uq}^k \neq -1\right) = \texttt{true},\ \forall u \in \mathcal{A}_k \setminus \{q\}.$$

d) Temporal Constraints: Temporal constraints may become violated during the task assignment process for a variety of reasons. Note that the task selection phase of CBBA is performed by agent i independently of all other agents, except for the information agent i posses in $\mathbf{z}_i$ and $\mathbf{y}_i$. Therefore, it is possible for several agents to bid on elements of an activity in the same bidding round, possibly resulting in conflicting arrival times. Additionally, when an agent selects the start time for a task k_q, it is only required to consider the constraints imposed by the tasks which k_q depends on. There may exist an agent k who is currently winning another task k_u which unilaterally depends on k_q. Now the start time chosen by agent i for task k_q may invalidate the start time already selected for task k_u by agent k.

Therefore, it is necessary for each agent to check the temporal constraints for all tasks in their bundle at each iteration. The method developed for enforcing timing constraints is described in this section for two cases: 1) temporal conflicts between tasks with unilateral dependency constraints, and 2) temporal conflicts between tasks with mutual dependency constraints. The process is as follows:

At each iteration, agent i checks each task $k_q \in \mathbf{b}_i$ for a

temporal constraint violation.

$$\text{tempSat}(k_q) = \bigwedge_{u \in S_{k_q}} \left(\zeta_{i(k_q)} \leq \zeta_{i(k_u)} + \mathcal{T}^k_{qu}\right) \wedge \left(\zeta_{i(k_u)} \leq \zeta_{i(k_q)} + \mathcal{T}^k_{uq}\right)$$

where $S_{k_q} = \{u \in \{1, \ldots, |\mathcal{A}_k|\} : z_{i(k_u)} \neq \emptyset\}$. If $\text{tempSat}(k_q) = \texttt{true}$, agent i keeps task k_q as it is. However, if a task k_u is found whose start time is in violation with the start time for k_q, then agent i *may* have to release task k_q.

If task k_q unilaterally depends on k_u, then agent i releases k_q, because it is violating a constraint imposed by a task assumed to be higher priority. The agent may re-bid on task k_q in the next iteration, as long as it can select a start time that does satisfy the constraints. If task k_u unilaterally depends on k_q, then agent i keeps task k_q in its bundle. The agent assumes that the agent winning task k_u will also realize that k_q and k_u are conflicted, and release task k_u.

If task k_q and k_u are mutually dependent, then a special procedure is followed: Assume that the score for a given task is monotonically decreasing within its time window. Then the agent arriving earliest (with respect to the start time of the task) is required to release that task. In this instance, it is assumed that the agent arriving later in their task's time window would have chosen an earlier start time if it were possible, since the score decays with time. Given that agent i is winning task k_q and agent k is winning task k_u, and the start times are conflicted, then agent i releases task k_q if

$$\zeta_{i(k_q)} - \tau^{\text{start}}_{k_q} \leq \zeta_{i(k_u)} - \tau^{\text{start}}_{k_u}$$

Now each of the cases for the (k_q, k_u) task pair have been accounted for when there is a temporal constraint violation between them. If an agent is required to release a task k_q because of a temporal constraint violation, and task k_q has an optimistic bidding strategy, then $w^{\text{solo}}_{i(k_q)}$ and $w^{\text{any}}_{i(k_q)}$ are each decremented by one.

V. Numerical Results

A. Simulation Setup

This section describes a simulation designed to compare the performance of Coupled-Constraint CBBA against the baseline CBBA. The simulated mission scenario involves a cooperative search, track, and engage mission. Participating in the mission, are three types of agents. Agents of type A are Neutralizing UAVs (NUAVs) which are capable of engaging hostile targets. Agents of type B are Sensor UAVs (SUAVs), and they are capable of providing the NUAVs with accurate measurements during an engagement. Finally, agents of type C are SUAVs, but are only capable of image capture; they cannot provide measurements to a NUAV. Agent capabilities are described in Table II.

There exist several objects of interest in the mission environment. Some of the objects are confirmed hostiles, and it is desired that all hostiles be engaged. Some of the objects are of unconfirmed identity, and more intelligence is desired regarding these objects. The two different types of objects each have a specific activity associated with them.

TABLE II
Agent Parameters for Simulated Mission Scenario

Agent Parameters		
Type A: NUAV	Units Available Capability Max Velocity	3 Neutralization 50 m/s
Type B: SUAV 1	Units Available Capability Max Velocity	5 Sensing for NUAV 25 m/s
Type C: SUAV 2	Units Available Capability Max Velocity	8 Sensing- Image only 15 m/s

Activity type 1 is a *service hostile* activity: hostile targets may be engaged, which requires one agent of type A to strike the target and one agent of type B to provide position measurements. The strike portion of the activity takes 120 seconds, and the NUAV and the SUAV must plan to arrive within 20 seconds of each other. If one type A and one type B agent are assigned to a service hostile activity, a type C agent may also be assigned to perform damage assessment (DA). The DA task takes 180 seconds, and must begin at least 60 seconds after the strike is completed. If there is no strike package available to engage the hostile target, a SUAV may be sent to observe the hostile, but it is assumed that the reward for performing the intelligence gathering task is much less than that for performing the strike task. The maximum score for a strike task is 100, the maximum score for a DA task is 50, and the maximum score for an intelligence gathering task is 10. The time window for the activity is 600 seconds.

Activity type 2 is a tracking activity and involves visiting one of the objects with unknown identity. This type of activity contains a single element which is an information gathering task that can be performed by either a type B or a type C agent. The duration is 60 seconds, and the maximum activity score is 10.

The dependency and temporal constraint matrices for activities of type 1 and 2 are

$$\mathcal{D}_1 = \begin{bmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & 1 & -1 \\ 0 & 0 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix}, \quad \mathcal{T}_1 = \begin{bmatrix} 0 & 20 & -180 & \infty \\ 20 & 0 & -180 & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 0 \end{bmatrix}$$

$$\mathcal{D}_2 = \begin{bmatrix} 0 \end{bmatrix}, \qquad \mathcal{T}_2 = \begin{bmatrix} 0 \end{bmatrix}$$

The simulation environment is 10 $\times$ 10 km. Activity locations and agent initial positions are randomly generated with uniform distribution over the environment. The number of activities is varied as a parameter, with half of the activities being of type 1, and half of type 2. The time windows for each activity are chosen such that the start time is uniformly random on [0, 300] seconds. A maximum bundle length of $L_t = 4$ is used.

The scoring for a given assignment is calculated such that only tasks which are assigned without constraint violation count toward the assignment score. Also, the baseline CBBA has no way to account for timing constraints, so the only way to enforce them is to shrink the time window for the two elements of the neutralization task to be 20 seconds long, and set the time window of the DA to begin after the

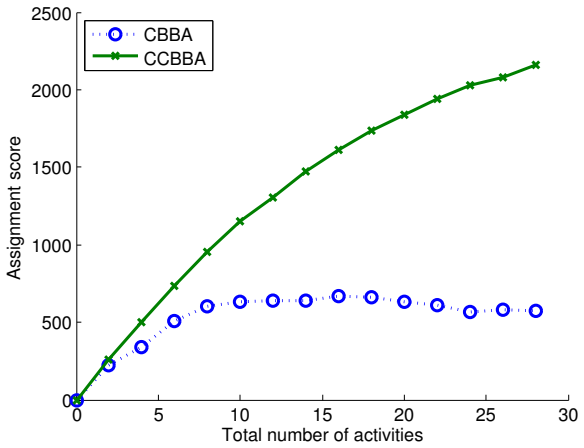

Fig. 2. Assignment Score for CCBBA vs. Baseline CBBA

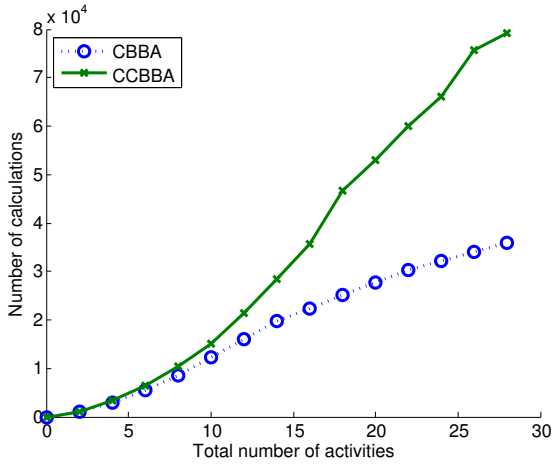

Fig. 4. Computation Complexity of CCBBA vs. Baseline CBBA

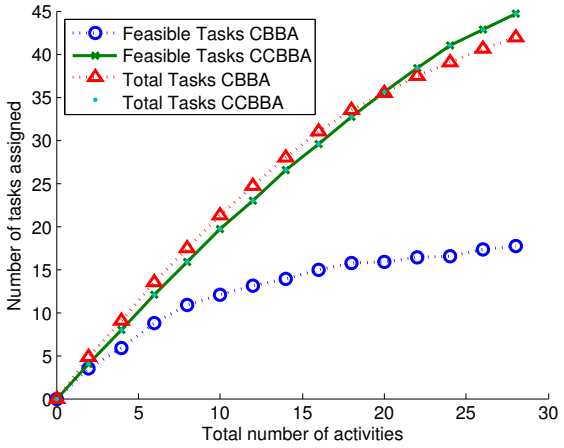

Fig. 3. Number of Tasks Assigned for CCBBA vs. Baseline CBBA

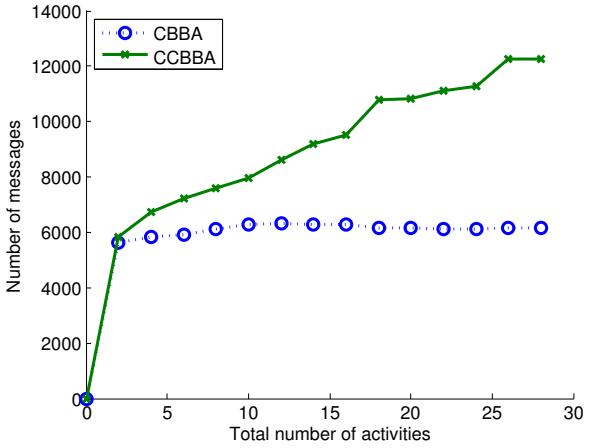

Fig. 5. Communication Complexity of CCBBA vs. Baseline CBBA

neutralization window ends.

B. Results

In this particular mission scenario, there are 16 agents and the number of activities is varied between 0 and 30 (0 to 75 tasks with coupled constraints.) The reported data is averaged over 80 trials for each case. Fig. 2 compares the average assignment scores of the baseline CBBA and CCBBA. Notice that the score for CBBA flattens out as the number of activities increases beyond 10. The reason for this behavior is that as the ratio of tasks to agents grows, the chance that all of the tasks in a single activity are assigned decreases, resulting in fewer feasible assignments for CBBA. However, notice that the score for CCBBA-generated assignments increases with the number of tasks, and does not posses the tendency to flatten out. This is because CCBBA is explicitly enforcing the coupled constraints so that agents select tasks that will actually be executable due to all of their constraints being met.

Fig. 3 further validates the advantage of using CCBBA for large task to agent ratios. The figure compares the total number of tasks assigned by each algorithm to the number of tasks that can actually be executed due to imposed coupled-constraints. Notice that the total number of tasks assigned (before considering constraints) is roughly the same for each approach. However, for the baseline CBBA, the number of tasks with unmet constraints grows as the number of activities increases, decreasing the amount of feasible tasks. Note that for CCBBA all assigned tasks satisfy the constraints, so the number of total tasks assigned is identical to the number of feasible tasks.

Fig. 4 shows the increase in computation for CCBBA compared to CBBA. The metric for comparing computation is the number of score calculations required to arrive at an assignment, summed over the whole network. We can see that although CCBBA requires increased computation, the algorithm remains computationally tractable for non-trivial problems.

Fig. 5 compares the communication requirements for the two algorithms. The metric for comparing communication is the total number of messages for the entire network parsed during the assignment process. This figure shows that for this choice of mission parameters, CBBA has approximately constant communication requirements with respect to the number of tasks, whereas CCBBA has approximately linear communication requirements. The additional complexity is associated with the optimistic bidding process, because agents must put forth additional effort to discover that a task will not have the required number of satisfied constraints.

One of CBBA's most valuable attributes is its polynomial computation and communication requirements. Although CCBBA is more expensive compared to CBBA in

terms of computation and communication, it still maintains polynomial-time convergence, providing a tractable scalable framework for solving task assignment problems for complex missions involving large teams and numerous tasks.

VI. Conclusions

Coupled-constraint CBBA (CCBBA) was developed as an efficient mechanism for solving the decentralized constrained task allocation problem. These algorithmic extensions to CBBA expand the types of constraints that CBBA can assign to include unilateral dependency constraints, mutual dependency constraints, mutual exclusion, and temporal constraints. Numerical examples validate that CCBBA enables incorporation of a wide range of mission constraints, providing measurable performance improvement to the baseline CBBA. Future work will include integration of the presented CCBBA into the Asynchronous CBBA framework [16] that has been developed as the most efficient implementation of CBBA in terms of computation and communication.

Acknowledgments

This work was sponsored (in part) by the AFOSR and USAF under grant (FA9550-08-1-0086) and MURI (FA9550-08-1-0356). 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. The second author gratefully acknowledges financial support by Korea Agency for Defense Development Grant ADDR-401-101761.

References

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

[2] C. Schumacher, P. Chandler, and S. Rasmussen, "Task allocation for wide area search munitions," in *Proceedings of the American Control Conference*, 2002.

[3] A. Casal, "Reconfiguration planning for modular self-reconfigurable robots," Ph.D. dissertation, Stanford University, Stanford, CA, 2002.

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

[5] D. Turra, L. Pollini, and M. Innocenti, "Fast unmanned vehicles task allocation with moving targets," in *Proceedings of the IEEE Conference on Decision and Control*, Dec 2004.

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

[7] T. W. McLain and R. W. Beard, "Coordination variables, coordination functions, and cooperative-timing missions," *Journal of Guidance, Control, and Dynamics*, vol. 28(1), pp. 150–161, 2005.

[8] D. A. Castanon and C. Wu, "Distributed algorithms for dynamic reassignment," *IEEE Conference on Decision and Control*, vol. 1, pp. 13–18 Vol.1, 9-12 Dec. 2003. [Online]. Available: 10.1109/CDC.2003.1272528

[9] J. Curtis and R. Murphey, "Simultaneous area search and task assignment for a team of cooperative agents," in *Proceedings of AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2003.

[10] T. Shima, S. J. Rasmussen, and P. Chandler, "Uav team decision and control using efficient collaborative estimation," in *Proceedings of the American Control Conference*, 8-10 June 2005, pp. 4107–4112 vol. 6. [Online]. Available: 10.1109/ACC.2005.1470621

[11] M. Alighanbari and J. How, "Robust and Decentralized Task Assignment Algorithms for UAVs," Ph.D. dissertation, MIT, 2007.

[12] H.-L. Choi, L. Brunet, and J. P. How, "Consensus-based decentralized auctions for robust task allocation," *IEEE Trans. on Robotics*, vol. 25 (4), pp. 912 – 926, 2009.

[13] S. Ponda, J. Redding, H.-L. Choi, J. P. How, M. A. Vavrina, and J. Vian, "Decentralized planning for complex missions with dynamic communication constraints," in *American Control Conference (ACC)*, July 2010.

[14] H.-L. Choi, A. K. Whitten, and J. P. How, "Decentralized task allocation for heterogeneous teams with cooperation constraints," in *American Control Conference (ACC)*, July 2010, pp. 3057–3062.

[15] A. K. Whitten, "Decentralized planning for autonomous agents cooperating in complex missions," Master's thesis, MIT, Department of Aeronautics and Astronautics, Aug. 2010.

[16] L. B. Johnson, S. Ponda, H.-L. Choi, and J. P. How, "Improving the efficiency of a decentralized tasking algorithm for UAV teams with asynchronous communications," in *AIAA Guidance, Navigation, and Control Conference (GNC)*, August 2010.

Appendix

A. Examples of Encoding Real-World Constraints

Encoding constraints into the dependency and temporal constraint matrices may not be intuitive. This section provides a few different examples of mission scenarios, and describes the corresponding constraint matrices. Further details and examples can be found in [15, Section 3.9].

a) Required Cooperation: Many real-world scenarios involve executing a set of tasks, with simultaneous arrival times, where all of them must be assigned for any of them to be valid. The dependency matrix for a set of mutually required tasks of arbitrary size is given by $\mathcal{D} = \mathbf{1}_{N_t \times N_t} - I_{N_t}$, and the corresponding temporal constraint matrix for this case is $\mathcal{T} = \mathbf{0}_{N_t \times N_t}$.

b) Not During: Consider a scientific operation where a series of measurements are taken by a fleet of robots. Suppose there are two measurements that are to be taken: task A and task B. Both measurements must be taken for either of them to be valid, however, they cannot be taken simultaneously, due to equipment interference. The constraints can be encoded by splitting one of the tasks into two instances - consider B into B^- and B^+ for example. Then, task B^- depends on task A, task B^+ depends on task A, and A depends on either B^- or B^+, which give

$$\mathcal{D} = \begin{bmatrix} 0 & 1 & 1 \\ 2 & 0 & -1 \\ 2 & -1 & 0 \end{bmatrix}, \mathcal{T} = \begin{bmatrix} 0 & \infty & -d_A \\ -d_B & 0 & \infty \\ \infty & \infty & 0 \end{bmatrix}.$$

c) Super-Additive Score Structure: Sometimes, it is desirable to send two UAVs to track a target if possible, severely reducing the probability of evasion. This situation often leads to a super-additive score structure for which the combined score is greater than the sum of the individuals. To encode this score structure, the combined assignment is represented by two tasks, each worth half of the combined score. A third task representing a single assignment is introduced; this task must be mutexed with each of the two tasks representing the combined assignment, which gives constraint matrices

$$\mathcal{D} = \begin{bmatrix} 0 & 1 & -1 \\ 1 & 0 & -1 \\ -1 & -1 & 0 \end{bmatrix}, \mathcal{T} = \begin{bmatrix} 0 & 0 & \infty \\ 0 & 0 & \infty \\ \infty & \infty & 0 \end{bmatrix}.$$