Distributed Algorithm for Solving the Bottleneck Assignment Problem

Mitchell Khoo, Tony A. Wood, Chris Manzie, Iman Shames.

Abstract— **Assignment problems are found in multiagent systems, where there is a need to allocate multiple tasks to agents. The bottleneck assignment problem (BAP) is an assignment problem where the objective is to minimise the worst individual cost in the assignment. Distributed algorithms for assignments with other objectives have been proposed, yet to date no distributed algorithm for the BAP exists. This paper addresses the gap; we develop a novel distributed algorithm that solves the BAP optimally. The algorithm does not require a centralised decision-maker having access to all information from each agent, which is an advantage over existing algorithms for solving the BAP. We use numerical simulations to compare the optimality of our algorithm against a greedy assignment-finding algorithm.**

I. INTRODUCTION

The bottleneck assignment problem (BAP) arises when a set of tasks must be allocated to a set of agents such that the most costly allocation is minimised. An example application of the BAP is the threat seduction problem in [1], where there are multiple incoming threats and a set of decoys. Each threat is to be jammed by one decoy. The worst-case time for an agent to position itself to jam a threat must be minimised.

Centralised algorithms to solve the BAP exist in literature, e.g., [1]–[5]. These algorithms are centralised in that they require a central decision-maker to have access to the costs of allocation of every task to every agent. A distributed algorithm removes the need for a central decision-maker to carry out all computations and aggregate all the necessary information from all agents. Currently, there does not exist a distributed algorithm for solving the BAP in literature.

The difficulty in developing a distributed algorithm that solves the BAP is due to the fact that existing centralised algorithms are not immediately amenable to being translated into distributed algorithms. The threshold algorithm in [2] involves a central decision-maker calculating an initial threshold cost and checking if an assignment can be created using only allocations with costs smaller than the threshold. The threshold is iteratively increased until such an assignment is found, and the first such assignment corresponds to the optimal assignment for the BAP. The threshold algorithm requires the central decision-maker to form a matrix with elements being the costs of every allocation, with each row corresponding to an agent and each column corresponding to a task. In [3], [4], improvements on the completion time of the threshold algorithm are made by sorting the costs and moving the threshold according to a binary search pattern rather than incrementally. In [5], an algorithm solves the BAP over a subset of the total agents and tasks, and iteratively increasing the size of the subset until it contains all the agents and tasks. The solution to the BAP over a given subset is found using the solution to the BAP over the previous subset.

All authors are with the Department of Electrical and Electronic Engineering at the University of Melbourne. Email: khoom1@student.unimelb.edu.au, {wood.t, manziec, iman.shames}@unimelb.edu.au

The concept of augmenting paths, defined in Definition 7 below, plays a crucial role in the BAP solution approaches given in [2]–[5]. Finding an augmenting path with distributed agents poses a challenge as doing so requires information from multiple agents. The algorithm developed in this paper also makes use of augmenting paths; finding an augmenting path in a bipartite graph in a distributed manner is one of the main contributions of this paper.

There are many distributed algorithms for assigning tasks to agents with other objectives, e.g., the linear assignment problem (LAP). In the LAP, tasks are allocated to agents such that the sum of the costs of the allocations is minimised. The Hungarian Method in [6] is a well-studied algorithm that solves the LAP in a centralised manner. In [7], a distributed method of executing the Hungarian Method is developed. Greedy algorithms have been used for finding a suboptimal solution to assignment problems. A greedy algorithm sequentially picks one allocation of a task to an agent with lowest cost from the remaining choices of allocations. Distributed versions of such greedy algorithms have been proposed in the past, e.g., the Consensus-Based Auction Algorithm (CBAA) in [8]. Some recent works consider distributed algorithms for assigning multiple tasks per agent, where each agent completes its allocated tasks sequentially; such problems are called time extended (TA) problems in [9]. In [8] the Consensus-Based Bundle Algorithm (CBBA) is introduced, which extends CBAA to TA problems. In [10], agents with limited fuel are considered and the objective is to maximise the number of assigned tasks. In [11], a TA problem with time varying costs is considered. In our paper, we consider problems where each agent carries out at most one task.

The main contribution of this paper is the development of a distributed algorithm for solving the BAP exactly, in contrast to the suboptimal solutions obtained from applying a greedy algorithm. We demonstrate this in a numerical case study where we analyse how the optimality gap of the greedy algorithms grows as the size of the problem grows.

II. BACKGROUND

Define a set of agents $\mathcal{D} := \{a_1, a_2, ..., a_m\}$ and a set of tasks $\mathcal{T} := \{b_1, b_2 ..., b_n\}$, with $m, n \in \mathbb{N}$. Assume that $m \geq n$. Represent all possible assignments of tasks to agents as a graph, i.e., let $\mathcal{G}_A = (\mathcal{V}_A, \mathcal{E}_A)$ be a complete weighted bipartite graph with vertex set $\mathcal{V}_A = \mathcal{D} \cup \mathcal{T}$ and edge set

$\mathcal{E}_A = \mathcal{D} \times \mathcal{T}$. Each edge $(i,j) \in \mathcal{E}_A$ is associated with a weight τ_{ij}, which corresponds to the cost for agent i to carry out task j; define the set $\tau = \{\tau_{ij} | i \in \mathcal{D}, j \in \mathcal{T}\}$. Definitions 1-8 hold given any arbitrary undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$.

Definition 1. *(Matching, [12]) A matching $\mathcal{M}$ in $\mathcal{G}$ is a set of edges such that $\mathcal{M} \subseteq \mathcal{E}$ and no vertex $v \in \mathcal{V}$ is incident with more than one edge in $\mathcal{M}$.*

Definition 2. *(Maximum Cardinality Matching, [12]) A maximum cardinality matching (MCM) is a matching $\mathcal{M}_{max}$ in $\mathcal{G}$ of maximum cardinality.*

There may be more than one MCM for a given graph; an MCM is not necessarily unique. Let $\mathcal{C}$ be the set of all MCMs of $\mathcal{G}_A$. The BAP is formulated as

$$\min_{\mathcal{M} \in \mathcal{C}} \max_{(i,j) \in \mathcal{M}} \tau_{ij}. \tag{1}$$

Since $m \geq n$, the cardinality of any $\mathcal{M} \in \mathcal{C}$ must be n. Next, we provide some graph theoretic definitions.

Definition 3. *(Neighbours) The set of neighbours of vertex $v \in \mathcal{V}$ in $\mathcal{G}$ is defined as $N_v = \{k | (v,k) \in \mathcal{E}\}$.*

Note that from Definition 3, given a vertex $v \in \mathcal{V}$, $\forall k \in N_v$, $v \in N_k$ since $\mathcal{G}$ is undirected.

Definition 4. *(Path, [12]) Let a sequence of distinct vertices $v_1, v_2, ..., v_l \in \mathcal{V}$ be such that for $k = 1, 2, ..., l-1$, v_k and v_{k+1} are neighbours in $\mathcal{G}$. The set of edges $\mathcal{P} = \{(v_k, v_{k+1})\}_{k=1,2,...,l-1}$ is then said to be a path between v_1 and v_l, with length $l-1$.*

Definition 5. *(Alternating path) Given a matching $\mathcal{M}$ and a path $\mathcal{P}$, $\mathcal{P}$ is an alternating path relative to $\mathcal{M}$ iff each vertex that is incident to an edge in $\mathcal{P}$ is incident with no more than one edge in $\mathcal{P} \cap \mathcal{M}$ and no more than one edge in $\mathcal{P} \backslash \mathcal{M}$.*

The definition of an alternating path is used to define an alternating tree in Section III.D. It is also used to define an augmenting path in Definition 7 below.

Definition 6. *(Free vertex, [12]) Given a matching $\mathcal{M}$, a vertex $v \in \mathcal{V}$ is free iff for all $w \in \mathcal{V}$, $(v,w) \notin \mathcal{M}$.*

Definition 7. *(Augmenting path, [12]) Given a matching $\mathcal{M}$ and a path $\mathcal{P}$ between vertices v_1 and v_l, $\mathcal{P}$ an augmenting path relative to $\mathcal{M}$ iff $\mathcal{P}$ is an alternating path relative to $\mathcal{M}$ and v_1 and v_l are both free vertices.*

The communication between agents is represented by a communication graph, as described in Assumption 4 below. The diameter of this graph bounds the amount of time a message from one agent takes to reach another agent.

Definition 8. *(Diameter) Let l_{ij} be the length of the shortest path between vertices $i, j \in \mathcal{V}$ in the graph. Define the diameter D of graph $\mathcal{G}$ to be $D = \max\limits_{i,j \in \mathcal{V}} l_{ij}$.*

The following assumptions model a distributed setting, where there are restrictions on the information available to each individual agent.

Assumption 1. *Assume an agent $i \in \mathcal{D}$ has access to the vector of costs $C_i = [\tau_{ib_1}, \tau_{ib_2}, ..., \tau_{ib_n}]^\top$. The entire set of costs τ is unknown to any single agent.*

Assumption 2. *Assume an agent $i \in \mathcal{D}$ has access to the set of edges $\mathcal{A}_i = \{(i,j) | (i,j) \in \mathcal{E}_A\}$. The entire set $\mathcal{E}_A$ is unknown to any single agent.*

Note, $\mathcal{E}_A = \bigcup\limits_{i=1}^{m} \mathcal{A}_i$ and $\mathcal{A}_s \cap \mathcal{A}_t = \emptyset$ for $s, t \in \mathcal{D}$, $s \neq t$.

Assumption 3. *Let $\mathcal{M}$ be an MCM of $\mathcal{G}_A$. For each agent $i \in \mathcal{D}$, let m_i be such that $(i, m_i) \in \mathcal{M}$. Assume agent i has access to m_i. The entire matching $\mathcal{M}$ is unknown to any single agent.*

Assumption 4. *Let communication between agents be modelled by a time invariant, undirected and connected graph $\mathcal{G}_C = (\mathcal{D}, \mathcal{E}_C)$ with vertex set $\mathcal{D}$, edge set $\mathcal{E}_C$ and diameter D. Agents can communicate with all their neighbours in $\mathcal{G}_C$. Assume agents communicate synchronously*[1].

We require a distributed algorithm to solve the BAP and we formally state this problem as follows.

Problem 1. *Under Assumptions 1-4, obtain an assignment of tasks to agents that corresponds to the minimiser of the problem given in (1).*

III. SOLVING THE BAP

We first present Algorithm 1 to solve the BAP, then discuss how each subroutine of Algorithm 1 can be carried out in the distributed setting.

A. Algorithm for Solving the BAP

Let $\mathcal{C}$ be the set of all MCMs of $\mathcal{G}_A$. To initialise Algorithm 1, we require an arbitrary MCM $\mathcal{M}_0 \in \mathcal{C}$ of $\mathcal{G}_A$; any MCM in $\mathcal{C}$ is applicable. One example would be to set $\mathcal{M}_0 = \{(a_p, b_q) | a_p \in \mathcal{D}, b_q \in \mathcal{T}, p = q\}$.

Lemmas 1 and 2 are used in the proof of Lemma 3, shown in Appendix A. The notation $A \oplus B$ denotes the symmetric difference of the sets A and B.

Lemma 1. *Given a graph $\mathcal{G}$, if $\mathcal{M}$ is a matching of $\mathcal{G}$ and $\mathcal{P}$ is an augmenting path relative to $\mathcal{M}$, then $\mathcal{M} \oplus \mathcal{P}$ is also a matching of $\mathcal{G}$ and $|\mathcal{M} \oplus \mathcal{P}| = |\mathcal{M}| + 1$.*

Proof: See [12, Lemma 1]. ■

Lemma 2. *Given a graph $\mathcal{G}$, a matching $\mathcal{M}$ is an MCM of $\mathcal{G}$ iff there is no augmenting path relative to $\mathcal{M}$ in $\mathcal{G}$.*

Proof: See [12, Corollary 1 (Berge's Theorem)]. ■

Assumption 5. *Given edge set $\bar{\mathcal{E}}$ and the costs of each edge τ, assume that the function* MaxEdge$(\bar{\mathcal{E}}, \tau)$ *returns the edge in $\bar{\mathcal{E}}$ with largest weight.*

Assumption 6. *Given a matching $\bar{\mathcal{M}}$, a graph $(\mathcal{V}_A, \bar{\mathcal{E}})$ and an edge $(\bar{i}, \bar{j})$, assume that the function* AugPath$((\bar{i}, \bar{j}), \bar{\mathcal{M}}, (\mathcal{V}_A, \bar{\mathcal{E}}))$ *checks if an augmenting path*

[1]Assume all agents share a global clock. At each time step of the clock, agents will exchange information with all their neighbours.

Algorithm 1 Algorithm for solving the BAP.

Input: Graph $\mathcal{G}_A = (\mathcal{V}_A, \mathcal{E}_A)$, cost set τ, an MCM $\mathcal{M}_0$.
Output: An MCM $\mathcal{M}$ of $\mathcal{G}_A$ that is a minimiser of (1).

```
1: $\mathcal{M} \leftarrow \mathcal{M}_0$
2: matching_exists $\leftarrow$ True
3: $\bar{\mathcal{E}} \leftarrow \mathcal{E}_A$
4: while matching_exists do
5:     $(\bar{i}, \bar{j}) \leftarrow$ MAXEDGE$(\bar{\mathcal{E}}, \tau)$
6:     $\bar{\mathcal{E}} \leftarrow \bar{\mathcal{E}} \backslash \{(\bar{i}, \bar{j})\}$
7:     if $(\bar{i}, \bar{j}) \in \mathcal{M}$ then
8:         $\bar{\mathcal{M}} \leftarrow \mathcal{M} \backslash \{(\bar{i}, \bar{j})\}$
9:         $\mathcal{M}_\nu \leftarrow$ AUGPATH$((\bar{i}, \bar{j}), \bar{\mathcal{M}}, (\mathcal{V}_A, \bar{\mathcal{E}}))$
10:        if $\mathcal{M}_\nu \neq \bar{\mathcal{M}}$ then        ▷ Augmenting path exists
11:            $\mathcal{M} \leftarrow \mathcal{M}_\nu$
12:        else
13:            matching_exists $\leftarrow$ False
14:        end if
15:    end if
16: end while
17: return $\mathcal{M}$
```

P exists relative to $\bar{\mathcal{M}}$ in $(\mathcal{V}_A, \bar{\mathcal{E}})$. If P exists, the function returns $\mathcal{M}_\nu = \bar{\mathcal{M}} \oplus \mathcal{P}$. If P does not exist, the function returns $\mathcal{M}_\nu = \bar{\mathcal{M}}$.

In sections III.C and III.D, we will verify that these two assumptions hold. For now, assume the functions are correctly implemented. We first verify the correctness of Algorithm 1.

Lemma 3. *If Assumptions 5 and 6 hold, then Algorithm 1 returns an MCM that is a minimiser of (1).*

Proof: See Appendix A. ■

Remark 1. *Consider Assumption 5. Let $\alpha \subseteq \bar{\mathcal{E}}$ be the set of edges in $\bar{\mathcal{E}}$ with largest weight. If α is not a singleton, we need a deterministic method of choosing one edge in α. One example would be to lexicographically order the edges in α according to the indices of agents and tasks, and to choose one edge according to this ordering.*

Next, we introduce the necessary ingredients for a distributed implementation of Algorithm 1.

B. Distributed Edge Removal

In Algorithm 1, we initialise the set $\bar{\mathcal{E}} = \mathcal{E}_A$. Then, an edge is removed from $\bar{\mathcal{E}}$ in line 6 of Algorithm 1. For distributed edge removal under Assumption 2, we instead initialise the set $\bar{\mathcal{A}}_i = \mathcal{A}_i$, for all $i \in \mathcal{D}$. Then line 6 of Algorithm 1 can be executed by particular agent $\bar{i}$ as

$$\bar{\mathcal{A}}_{\bar{i}} \leftarrow \bar{\mathcal{A}}_{\bar{i}} \backslash \{(\bar{i}, \bar{j})\}, \tag{2}$$

with the same resulting effect. Similarly edge $(\bar{i}, \bar{j})$ is removed from the matching $\mathcal{M}$ in line 8 of Algorithm 1. For distributed edge removal under Assumption 3,

$$m_{\bar{i}} = -1, \tag{3}$$

corresponds to $(\bar{i}, \bar{j})$ being removed from $\mathcal{M}$. This notation is consistent to that used in section III.D.

C. Distributed Implementation of MAXEDGE$(\cdot)$

We now discuss a function that allows Assumption 5 to be satisfied under Assumptions 1-4. Given an edge set $\bar{\mathcal{E}} \subseteq \mathcal{E}_A$, agents must find an edge

$$(\bar{i}, \bar{j}) = \arg \max_{(i,j) \in \bar{\mathcal{E}}} \tau_{ij}, \tag{4}$$

without conflict[2]. The problem in (4) is also known as the max-consensus problem [13], [14]. Denote $N_i = \{k | (i,k) \in \mathcal{E}_C\}$ the set of neighbours of an agent $i \in \mathcal{D}$ in $\mathcal{G}_C$.

Lemma 4. *Under Assumptions 1-4, the max-consensus algorithm [14, eq. (3)] solves the problem given by (4) and fulfills Assumption 5.*

Proof: Let $x_i(0) = \max_{(i,j) \in \bar{\mathcal{A}}_i} \tau_{ij}$, for all $i \in \mathcal{D}$. Apply the max-consensus algorithm $x_i(k) = \max_{j \in N_i} x_j(k)$, for all $i \in \mathcal{D}$, where k denotes the kth communication instance. Agents $i \in \mathcal{D}$ converge to $x_i(D) = \max_{i \in \mathcal{D}} \max_{(i,j) \in \bar{\mathcal{A}}_i} \tau_{ij} = \max_{(i,j) \in \bar{\mathcal{E}}} \tau_{ij}$ after D communication instances, where D is the diameter of $\mathcal{G}_C$. See [14, Theorem 4.1] for proof of convergence of the max-consensus algorithm under Assumption 4; $\mathcal{G}_C$ is undirected and connected. Note in (4) we require the argmax rather than the maximum; besides finding the maximum weight amongst its neighbours, agents $i \in \mathcal{D}$ must additionally take note of the corresponding edge in each communication instance. ■

D. Distributed Implementation of AUGPATH$(\cdot)$

We now present a distributed function that satisfies Assumption 6. Given an edge set $\bar{\mathcal{E}}$, we must search for an augmenting path relative to a given matching $\bar{\mathcal{M}}$ under Assumptions 1-4.

Definition 9. *(Alternating tree) A tree is a graph in which any two vertices are connected by exactly one path. Given a graph $\mathcal{G}$ and a matching $\mathcal{M}$, $\mathcal{G}$ is an alternating tree relative to $\mathcal{M}$ iff $\mathcal{G}$ is a tree and there exists a vertex (called the root) in $\mathcal{G}$ such that any path between the root and every other vertex in $\mathcal{G}$ is an alternating path relative to $\mathcal{M}$.*

From [15] and given a root vertex, a depth-first search (DFS) is systematic and will visit every vertex in the graph that has a path to the root vertex. This holds when the number of vertices in the graph is finite. A tree is formed from the explored vertices of a DFS. By construction, Function 1 implements a DFS with $\bar{j}$ as the root. In particular, only alternating paths are considered in Function 1. An alternating tree that is a subgraph of $(\mathcal{V}_A, \bar{\mathcal{E}})$ relative to matching $\bar{\mathcal{M}}$ is formed from the explored vertices. To observe this, note that the only child of every agent $i \in \mathcal{D}$ in the tree is m_i, which is the task that i is matched to in $\mathcal{M}$; this is seen in line 23 of Function 1. By enforcing this pattern, all paths in the tree formed by this DFS are alternating paths.

[2]See Remark 1.

Function 1 Distributed function to find an augmenting path.

Input: Edge $(\bar{i}, \bar{j})$, matching $\bar{\mathcal{M}}$, graph $(\mathcal{V}_A, \bar{\mathcal{E}})$.
Output: New MCM $\mathcal{M}_\nu$.

```
 1: function AUGPATH($(\bar{i},\bar{j})$, $\bar{\mathcal{M}}$, $(\mathcal{V}_A,\bar{\mathcal{E}})$)
 2:     $F \leftarrow \emptyset$
 3:     $m_i \leftarrow j|_{(i,j)\in\bar{\mathcal{M}}}, \forall i \in \{a|a \in \mathcal{D}, (a,b) \in \bar{\mathcal{M}}\}$
 4:     $m_i \leftarrow -1, \forall i \in \{a|a \in \mathcal{D}, (a,b) \notin \bar{\mathcal{M}}\}$        ▷ eq. (3)
 5:     $\nu_i \leftarrow m_i, \forall i \in \mathcal{D}$
 6:     search_complete ← False
 7:     $t \leftarrow \bar{j}$
 8:     while ¬search_complete do
 9:         $S_t \leftarrow \{i|i \in \mathcal{D}, i \notin F, (i,t) \in \bar{\mathcal{E}}\}$
10:         $a^* \leftarrow \arg\min_{i\in S_t} \tau_{it}$
11:         if $S_t = \emptyset$ and $t = \bar{j}$ then
12:             search_complete ← True
13:         else if $S_t = \emptyset$ and $t \neq \bar{j}$ then
14:             $a^* \leftarrow k$, s.t. $m_k = t$
15:             $t \leftarrow \nu_{a^*}$
16:             $\nu_{a^*} \leftarrow m_{a^*}$
17:         else if $a^*$ is free then
18:             $\nu_{a^*} \leftarrow t$
19:             search_complete ← True
20:         else
21:             $\nu_{a^*} \leftarrow t$
22:             $F \leftarrow F \cup \{a^*\}$
23:             $t \leftarrow m_{a^*}$
24:         end if
25:     end while
26:     $\mathcal{M}_\nu \leftarrow \{(i,\nu_i)\}_{i\in\mathcal{D},\nu_i\neq-1}$
27:     return $\mathcal{M}_\nu$
28: end function
```

Remark 2. *Function 1 implements a search to find a free agent. Each agent is checked at most once for this success condition. Let set F contain all agents $i \in \mathcal{D}$ that have been so explored. If the search is successful, the function must return the augmenting path. Variable ν_i stores information about the path between current vertex t and root $\bar{j}$.*

Remark 3. *Finding $a^* = \arg\min_{a\in S_t} \tau_{at}$ in line 10 of Function 1 corresponds to the min-consensus problem. This is the same as the max-consensus problem in (4), only the max is replaced with min. Note, only a subset of agents and their edges are considered here, $S_t \subseteq \mathcal{D}$.*

Lemma 5. *Under Assumptions 1-4, the function* AUGPATH$(\cdot)$ *in Function 1 fulfills Assumption 6.*

Proof: See Appendix B. ■

Algorithm 1 is a procedure for solving the BAP. Each step in this procedure can be implemented under Assumptions 1-4. Thus, we have the following theorem.

Theorem 1. *Algorithm 1 solves Problem 1.*

Proof: From Lemma 3, Algorithm 1 returns an MCM that is a minimiser of (1). From Lemmas 4 and 5, the

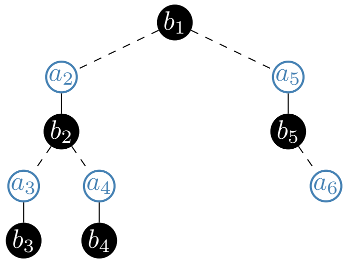

Fig. 1. An alternating tree relative to a matching $\mathcal{M}$. Dotted lines represent edges not in the matching $\mathcal{M}$, solid lines represent edges in $\mathcal{M}$. Shown here, a set of agents $\{a_2, a_3, a_4, a_5, a_6\}$ and a set of tasks $\{b_1, b_2, b_3, b_4, b_5\}$.

functions MAXEDGE$(\cdot)$ and AUGPATH$(\cdot)$ in Algorithm 1 can be implemented such that they satisfy Assumptions 1-4. Edge removal satisfying Assumptions 1-4 can be implemented as in (2) and (3). ■

Remark 4. *With the following adjustment, we can remove multiple edges per iteration of the while-loop in Algorithm 1. In line 5 of Algorithm 1, rather than finding the largest weighted edge $(\bar{i}, \bar{j})$ in $\bar{\mathcal{E}}$, instead find the largest weighted edge in $\mathcal{M}$; denote this critical edge as $(\bar{\bar{i}}, \bar{\bar{j}}) = \arg\max_{(i,j)\in\mathcal{M}} \tau_{ij}$. Then, all agents remove all edges with weights larger than or equal to the weight of $(\bar{\bar{i}}, \bar{\bar{j}})$, sans edges that are in $\mathcal{M}$. Then proceed with the removal of $(\bar{\bar{i}}, \bar{\bar{j}})$ triggering a search for an augmenting path.*

IV. NUMERICAL ANAYSIS

Recall the complete bipartite graph $\mathcal{G}_A$. A greedy algorithm is one that sequentially chooses the edge with lowest cost given prior selections while ensuring the chosen edges form a matching of $\mathcal{G}_A$. We compare the performance of the distributed BAP algorithm, henceforth referred to as distBAP, with a greedy algorithm. The greedy algorithm we use in this section is CBAA from [8]. We implemented distBAP with the edge removal method in Remark 4.

All assignment costs generated in the following examples are integers drawn from a random uniform distribution over the interval $[1, 50]$. The number of agents and tasks are equal in all the following examples, $m = n$. By Assumption 4, agents communicate synchronously and share a global clock. We set $\mathcal{G}_C$ to be a cycle graph with diameter $D = \lceil((m - 1)/2)\rceil$, where $\lceil x \rceil = \min\{n \in \mathbb{Z}|n \geq x\}$.

A. Average Number of Iterations to Converge

The worst-case completion time of distBAP is $|\mathcal{E}_A|(D + 2nD)$ time steps of the global clock. Observe that MAXEDGE$(\cdot)$ and AUGPATH$(\cdot)$ appear within the loop of Algorithm 1. The loop in Algorithm 1 executes at most $|\mathcal{E}_A|$ times; a finite number of edges are removed. The function MAXEDGE$(\cdot)$ requires at most D time steps for completion. The function AUGPATH$(\cdot)$ contains a loop with D time steps per loop iteration and at most $2n$ loop iterations; n for exploring plus n for backtracking. If $m = n$, $|\mathcal{E}_A| = n^2$ and distBAP has a worst-case completion time of order $\mathcal{O}(n^3 D)$.

Fig. 2 shows the completion time of distBAP versus the number of tasks n, averaged over 100 simulations per

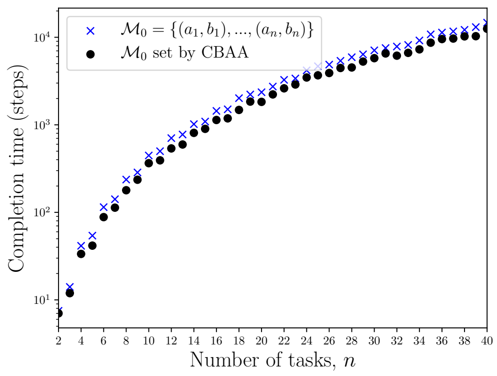

Fig. 2. Number of steps necessary to solve distBAP versus n.

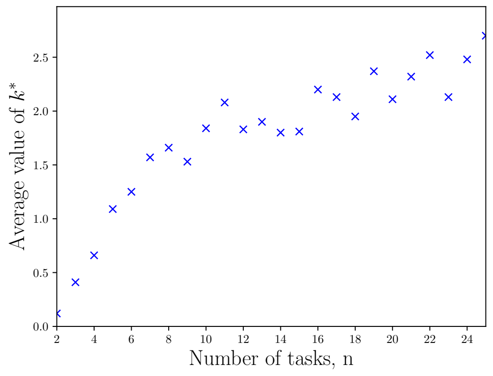

Fig. 4. Average number of iterations of Algorithm 1 for the assignment via distBAP to have lower cost than the assignment via CBAA versus n.

instance of n. The average completion time is polynomial in n. We also plotted the completion time of distBAP when the initial matching $\mathcal{M}_0$, which is an input of Algorithm 1 is set to be the assignments found via a greedy algorithm. On average, there is an improvement in the completion time by choosing the initial matching via the greedy algorithm.

B. Cost Versus the Number of Iterations

We now consider one example simulation with $m = n = 25$. For this simulation, the assignment via CBAA has associated with it an edge with the largest cost of that assignment; we denote this cost to be g. We are interested in the number of iterations of Algorithm 1 it takes for distBAP to find an MCM $\mathcal{M}$ such that the largest cost of $\mathcal{M}$ becomes smaller than g. Fig. 3 shows the results of this simulation. The blue crosses show the cost of the largest weighted edge in $\mathcal{M}$ at a given iteration of Algorithm 1. The solid line shows g. Let k^* be the number of iterations of Algorithm 1 before the assignment via distBAP has lower cost than the assignment via CBAA; for this example, $k^* = 3$.

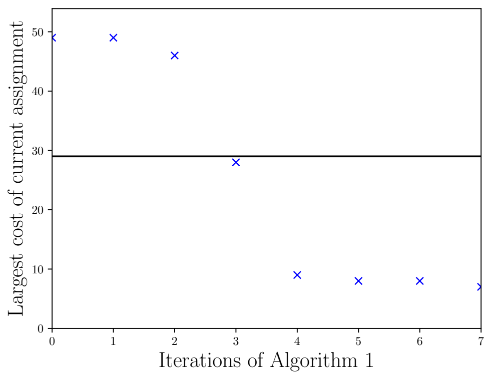

Fig. 3. It takes 3 iterations of Algorithm 1 for the assignment via distBAP to have lower cost than the assignment via CBAA. Thus, $k^* = 3$.

Fig. 3 shows the results from only one example simulation. The average k^* required for different sizes of n is shown in Fig. 4. To generate this plot, 100 simulations were run for each size of n.

By observing Fig. 4, we can see that the average k^* increases as n increases. This is because as n increases, there are more edges in the problem. Consequently, for a given maximum cost g in an assignment found via CBAA, more edges must be removed for the assignment via distBAP to have lower cost than g.

C. Optimality Gap of Greedy Assignment

Let the cost of the largest edge in the assignment found via CBAA be g. Let the cost of the largest edge in the final assignment found via distBAP be h. Let r be the largest possible cost that can be generated. Here $r = 50$ since we draw costs from the interval $[1, 50]$. Fig. 5 shows the normalised optimality gap $opt = \frac{g-h}{r}$ averaged across 100 simulations per instance of n.

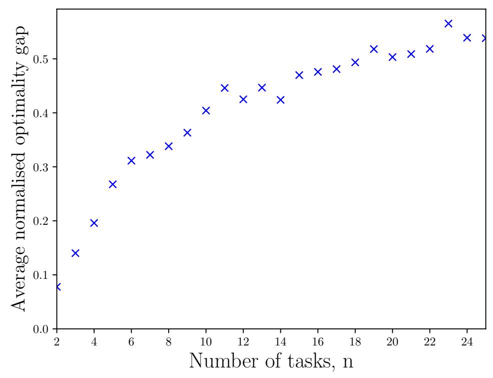

Fig. 5. Average normalised optimality gap, $(g - h)/r$ versus n.

From Fig. 5, the optimality gap between CBAA and distBAP increases as n increases. This is expected since the cost of the largest edge in the final assignment found via distBAP decreases as n increases. As n increases, the number of MCMs in the problem increase, so it becomes more likely that an MCM with a smaller cost can be found. The maximum cost g in an assignment found via CBAA will not decrease as the number of tasks increases since it depends only on the cost of the last chosen edge.

As the number of tasks increases, there are more edges to be removed. Thus, on average k^* increases with n. But the largest cost of the assignment decreases with every iteration, so we are guaranteed to get assignments with costs closer to the optimal assignment with every iteration. If distBAP is run until completion, it returns the optimal assignment. This is a crucial property that cannot be guaranteed by a greedy algorithm. Fig. 5 shows the largest cost in an assignment found via a greedy algorithm can be twice as high as the largest cost in an assignment that solves the BAP exactly.

V. CONCLUSIONS AND FUTURE WORKS

In this paper, we presented a distributed algorithm to solve the BAP. The algorithm begins with an initial valid assignment and iteratively modifies the assignment such that the largest cost in the assignment decreases with every iteration. When no further modification can be performed to produce a valid assignment with lower largest cost, the assignment is optimal with respect to the BAP. In order for the assignment to be iteratively modified in this way, a novel distributed approach to search for an augmenting path in a bipartite graph was presented. The optimality gap that occurs when an assignment is made via a greedy algorithm contrasts the optimality guarantee provided by the presented distributed algorithm as demonstrated in the numerical case study. An immediate extension to this work is the application of the algorithm to problems where agents can carry out more than one task at a time and or where tasks require more than one agent to be completed. Interesting future work includes improving of the convergence rate/completion time of distributed algorithm by bounding the assignment costs.

APPENDIX A
PROOF OF LEMMA 3

At iteration $k = 1, 2, ..., f$ of the while-loop of Algorithm 1 edge $(\bar{i}^k, \bar{j}^k)$ is removed in line 6, where f is some final iteration. Denote $\mathcal{G}^k := (\mathcal{V}_A, \bar{\mathcal{E}}^k)$, where $\bar{\mathcal{E}}^k = \mathcal{E}_A \backslash \{(\bar{i}^1, \bar{j}^1), (\bar{i}^2, \bar{j}^2), ..., (\bar{i}^k, \bar{j}^k)\}$ and $\mathcal{G}^0 := (\mathcal{V}_A, \bar{\mathcal{E}}^0) = (\mathcal{V}_A, \mathcal{E}_A)$. Recall any MCM of $\mathcal{G}_A$ has cardinality n.

The graph $\mathcal{G}^0$ is complete and has a matching $\mathcal{M}^0$ of size n. Assume at some iteration k of the while-loop in Algorithm 1, we have a graph $\mathcal{G}^{k-1}$, an MCM $\mathcal{M}^{k-1}$ of size n and edge to be removed $(\bar{i}^k, \bar{j}^k)$. If the edge $(\bar{i}^k, \bar{j}^k) \notin \mathcal{M}^{k-1}$, then $\mathcal{M}^{k-1}$ is also a matching of $\mathcal{G}^k$ with size n. If the edge $(\bar{i}^k, \bar{j}^k) \in \mathcal{M}^{k-1}$, then $\mathcal{M}^{k-1} \nsubseteq \bar{\mathcal{E}}^k$ and $\mathcal{M}^{k-1}$ is not matching of $\mathcal{G}^k$; $\bar{\mathcal{M}} = \mathcal{M}^{k-1} \backslash \{(\bar{i}^k, \bar{j}^k)\}$ is a matching of $\mathcal{G}^k$ of size $n-1$. Given $\bar{\mathcal{M}}$, Lemmas 1 and 2 allow us to determine if it is possible to recover an MCM $\mathcal{M}^k$ of size n depending the existance of an augmenting path P in $\mathcal{G}^k$ relative to $\bar{\mathcal{M}}$. Algorithm 1 terminates at iteration $k = f$ when no such P exists.

By Assumption 5, the order of edge removal is $\tau_{\bar{i}^1\bar{j}^1} \geq \tau_{\bar{i}^2\bar{j}^2} \geq ... \geq \tau_{\bar{i}^f\bar{j}^f}$. All the edges in $\mathcal{G}_A$ with weights strictly less than $\tau_{\bar{i}^k\bar{j}^k}$ appear as edges in $\mathcal{G}^k$, for $k = 1, ..., f$. If $\mathcal{G}_A$ contains a matching $\mathcal{M}'$ of size n with all edges in $\mathcal{M}'$ having weights strictly less than $\tau_{\bar{i}^k\bar{j}^k}$, then $\mathcal{G}^k$ contains a matching of size n. Then from the contrapositive, if $\mathcal{G}^k$ does not contain a matching of size n, then $\mathcal{G}_A$ does not contain a matching $\mathcal{M}'$ of size n with all edges in $\mathcal{M}'$ having weights strictly less than $\tau_{\bar{i}^k\bar{j}^k}$. From Lemma 2, the while-loop terminates at some iteration $k = f$ when $\mathcal{G}^f$ does not contain a matching of size n. Then, $\mathcal{G}_A$ does not contain an MCM $\mathcal{M}'$ with all edges in $\mathcal{M}'$ having weights strictly less than $\tau_{\bar{i}^f\bar{j}^f}$. But from the prior iteration $k = f-1$, $\mathcal{M}^{f-1}$ is a matching of size n of $\mathcal{G}_A$ with weights less than or equal to $\tau_{\bar{i}^f\bar{j}^f}$. Algorithm 1 returns $\mathcal{M}^{f-1}$, a minimiser of (1).

APPENDIX B
PROOF OF LEMMA 5

At every iteration of the while-loop in Function 1, there exists an alternating path P between the current vertex t and $\bar{j}$. Let $K_P = \{a \in \mathcal{D} | (a, b) \in P\}$. Define a function $\phi : \mathcal{D} \mapsto \mathcal{T}$ mapping an agent in the tree to its parent vertex. From lines 18 and 21, for all agents $i \in K_P$, $\nu_i = \phi(i)$. From lines 5 and 16, for all agents $i \notin K_P$, $\nu_i = m_i$. Thus, we have at every iteration of the while-loop $\{(i, \nu_i) | i \in \mathcal{D}\} = \{(i, m_i) | i \notin K_P\} \cup \{(i, \phi(i)) | i \in K_P\}$. Observe that $\{(i, \nu_i) | i \in \mathcal{D}, \nu_i \neq -1\} = \{(i, m_i) | i \notin K_P, m_i \neq -1\} \cup \{(i, \phi(i)) | i \in K_P\} = \bar{\mathcal{M}} \oplus P$. Thus, if P is an augmenting path, then the function returns $\mathcal{M}_\nu = \bar{\mathcal{M}} \oplus P$ as required. On the other hand, if no augmenting path exists, the search terminates at $t = \bar{j}$, i.e., $P = \emptyset$ and $K_P = \emptyset$; so $\{(i, \nu_i) | i \in \mathcal{D}, \nu_i \neq -1\} = \{(i, m_i) | i \in \mathcal{D}, m_i \neq -1\} = \bar{\mathcal{M}}$.

REFERENCES

[1] I. Shames, A. Dostovalova, J. Kim, H. Hmam, Task Allocation and Motion Control for Threat-Seduction Decoys, *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, Melbourne, VIC, 2017, pp. 4509-4514.

[2] R. Garfinkel, An Improved Algorithm for the Bottleneck Assignment Problem, *Operations Research*, 19(7), 1971, pp. 1747-1751.

[3] H. N. Gabow, R. E. Tarjan, Algorithms for Two Bottleneck Optimization Problems, *Journal of Algorithms*, 9(3), 1988, pp. 411-417.

[4] A. P. Punnen, K. P. K. Nair, Improved Complexity Bound for the Maximum Cardinality Bottleneck Bipartite Matching Problem, *Discrete Applied Mathematics*, 55(1), 1994, pp. 91-93.

[5] U. Derigs, U. Zimmermann, An Augmenting Path Method for Solving Linear Bottleneck Assignment Problems, *Computing*, 19(4), 1978, pp. 285-295.

[6] H. W. Kuhn, The Hungarian Method for the Assignment Problem, *Naval Research Logistics Quarterly 2*, 1955, pp. 83-97.

[7] S. Chopra, G. Notarstefano, M. Rice, M. Egerstedt, A Distributed Version of the Hungarian Method for Multi-Robot Assignment, *IEEE Transactions on Robotics*, 33(4), 2017, pp. 932-947.

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

[9] B. P. Gerkey, M. J. Matarić, A Formal Analysis and Taxonomy of Task Allocation in Multi-robot Systems, *The International Journal of Robotics Research 23(9)*, 2004, pp. 939-954.

[10] J Turner, Q. Meng, G. Schaefer, A. Soltoggio, Fast Consensus for Fully Distributed Multi-agent Task Allocation, *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, 2018, pp. 832-839.

[11] A. J. Smith, G. Best, J. Yu, G. A. Hollinger, Real-time Distributed Non-myopic Task Selection for Heterogeneous Robotic Teams, *Autonomous Robots 43(3)*, 2019, pp. 789-811.

[12] J. E. Hopcroft, R. M. Karp, An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs, *SIAM Journal on Computing*, 2(4), 1973, pp. 225-231.

[13] A. Tahbaz-Salehi, A. Jadbabaie, A One-Parameter Family of Distributed Consensus Algorithms with Boundary: From Shortest Paths to Mean Hitting Times, *Proceedings of the 45th IEEE Conference on Decision and Control*, IEEE, 2006, pp. 4664-4669.

[14] B. M. Nejad, S. A. Attia, J. Raisch, Max-consensus in a Max-plus Algebraic Setting: The Case of Fixed Communication Topologies, *2009 XXII International Symposium on Information, Communication and Automation Technologies*, IEEE, 2009, pp. 1-7.

[15] S. M. LaValle, Planning Algorithms, *Cambridge University Press*, 2006.