Exploiting Structure in the Bottleneck Assignment Problem

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

* Department of Electrical and Electronic Engineering at the University of Melbourne, (e-mail: khoom1@student.unimelb.edu.au).
∗∗ (e-mail: {wood.t,manziec,iman.shames}@unimelb.edu.au)

Abstract: An assignment problem arises when there exists a set of tasks that must be allocated to a set of agents. The bottleneck assignment problem (BAP) has the objective of minimising the most costly allocation of a task to an agent. Under certain conditions the structure of the BAP can be exploited such that subgroups of tasks are assigned separately with lower complexity and then merged to form a combined assignment. In particular, we discuss merging the assignments from two separate BAPs and use the solution of the subproblems to bound the solution of the combined problem. We also provide conditions for cases where the solution of the subproblems produces an exact solution to the BAP over the combined problem. We then introduce a particular algorithm for solving the BAP that takes advantage of this insight. The methods are demonstrated in a numerical case study.

Keywords: Algorithms; Decision-making; Graph theory; Agents; Optimization problems.

1. INTRODUCTION

An assignment problem arises when multiple tasks are to be allocated to multiple agents. For example, situations where jobs are to be assigned to a group of workers or passengers positioned at different locations in a city are to be picked up by a fleet of cars. Tasks can be assigned based on many different criteria. See Gerkey and Matarić (2004), Burkard et al. (2009), and Pentico (2007) for reviews on the different objectives for assignment problems.

One particular objective is to assign tasks to agents such that the total cost of the assignment is minimised. This type of assignment problem is called the linear assignment problem (LAP). The Hungarian Method in Kuhn (1955) is a well-studied algorithm for solving the LAP. In Chopra et al. (2017), a distributed version of the Hungarian Method is presented; a distributed algorithm is one that does not rely on a centralised decision-maker for computation. In Bertsekas and Castaño (1991) and Zavlanos et al. (2008), so-called auction algorithms are presented to solve the LAP. A greedy algorithm is one where tasks are allocated to agents sequentially. Each allocation is made according to the lowest cost amongst the remaining choices. In Choi et al. (2009), the Consensus-Based Auction Algorithm (CBAA) is presented, which is a greedy algorithm used to obtain suboptimal solutions to the LAP with low computational cost compared to algorithms for solving the LAP exactly.

Another objective is to assign tasks to agents such that the costliest allocation is minimised, which corresponds to the bottleneck assignment problem (BAP). The BAP has application in time-critical problems. For example in Shames et al. (2017), a set of decoys must travel to a set of positions such that the worst-case positioning time is minimised. In Garfinkel (1971), a threshold algorithm is presented, where a threshold is iteratively increased until it is possible to find an assignment containing only allocations of tasks to agents with costs smaller than the threshold. In Gabow and Tarjan (1988); Punnen and Nair (1994), the bound on the completion time of the threshold algorithm is reduced moving the threshold according to a binary search pattern. In Derigs and Zimmermann (1978), an algorithm is presented that iteratively solves the BAP over an increasing subset of agents and tasks. The subset size is increased until it contains all the agents and tasks. In Khoo et al. (2019), a distributed algorithm for solving the BAP is introduced. There are other variants of the BAP. Such variants include the scheduling problems in Carraresi and Gallo (1984) and Aggarwal et al. (1986), which require assigning more than one task per agent. In fact, this can be regarded as an example of a time-extended assignment from the taxonomy in Gerkey and Matarić (2004).

In this paper, we focus on the BAP and restrict the scope to having each agent carry out at most one task and each task requiring at most one agent for completion. The contribution of this work is to investigate structure that can be exploited to solve the BAP efficiently. Consider partitioning the sets of agents and tasks, i.e., splitting the assignment problem into two smaller BAPs. We can use the two solutions of the subproblems for solving the combined BAP. Consider the following three ways to exploit the structure of the BAP. We relate each scenario to a ride-sharing application for illustration.

For the first scenario, assume the sets of agents and tasks were partitioned equitably, i.e., none of the subproblems has fewer tasks than agents. Merging the solutions of the

* The research is funded by Defence Science and Technology Group through research agreements MyIP: 7558 and MyIP: 7562.
two subproblems forms a valid but possibly suboptimal assignment in the combined problem. In fact, the cost of the merged assignment is an upper bound on the cost of the optimal BAP solution. In a ride-sharing application, two rival companies may assign their own vehicles to their own customers. However, they may find that pooling their resources allows a better service for all customers.

For the second scenario, we define a bottleneck cluster as a group of agents and tasks with small allocation costs amongst each other. When the two subproblems consist of two separate bottleneck clusters, we can determine conditions under which the solutions of the subproblems form an exact solution to the combined problem. Consider two cities each with their own sets of vehicles and customers. If the cities were geographically far apart, there is no benefit for vehicles in one city to serve customers in the other city.

The third scenario relates to the algorithm in Khoo et al. (2019). Knowing the solutions to the two subproblems leads to information about task-to-agents allocations that are particularly costly. We can eliminate suboptimal options when the algorithm is initialised to solve the combined problem. Assume a group of customers has been assigned to vehicles. Then, a new group of customers requests to be picked up. By only considering the idle vehicles for the new customers and not the previously assigned ones, the resulting assignment problem has lower complexity. The solution from the two subproblems can be used as a warm-start to solving the combined problem.

2. PRELIMINARIES

Given an arbitrary directed graph $G = (V, E)$ with vertex set $V$ and edge set $E$, consider the following definitions found in Hopcroft and Karp (1973) and Khoo et al. (2019).

Definition 1. (Maximum Cardinality Matching). A matching $M$ in $G$ is a set of edges such that $M \subseteq E$ and no vertex $v \in V$ is incident with more than one edge in $M$. A maximum cardinality matching (MCM) is a matching $M_{\text{max}}$ in $G$ of maximum cardinality.

Let $A_b$ be a set of agents and $B_b$ be a set of tasks, where $A_b \cap B_b = \emptyset$. Consider an arbitrary complete bipartite graph $G_b = (V_b, E_b)$ with vertex set $V_b = A_b \cup B_b$ and edge set $E_b = \{(i, j) | i \in A_b, j \in B_b\}$. Let $C(G_b)$ be the set of all MCMs of $G_b$. Let $w : E_b \mapsto \mathbb{R}$ map edges to real-valued weights. The BAP for graph $G_b$ is formulated as

$$\text{BOT}(G_b) : \min_{M \in C(G_b)} \max_{(i,j) \in M} w((i,j)).$$

Definition 2. (Bottleneck edge). A bottleneck edge of graph $G_b$ is any $e \in \arg \max_{(i,j) \in M} w((i,j))$, for any MCM $M$ that is a minimiser of $\text{BOT}(G_b)$.

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

Note that given a vertex $v \in V, \forall k \in N_v, v \notin N_k$.

Definition 4. (Path). Let a sequence of distinct vertices $v_1, v_2, ..., v_k+1 \in V$ be such that for $k = 1, 2, ..., l, v_{k+1} \in N_{v_k}$. The set of edges $P = \{(v_k, v_{k+1}) | k=1,2, ..., l\}$ is then said to be a path between $v_1$ and $v_{l+1}$, with length $l$.

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

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

Definition 7. (Augmenting path). Given a matching $M$ and a path $P$ between vertices $v$ and $v'$. $P$ is an augmenting path relative to $M$ if and only if $P$ is an alternating path relative to $M$ and $v$ and $v'$ are both free vertices.

Definition 8. (Alternating tree) Given a matching $M$, $G$ is an alternating tree relative to $M$ if and only if $G$ is a tree and any path between the root vertex of $G$ and every other vertex in $G$ is an alternating path relative to $M$.

3. PROBLEM FORMULATION

Let there be two sets of agents $A_1 = \{a_1, a_2, ..., a_{m_1}\}$ and $A_2 = \{a_1, a_2, ..., a_{m_2}\}$ and two sets of tasks $B_1 = \{b_1, b_2, ..., b_{n_1}\}$ and $B_2 = \{b_1, b_2, ..., b_{n_2}\}$. Define the sets $A_3 := A_1 \cup A_2$ and $B_3 := B_1 \cup B_2$. Let $m_3 = m_1 + m_2$ and $n_3 = n_1 + n_2$ and assume $m_1 \geq n_1$ and $m_2 \geq n_2$.

For $k = 1, 2, 3$, define $\mathcal{V}_k := A_k \cup B_k, \mathcal{E}_k := \{(i, j) | i \in A_k, j \in B_k\}$ and graph $G_k := (\mathcal{V}_k, \mathcal{E}_k)$. Define $\text{BOT}(G_k) := \arg \max_{M \in C(G_k)} \max_{(i,j) \in M} w((i,j))$, the set of solutions to $\text{BOT}(G_k)$ for any bipartite graph $G_k$.

Assumption 1. Assume we have $M_1 \in \mathcal{D}(G_1)$ and $e_1 \in \arg \max_{(i,j) \in M_1} w((i,j))$, i.e., an arbitrary solution to $\text{BOT}(G_1)$ and a corresponding bottleneck edge of $G_1$.

Assumption 2. Assume we have $M_2 \in \mathcal{D}(G_2)$ and $e_2 \in \arg \max_{(i,j) \in M_2} w((i,j))$, i.e., an arbitrary solution to $\text{BOT}(G_2)$ and a corresponding bottleneck edge of $G_2$.

Problem 1. Given Assumptions 1 and 2, find a solution to $\text{BOT}(G_3)$, i.e., find some matching $M_3 \in \mathcal{D}(G_3)$.

In Section 4, we define structures of the BAP that can be exploited to solve Problem 1. Then in Section 5, we discuss a specific algorithm that allows us to exploit some structure of the BAP discussed in Section 4.

4. STRUCTURE OF THE BAP

In this section, we discuss structures of the BAP that can be exploited. We first introduce an upper bound on the weight of a bottleneck edge of $G_3$, in terms of the bottleneck edges $G_1$ and $G_2$. Then, we introduce bottleneck clusters and provide conditions when the solution to Problem 1 is found by merging matchings $M_1$ and $M_2$.
solution is sufficient in our application, there is no need to invest further resources to solve Problem 1 exactly.

4.2 Bottleneck Clusters

We now introduce the novel concept of a bottleneck cluster. In Khoo et al. (2019), conditions for determining if an edge is a bottleneck edge of a given graph are presented. We build on this result and discuss corresponding conditions under which $M_2 = M_1 \cup M_2$ is an exact solution to $BOT(G_3)$ when $G_1$ and $G_2$ are both bottleneck clusters.

Once again, consider an arbitrary complete bipartite graph $G$. Let $\phi(G) = (\nu, \mu)$ denote $\phi(G) = (\nu, \mu)$. Let $\nu_0$ contain the bottleneck agent $a_c$, and let $\nu_v$ contain the bottleneck task $b_c$. Let $\nu_b = \nu_v \cup \nu_a$ and $\nu_v \cap \nu_a = \emptyset$. By Definition 9, it must be possible to construct both $S_\nu(G)$ and $S_\nu(G)$ to be alternating trees such that $\nu_\nu \cup \nu_\mu \subseteq \phi(G, M_\nu)$. By Proposition 1, for all agents $a' \in \nu_v \cup \nu_\mu$ and for all tasks $b' \in \nu_v \cup \nu_b$, $\{a', b\} \notin \phi(G, M_\nu)$.

Corollary 2. Consider Assumption 3 and let $G_3$ be a bottleneck cluster with respect to the critical bottleneck edge $e_c$. We form two subgraphs of $G_3$, denoted as $S_\nu(G_3) = (\nu_v, \mu)$ and $S_\nu(G_3) = (\nu_v, \mu)$. Let $\nu_a$ contain the bottleneck agent $a_c$, and let $\nu_v$ contain the bottleneck task $b_c$. Let $\nu_b = \nu_v \cup \nu_a$ and $\nu_v \cap \nu_a = \emptyset$. By Definition 9, it must be possible to construct both $S_\nu(G)$ and $S_\nu(G)$ to be alternating trees such that $\nu_\nu \cup \nu_\mu \subseteq \phi(G_3, M_3)$. By Definition 1, for all agents $a' \in \nu_v \cup \nu_\mu$ and for all tasks $b' \in \nu_v \cup \nu_b$, $\{a', b\} \notin \phi(G_3, M_3)$.

Fig. 1. A bottleneck cluster. Dotted lines represent edges not in matching $M_\nu$, solid lines represent edges in $M_\nu$. Shown here, a set of agents $\{a_1, a_2, ..., a_9\}$ and a set of tasks $\{b_1, b_2, ..., b_9\}$. Edge $\{a_1, b_1\}$ is a critical bottleneck edge so Corollary 2 applies.
In general, the converse of Lemma 2 does not hold unless we apply some additional assumptions. This leads to the following theorem.

**Theorem 2.** Given Assumptions 1 and 2, assume both $G_1$ and $G_2$ are bottleneck clusters with respect to $e_1$ and $e_2$ respectively. Assume $e_1$ is a critical bottleneck edge of $G_1$ relative to $M_1$ and $e_2$ is a critical bottleneck edge of $G_2$ relative to $M_2$. Assume that $w(e_1) > w(e_2)$. Let $\arg\max_{(i,j)\in M_2} w((i,j))$ be a singleton. It holds that $w(e_3) < w(e_1)$ if and only if there exists vertices $i, j \in V_2$ such that conditions i., ii., and iii. from Lemma 2 are true.

**Proof.** The necessary condition for $w(e_3) < w(e_1)$ holds from Lemma 2. We now prove the sufficient condition. Assume there exist vertices $i$ and $j$ such that all i., ii., and iii. are true. Then, aside from path $P = \{e_1 = \{a_1, b_1\}\}$, there exists another alternating path $P'$ between $a_1$ and $b_1$, which does not contain the edge $e_1$. Namely, the alternating path $P'$ constructed from the union of the alternating paths between $a_1$ and $b'$ and $i$ and $j$ and $a'$ and $b'$. Thus, there exists an augmenting path $P'' \subseteq \phi(G_3, M_1 \cup M_2) \setminus \{e_1\}$ relative to $(M_1 \cup M_2) \setminus \{e_1\}$. From Lemma 1, there exists an MCM $\mathcal{M}'$ of $G_3$ such that $\mathcal{M}' \in \phi(G_3, M_1 \cup M_2) \setminus \{e_1\}$. By the assumptions on $e_1$, $\phi(G_3, M_1 \cup M_2) \setminus \{e_1\}$ contains only edges with weights strictly smaller than $w(e_1)$. Thus, there exists an MCM of $G_3$ with all edges having weight smaller than $w(e_1)$, i.e., $w(e_3)$ must be smaller than $w(e_1)$.

---

**Problem 1.** Let us refer to this algorithm as pruneBAP. Fig. 3 is an illustration of this algorithm.

**Iteration 1:**

$e_24 \ e_42 \ e_34 \ e_32 \ e_21 \ e_12 \ e_13 \ e_23 \ e_25 \ e_35 \ e_33 \ e_31 \ e_41 \ e_42 \ e_43 \ e_44$

**Iteration 2:**

$e_24 \ e_42 \ e_34 \ e_32 \ e_43 \ e_41 \ e_42 \ e_43 \ e_44$

**Iteration 3:**

$e_24 \ e_42 \ e_43 \ e_44$

**Fig. 3.** A demonstration of pruneBAP with $A_3 = \{a_1, a_2, a_3, a_4\}$ and $B_3 = \{b_1, b_2, b_3, b_4\}$. Edges in $E_3$ are arranged in order of ascending weight, where $e_{pq}$ is the edge between agent $a_p$ and task $b_q$. At iteration 1, the initial arbitrary MCM is denoted by the 4 circled edges. Edges to the right of the dashed lines have been pruned from $E_3$. Note, $w(e_{11}) \geq w(e_{21})$, i.e., with each iteration the weight of the largest edge in the current MCM is non-increasing. The algorithm terminates when a matching of size 4 does not exist in the remaining edges to the left of the dashed line.

---

### 5.1 Warm-starting Versus Cold-starting pruneBAP

Solving Problem 1 by pruneBAP with an arbitrary MCM $M_0$ at initialisation does not make use of Assumptions 1 and 2. We denote this as a cold-start to pruneBAP. Given Assumptions 1 and 2, consider the following. It holds that the set $M := M_1 \cup M_2$ is an MCM of the graph $G_3$. Without loss of generality, let $w(e_1) \geq w(e_2)$. Then, it also holds that $e_1$ is the largest edge in $M$. We make use of $M$ to solve Problem 1 by choosing it as the initial MCM of pruneBAP. Edges in the set $\{e \in E_3 | w(e) \geq w(e_1), e \notin M_0\}$ are removed from $E_3$ in the first iteration. We denote this as a warm-start to pruneBAP. Fig. 4 illustrates a warm-start to pruneBAP.

**Fig. 4.** A demonstration of pruneBAP with $A_3 = \{a_1, a_2, a_3, a_4\}$ and $B_3 = \{b_1, b_2, b_3, b_4\}$. Edges in $E_3$ are arranged in order of ascending weight, where $e_{pq}$ is the edge between agent $a_p$ and task $b_q$. At iteration 1, the initial arbitrary MCM is denoted by the 4 circled edges. Edges to the right of the dashed lines have been pruned from $E_3$. Note, $w(e_{11}) \geq w(e_{21})$. Edges in the set $\{e \in E_3 | w(e) \geq w(e_1), e \notin M_0\}$ are removed from $E_3$ in the first iteration. We denote this as a warm-start to pruneBAP. Fig. 4 illustrates a warm-start to pruneBAP.

---

### 5. ALGORITHM FOR SOLVING THE BAP

In this section, we discuss how the algorithm from Khoo et al. (2019) makes use of Assumptions 1 and 2 to solve Problem 1.

---

### 6. CASE STUDIES

Consider agents and tasks represented by points in a vector space $S$ with a distance function $D : S \times S \mapsto \mathbb{R}^+$. 
For example, this could be a ride-sharing application, where agents are vehicles and their tasks are to pick up customers. Here we consider a 2-dimensional space $S = \mathbb{R}^2$ and Euclidean distance $D(x, y) = \|x - y\|_2$. Agents are to be assigned to move from their initial positions to target destinations based on the BAP with distance as weights.

6.1 Task Reassignment

Case Study 1. Let $A = \{a_1, a_2, ..., a_{n_1}\} \subset S$ be the initial locations of a set of agents. Let $B_1 = \{b_1, b_2, ..., b_{n_2}\} \subset S$ be the set of goal locations. Assume $m_3 > n_1$. We first solve $BOT((A \cup B_1, \mathcal{E}))$, where $\mathcal{E} = \{(i, j) | i \in A, j \in B_1\}$ to determine an assignment of tasks to agents that minimises the worst-case distance an agent must travel to reach a goal location. Without loss of generality, assume vehicles at positions $A_1 = \{a_1, a_2, ..., a_{n_1}\}$ are assigned to goals at $B_1$. Now assume that a second set of goal locations becomes available to agents. Let $B_2 = \{\beta_1, \beta_2, ..., \beta_{n_2}\} \subset S$ be the set of new goal locations. Assume that $m_3 \geq n_1 + n_2$. Let $A_2 = \{a_{n_1+1}, a_{n_1+2}, ..., a_{n_1+n_2}\}$ be the locations of the remaining unassigned agents. We now assign the new goals to the remaining agents, i.e., solve $BOT((A_2 \cup B_2, \mathcal{E}_2))$, where $\mathcal{E}_2 = \{(i, j) | i \in A_2, j \in B_2\}$.

By Theorem 1, the assignment obtained from solving $BOT((A \cup B_1, \mathcal{E}))$ and $BOT((A_2 \cup B_2, \mathcal{E}_2))$ in Case Study 1 is not necessarily the optimal solution to $BOT((A \cup B_1 \cup B_2, \mathcal{E}_3))$, where $\mathcal{E}_3 = \{(i, j) | i \in A, j \in B_1 \cup B_2\}$. Fig. 5 shows a numerical example of a case where the optimal assignment is of lower cost than the assignment used to warm-start pruneBAP. For this example, $m_3 = 40$, $n_1 = 20$ and $n_2 = 20$. The data was generated using continuous uniform distributions with range $[0, 100]$ for both coordinates $x$ and $y$. Fig. 6 shows a plot of the average cost of the assignment used as warm-start to initialise pruneBAP and the average cost of the optimal assignment after pruneBAP has terminated. For all simulations, $m_3 = n_1 + n_2$. For each even value of $m_3$, 100 simulations were generated. We observe that the cost of the assignment obtained from the subproblems is never greater than the cost of an optimal solution to $BOT(G_1)$, in accordance with Theorem 1. In this case, the unstructured distribution of the locations results in all of the conditions in Theorem 2 being satisfied and we observe that $w(e_3) < \max\{w(e_1), w(e_2)\}$ as expected.

6.2 Clustering of Agents and Tasks

Case Study 2. Let $A_1 = \{a_1, a_2, ..., a_{m_1}\} \subset S$ and $A_2 = \{a_{m_1+1}, a_{m_1+2}, ..., a_{m_2}\} \subset S$ be the initial locations of two sets of agents. Let $B_1 = \{b_1, b_2, ..., b_{m_2}\} \subset S$ and $B_2 = \{\beta_1, \beta_2, ..., \beta_{m_2}\} \subset S$ be the set of goal locations. Assume that $m_1 \geq n_1$ and $m_2 \geq n_2$, i.e., there are more agents than there are goals. Assume the set of locations $A_1$ and $B_1$ are separated geographically from $A_2$ and $B_2$.

In Case Study 2, we illustrate an example where not all of the conditions i., ii., and iii. in Lemma 2 hold. Fig. 7 shows a numerical example where the initial assignment $M_1 \cup M_2$ used to warm-start pruneBAP is in fact the optimal assignment of $B_1 \cup B_2$ to $A_1 \cup A_2$. In this example, $m_1 = 20$, $m_2 = 20$, $n_1 = 20$ and $n_2 = 20$. The data in Fig. 7 was generated using independent normal distributions with a variance of 100 for each distribution. The distributions for sets $A_1$ and $B_1$ are centred at the point $(x, y) = (40, 60)$. The distributions for sets $A_2$ and $B_2$ are centred at the point $(x, y) = (60, 40)$. Fig. 8 shows the number of instances out of 100 simulations for which the behaviour in Fig. 7 is observed. That is, the instances where the bottleneck edges obtained from the subproblems directly results in an optimal solution to $BOT(G_3)$, where $G_3$ is defined as in Section 3. The number of agents equals the number of tasks for each simulation, i.e., $m_1 = n_1 = m_2 = n_2$. For each simulation, positions were generated using the same normal distribution as in Fig. 7. We now observe realisations where the cost of the assignment obtained from the subproblems is equal to the cost of an optimal solution to $BOT(G_3)$. This illustrates that with this distribution of agents and tasks there are instances where there is structure such that the conditions in Theorem 2 do not all hold and $w(e_3) = \max\{w(e_1), w(e_2)\}$.

7. CONCLUSION

We discussed properties of pruneBAP that allow us to warm-start the algorithm given BAP solutions to divided
sets of tasks and agents. The solutions based on the divided problems forms an MCM of the combined problem. The pruneBAP algorithm can be initialised with any MCM, and thus allows us to make use of the solutions based on the divided sets. We then have an upper bound on the BAP solution to the combined problem in terms of the bottleneck edges of the divided problems. We also introduced the novel concept of a bottleneck cluster relative to a bottleneck edge. This idea is inspired by the pruneBAP algorithm and the alternating tree that is obtained as a result of the algorithm. Using bottleneck clusters, we provided conditions such that the initial MCM used to warm-start pruneBAP is a solution to the BAP. From numerical simulations motivated by ride-sharing, we illustrate an example where the conditions hold if there exist clusters that are separated in space.

An interesting future direction is the investigation of methods to optimally partition agents and tasks. Another direction would be to investigate clustering properties for the LAP.

REFERENCES

Aggarwal, V., Tikekar, V.A., and Hsu, L.F. (1986). Bottleneck assignment problems under categorization. *Computers and Operations Research*, 13, 11–26.

Bertsekas, D.P. and Castaño, D.A. (1991). Parallel synchronous and asynchronous implementations of the auction algorithm. *Parallel Computing*, 17, 707–732.

Burlard, R.E., Dell’Amico, M., and Martello, S. (2009). *Assignment problems*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia.

Carraresi, P. and Gallo, G. (1984). A multi-level bottleneck assignment approach to the bus drivers’ rostering problem. *European Journal of Operational Research*, 16, 163–173.

Choi, H.L., Brenet, L., and How, J.P. (2009). Consensus-based decentralized auctions for robust task allocation. *IEEE Transactions on Robotics*, 25, 912–926.

Chopra, S., Notarstefano, G., Rice, M., and Egerstedt, M. (2017). A distributed version of the hungarian method for multi-robot assignment. *IEEE Transactions on Robotics*, 33, 932–947.

Derigs, U. and Zimmermann, U. (1978). An augmenting path method for solving linear bottleneck assignment problems. *Computing*, 19, 285–295.

Gabow, H.N. and Tarjan, R.E. (1988). Algorithms for two bottleneck optimization problems. *Journal of Algorithms*, 9, 411–417.

Garfinkel, R.S. (1971). An improved algorithm for the bottleneck assignment problem. *Operations Research*, 19, 1747–1751.

Gerkey, B.P. and Matarić, M.J. (2004). A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, 23, 939–954.

Hopcroft, J.E. and Karp, R.M. (1973). An \( n^{5/2} \) algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2, 225–231.

Khoo, M., Wood, T.A., Manzie, C., and Shames, I. (2019). Distributed algorithm for solving the bottleneck assignment problem. *IEEE 58th Conference on Decision and Control*, 1850–1855.

Kuhn, H.W. (1955). The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2, 83–97.

Pentico, D.W. (2007). Assignment problems: A golden anniversary survey. *European Journal of Operational Research*, 176, 774–793.

Punnen, A.P. and Nair, K.P.K. (1994). Improved complexity bound for the maximum cardinality bottleneck bipartite matching problem. *Discrete Applied Mathematics*, 55, 91–93.

Shames, I., Dostoalova, A., Kim, J., and Hmam, H. (2017). Task allocation and motion control for threat-seduction decoys. *IEEE 56th Annual Conference on Decision and Control*, 4509–4514.

Zavlanos, M.M., Spesivtsev, L., and Pappas, G.J. (2008). A distributed auction algorithm for the assignment problem. *IEEE Conference on Decision and Control*, 1212–1217.