A Distributed Augmenting Path Approach for the Bottleneck Assignment Problem

Mitchell Khoo, Tony A. Wood, Member, IEEE, Chris Manzie, Senior Member, IEEE, and Iman Shames, Member, IEEE

Abstract—We develop an algorithm to solve the bottleneck assignment problem (BAP) that is amenable to having computation distributed over a network of agents. This consists of exploring how each component of the algorithm can be distributed, with a focus on one component in particular, i.e., the function to search for an augmenting path. An augmenting path is a common tool used in most BAP algorithms and poses a particular challenge for this distributed approach. Given this significance, we compare the properties of two different methods to search for an augmenting path in a bipartite graph. We evaluate the derived approaches with a simulation-based complexity investigation.

Index Terms—Autonomous agents, autonomous systems, distributed algorithms, multi-agent systems.

I. INTRODUCTION

An assignment problem emerges when multiple tasks must be allocated to multiple agents. In [1], [2], and [3], reviews of the different types of assignment problems are presented and categorized according to objective functions, the number of agents required to complete tasks and the number of tasks individual agents can carry out. The bottleneck assignment problem (BAP) is a particular assignment problem with the objective of minimizing the costliest agent-task pairing. Such an assignment is typically necessary in time-critical problems, where tasks are carried out by agents simultaneously and the time taken to complete all tasks must be minimized. An example application of the BAP is the threat seduction problem in [4], where decoys are assigned to multiple incoming threats.

The BAP is well-studied and there exist many centralized algorithms to solve it, e.g., [4], [5], [6], [7], [8]. Centralized computation refers to the case where one decision-maker has access to the allocation cost of every potential agent-task pairing. The threshold algorithm in [5] involves an initial threshold cost and checking if an assignment can be created with allocation costs smaller than the threshold. The threshold is iteratively increased until a valid assignment is found, which then corresponds to the optimal assignment for the BAP. In [6] and [7], improvements on the complexity of the threshold algorithm are made by moving the threshold according to a binary search pattern rather than incrementally. In [8], the BAP is solved for a subset agents and tasks, where the size of the subset is increased until it contains all the agents and tasks.

In many applications, no centralized decision-maker is available and there are restrictions on the information shared between agents. This is motivation to consider solving the BAP with computation distributed over a network of agents. As a comparison, there are many distributed algorithms for assignment problems with other objectives and in particular, the linear assignment problem (LAP). In the LAP, tasks are allocated to agents such that the sum of the costs of the allocations is minimized. In [9], a distributed version of the well-known Hungarian method from [10] for solving the LAP is developed. In [11] and [12], so-called auction algorithms are presented. The LAP can be cast as a linear program with binary decision variables to represent the allocation of an agent to a task. In [13], a distributed simplex algorithm is developed to solve linear programs and the LAP is used as a motivating example. Greedy algorithms sequentially pick one agent-task allocation with lowest cost from the remaining choices of allocations. There exist distributed greedy algorithms, e.g., the consensus-based auction algorithm (CBAA) in [14]. This article presents a method to solve BAP with distributed computation. The challenging part of this approach lies in searching for augmenting paths when information and computation is distributed over agents. We investigate two ways to conduct such a search and discuss the tradeoffs between them. This approach for solving the BAP is based on preliminary work in [15], where a distributed algorithm of this kind was first conceived. The novel contributions are three-fold. First, we define a so-called critical edge and relate it to a solution of the BAP, i.e., we prove that finding such a critical edge is equivalent to finding a solution to the BAP. This is a general graph theoretical result, but we apply it specifically to extend the analysis in [15] on the convergence of the distributed BAP algorithm to a solution and provide more insight into the role of augmenting paths in obtaining a solution. Second, a general framework for an augmenting path search is identified, i.e., the two augmenting path search methods provided in this article are not the only ways of finding an augmenting path and we determine steps that guarantee a search will find an augmenting path should one exist and terminate should one not exist. Third, a new method to search for an augmenting path in a bipartite graph with distributed computation is developed and compared to the existing one from [15].

In Section II, the graph theoretical background is introduced, including several novel definitions for analyzing the structure of the BAP. In Section III, we present an algorithm for solving the BAP with computation distributed over a network of agents, which includes a distributed search for an augmenting path. In Section IV, we present an alternative approach for conducting the distributed search for an
augmenting path and compare it to the approach in Section III. Finally, in Section V, we present a simulation-based investigation into the complexity and implications of the different algorithms.

II. PRELIMINARIES

Consider an undirected graph \( G = (V, E) \), where \( V \) is a set of vertices and \( E \) is a set of edges, and every edge is a set of two vertices. We provide the following graph theoretical definitions, which are similar to the ones found in [2] and [16].

Definition 1 (Matching, maximum cardinality matching): A matching \( M \) of graph \( G \) is a set of edges such that \( M \subseteq E \) and no vertex \( v \in V \) is incident to more than one edge in \( M \). A maximum cardinality matching (MCM) is a matching \( M^{\max} \) of \( G \) with maximum cardinality.

Definition 2 (Neighbors): The set of neighbors of vertex \( v \in V \) in graph \( G = (V, E) \) is denoted as \( \mathcal{N}(v) := \{k \in V | (v, k) \in E \} \).

Definition 3 (Path): Given distinct vertices \( v_1, v_2, \ldots, v_k \in V \), the set of edges \( E(v_1, v_2, \ldots, v_k) = \{(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k)\} \) is a path between \( v_1 \) and \( v_k \) of length \( k \).

Definition 4 (Diameter): The diameter of graph \( G = (V, E) \) is defined as \( \text{Diam}(G) := \max_{v \in V} \sum_{v' \in V} d(v, v') \), where \( d(v, v') \) is the length of the shortest path between vertices \( v, v' \in V \).

Definition 5 (Alternating path): Given a matching \( M \), a path \( P \) is called an alternating path relative to \( M \) if and only if each vertex that is incident to an edge in \( P \) is incident to no more than one edge in \( P \cup M \) and no more than one edge in \( P \setminus M \).

Definition 6 (Free vertex): Given a matching \( M \), vertex \( v \in V \) is free if and only if \( \{v, w\} \notin M \) for all \( w \in V \).

Definition 7 (Augmenting path): Given a matching \( M \) and a path \( P \) between vertices \( v_1, v_2, \ldots, v_k \in V \), \( P \) is an augmenting path relative to \( M \) if and only if \( P \) is an alternating path relative to \( M \) and \( v_1 \) and \( v_k \) are both free vertices.

A tree is a graph \( G_{\text{t}} = (V_{\text{t}}, E_{\text{t}}) \) for which any two vertices in \( V_{\text{t}} \) are connected by exactly one path, and one vertex \( r \in V_{\text{t}} \) is designated as the root vertex.

Definition 8 (Level): Given a tree \( G_{\text{t}} = (V_{\text{t}}, E_{\text{t}}) \) with root vertex \( r \in V_{\text{t}} \), the level of a vertex \( v \in V_{\text{t}} \) in \( G_{\text{t}} \) is the length of the path between \( v \) and \( r \).

Definition 9 (Parent and child): Given a tree \( G_{\text{t}} = (V_{\text{t}}, E_{\text{t}}) \) with root vertex \( r \in V_{\text{t}} \), vertex \( v \in V_{\text{t}} \) is a parent of vertex \( v' \in V \) if and only if \( L_{v'} = L_v + 1 \), where \( L_\cdot \) is the level of \( \cdot \) in \( G_{\text{t}} \), respectively. Vertex \( v' \in V_{\text{t}} \) is a child of vertex \( v \in V_{\text{t}} \) if and only if \( v' \) is a parent of \( v \).

Definition 9 can be extended to define a descendant, i.e., vertices \( v' \in V_{\text{t}} \) with levels \( L_{v'} = L_v + k \), \( k \in \{1, 2, 3, \ldots\} \). Consider a set of agents \( A = \{a_1, a_2, \ldots, a_m\} \) and a set of tasks \( B = \{b_1, b_2, \ldots, b_n\} \), with \( m, n \in \mathbb{N} \). Assume that \( A \cap B = \emptyset \) and that \( m \geq n \). We represent all possible assignments of tasks to agents as a graph, i.e., let \( G_0 = (V_0, E_0) \) be a complete weighted bipartite graph with vertex set \( V_0 = A \cup B \) and edge set \( E_0 = \{(i, j)| i \in A, j \in B\} \). Each edge \( i \in A \) is associated with a weight \( \tau_{i,j} \in \mathbb{R} \), which corresponds to the cost for agent \( i \) carrying out task \( j \). We define the function \( w : E_0 \rightarrow \mathbb{R} \) that maps the edges of \( G_0 \) to their weights. Let \( C(G_0) \) be the set of all MCMs of \( G_0 \). The BAP for graph \( G_0 \) is formulated as

\[
\text{BAP} : \min_{M \in C(G_0)} \max_{e \in M} w(e). \tag{1}
\]

We also define the set of MCMs that are solutions to the BAP given in (1).

Definition 10 (Bottleneck assignment): The set of all bottleneck assignments of \( G_0 \) is defined as \( S(G_0) := \arg \min_{M \in C(G_0)} \max_{e \in M} w(e) \).

Definition 11 (Bottleneck edge): Given any bottleneck assignment \( M \in S(G_0) \), any \( e \in \arg \max_{M \in C(G_0)} w(e) \) is a bottleneck edge of \( G_0 \).

Finally, we introduce the novel concepts of a pruned edge set and a critical edge based on [17].

Definition 12 (Pruned edge set): Given an MCM \( M \) of graph \( G_0 \), a pruned edge set is defined as \( \phi(G_0, M) = M \cup \{e \in E_0 | w(e) < \max_{M \in C(G_0)} w(e)\} \).

Definition 13 (Critical edge): An edge \( e \in \mathcal{E}_0 \) is a critical edge of graph \( G_0 \) if and only if there exists an MCM \( G_0 \) such that \( e \in \arg \max_{M \in C(G_0)} w(e) \) and \( \phi(G_0, M) \setminus \{e\} \) does not contain an augmenting path relative to \( M \setminus \{e\} \).

III. DISTRIBUTED ALGORITHM FOR SOLVING THE BAP

We outline a modular approach for solving the BAP and show that all components can be applied in a distributed setting. The following assumptions model a distributed setting, where information available to each agent may be limited.

Assumption 1: Assume an agent \( i \in A \) has access to the set of incident edges in graph \( G_i, E_i = \{i, j\} | j \in B_i \} \), and the weight of each edge in \( E_i \), i.e., the tuple \( W_i = \{w(i, b_1), w(i, b_2), \ldots, w(i, b_n)\} \).

Assumption 2: Let communication between agents be modeled by a time invariant, undirected and connected graph \( G_C = (A, E_C) \), with vertex set being the set of agents, edge set \( E_C \), and diameter \( D \). Agents communicate synchronously, i.e., all agents share a global clock and at discrete time steps of the clock, all agents \( i \in A \) exchange information with their neighbors \( i' \in N(G_0, i) \). All agents have knowledge of \( D \).

Remark 1 does not imply centrality. No agent knows the whole \( M \). The function \( Q \) is just for the aid of the reader, we apply it later to refer to each element of matching \( M \) that is being stored by a different agent.

Remark 1: An MCM \( M \) of \( G_0 \) can be jointly stored by agents. For each agent \( i \in A \) that is not free, agent \( i \) stores its matched task \( m_i \), where \( \{m_i\} \in M \). For free agent \( i' \in A \), we henceforth use \( m_i = b \) to denote it is unmatched, i.e., \( b \notin \mathcal{B} \) denotes doing no task. The function \( Q : C(G_0) \times A \rightarrow B \cup \{\emptyset\} \) maps agents to their matched tasks in an MCM of \( G_0 \).

The problem of solving the BAP in a distributed manner is formally stated as follows.

Problem 1: Under Assumptions 1 and 2, obtain a solution to the BAP given by (1).

A. Algorithm for Solving the BAP

We first present PruneBAP for solving the BAP. Then, we discuss properties of this algorithm, including how each subroutine of PruneBAP can be implemented in the distributed setting. To initialize PruneBAP, we require an arbitrary MCM \( M_D \in C(G_0) \) of \( G_0 \), e.g., agents and tasks can be initially matched based on arbitrary indexing, \( M_D = \{a_1, b_2\}, a_2, b_1 \in A, b_2 \in B, p = q\). Proposition 1 relates the existence of an augmenting path to the existence of an MCM within a set of edges. Corollary 1 follows from combining Definition 13 and Proposition 1, with the pruned edge set \( \phi(G_0, M) = E' \).

Proposition 1 (Proof in Appendix B1): Consider an MCM \( M \) of graph \( G = (V, E) \), a set of edges \( E' \subset E \) such that \( M \subseteq E' \subseteq E \), and an edge \( e \in M \). An augmenting path \( P \subseteq E \setminus \{e\} \) exists relative to \( M \setminus \{e\} \) if and only if there exists an MCM \( M' \) of \( G \) such that \( M' \subseteq E \setminus \{e\} \).
Algorithm: PruneBAP.

Input: Graph $G_b = (V_b, E_b)$ and an MCM $M_b$.
Output: An MCM $M$ of $G_b$ that is a minimizer of $(1)$. \\
1: $M ← M_b$ \\
2: matching_exists ← True \\
3: while matching_exists do \\
4:  $(\hat{e}, w(\hat{e})) ← \text{MAXEdge}(M)$ \\
5:  $\tilde{E} ← \phi(G_b, M) \setminus \{\hat{e}\}$ \\
6:  $M ← M \setminus \{\hat{e}\}$ \\
7:  $M_\nu ← \text{AUGPATH}(\hat{e}, M, (V_b, \tilde{E}))$ \\
8:  if $M_\nu \neq M$ then \hspace{1cm} $\triangleright$ Augmenting path exists \\
9:  $M ← M_\nu$ \\
10: else \\
11:  matching_exists ← False \\
12: end if \\
13: end while \\
14: return $M$

In Sections III-B1 and III-B3, we verify that the following two requirements can be met and that they can be implemented in a distributed setting. The notation $A \oplus B$ denotes the symmetric difference of the sets $A$ and $B$.

**Requirement 1:** Given an MCM $M$ of $G_b$, we require a function $\text{MAXEdge}(M)$ that returns the tuple $(\hat{e}, w(\hat{e}))$, where $\hat{e} \in \arg \max_{e \in E} w(e)$.

**Requirement 2:** Given a graph $(V_b, E')$, an MCM $M$, and a matching $M = M \setminus \{\hat{e}\}$, we require a function $\text{AUGPATH}(\hat{e}, M, (V_b, E'))$ that checks if there exists an augmenting path $P$ relative to $M$ in $E' \setminus \{\hat{e}\}$. If $P$ exists, the function returns an MCM $M_\nu = M \oplus P$. If $P$ does not exist, the function returns $M_\nu \equiv M$.

**Proposition 2 (Proof in Appendix B2):** If Requirements 1 and 2 are met, then PruneBAP returns a bottleneck assignment of $G_b$.

We do not consider PruneBAP a “centralized algorithm.” PruneBAP and all its modular components can be implemented in different ways, some of which may be centralized and some distributed. A distributed implementation of PruneBAP is just one where all the components have distributed implementation.

### B. Implementing PruneBAP With Distributed Computation

We now discuss how the individual components of PruneBAP can be implemented with the distributed setting outlined in Assumptions 1 and 2 and Remark 1. There are three main components to consider, i.e., finding the largest edge within a set of edges, removal of edges to form a pruned edge set, and searching for an augmenting path.

#### 1) Distributed Implementation of MAXEdge():

We discuss a function that allows Requirement 1 to be satisfied under Assumptions 1 and 2. Given an edge set $\tilde{E} \subseteq E_b$, agents must find an edge

$$\hat{e} = \arg \max_{e \in \tilde{E}} w(e) \quad (2)$$

without conflict, i.e., agents reach consensus on the next edge to test as a critical edge. See Remark 2 about resolving ties should they arise. The problem in (2) is known as the max-consensus problem [18], [19].

**Lemma 1 (Proof in [19]):** Under Assumptions 1 and 2, the max-consensus algorithm in [19, eq. (3)] solves the problem given by (2) and fulfills Requirement 1.

**Remark 2:** Given an arbitrary set of edges $\tilde{E} \subseteq E_b$, let $\alpha = \arg \max_{e \in \tilde{E}} w(e)$. If $\alpha$ is not a singleton, a deterministic method is required for selecting one edge to satisfy Requirement 1. For instance, given indexed agents, the edge incident to the agent with the lower index number can be selected. If we wish to maintain anonymity of agents, then another method of tiebreaking, such as using timestamps, could be employed.

For Requirement 1, both the edge $\hat{e}$ and the corresponding weight $w(\hat{e})$ are required from the max-consensus algorithm.

#### 2) Distributed Edge Removal:

Given the tuple $(\hat{e}, w(\hat{e}))$ from Requirement 1 and given that each agent $i \in A$ has access to $E_i$ from Assumption 1 and its matched task $m_i$ from Remark 1, each agent $i$ can locally determine the set $\hat{E}_i = \{i, m_i\} \cup \{e \in E_i | w(e) < w(\hat{e})\}$. Thus, we have a distributed representation of the pruned edge set, since $\phi(G_b, M) = \bigcup_{i \in A} \hat{E}_i$. To represent $\hat{E} = \phi(G_b, M) \setminus \{\hat{e}\}$ it remains for the particular agent $i = A \cap \hat{e}$ to additionally prune $\hat{e} = \{i, m_i\}$, i.e., $\hat{E}_i = \{e \in E_i | w(e) < w(\hat{e})\}$.

Similarly edge $\hat{e}$ is locally removed from the matching $M$ in Line 6 of PruneBAP. To complete the distributed edge removal according to Remark 1, agent $i$ is labeled as free, i.e., $m_i = \hat{b}$.

#### 3) Distributed Implementation of AUGPATH():

We present a general framework for an augmenting path search and show that regardless of the order in which agents are explored, this framework guarantees all agents with an alternating path to the root are explored. This method exploits the fact that only one edge is removed from an MCM of $G_b$ at a time.

**Definition 14 (Alternating search):** Given a complete bipartite graph $G_b = (A \cup B, E_b)$, consider a subgraph $(A \cup B, \tilde{E})$, where $\tilde{E} \subseteq E_b$. Consider an MCM of $G_b$, $M = \{\{a_1, m_{a_1}\}, \{a_2, m_{a_2}\}, \ldots, \{a_n, m_{a_n}\}\}$, such that $M \setminus \{\{a_i, m_{a_i}\}\} \subseteq \tilde{E}$ and $\{a_i, m_{a_i}\} \notin \tilde{E}$. An alternating search is defined using the following vertex and edge set constructions.

Consider a tree $(V_1, E_1)$ consisting of only the free root task vertex $m_{a_n} \in B$, i.e., $V_1 = \{m_{a_n}\}$ and $E_1 = \emptyset$. For all $k \in [1, 2, \ldots, f-1]$, we construct vertex and edge sets $V_{k+1} = V_k \cup \{i_k, m_{i_k}\}$ and $E_{k+1} = E_k \cup \{i_k, j_k\}$, satisfying the condition that agent $i_k \in A \setminus V_k$ and task $j_k \in B \cap V_k$ are neighbors in the graph $(A \cup B, \tilde{E})$, i.e., $\{i_k, j_k\} \in \tilde{E}$. Let $f$ be the iteration at which for all tasks $j_f \in B \cap V_f$, there does not exist an agent $i_f \in A \setminus V_f$ with $\{i_f, j_f\} \in \tilde{E}$.

**Proposition 3 (Proof in Appendix B3):** Given the tree $(V_f, E_f)$ constructed in Definition 14, the following statement holds. For any agent $i \in A$, if there exists an alternating path between $i$ and a free task $m_{a_n}$ containing only elements of $\tilde{E}$, then the agent is in the tree, i.e., $i \in V_f$.

AugDFS is a distributed function satisfying Requirement 2 under Assumptions 1 and 2.

AugDFS implements a depth-first search (DFS) [20] that adheres to the pattern of agent exploration in Definition 14. The goal of the search is to find a free agent and the root $j$ is a free vertex. Therefore, by construction the path between that free agent and the root is an augmenting path. On the other hand, Proposition 3 guarantees that failure to find a free agent means an augmenting path does not exist.

**Remark 3:** AugDFS requires storage of the alternating path between the root and the current vertex $t$. The following is a suggested method to achieve this with distributed storage. All agents keep track of the current vertex $t$ by storing a first-in/fast-out stack of tasks; a task is added to the stack when the search proceeds in Line 19 and removed when the search backtracks in Line 12.

**Remark 4:** An approach for choosing the next agent to explore in Line 9 in AugDFS is to have $a^* = \arg \min_{i \in S_b} w((i, t))$. This corresponds to the min-consensus problem and greedily explores edges with smaller weights first.

**Lemma 2 (Proof in Appendix B4):** Under Assumptions 1 and 2, AugDFS fulfills the requirements for AUGPATH() and therefore guarantees that Requirement 2 is satisfied.
4) Distributed Implementation of PruneBAP: Theorem 1 combines the abovementioned results. We also derive a bound on the complexity of PruneBAP in Proposition 4.

Theorem 1 (Proof in Appendix B5): PruneBAP solves Problem 1.

Definition 15 (Time step): Given Assumption 2, one time step refers to one time step or tick of the global clock shared by all agents.

Proposition 4 (Proof in Appendix B6): The worst-case complexity, in terms of time steps, of the distributed implementation of PruneBAP is order $O(mn^2D)$.

If $m = n$, then $|E| = n^2$ and PruneBAP has a worst-case complexity of order $O(n^3D)$. The number of iterations of the while-loop in AugDFS is at most $2n - 1$, where in the worst-case the search explores $n$ matched agents and backtracks $n - 1$ times before terminating. The root $j$ is the only free task vertex in the graph, which narrows down the search for an augmenting path. Without exploiting the fact that $j$ known, finding an augmenting path in a bipartite graph in general has complexity $O(n^2)$, from [2] and [21]. We discuss the complexity of AugDFS in more detail in the following section. Pseudocode for PruneBAP with AugDFS from an agent’s perspective can be found at.

IV. ALTERNATIVE IMPLEMENTATION OF DISTRIBUTED AUGMENTING PATH SEARCH

AugDFS conducts a distributed DFS that allows Requirement 2 to be satisfied. However, this is not the only function that fulfills Requirement 2. We introduce a distributed breadth-first search (BFS) approach.

A. BFS Augmenting Path Search

AugDFS does not implement a standard BFS. Multiple agents $i \in A$ are explored simultaneously rather than only one agent at a time.

Each line in AugDFS can be implemented with distributed computation, i.e., under Assumptions 1 and 2. AugDFS has two main steps. The first step is to determine the set of agents to explore next, shown in Line 8. To achieve this in a distributed setting, each unexplored agent individually checks for membership to the set $S$. The second step is to store all the alternating paths in the alternating search, shown in Line 11. The following remark outlines a method to implement this step in the distributed setting.

Remark 5: Each time an agent $i \in A$ is explored, agent $i$ stores its parent and child vertices, $\nu_i$ and $m_i$, respectively. This pair of tasks $(\nu_i, m_i)$ is then communicated to all other agents. All explored agents $i' \in F$ receive the pair of tasks and determine if task $m_i$ is to be added to their individual sets of descendents. Given that an agent $i' \in F$ receives a pair of tasks $(\nu_i, m_i)$, if $\nu_i$ is a descendant of $i'$, then $m_i$ is also a descendant of $i'$.

Lemma 3 (Proof in Appendix B7): Under Assumptions 1 and 2, AugBFS fulfills the requirements for AugPATH() and therefore guarantees that Requirement 2 is satisfied.

Agents can execute Lines 8–12 without waiting for other agents. The key point is that agents are explored in a way that adheres to Definition 14. This indicates that the synchronous communication between agents in Assumption 2 is a stronger assumption than necessary. However, we limit the discussion to the synchronous case, in this article.

B. Comparing Distributed Search Methods

Table I shows a comparison of AugDFS to AugBFS. Iterations refers to the number of iterations of the while-loops of AugDFS or AugBFS,
Average number of time steps for completion of PruneBAP. For shows the average number of iterations of $n^2 \cdot G$ to be the Euclidean distance $G$ is sparse compared to the complete graph when it is sparse. As expected, the average number of iterations of the while-loop of PruneBAP for $n$ agents is the cardinality of an MCM of the given bipartite graph, and $D$ is the diameter of the communication graph. Refer to Remark 1 about $m_1$.

In AugDFS, only one vertex is explored in every iteration. To reach consensus on which agents to explore next, agents collect information from their neighbors, choose a local candidate and send their choice to their neighbors. In contrast, AugBFS explores all vertices with the same level in the tree at every iteration. Therefore, it returns the shortest augmenting path. However, since multiple agents are explored, agents collect information from their neighbors and then in turn send all this information to their neighbors. Thus, there is a tradeoff between exploring multiple vertices per iteration and communicating more information between agents per iteration. Fig. 1 demonstrates the difference between the alternating trees constructed by the searches in AugDFS and AugBFS.

| Function | AugDFS | AugBFS |
|----------|--------|--------|
| Num. agents explored per iteration | 1 (can be greedily chosen, see Remark 4) | $m$ agents, where $1 \leq m \leq n$ |
| Iterations to terminate | $2n - 1$ | $n$ |
| Time steps per iteration | $D$ | $D$ |
| Worst-case message size sent between agents per iteration | A single 3-tuple, which contains $(m_i, w(i, m_i), i)$ | A set of $m$ 3-tuples, i.e., when number of explored agents $m = n$. Each 3-tuple contains $(m_i, w_i, i)$ |
| Worst-case total info stored by any agent | Set of all $m$ matched tasks, see Remark 3 | Set of all $m$ matched tasks, see Remark 5 |

Fig. 1. Comparison of alternating trees constructed by the two different searches. The edge weights are not listed, but assume that $(a_1, b_1)$ has the largest weight amongst all the drawn edges. Assume undrawn edges have weight larger than the weight of $(a_1, b_1)$. Given an MCM $M$ represented by the solid lines, the top bipartite graph shows the edges in a pruned edge set $\delta(G, M)$. The edge highlighted in red is the edge to be removed, and $b_1$ is the root of the search for an augmenting path. The bottom left tree is constructed via AugDFS and the bottom right tree via AugBFS.

Fig. 2. Average number of iterations of the while-loop of PruneBAP for completion versus the number of tasks, $n$.

Fig. 3. Average number of time steps for completion of PruneBAP versus the number of tasks, $n$.

In this section, we present a numerical analysis of PruneBAP and the functions AugDFS and AugBFS. A time step is described in Definition 15. An iteration of PruneBAP represents the number of iterations of the while-loop of PruneBAP.

Consider an equal number of agents and tasks, i.e., $n = m$, that are all represented by points in $\mathbb{R}^2$. Agents are to be assigned to move from their initial positions to assigned target positions such that a BAP with distance as weights is solved. To this end, we define the weights of the complete bipartite assignment graph $G_0$ to be the Euclidean distance between agents and tasks. All coordinates are generated from a uniform distribution between 0 and 100 normalized distance units.

By Assumption 2, agents communicate synchronously and share a global clock. For the following examples, the communication between agents is modeled as a complete graph $G_C$, i.e., all agents have a communication link to all other agents and the diameter $D = 1$. For connected communication graphs that are not complete, the number of required time steps scales proportionally with the diameter $D$. Thus, the relative comparison of methods is fully illustrated by the case with fully connected communication.

V. COMPLEXITY INVESTIGATION

We evaluate the number of time steps it takes to run PruneBAP and compare the two implementations of an augmenting path search, AugDFS and AugBFS. Fig. 2 shows the average number of iterations of the while-loop of PruneBAP for completion versus the number of tasks, $n$. For every value of $n$, the figure shows the average of 100 realizations of the agent and task positions. As expected, AugBFS requires more iterations of the while-loop of PruneBAP than AugDFS. This is because in AugDFS the augmenting path is constructed by a greedy minimization of weights as described in Remark 4. Therefore, the augmenting path that is found typically contains edges with small weights and fewer iterations of the while-loop of PruneBAP are required. We recall that the worst-case complexity of PruneBAP is $O(n^3)$. Fig. 3 shows the empirical average number of time steps for completion of PruneBAP versus the number of tasks $n$. Again, the values are averaged over 100 realizations of the agent and task positions. Although AugDFS results in fewer iterations of the while-loop of PruneBAP than AugBFS, AugBFS results in fewer time steps for completion. Note that we also plotted cases when $G_0$ is complete and when it is sparse. As expected, the PruneBAP converges faster on average when $G_0$ is sparse compared.
Average maximum number of explored agents and overall average number of explored agents per $D$ time steps for AugBFS versus number of tasks, $n$. For AugDFS, one agent is explored per $D$ time steps. 

Average maximum number of explored agents and overall average number of explored agents per $D$ time steps for AugBFS versus number of tasks, $n$. For AugDFS, one agent is explored per $D$ time steps. 

Weight of largest edge in MCM versus number of time steps. 

Weight of largest edge in MCM versus number of time steps. 

PruneBAP requires an MCM $M$ with largest weight smaller than $g$. This is indicated by the time step at which the blue and red marks drop below the black line in Fig. 5. CBAA only produces an MCM in the final time step, whereas PruneBAP produces a series of MCMs with a nonincreasing largest edge weight.

Fig. 6 shows the average number of time steps required for PruneBAP to find an MCM $M$ that has a largest weight smaller than $g$. Fig. 6 also shows the number of time steps for CBAA to find an assignment. Once again, the averages were taken across 100 realizations for every $n$. Fig. 6 suggests that on average, PruneBAP would converge faster by warm-starting, i.e., initializing PruneBAP with the assignment found via CBAA.

VI. CONCLUSION

We introduced tools, i.e., a pruned edge set and a critical edge to present an algorithm to solve the BAP. PruneBAP iteratively produces an assignment with lower bottleneck weight with the final assignment being a solution to the BAP. The algorithm has several components, i.e., finding a pruned edge set, finding the largest edge amongst agents, and searching for an augmenting path. We derived methods to distribute the execution of each individual component over a network of agents with limited information. In particular, we compared two methods, AugDFS and AugBFS, for conducting a distributed search for an augmenting path. The two methods provide a tradeoff between computational complexity and amount of information that needs to be communicated between agents. We investigated the average numerical complexity of PruneBAP, AugDFS, and AugBFS. As a benchmark, we compared PruneBAP to CBAA, a greedy distributed assignment-finding algorithm.

Aside from their use for solving the BAP, augmenting path searches are applied in many approaches for solving other combinatorial optimization problems. These include the problems of finding an MCM of a bipartite graph, finding the maximum cardinality intersection of a matroid, and finding the minimum of submodular functions, see [21], [22], [23], [24]. The distributed augmenting path search methods derived in this article have the potential to provide benefits for solving other combinatorial optimisation problems. The extension of the methods to other applications is the subject of future work. Other future work might include considering protocols to handle failure of one or more agents midway through PruneBAP.

APPENDIX

A. Auxiliary Lemmas

Lemmas 4 and 5 relate the existence of an augmenting path to the cardinality of matchings in a graph and are used to prove Proposition 1.
The notation $A \oplus B$ denotes the symmetric difference of the sets $A$ and $B$.

Lemma 4 (Proof in [21]): Given a graph $G$, if $M$ is a matching of $G$ and $P$ is an augmenting path relative to $M$, then $M \oplus P$ is also a matching of $G$ and $|M \oplus P| = |M| + 1$.

Lemma 5 (Berge’s theorem, proof in [21]): Given a graph $G$, a matching $M$ is an MCM of $G$ if and only if there is no augmenting path relative to $M$ in the edge set of graph $G$.

B. Proofs

Proof of Proposition 1: First, we prove sufficiency. Assume there exists an augmenting path $P \subseteq E' \setminus \{e\}$ relative to $M \setminus \{e\}$. By Lemma 4, $M' = M \setminus \{e\} \oplus P$ is an MCM of $G$. Since both $P$ and $M \setminus \{e\}$ are subsets of $E' \setminus \{e\}$, their symmetric difference is also a subset of $E' \setminus \{e\}$.

Next, we prove necessity. Assume there does not exist an augmenting path $P \subseteq E' \setminus \{e\}$ relative to $M \setminus \{e\}$. By Lemma 5, the matching $M \setminus \{e\}$ is an MCM of the graph $(V, E)$ with edge set $E' \setminus \{e\}$. We know that $M$ is an MCM of $G$ and it has cardinality $|M|$. Since $M \setminus \{e\}$ has cardinality $|M| - 1$, it is not an MCM of $G$. Thus, there does exist an MCM of $G$ within the set $E' \setminus \{e\}$.

2) Proof of Proposition 2: Given Requirements 1 and 2, we observe that Lines 5, 6, and 7 of PruneBAP test the sufficient and necessary conditions for the existence of the MCM $M'$ based on Proposition 1, with $e = e$ and $E = \phi(G_0, M)$.

Now, let $k \in \{1, 2, \ldots, f\}$ denote the iterations of the while-loop of PruneBAP, where without loss of generality, $f$ is the final iteration. Let $E_k$ denote the set $E$ in Line 5 at iteration $k$ of the while-loop. By Proposition 1, we have that for all iterations $k < f$, there exists an MCM $M_k$ of $G_0$ such that $M_k \subseteq E_k$. At the final iteration $k$, an augmenting path is not found so PruneBAP returns $M = M_{f-1}$. It remains to show that the MCM $M_{f-1} \subseteq E_{f-1}$ is a bottleneck assignment of $G_0$. Assume for contradiction there exists an MCM $M_f$ of $G_0$ with all edges having weights strictly less than $e \in \arg \max_{e \in M_{f-1}} w(e)$. This implies that there exists an MCM $M_f$ of $G_0$ such that $M_f \subseteq \phi(G_0, M_{f-1}) \setminus \{e\} = E_{f-1}$, which contradicts the assumption that $f$ is the final iteration. Following the abovementioned arguments, $e$ is a critical edge of $G_0$ and consequently $M_{f-1}$ is a bottleneck assignment.

3) Proof of Proposition 3: By construction, if task $j \in B$ is in $V_f$, then for all agents $i \in A$ with $i, j \in E_i$, we have $i \not\in V_f$. By contrapositive, if an agent $i \in A$ with $i, j \in E_i$ is not in $V_f$, then task $j \in B$ is not in $V_f$. Similarly, if an agent $i \in A$ is in $V_f$, then $m_i \not\in V_f$, and if $m_i \not\in V_f$, then agent $i \in A$ is not in $V_f$.

Assume for contradiction that there exists $i \in A$ such that there exists an alternating path $P$ between $i$ and $m_{a_1}$, and that $i \not\in V_f$. Let $P := \{v_k, v_k + 1\}$, where $k = 0, 1, \ldots, K$. We assume that $v_k = i$, $v_{K+1} = m_{a_1}$. We have that $v_k \not\in V_f$. From the abovementioned arguments, if $v_k \not\in V_f$, then $v_{K+1} \not\in V_f$. By applying this to all edges in $P$, we have that $v_{K+1} = m_{a_1} \not\in V_f$. This is a contradiction, as $m_{a_1}$ is the root of the tree. Therefore, $i$ must be in $V_f$.

4) Proof of Lemma 2: At every iteration of the while-loop in AugDFS, there exists an alternating path $P$ between the current vertex $t$ and root $\hat{t}$. Consider the set of agents incident to edges in $P$, i.e., $K := \{i \in A|\{i, j\} \in P\}$, and consider function $g: A \rightarrow B$ mapping an agent in the tree to its parent vertex. From Lines 17 and 20, for all agents $i \in K_P$, $i = g(i)$. From Lines 4 and 11, we have that $i \not\in K_P$. Thus, at every iteration of the while-loop we have $\{i, i\} \in A = \{\{i, m_i\} | i \not\in K_P \}$. From Line 25, we get $\{i, i\} \in A, i, j \not\in K_P \}$. Thus, if $P$ is an augmenting path, then the function returns $M_P = M \oplus P$ as required.

On the other hand, if no augmenting path exists, the search terminates at $t = j$, i.e., $P = \emptyset$, $K_P = \emptyset$, and $\{\{i, i\} | i \in A, i, j \not\in K_P \} = \emptyset$.

5) Proof of Theorem 1: From Proposition 2, PruneBAP returns a bottleneck assignment of $G_0$. From Lemmas 1 and 2, the functions MaxEdge() and AugPath() in PruneBAP can be implemented such that they satisfy Assumptions 1 and 2. Edge removal satisfying Assumptions 1 and 2 can be implemented, as shown in Section III-B2.

6) Proof of Proposition 4: In every iteration of the while-loop of PruneBAP, at least one edge in $E_k$ is removed. There are, therefore, at most $|E_k| = mn$ iterations of the while-loop. The while-loop of PruneBAP itself contains two components that depend on time steps. The function MaxEdge() requires at most $D$ time steps for completion. The distributed search for an augmenting path has order $O(nD)$, where each iteration of the while-loop in AugDFS requires $D$ time steps and there are at most $2n - 1$ iterations.

7) Proof of Lemma 3: At every iteration of the while loop in AugDFS, for all $t \in B$, there exists an alternating path $P_t$ between $i$ and $j$. This holds by construction, since an agent $i \in F$ has its matched task $m_i$ inserted into the set $B$ only after $i$ is explored. We alternate between edges in the matching and not in the matching as outlined in Definition 14. If one or more unmatched agents are explored in Line 8, then the search is successful and it remains for one of the augmenting paths to be selected. Following Proposition 3, no augmenting path exists if $S$ is empty. In such a case, all agents $i \in A$ for which there exists an alternating path between $i$ and root $\hat{t}$ have been explored and none of these agents are free vertices.

References

[1] P. G. Gerkey and M. J. Mataric, “A formal analysis and taxonomy of task allocation in multi-robot systems,” Int. J. Robot. Res., vol. 23, pp. 939–954, 2004.
[2] R. E. Burkard, M. Dell’Amico, and S. Martello, Assignment Problems. Philadelphia, PA: SIAM, 2009.
[3] D. W. Penticio, “Assignment problems: A golden anniversary survey,” Eur. J. Oper. Res., vol. 176, pp. 774–793, 2007.
[4] I. Shames, A. Dostovalova, J. Kim, and H. Hmam, “Task allocation and motion control for threat-seduction decoys,” in Proc. IEEE 56th Ann. Conf. Decis. Control, 2017, pp. 4509–4514.
[5] R. S. Garfinkel, “An improved algorithm for the bottleneck assignment problem,” Operations Res., vol. 19, pp. 1747–1751, 1971.
[6] H. N. Gabow and R. E. Tarjan, “Algorithms for two bottleneck optimization problems,” J. Algorithms, vol. 9, pp. 411–417, 1988.
[7] A. P. Punnen and K. P. K. Nair, “Improved complexity bound for the maximum cardinality bottleneck bipartite matching problem,” Discrete Appl. Math., vol. 55, pp. 91–93, 1994.
[8] U. Derigs and U. Zimmermann, “An augmenting path method for solving linear bottleneck assignment problems,” Computing, vol. 19, pp. 285–295, 1978.
[9] S. Chopra, G. Notarstefano, M. Rice, and M. Egerstedt, “A distributed version of the Hungarian method for multirobot assignment,” IEEE Trans. Robot., vol. 33, no. 4, pp. 932–947, Aug. 2017.
[10] H. W. Kuhn, “The Hungarian method for the assignment problem,” Nav. Res. Logistics Quart., vol. 2, pp. 83–97, 1955.
[11] D. P. Bertsekas and D. A. Castaño, “Parallel asynchronous and asynchronous implementations of the auction algorithm,” Parallel Comput., vol. 17, pp. 707–732, 1991.
[12] M. M. Zavlanos, L. Spenstveit, and G. J. Pappas, “A distributed auction algorithm for the assignment problem,” in Proc. IEEE 47th Conf. Decis. Control, 2008, pp. 1212–1217.
[13] M. Bürger, G. Notarstefano, F. Bullo, and F. Allgöwer, “A distributed simplex algorithm for degenerate linear programs and multi-agent assignments,” Automatica, vol. 48, pp. 2298–2304, 2012.
[14] H.-L. Choi, L. Breen, and J. P. How, “Consensus-based decentralized auctions for robust task allocation,” IEEE Trans. Robot., vol. 25, no. 4, pp. 912–926, Aug. 2009.
[15] M. Khoo, T. A. Wood, C. Manzie, and I. Shames, “Distributed algorithm for solving the bottleneck assignment problem,” in *Proc. IEEE 58th Conf. Decis. Control*, 2019, pp. 1850–1855.

[16] C. Godsil and G. F. Royle, *Algebraic Graph Theory*. Berlin, Germany: Springer, 2013.

[17] M. Khoo, T. A. Wood, C. Manzie, and I. Shames, “Exploiting structure in the bottleneck assignment problem,” in *Proc. IFAC World Congr.* 2020, vol. 53, pp. 3310–3315.

[18] A. Tahbaz-Salehi and A. Jadbabaie, “A one-parameter family of distributed consensus algorithms with boundary: From shortest paths to mean hitting times,” in *Proc. IEEE 45th Conf. Decis. Control*, 2006, pp. 4664–4669.

[19] B. M. Nejad, S. A. Attia, and J. Raisch, “Max-consensus in a max-plus algebraic setting: The case of fixed communication topologies,” in *Proc. 22nd Int. Symp. Inf., Commun. Autom. Technol.*, 2009, pp. 1–7.

[20] S. M. LaVelle, *Planning Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2006.

[21] J. E. Hopcroft and R. M. Karp, “An \(n^{5/2}\) algorithm for maximum matchings in bipartite graphs,” *SIAM J. Comput.*, vol. 2, pp. 225–231, 1973.

[22] N. J. A. Harvey, “Matchings, matroids and submodular functions,” Ph.D. thesis, Massachusetts Inst. Technol., Cambridge, MA, USA, 2008.

[23] W. H. Cunningham, “On submodular function minimization,” *Combinatorica*, vol. 5, pp. 185–192, 1985.

[24] A. Chakrabarti and S. Kale, “Submodular maximization meets streaming: Matchings, matroids, and more,” *Math. Program.*, vol. 154, pp. 225–247, 2015.