Swarm Intelligence (2022) 16:233–260
https://doi.org/10.1007/s11721-022-00213-0

Sample greedy based task allocation for multiple robot systems

Hyo-Sang Shin[1] · **Teng Li**[1] · **Hae-In Lee**[1] · **Antonios Tsourdos**[1]

Received: 4 March 2021 / Accepted: 18 July 2022 / Published online: 13 August 2022

Abstract
This paper addresses in-schedule dependent task allocation problems for multi-robot systems. One of the main issues with those problems is the inherent NP-hardness of combinatorial optimisation. To handle this issue, this paper develops a decentralised task allocation algorithm by leveraging the submodularity concept and a sampling process of task sets. Our theoretical analysis reveals that the proposed algorithm can provide an approximation guarantee of 1/2 of the optimal solution for the monotone submodular case and 1/4 for the non-monotone submodular case, both with polynomial time complexity. To examine the performance of the proposed algorithm and validate the theoretical analysis, we introduce two task allocation scenarios and perform numerical simulations. The simulation results confirm that the proposed algorithm achieves a solution quality which is comparable to state-of-the-art algorithms in the monotone case and much better quality in the non-monotone case with significantly lower computational complexity.

Keywords Task allocation · Multi-robot system · Approximation guarantee · Submodularity · Sampling greedy

1 Introduction

This paper addresses ID [ST, SR, TA] multi-robot task allocation (MRTA) problems, according to the taxonomy of MRTA problems defined by Gerkey and Matarić (2004) and Korsah et al. (2013). In-schedule dependent (ID) problems mean that the utility, or gain that is obtained by performing some tasks, of each robot depends on a robot's own schedule and the other tasks that the robot has already performed. The problem is to find the time-extended assignment (TA) of a set of single-robot (SR) tasks to a group of single-task (ST) robots. Each robot can perform one task at a time, and one task cannot be performed by multiple robots. The well-known travelling sales man problem and many multi-robot coordination problems also belong to this category of MRTA. The issue is that the MRTA

✉ Hyo-Sang Shin
h.shin@cranfield.ac.uk

[1] School of Aerospace, Transport and Manufacturing, Cranfield University, Cranfield MK43 0AL, UK

problems in this category are known to be NP-hard, requiring extensive computational cost to find the exact solution.

The task allocation algorithm under consideration in this paper exploits an approximation approach, and the algorithm is fully decentralised. Approximation approaches can balance the optimality of a solution and computational cost by approximating the MRTA problem. Moreover, they typically provide a mathematical guarantee on the quality of solution and computational complexity if the problem satisfies certain conditions. Note that the centralised solution of the task allocation problem involves having to communicate all the robots and environment data to a centralised entity. This may not be possible in some realistic scenarios because relying on a central entity could remove resilience, or the bandwidth to communicate all the information may not be available. Employing a decentralised algorithm can improve resilience and relieve communication requirements compared with a centralised algorithm, and thus enabling decentralisation could be more practical for multi-robot operations.

Decentralised approximation approaches have been frequently applied to solve MRTA problems. Choi et al. presented the Consensus-Based Bundle Auction (CBBA) algorithm, which is the first decentralised approximate algorithm that provides a solution guaranteed to be within a constant factor of the optimal (Choi et al., 2009). By assuming that the utility functions of the robots are monotone non-decreasing and submodular, their algorithm is proven to provide a solution with a guarantee of at least 50% of the value of the optimal solution.

Submodularity is a property commonly required in approximate approaches to obtain a mathematical guarantee. The marginal gain that a robot obtains by executing an extra task diminishes as the number of tasks that need to be carried out by the robot increases. Williams et al. (2017) investigated a surveillance mission in an urban environment applying decentralised Sequential Greedy Algorithm (SGA) considering multiple matroid constraints. Sun et al. (2019) solved a target covering problem using distributed SGA in an environment with obstacles and provided a tighter optimality bound by analysing the curvature of submodularity. To improve the efficiency, Qu et al. (2015), Qu et al. (2019) developed a Distributed Greedy Algorithm (DGA) for a large group of Earth-observing satellites to automatically assign locations based on local information and communication. DGA introduces the concept of *admissible task set*, which is valid for spatially stationary robots, but difficult to apply to mobile robots. Decentralised approximation approaches were also used in search and localisation (Ding & Castanón, 2017), and sensor networks (Kumar et al., 2017; Corah & Michael, 2018) to efficiently solve MRTA problems.

The issue with the aforementioned algorithms is that they can provide approximation guarantees only for monotone submodular objective functions: they cannot provide any approximation guarantee for maximising non-monotone objective functions. The objective functions are not necessarily monotone, e.g., if concentrating too many tasks to one robot is not beneficial due to battery constraints, then the objective functions could become non-monotone.

This paper aims to develop a decentralised task allocation algorithm that can be implemented in practice for large-scale multi-robot systems (MRS) based on submodular maximisation. The focus is to provide guarantees on optimality not only for monotone submodular functions, but also for non-monotone ones with low computational complexity. The main contribution of our work is summarised in the following:

This paper introduces a sampling process of task sets to the main concepts of SGA which sequentially allocates tasks in a greedy manner considering all tasks available.

The proposed algorithm is named DSTA (Decentralised Sample-based Task Allocation). In DSTA, each robot randomly selects tasks with a uniform probability of $p \in (0, 1]$ from the task set. Then, DSTA runs decentralised SGA based on the task samples randomly selected. The DSTA algorithm offers three advantages. First, the algorithm achieves approximation guarantees in expectation for both monotone and non-monotone submodular functions. Second, the computational complexity can be reduced as the algorithm considers only sampled tasks, not all tasks in the greedy selection phase. Third, DSTA is able to balance the approximation guarantee and computational complexity by adjusting the sampling probability p.

To examine the advantages of the proposed algorithm, we conduct a theoretical analysis. The analysis focuses on the expected approximation guarantee and variance of the solution and the computational complexity. To the best of our knowledge, this is the first attempt to theoretically investigate the variance of the solutions in any randomised SGAs.

We consider two simple surveillance mission scenarios for the validation of the proposed algorithm through Monte Carlo simulations. Using simulations, we investigate the performance of the proposed algorithm, i.e., optimality and variance of the solution and computational complexity. The simulation results confirm the theoretical analysis. For rigorous validation, the performance of the proposed algorithm is compared with that of state-of-the-art algorithms such as Genetic Algorithm (GA) (Kotwal & Dhope, 2015), CBBA (Choi et al., 2009), and DGA (Qu et al., 2019).

The rest of this paper is organised as follows. Section 2 presents preliminaries and backgrounds. Section 3 describes our task allocation algorithm, and its analysis is detailed in Sect. 4. The performance and validity of the analysis are investigated through numerical simulations in Sect. 5. Section 6 offers conclusions and discusses future research directions.

2 Preliminaries and backgrounds

Following the general definition of the task allocation problems from (Dias et al., 2006), the particular instance of the task allocation problems under consideration is defined as follows:

Definition 1 (Task allocation) Given a set of tasks $\mathcal{T}$, a set of robots $\mathcal{A}$, and a utility function $f_a : 2^{\mathcal{T}} \to \mathbb{R}^+$ for each robot $a \in \mathcal{A}$, find a non-overlapping allocation, $S \in \mathcal{A} \times \mathcal{T}$, that maximises an objective function $f : 2^{\mathcal{A}\times\mathcal{T}} \to \mathbb{R}^+$ defined as $f(S) = \sum_{a\in\mathcal{A}} f(S_a)$ where $f(S_a) := f_a(\mathcal{T}_a)$ and S_a denotes the allocation subset for Robot a. Here, $S = \cup_{a\in\mathcal{A}} S_a$ and $\mathcal{T}_a$ is the task subset allocated to Robot a.

The task-robot pair for Robot a and Task j is denoted as $u_{aj} := (a, j)$. Then, S_a can be expressed as $S_a = \{u_{aj} : \forall j \in \mathcal{T}_a\}$. In the rest of this paper, we use the term "pair" to refer to "task-robot pair" for simplicity. The non-overlapping allocation implies that one task cannot be allocated to more than one robot, i.e., the sets $\{\mathcal{T}_a\}_{a\in\mathcal{A}}$ are disjoint, but one robot can take more than one task.

Definition 2 (Marginal gain value (*mgv*) (Krause & Golovin, 2014)) For a set function $f : 2^{\mathcal{N}} \rightarrow \mathbb{R}$, $S \subset \mathcal{N}$ and $u \in \mathcal{N}$, the *marginal gain value (mgv)* of f at S with respect to u is defined as

$$\Delta f(u|S) := f(S \cup \{u\}) - f(S). \tag{1}$$

This paper assumes that the utility function for each robot is submodular. The definition of submodularity follows.

Definition 3 (Submodularity (Krause & Golovin, 2014)) Let $\mathcal{N}$ be a finite set. A real-valued set function $f : 2^{\mathcal{N}} \rightarrow \mathbb{R}$ is *submodular* if, for all $X, Y \subseteq \mathcal{N}$,

$$f(X) + f(Y) \geq f(X \cap Y) + f(X \cup Y). \tag{2}$$

Equivalently, $\forall\, A \subseteq B \subset \mathcal{N}$ and $u \in \mathcal{N} \backslash B$,

$$\Delta f(u|A) \geq \Delta f(u|B). \tag{3}$$

We use " \ " as the subtraction operation of two sets. The submodular functions considered are normalised (i.e., $f(\emptyset) = 0$) and non-negative (i.e., $f(S) \geq 0$ for all $S \subseteq \mathcal{N}$).

Submodularity is an intuitive notion, meaning that the *mgv* that a robot obtains by executing an extra task diminishes, as the number of tasks carried out by the robot increases. We believe that submodularity is a good modelling tool in designing utility functions for task allocation problems. In the machine learning community, there has been a great effort spearheading this idea: finding suitable submodular models to solve inherently discrete tasks, such as summarising documents, scene segmentation, or pattern discovery (Song et al., 2014; Mirzasoleiman et al., 2016). Task allocation is an inherently discrete problem, and thus "submodularising" task allocation problems should yield useful applications of efficient MRS cooperation.

Equation 3 is known as *diminishing returns*, which is one of the core properties of submodular functions (Krause & Golovin, 2014). Diminishing returns mean that the *mgv* of a given element u will never increase as more elements have already been added into the set S. The range of applications holding this property is wide (Choi et al., 2009), e.g., a surveillance mission in which the available time for a robot to monitor an additional point decreases as a given robot is assigned more points to monitor.

Definition 4 (Monotonicity (Krause & Golovin, 2014)) A set function $f : 2^{\mathcal{N}} \rightarrow \mathbb{R}$ is *monotone* if, $\forall\, A \subseteq B \subseteq \mathcal{N}$, $f(A) \leq f(B)$.

Developing an algorithm that works with non-monotone submodular utility functions could be of significant importance since non-monotonicity is a feature that arises naturally in many practical scenarios. For example, in a multi-robot surveillance mission, if a robot is assigned too many targets to track, it might end up spending its time travelling between targets and not gathering enough useful information at the targets' locations. Therefore, adding tasks to a robot's assignment could reduce the final utility. Monotone submodular functions are structurally ill-suited to model such a scenario since, by definition, they do not contemplate reduction in the utility. Therefore, this paper considers both monotone and non-monotone submodular functions.

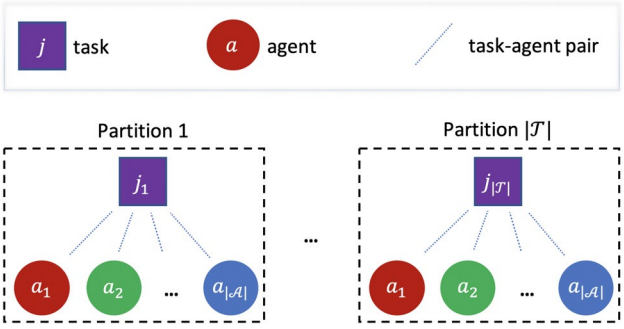


Fig. 1 Demonstration of task-robot pairs grouped by different partitions

The non-overlapping constraint in Definition 1 can be classified as a partition matroid constraint. The definition of *matroid* is given next.

Definition 5 (Matroid (Badanidiyuru & Vondrák, 2014)) A matroid $\mathcal{M}$ is a pair of $\mathcal{N}$ and $\mathcal{I}$, i.e., $\mathcal{M} = (\mathcal{N}, \mathcal{I})$ where $\mathcal{N}$ is a finite set, called *ground set*, and $\mathcal{I} \subseteq 2^{\mathcal{N}}$ is a collection of independent sets, satisfying:

- $\emptyset \in \mathcal{I}$
- $A \subseteq B, B \in \mathcal{I} \Rightarrow A \in \mathcal{I}$
- $A, B \in \mathcal{I}, |A| < |B| \Rightarrow \exists\, b \in B \backslash A$ such that $A \cup \{b\} \in \mathcal{I}$.

where $|\cdot|$ is the cardinality operator.

Representative examples of matroid constraints are uniform and partition matroid constraints. For the uniform matroid constraint, which is also called *cardinality constraint*, any subset $S \subseteq \mathcal{N}$ satisfying $|S| \leq k$ is independent, i.e., $S \in \mathcal{I}$. The *partition matroid constraint* means that an independent subset S can contain at most a certain number of elements from each partition of $\mathcal{N}$.

For the task allocation problem under consideration, we define the ground set as $\mathcal{N} := \{u_{aj} : \forall a \in \mathcal{A}, \forall j \in \mathcal{T}\}$, i.e., the set of all pairs, and define $\mathcal{I}$ as the collection of all possible non-overlapping allocation solutions. Then the ground set $\mathcal{N}$ can be divided into $|\mathcal{T}|$ partitions with respect to the tasks $j \in \mathcal{T}$, as shown in Fig. 1. According to the non-overlapping constraint in Definition 1, a valid allocation solution S can contain at most one pair from each partition. For example, if we have already selected the pair $u_{a_1 j_1}$ from Partition 1 which contains available pairs corresponding to Task j_1, then we cannot select any other task-robot pairs from Partition 1. This implies that the non-overlapping constraint in our task allocation problem is a partition matroid constraint.

This paper considers a bidirectional communication graph in which any robot can communicate with its neighbouring robots. The definition of neighbouring robot is provided in Definition 6. For ease of analysis, we assume that the communication network is strongly connected and all communication links are stable. Under the assumption, it is proven that the convergence of the max-consensus protocol is guaranteed (Giannini et al., 2016). The communication error and the specific communication techniques are out of the scope of this work.

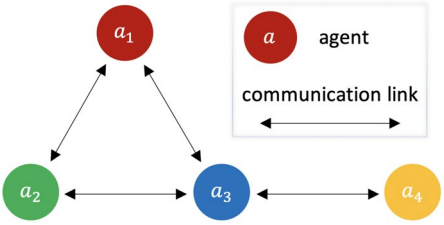


Fig. 2 Illustration of a communication graph with bidirectional links

Definition 6 (Neighbouring robots (Macal & North, 2010; Shin et al., 2020)) For Robot a in a communication network, the robots that have a direct bidirectional communication link with Robot a are termed as *neighbouring robots* of Robot a.

A simple communication graph is illustrated in Fig. 2. Two robots build a bidirectional communication link and become neighbouring robots if their physical distance is within the transmission range of their onboard communication equipment. Robot a_4 and Robot a_2 are not neighboring robots because they have no direct communication link with each other. However, Robot a_3 can transmit the information of Robot a_4 to Robot a_2.

Definition 7 (Consensus step) In a connected network, a consensus step is one round of communication after which each robot share the information of all other robots in the network.

The following theorem adapted from (Buchbinder et al., 2014) will be used as the mathematical foundation to analyse the approximation guarantee of the proposed task allocation algorithm. This shows the bound of expectation of $h(S)$, i.e., $\mathbb{E}[h(S)]$ with respect to the the utility of an empty set, $h(\emptyset)$ for all submodular functions $h(\cdot)$. Please refer to (Buchbinder et al., 2014) for the proof of Theorem 1.

Theorem 1 ((Buchbinder et al., 2014)) *Let* $h : 2^{\mathcal{N}} \rightarrow \mathbb{R}_{\geq 0}$ *be a submodular function, and let* S *be a random subset of* $\mathcal{N}$ *where each element appears with probability at most* p *(not necessarily independently). Then,* $\mathbb{E}[h(S)] \geq (1-p)h(\emptyset)$.

3 Algorithm

As summarised in Algorithm 1, DSTA consists of two phases: the sampling phase (Algorithm 1, lines 1 – 6) and the allocation phase (Algorithm 1, lines 7 – 20).

In the sampling phase, the ground set $\mathcal{N}_a$ containing all pairs that are related to Robot a is generated. Then, Robot a randomly samples pairs from $\mathcal{N}_a$ with a uniform probability $p \in (0, 1]$ to form its own sample set, $\mathcal{R}_a$ (Algorithm 1, lines 3 – 6). This means that the probability of a pair u_{aj} being added into the sample set $\mathcal{R}_a$ is p.

In the allocation phase, each robot selects a pair and negotiates with other robots in a greedy manner. Robot a selects a pair that provides the maximum *mgv* from its own sample set $\mathcal{R}_a$ (Algorithm 1, lines 8 – 9). Then, Robot a uses the information of the selected pair to negotiate with other robots through the *MaxCons* function (Algorithm 1, line 10). The notation "==" in line 11 is used to judge whether the two elements have the same value. "$a^* == a$" means that Robot a provides the globally largest *mgv* and wins the auction in the current iteration. Robot a adds its pair u_{aj}^m to its selection set and removes the pair from its sample set (Algorithm 1, lines 11 – 13). Otherwise, if Robot a has the pair u_{aj^*} in its sample set, the robot removes u_{aj^*} from its $\mathcal{R}_a$ (Algorithm 1, lines 15 – 17). Robots repeat this procedure until there is no more task to allocate.

MaxCons in line 10 of Algorithm 1 is the max-consensus function, which represents the negotiation among all robots. For the max-consensus, Robot a sends its current best *mgv*, i.e., ω_a^m together with the corresponding robot id a and task id j^m to its neighbours. At the same time, each robot receives the same information from all its neighbours. Robots keep transferring the information they have received until reaching global consensus. Finally, the *MaxCons* function returns the corresponding robot id a^* and the task id j^* of the pair that provides the globally largest *mgv*.

Algorithm 1 DSTA for Robot a

Input: $f : 2^{\mathcal{A}\times\mathcal{T}} \to \mathbb{R}^+, \mathcal{T}, \mathcal{A}, p$.
Output: A set $S_a \subseteq \{u_{aj} : \forall j \in \mathcal{T}\}$.

1: $\mathcal{R}_a \leftarrow \emptyset$, $S_a \leftarrow \emptyset$
2: $\mathcal{N}_a \leftarrow \{u_{aj} : \forall j \in \mathcal{T}\}$
3: **for** $u_{aj} \in \mathcal{N}_a$ **do**
4: with probability p,
5: $\mathcal{R}_a \leftarrow \mathcal{R}_a \cup \{u_{aj}\}$
6: **end for**
7: **while** $\exists\ u_{aj} \in \mathcal{R}_a$ s.t. $\Delta f(u_{aj}|S_a) > 0$ **do**
8: $j^m, u_{aj}^m \leftarrow \underset{j\in\mathcal{T}, u_{aj}\in\mathcal{R}_a}{\arg\max} \Delta f(u_{aj}|S_a)$
9: $\omega_a^m \leftarrow \Delta f(u_{aj}^m|S_a)$
10: $a^*, j^* \leftarrow MaxCons(a, j^m, \omega_a^m, \mathcal{A})$
11: **if** $a^* == a$ **then**
12: $S_a \leftarrow S_a \cup \{u_{aj}^m\}$
13: $\mathcal{R}_a \leftarrow \mathcal{R}_a \backslash \{u_{aj}^m\}$
14: **else**
15: **if** $u_{aj^*} \in \mathcal{R}_a$ **then**
16: $\mathcal{R}_a \leftarrow \mathcal{R}_a \backslash \{u_{aj^*}\}$
17: **end if**
18: **end if**
19: **end while**
20: **return** S_a

Remark 1 Thanks to sampling, each robot in DSTA is required to evaluate function values only for a portion of entire pairs, i.e., only for those sampled. This should accelerate the task allocation process by some degree, which depends on the sampling probability p.

Remark 2 Developing a max-consensus protocol is beyond the scope of this study. There are many max-consensus algorithms available in the existing literature. Also, the convergence characteristics of such algorithms are well studied considering various practical aspects of communication, e.g. dynamic networks, asynchronous communication, and time delay (Cortés, 2008; Giannini et al., 2016; Iutzeler et al., 2012; Olfati-Saber & Murray, 2004). For the application of the proposed DSTA algorithm, one can select an efficient consensus protocol considering the practical aspects.

Algorithm 2 An Equivalent View of DSTA

Input: $f : 2^{\mathcal{A}\times\mathcal{T}} \to \mathbb{R}^+, \mathcal{T}, \mathcal{A}, p$.
Output: Sets $S_a \subseteq \{u_{aj} : \forall j \in \mathcal{T}\}, \forall a \in \mathcal{A}$.

1: **for** $a \in \mathcal{A}$ **do**
2: $\mathcal{R}_a \leftarrow \emptyset$, $S_a \leftarrow \emptyset$
3: $\mathcal{N}_a \leftarrow \{u_{aj} : \forall j \in \mathcal{T}\}$
4: **for** $u_{aj} \in \mathcal{N}_a$ **do**
5: with probability p,
6: $\mathcal{R}_a \leftarrow \mathcal{R}_a \cup \{u_{aj}\}$
7: **end for**
8: **end for**
9: **while** $\exists\ u_{aj} \in \mathcal{R}_a$ s.t. $\Delta f(u_{aj}|S_a) > 0$ **do**
10: $a^*, j^* \leftarrow \underset{a\in\mathcal{A}, j\in\mathcal{T}, u_{aj}\in\mathcal{R}_a}{\arg\max} \Delta f(u_{aj}|S_a)$
11: **for** $a \in \mathcal{A}$ **do**
12: **if** $a^* == a$ **then**
13: $S_a \leftarrow S_a \cup \{u_{aj^*}\}$
14: $\mathcal{R}_a \leftarrow \mathcal{R}_a \backslash \{u_{aj^*}\}$
15: **else**
16: **if** $u_{aj^*} \in \mathcal{R}_a$ **then**
17: $\mathcal{R}_a \leftarrow \mathcal{R}_a \backslash \{u_{aj^*}\}$
18: **end if**
19: **end if**
20: **end for**
21: **end while**
22: **return** $S_a, \forall a \in \mathcal{A}$

An equivalent view of DSTA is demonstrated in Algorithm 2, which handles task allocation procedures of all robots in one algorithm framework. The lines 1 – 8 of Algorithm 2 describe the sampling process of DSTA for all robots. The greedy selection and *MaxCons* (Algorithm 1, lines 8 – 10) are represented by the line 10 of Algorithm 2. This line finds the robot id a^* and task id j^* corresponding to the pair that can provide the globally largest *mgv* among all robots in the current iteration. Then, in lines 11 – 20 of Algorithm 2, the winner Robot a^* puts its selected pair to its allocation set S_a. Next, all robots remove the pairs related to the corresponding task j^* from their sample sets to avoid conflicts.

Remark 3 It is possible that some tasks are not allocated to any robot due to random sampling, especially when the number of robots is small. If coverage of tasks is critical, we can increase the sampling probability. Nonetheless, as indicated in Definition 1, allocating all tasks is not required in the task allocation problem considered in this paper. This is natural as we consider not only monotone, but also non-monotone cases where the algorithm should be terminated when a marginal gain value becomes negative. In addition, for some repetitive missions such as a multi-target surveillance mission, it is unnecessary to cover all targets every time when there are too many targets compared with the number of robots.

Remark 4 CBBA algorithm (Choi et al., 2009) is used as a benchmark algorithm as it is widely used and is developed based on the decentralised SGA. Each robot with CBBA first constructs a task bundle by continually adding tasks in a greedy manner. Then, CBBA applies a sophisticated consensus strategy to enable convergence on the list of winning bids and robots. The task bundles help CBBA to reduce communication burden but incur dramatic increase in the computational complexity. The reason is that if a task *j* in a bundle of a robot is allocated to another robot during one consensus step, the robot must release all the tasks that are added to the bundle after the task *j*, and reconstruct the bundle in the next iteration. Reconstructing the bundle requires additional function evaluations and thus increases the computational complexity.

4 Analysis

This section analyses the performance, especially optimality and computational complexity, of the DSTA algorithm. As discussed in Sect. 3, for ease of analysis, it is assumed that the communication network is strongly connected and stable. Under the assumption, it is well known that the convergence of the max-consensus is established (Cortés, 2008; Giannini et al., 2016; Iutzeler et al., 2012; Olfati-Saber & Murray, 2004) and thus the decentralised algorithm can be understood in an equivalent view shown in Algorithm 2.

4.1 Algorithm analysis

For the convenience of analysing the theoretical performance of the proposed decentralised DSTA, we transform Algorithm 2 to an equivalent version, Algorithm 3, which first samples the task-robot pairs and then performs a greedy algorithm for the sampled set. We define the ground set $\mathcal{N}$ as a set containing all task-robot pairs, i.e., $\mathcal{N} := \{u_{aj} : \forall a \in \mathcal{A}, \forall j \in \mathcal{T}\} = \cup_{a\in\mathcal{A}}\mathcal{N}_a$. $\mathcal{I}$ is defined as the collection of all independent sets of task-robot pairs. An independent set of task-robot pairs means that this set can contain at most one task-robot pair from each partition determined by each task as described in Fig 1.

According to Definition 1, the sets $\{\mathcal{T}_a\}_{a\in\mathcal{A}}$ are disjoint, i.e., $\cup_{a\in\mathcal{A}}S_a \in \mathcal{I}$. Also, from Definition 1, $S = \cup_{a\in\mathcal{A}}S_a$ and $f(\cup_{a\in\mathcal{A}}S_a) = \sum_{a\in\mathcal{A}}f(S_a)$. Denoting the i^{th} robot as a_i, we have:

$$\begin{aligned}\Delta f(u_{aj}|S) &= f\left(u_{aj} \cup \left(\cup_{a_i\in\mathcal{A}}S_{a_i}\right)\right) - f\left(\cup_{a_i\in\mathcal{A}}S_{a_i}\right)\\ &= f(u_{aj}\cup S_a) + \sum_{a_i\in\mathcal{A},a_i\neq a}f(S_{a_i}) - \sum_{a_i\in\mathcal{A}}f(S_{a_i})\\ &= f(u_{aj}\cup S_a) - f(S_a)\\ &= \Delta f(u_{aj}|S_a)\end{aligned} \tag{4}$$

We can thus represent the DSTA algorithm with respect to S, instead of S_a, in a centralised view under the assumption of the consensus convergence. The resulting algorithm given in Algorithm 3 is equivalent to the sampling greedy algorithm for submodular maximisation subject to a partition matroid constraint. Note that the while loop condition $\exists\, u_{aj} \in \mathcal{R}_a$ in Algorithm 2 is replaced by $\exists\, u_{aj} \in \mathcal{N}_s\backslash S$ s.t. $S\cup\{u_{aj}\} \in \mathcal{I}$ in Algorithm 3. The rationale behind this replacement is that, if the allocated tasks are subtracted from $\mathcal{R}_a$ for all robots, the solution set S should satisfy $S\cup\{u_{aj}\} \in \mathcal{I}$ thanks to the partition matroid properties. The line 8 of Algorithm 3 is to find the globally best task-robot pair.

Algorithm 3 Sample Greedy

Input: $f : 2^{\mathcal{A}\times\mathcal{T}} \to \mathbb{R}^+, \mathcal{T}, \mathcal{A}, \mathcal{I}, p$.
Output: A set $S \in \mathcal{I}$.

1: $\mathcal{N}_s \leftarrow \emptyset, S \leftarrow \emptyset$
2: $\mathcal{N} \leftarrow \{u_{aj} : \forall a \in \mathcal{A}, \forall j \in \mathcal{T}\}$
3: **for** $u_{aj} \in \mathcal{N}$ **do**
4: with probability p,
5: $\mathcal{N}_s \leftarrow \mathcal{N}_s \cup \{u_{aj}\}$
6: **end for**
7: **while** $\exists\, u_{aj} \in \mathcal{N}_s\backslash S$ s.t. $S\cup\{u_{aj}\} \in \mathcal{I}$ and $\Delta f(u_{aj}|S) > 0$ **do**
8: $u^*_{aj} \leftarrow \underset{u_{aj}\in\mathcal{N}_s\backslash S}{\arg\max}\ \Delta f(u_{aj}|S)$
9: $S \leftarrow S \cup \{u^*_{aj}\}$
10: **end while**
11: **return** S

This paper considers Algorithm 3 as the baseline algorithm for the performance analysis. Note that (Feldman et al., 2017) provides a good analysis scheme for the sample greedy for k-extendable systems, especially for the approximation guarantee. Since the partition matroid constraint is a special case of k-extendable systems, we can borrow the analysis scheme in Feldman et al. (2017) for the analysis of the proposed DSTA algorithm. Therefore, we follow the analysis scheme in Feldman et al. (2017), but make necessary modifications to facilitate the partition matroid constraint to examine the expected approximation guarantee. To further investigate properties, we derive the upper bound of the variance of the solution in the proposed DSTA algorithm. Note that this is the first attempt to examine the variance of any randomised greedy algorithm.

Algorithm 4 Equivalent Sample Greedy

Input: $f : 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0}, \mathcal{N}, \mathcal{I}, p$.
Output: A set $S \in \mathcal{I}$.

1: $\mathcal{N}_s \leftarrow \emptyset, S \leftarrow \emptyset, R \leftarrow \mathcal{N}, C \leftarrow \emptyset, Q \leftarrow OPT, K_u = \emptyset$ for all $u \in \mathcal{N}$
2: **for** $u \in \mathcal{N}$ **do**
3: with probability p,
4: $\mathcal{N}_s \leftarrow \mathcal{N}_s \cup \{u\}$
5: **end for**
6: **while** $\exists\ u \in R$ s.t. $S \cup \{u\} \in \mathcal{I}$ and $\Delta f(u|S) > 0$ **do**
7: $c \leftarrow \arg\max_{u \in R} \Delta f(u|S)$
8: $S_c \leftarrow S$
9: $C \leftarrow C \cup \{c\}$
10: $R \leftarrow R \backslash \{c\}$
11: **if** $c \in \mathcal{N}_s$ **then**
12: $S \leftarrow S \cup \{c\}$
13: $Q \leftarrow Q \cup \{c\}$
14: Let $K_c \subseteq Q \backslash S$ be the smallest set s.t. $Q \backslash K_c \in \mathcal{I}$
15: **else**
16: **if** $c \in Q$ **then**
17: $K_c \leftarrow \{c\}$
18: **else**
19: $K_c \leftarrow \emptyset$
20: **end if**
21: **end if**
22: $Q \leftarrow Q \backslash K_c$
23: **end while**
24: **return** S

Following (Feldman et al., 2017), we present Algorithm 4, which results in the output S equivalent to Algorithm 3, but contains a few auxiliary variables. Algorithm 4 allows us to ease the performance analysis. To derive the theoretical approximation guarantee in the average sense, it is necessary to take all task-robot pairs into account, even those that are not in the sample set. Then, in line 11 of Algorithm 4, we check whether the current considered pair c is in the sample set. If c is in $\mathcal{N}_s$, it means that c has been sampled and should be put into the solution set S. In this way, we can analyse the theoretical performance of the random sampling in the average sense. Note that Algorithm 4 introduces a few auxiliary variables, such as R, OPT, C, S_c, Q, and K_c, which are highlighted in magenta. These variables have no effect on procedures of generating S and hence on the output S, but are used for the convenience of analysis. Therefore, Algorithm 4 is equivalent to Algorithm 3.

Let us briefly discuss the meanings and roles of the auxiliary variables. The set R in Algorithm 4 is for remaining pairs, i.e., $R = \mathcal{N} \backslash C$, and OPT is the optimal solution. There is no need to know the exact value of OPT because it is introduced only for the theoretical analysis.

The variable C is a set that contains all pairs that have already been considered by Algorithm 4 regardless of whether they are added to S or not.

S_c is a set that contains the selected pairs at the beginning of the current iteration. At the end of the current iteration, $S = S_c \cup \{c\}$ if c is added into S and Q, otherwise $S = S_c$.

Q is an independent set introduced to help to prove the relationship between the expected function utility of the solution set S of DSTA and that of the optimal solution set OPT.

K_c is a set that is introduced to ensure Q remains independent even as c is added. Following the matroid properties, K_c is removed from Q to ensure the independence of Q. Before c is added, Q is independent and thus satisfies the partition matroid constraint at each iteration. Once c is added, Q could have at most two common elements within one partition. This implies that the set K_c is either an empty set or a singleton, i.e., $|K_c| \leq 1$.

The variables with subscript in Algorithm 4, e.g. K_c and S_c, are not single variables, but they have a distinct value for each $c \in C$. On the other hand, variables denoted without any subscript, e.g. S or Q, are single variables.

4.2 Performance analysis

The main characteristics of the approximation guarantee and computational complexity of the proposed DSTA algorithm are summarised in Theorem 2.

Theorem 2 *Suppose the max-consensus in Algorithm* 1 *assures convergence*. *Then*, *the DSTA algorithm achieves the following expected approximation guarantees*, G_a, *for submodular objective functions*:

$$G_a = \begin{cases} \frac{p}{p+\max(p,1-p)} & \text{for monotone} \\ \frac{p(1-p)}{p+\max(p,1-p)} & \text{for non-monotone} \end{cases} \tag{5}$$

with an expected total computational complexity of $O(pnr)$ *and individual complexity of* $O(pr^2)$ *for each robot*, *where* p *is the sampling probability*, r *is the number of tasks*, *i.e.*, $r = |\mathcal{T}|$, *and* n *is the number of pairs*, *i.e.*, $n = |\mathcal{T}| \times |\mathcal{A}|$.

The computational complexity of DSTA can be easily proven. As shown in Algorithm 2, there are at most r rounds of auctions. In each round, each robot requires function evaluations at most pr times on average. Since there are $|\mathcal{A}|$ number of robots, the total number of utility function evaluations in each auction is $O(pn)$. Therefore, the average total time complexity is $O(pnr)$. For each robot, the average individual time complexity is equivalent to the average total complexity divided by the number of robots, i.e., $O(pr^2)$.

Let us now investigate the approximation guarantee of DSTA through Algorithm 4. To prove the approximation guarantee given in Theorem 2, we first show the bound of expectation of $f(S)$, i.e., $\mathbb{E}\left[f(S)\right]$, with respect to $\mathbb{E}[f(S \cup OPT)]$ in Lemma 3. Lemmas 1 and 2 will be required to prove the bound of $\mathbb{E}\left[f(S)\right]$ in Lemma 3.

Lemma 1 $\mathbb{E}[|K_u|] \leq \max(p, 1-p)$ *for all* $u \in \mathcal{N}$.

Proof See Appendix A. □

Lemma 2 $\mathbb{E}[f(S)] = \sum_{u \in \mathcal{N}} p\mathbb{E}\left[\Delta f(u|S_u)\right]$.

Proof See Appendix B. □

Now, let us derive the lower bound of $\mathbb{E}\left[f(S)\right]$ with respect to $\mathbb{E}[f(S \cup OPT)]$ based on the results of Lemmas 1 and 2.

Lemma 3 $\mathbb{E}[f(S)] \geq \frac{p}{p+\max(p,1-p)}\mathbb{E}[f(S \cup OPT)]$.

Proof According to the evolution of Q in Algorithm 4, Q is independent at the end of each iteration, i.e., $Q \in \mathcal{I}$. S is a subset of Q i.e., $S \subseteq Q$, since every element c that is added to S is also in Q. Therefore, from Definition 5, we have $S \cup \{q\} \in \mathcal{I}\ \forall q \in Q \backslash S$. By the termination condition of Algorithm 4, $\Delta f(q|S) \leq 0\ \forall q \in Q \backslash S$. Hence, at the termination of Algorithm 4, it holds that

$$\sum\nolimits_{q \in Q \backslash S} \Delta f(q|S) \leq 0.$$

Let $Q \backslash S = \{q_1, q_2, \ldots, q_{|Q \backslash S|}\}$, then

$$\begin{aligned} f(S) &= f(Q) - \sum\nolimits_{i=1}^{|Q \backslash S|} \Delta f(q_i | S \cup \{q_1, \ldots, q_{i-1}\}) \\ &\geq f(Q) - \sum\nolimits_{i=1}^{|Q \backslash S|} \Delta f(q_i|S) \quad \text{(submodularity)} \\ &\geq f(Q) \end{aligned}$$

If $u \in C$, it implies that the *mgv* of u is no less than any other element from $K_u \backslash S$ at that iteration, i.e.,

$$\Delta f(u|S_u) \geq \Delta f(q|S_u),\ \forall q \in K_u \backslash S. \tag{6}$$

Additionally, any pair can be removed from Q at most once. In other words, the pair that is contained in K_u at one iteration is always different from other iterations when K_u is not empty. Hence, the sets in the sequence $\{K_u \backslash S\}_{u \in \mathcal{N}}$ are disjoint. According to the definition and evolution of Q, Q can be expressed as

$$Q = (S \cup OPT) \backslash \cup_{u \in \mathcal{N}} (K_u \backslash S). \tag{7}$$

Note that if $u \notin C$, then $K_u = \emptyset$ and $S_u = \emptyset$ by convention. Denoting $\mathcal{N}$ as $\{u_1, \ldots, u_{|\mathcal{N}|}\}$, we define Q_u^i as:

$$Q_u^i := (S \cup OPT) \backslash \cup_{u \in \mathcal{N}_i} (K_u \backslash S) \tag{8}$$

where $\mathcal{N}_i = \{u_1, \ldots, u_i\}$. Then, it is clear that $S_u \subseteq S \subseteq Q_u^i$. From Eq. (7), we obtain

$$\begin{aligned} f(Q) &= f(S \cup OPT) - \sum\nolimits_{i=1}^{|\mathcal{N}|} \Delta f(K_u^i \backslash S | Q_u^i) \\ &\geq f(S \cup OPT) - \sum\nolimits_{i=1}^{|\mathcal{N}|} \Delta f(K_u^i \backslash S | S_u^i) && \text{(submodularity)} \\ &\geq f(S \cup OPT) - \sum\nolimits_{u \in \mathcal{N}} |K_u \backslash S| \Delta f(u|S_u) && \text{(Eq. (6))} \\ &\geq f(S \cup OPT) - \sum\nolimits_{u \in \mathcal{N}} |K_u| \Delta f(u|S_u) \end{aligned}$$

where K_u^i and S_u^i denote K_u and S_u corresponding to u_i, respectively.

By taking an expectation over $f(S)$, we have

$$\begin{aligned}
\mathbb{E}[f(S)] &\geq \mathbb{E}[f(Q)] \\
&\geq \mathbb{E}[f(S \cup OPT)] - \mathbb{E}[|K_u|] \cdot \mathbb{E}\left[\sum\nolimits_{u \in \mathcal{N}} \Delta f(u|S_u)\right] \\
&\geq \mathbb{E}[f(S \cup OPT)] - \max(p, 1-p) \cdot \sum\nolimits_{u \in \mathcal{N}} \mathbb{E}[\Delta f(u|S_u)] \qquad \text{(Lemma 1)} \\
&= \mathbb{E}[f(S \cup OPT)] - \max(p, 1-p) \cdot \frac{1}{p} \mathbb{E}[f(S)]. \qquad \text{(Lemma 2)}
\end{aligned}$$

The result is clear by rearranging the above inequality. □

We are now ready to complete the proof of Theorem 2. We denote the objective function f as f_m and f_n for the monotone and non-monotone cases, respectively.

Proof of Theorem 2 To obtain the approximation guarantees for both monotone and non-monotone submodular utility functions, we need to analyse the relationship between $f(S \cup OPT)$ and $f(OPT)$. If f is monotone, then

$$f_m(S \cup OPT) \geq f_m(OPT). \tag{9}$$

From Lemma 3 and Eq. (9), it is clear that

$$\begin{aligned}
\mathbb{E}[f_m(S)] &\geq \frac{p}{p + \max(p, 1-p)} \cdot \mathbb{E}[f_m(S \cup OPT)] \quad \text{(Lemma 3)} \\
&\geq \frac{p}{p + \max(p, 1-p)} \cdot f_m(OPT).
\end{aligned}$$

For the non-monotone case, we define a new submodular function $h : 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0}$ as $h(X) = f_n(X \cup OPT)\ \forall X \subseteq \mathcal{N}$. Since S contains every element with probability at most p, Theorem 1 yields

$$\mathbb{E}[f_n(S \cup OPT)] = \mathbb{E}[h(S)] \geq (1-p)h(\emptyset) = (1-p)f_n(OPT).$$

Hence, we have

$$\mathbb{E}[f_n(S)] \geq \frac{p(1-p)}{p + \max(p, 1-p)} \cdot f_n(OPT).$$

□

Corollary 1 *The trade-off between approximation guarantee and computational complexity can be controlled by adjusting the sampling probability p. When $p = 0.5$, the solution given by DSTA is lower-bounded by $\frac{1}{2}$ and $\frac{1}{4}$ of the optimal solution for the monotone and non-monotone cases, respectively.*

Proof Recalling that p is the sampling probability and $p \in (0, 1]$, we have

$$\max(p, 1-p) = \begin{cases} 1-p & \text{for } p \in (0, 0.5] \\ p & \text{for } p \in (0.5, 1]. \end{cases}$$

Therefore, in the monotone case, the expected approximation ratios are obtained as:

$$\mathbb{E}[f_m(S)] \geq \begin{cases} p \cdot f_m(OPT) & \text{for } p \in (0, 0.5] \\ 1/2 \cdot f_m(OPT) & \text{for } p \in (0.5, 1] \, . \end{cases} \tag{10}$$

In the non-monotone case,

$$\mathbb{E}[f_n(S)] \geq \begin{cases} p(1-p) \cdot f_n(OPT) & \text{for } p \in (0, 0.5] \\ (1-p)/2 \cdot f_n(OPT) & \text{for } p \in (0.5, 1] \, . \end{cases} \tag{11}$$

As shown in Eqs. (10) and (11), for $p \in (0.5, 1]$, the lower bound of the expected approximation ratio becomes stagnated in the monotone case and decreasing in the non-monotone case. Moreover, it is clear that the computational complexity increases as the sampling probability increases. On the other side, for $p \in (0, 0.5]$, the sampling probability provides trade-off capability between the approximation ratio and computational complexity. As the probability increases for $p \in (0, 0.5]$, the expected approximation ratios improve for both monotone and non-monotone cases, but the computational complexity also increases. From Eqs. (10) and (11), the best expected approximation guarantees can be readily obtained, when $p = 0.5$, as:

$$\mathbb{E}[f(S)] \geq \begin{cases} 1/2 \cdot f(OPT) & \text{if } f \text{ is monotone} \\ 1/4 \cdot f(OPT) & \text{if } f \text{ is non-monotone} \, . \end{cases}$$

□

Now, let us investigate another key property of the DSTA algorithm, that is the variance of the functional value of the solution.

Theorem 3 *Suppose the max-consensus in Algorithm* 1 *assures convergence. Then, the variance of the converged objective function is bounded as*:

$$Var(f(S)) \leq \left(\frac{1}{p} - 1\right) f^2(OPT) + p \cdot Var\left(\sum_{u \in \mathcal{N}} \Delta f(u|S_u)\right). \tag{12}$$

Proof $f^2(S)$ can be obtained as:

$$\begin{aligned} f^2(S) &= \left(\sum_{u \in S} \Delta f(u|S_u)\right)^2 \\ &= \sum_{u \in S} \Delta f^2(u|S_u) + 2 \sum_{u_i \in S} \sum_{u_j \in S^i} \Delta f(u_i|S_{u_i}) \Delta f(u_j|S_{u_j}), \end{aligned} \tag{13}$$

where $S^i \triangleq \mathcal{N}_i \cap S$. Its expectation is obtained as:

$$\mathbb{E}[f^2(S)] = p \cdot \mathbb{E}\left[\sum_{u\in\mathcal{N}} \Delta f^2(u|S_u)\right] + 2p^2 \cdot \mathbb{E}\left[\sum_{u_i\in\mathcal{N}}\sum_{u_j\in\mathcal{N}_i} \Delta f(u_i|S_{u_i})\Delta f(u_j|S_{u_j})\right]$$

$$\leq p\mathbb{E}\left[\left(\sum_{u\in\mathcal{N}} \Delta f(u|S_u)\right)^2\right]$$

$$= \frac{1}{p}(\mathbb{E}[f(S)])^2 + p \cdot \mathrm{Var}\left(\sum_{u\in\mathcal{N}} \Delta f(u|S_u)\right). \quad \text{(Lemma 2)}$$

Hence, the variance of $f(S)$ is bounded as:

$$\begin{aligned}\mathrm{Var}(f(S)) &= \mathbb{E}[f^2(S)] - (\mathbb{E}[f(S)])^2 \\ &\leq \left(\frac{1}{p} - 1\right)(\mathbb{E}[f(S)])^2 + p \cdot \mathrm{Var}\left(\sum_{u\in\mathcal{N}} \Delta f(u|S_u)\right) \\ &\leq \left(\frac{1}{p} - 1\right) f^2(OPT) + p \cdot \mathrm{Var}\left(\sum_{u\in\mathcal{N}} \Delta f(u|S_u)\right).\end{aligned} \tag{14}$$

□

Remark 5 Theorem 3 shows that the upper bound of the variance of $f(S)$ depends on the sampling probability p and variance of the summation of the marginal gain values. It is clear that the first term on the right-hand side of Eq. (14) is exponentially diminishing as the sampling probability increases. For the monotone case, the variance of the random variable S_u becomes smaller, approaching zero, as p increases. This implies that the variance of the summation of the marginal gain values decreases, converging to zero, as p increases to one. Therefore, it is expected that the upper bound of the variance decreases as the sampling probability increases. For the non-monotone case, the variance of S_u would not become smaller even if p increases since the DSTA algorithm will be terminated when $\Delta f(u|S)$ becomes negative. Note that negative marginal gain value is possible not in the monotone case, but only in the non-monotone case. This means that the second term on the right-hand side of Eq. (14) and consequently the variance could become larger as p increases for the non-monotone cases.

5 Numerical simulations

In this section, we validate the theoretical analysis and examine the performance of the DSTA algorithm. For rigorous validation, we consider two application scenarios of a multi-target surveillance mission. In the first scenario, we use unmanned aerial vehicles (UAVs) and compare DSTA with the benchmark algorithms. The first benchmark algorithm is a widely accepted decentralised task allocation algorithm, CBBA (Choi et al., 2009). The second benchmark algorithm is a state-of-the-art task allocation algorithm denoted as GA (Kotwal & Dhope, 2015) that is based on a genetic algorithm. A simple demonstration of

the multi-target surveillance mission can be found in the literature (Li et al., 2019). While in the second scenario, we compare DSTA with the state-of-the-art algorithm DGA (Qu et al., 2019) using spatially static Earth-observing satellites as robot platforms. Both monotone and non-monotone objective functions are considered for each scenario.

The computational complexity of task allocation algorithms is measured by the number of *function evaluations* (objective function queries), which is independent from the computer status. Since DSTA requires global consensus among robots, we count the number of *consensus steps* to measure the communication complexity. Note that the maximum number of consensus steps is equal to the number of tasks. To investigate the variances of DSTA performance, we run 1000 rounds of DSTA in each case. The results are depicted in bar graph to also indicate the variances. For comparison, we run CBBA (Choi et al., 2009) or DGA (Qu et al., 2019) for one round in each case because they are deterministic algorithms.

5.1 Scenario 1: Surveillance mission using UAVs

In this scenario, we assess the performance of DSTA and compare it with CBBA (Choi et al., 2009) and GA (Kotwal & Dhope, 2015) based on a monotone submodular objective function (Case 1) and a non-monotone submodular objective function (Case 2). Then, we examine the trade-off performance of DSTA with respect to different sampling probabilities (Case 3).

We assume that a group of UAVs needs to perform 60 tasks (waypoint targets). Tasks are randomly located on a $W \times W$ 2D space ($W = 10$ km). In Case 1 and Case 2, the sampling probability is fixed to 0.5. In Case 3, we fix the number of UAVs to 15 and increase the sampling probability from 0.1 to 0.9 with a step of 0.1.

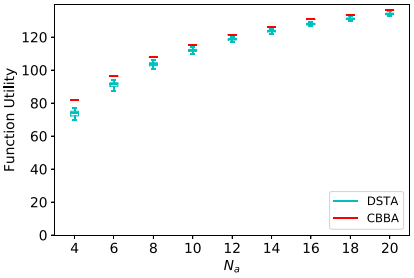
Case 1: Monotone submodular objective function

We adapt a monotone submodular utility function from (Segui-Gasco et al., 2015) which is a coverage-type function for the surveillance mission. Different tasks are marked with a *task value factor* v_j according to their values. Assume that UAVs are equipped with different sensors that are suitable for different tasks. The *task-robot fitness factor* m_{aj} represents the match fitness between the capabilities of Robot a and the requirements of Task j. The utility of executing the tasks $j \in \mathcal{T}_a$ is measured as the sum of the product of m_{aj} and v_j. For the tasks $j \notin \mathcal{T}_a$, we add an exponentially decaying term related to the shortest distance between Task j and tasks in $\mathcal{T}_a$ which is denoted as $d_{\min}(j, \mathcal{T}_a)$. When Robot a is carrying out a task at the location of this task, it can partially serve another one nearby. The objective function for Robot a is

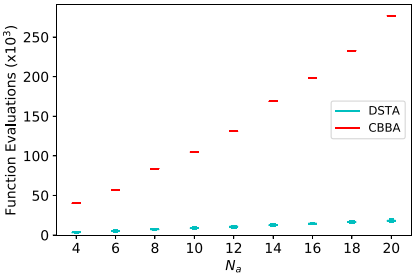
$$f(S_a) = \sum_{j \in \mathcal{T}_a} m_{aj} v_j + \sum_{j \notin \mathcal{T}_a} m_{aj} v_j e^{-\frac{d_{\min}(j, \mathcal{T}_a)}{d_0}}, \tag{15}$$

where d_0 is a reference distance. The overall objective function of the surveillance mission is obtained by combining Eq. (15) and Definition 1:

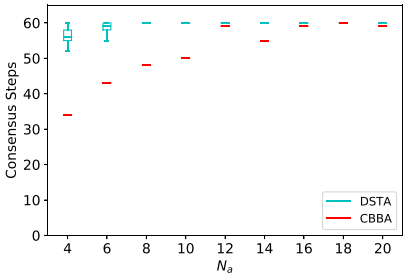
$$f(S) = \sum_{a \in \mathcal{A}} \left[\sum_{j \in \mathcal{T}_a} m_{aj} v_j + \sum_{j \notin \mathcal{T}_a} m_{aj} v_j e^{-\frac{d_{\min}(j, \mathcal{T}_a)}{d_0}} \right]. \tag{16}$$



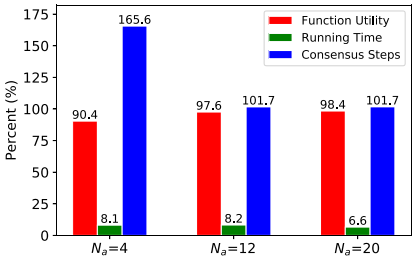
(a) Solution quality represented by objective function utilities.



(b) Computational complexity measured by numbers of function evaluations.



(c) Communication complexity measured by numbers of consensus steps.



(d) Percentages of performance of DSTA with CBBA as a baseline.

Fig. 3 Performance comparison of DSTA and CBBA in the monotone case

In the simulations, the task value factors are uniformly random numbers $v_j \in [0.6, 1.0]$. Each task-robot fitness factor is set as a uniformly random number $m_{aj} \in [0.5, 1.0]$. The reference distance is set as $d_0 = 1$ km.

Figure 3 demonstrates the simulation results comparing the performance of DSTA with CBBA. On the one hand, as shown in Fig. 3a, the function utilities achieved by CBBA and DSTA increase with a decreasing increment rate as the number of robots increases. The nonlinearity of increment is attributed to the submodularity of the objective function. The proposed DSTA algorithm achieves a comparable quality of solutions, i.e., comparable values of the utility function, to CBBA. On average, the quality of the DSTA solution increases from 90% to 98% with CBBA as a baseline as shown in Fig. 3d. On the other hand, Figure 3b shows that the computational complexity of CBBA and DSTA increases linearly as the number of robots increases. As shown in Fig. 3 b and d, the computational time of the DSTA algorithm is significantly lower compared with that of CBBA (less than 10%). It is worth noting that as the size of the problem increases, the difference between the two algorithms on the function value becomes tighter, but the difference in computational time becomes more significant. Overall, DSTA has minor variations in terms of function utilities, computational complexity, and consensus steps in the monotone case.

One advantage of CBBA is that it requires fewer consensus steps than DSTA does, especially when there are small numbers of UAVs, as shown in Fig. 3c. Therefore, CBBA has lower potential communication complexity. However, UAVs communicate

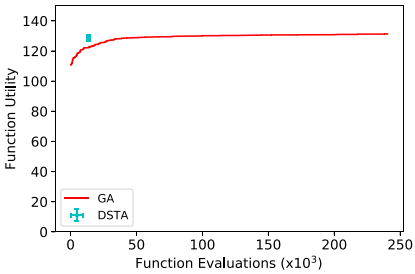


Fig. 4 Performance comparison of DSTA and GA in the monotone case

using task bundle information with CBBA while using single task information with DSTA. This means that more information needs to be transferred in each consensus step with CBBA. In addition, the number of consensus steps required by CBBA gets closer to that required by DSTA as N_a goes up. When there are fewer UAVs, some tasks could be more likely ignored by all UAVs due to random sampling. This is the reason why DSTA requires fewer consensus steps and shows larger spreads when N_a is small as shown in Fig. 3c.

Next, we compare the performance of DSTA with that of GA (Kotwal & Dhope, 2015). The number of robots is fixed as 15. For GA, the size of the population is set as 20. In each iteration, we store the largest function utility among the population of 20. In order to avoid the local optima, we set the mutation probability to 0.5. We set the maximum number of iterations to 8000.

Figure 4 reports the simulation results. The number of function evaluations of GA is proportional to the number of iterations. According to Fig. 4, the maximum utility achieved by GA is around 131. DSTA achieves an average utility of about 129 which is 98.5% of GA. On average, DSTA executes 13.6×10^3 function evaluations. In contrast, GA executes more function evaluations than DSTA to achieve the same function utility.

Case 2: Non-monotone submodular objective function

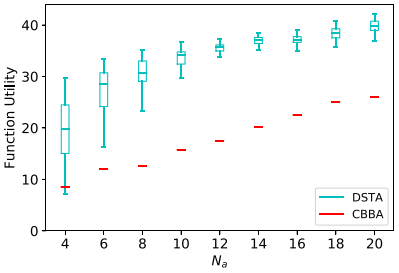
A non-monotone submodular objective function is designed by modifying the monotone objective function. Assume that tasks with high importance are usually difficult to execute. Allocating many difficult tasks to one robot could exceed the capability of the robot and cause mission failure. Therefore, it is risky to allocate many important tasks to one robot, but only a few to others. We introduce an *inter-task effect factor* x_{ij} as a penalty to the concentration of important tasks for each robot.

The overall objective function in the non-monotone case is modelled as

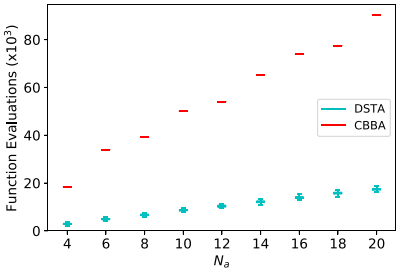
$$f(S) = \sum_{a \in \mathcal{A}} \left[\sum_{j \in \mathcal{T}_a} m_{aj} v_j - \lambda_x \sum_{i,j \in \mathcal{T}_a, i<j} x_{ij} \right], \tag{17}$$

where λ_x is a scaling parameter of the penalty term. The model uses $x_{ij} = e^{v_i v_j}$ to discourage the concentration of important tasks in one robot. Note that the task allocation algorithms will stop once the *mgv*s of all remaining tasks become negative.

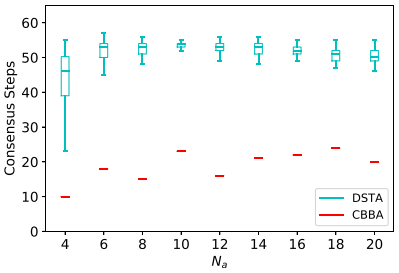
It is assumed that there are N_a special tasks that are more important and difficult than others. We set the importance factors of these tasks as uniformly random numbers $v_j \in [5.0, 6.0]$. For these tasks, the match fitness factors m_{aj} are set to be 0.2 for certain UAVs and 0.1 for others. The importance factors of other tasks are uniformly distributed over [0.6, 1.0] and the match fitness factors related to these tasks are uniformly distributed over [0.5, 1.0]. To examine the variance of the performance of DSTA, we fix the values of



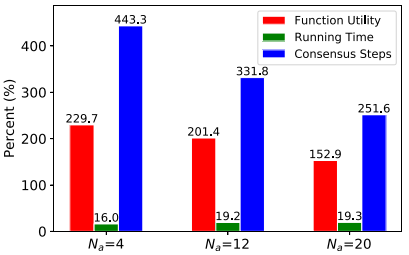
(a) Solution quality represented by objective function utilities.



(b) Computational complexity measured by numbers of function evaluations.



(c) Communication complexity measured by numbers of consensus steps.



(d) Percentages of performance of DSTA with CBBA as a baseline.

Fig. 5 Performance comparison of DSTA and CBBA in the non-monotone case

these importance factors and match fitness factors once they are assigned at the beginning of the algorithm. We set the scaling factor of the penalty term to $\lambda_x = 0.01$.

The simulation results are reported in Fig. 5. The results confirm that the proposed DSTA achieves better average function values and is still significantly faster than CBBA. Figure 5a shows that the average utilities achieved by DSTA are higher than the utilities achieved by CBBA. Compared with the monotone case in Fig. 3a, the spreads in Fig. 5a are larger. This is because whether the special tasks are selected or not has a significant impact on the results of DSTA and consequently its variation. When the special tasks are sampled, DSTA is most likely to select them greedily at the beginning, considering their relatively large *mgv*s. Unlike in the monotone case, this could quickly make the *mgv*, $\Delta f(u|S)$, negative and terminate the algorithm. Nonetheless, as shown in Fig. 5a, even the worst cases of the function values achieved by DSTA are comparable to those of CBBA. The potential attribute of the relatively poor performance of CBBA is that robots greedily select more important tasks at early iterations. If robots select more tasks in later iterations, the total function values could start to decrease due to the non-monotonicity of the utility function. Hence, CBBA gets trapped in local optima with a few allocated tasks per robot. Note that CBBA only has constant approximation guarantee for monotone submodular functions. By contrast, it may exhibit arbitrarily poor performance with non-monotone utility functions, which is confirmed by the simulation results. However, the sampling procedure in the proposed DSTA might enable abandoning the special tasks with a certain probability and hence utilising most of the available tasks to find solutions without getting trapped in local optima.

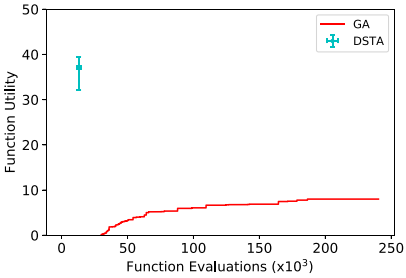


Fig. 6 Performance comparison of DSTA and GA in the non-monotone case

Then, we compare the performance of DSTA with that of GA (Kotwal & Dhope, 2015) using the same parameter settings as before. According to Fig. 6, DSTA achieves an average utility of 37 and executes 13.2×10^3 function evaluations on average. In contrast, the maximum utility achieved by GA is only 8. Moreover, GA executes much more function evaluations than DSTA. This is because GA tends to allocate more tasks to each robot. However, in the non-monotone case, adding too many tasks likely decreases the overall utility due to the negative marginal gain caused by the penalty term. On the contrary, DSTA will allocate a proper number of tasks to each robot. In other words, the DSTA algorithm will terminate once the largest *mgv* of the remaining tasks becomes negative. The simulation result indicates that DSTA can still work well in extreme cases where the objective function has significant non-monotonicity, but the benchmark algorithms (CBBA and GA) fail to provide a satisfactory solution.

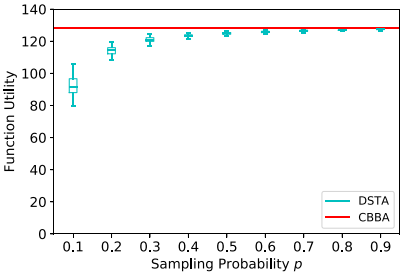
Case 3: Trade-offs with respect to different sampling probabilities

To validate the trade-off analysis in Sect. 4, we run another set of simulations with respect to different sampling probabilities. Parameters related to the monotone and non-monotone objective functions have the same settings as in Case 1 and Case 2, respectively, except the number of robots and the sampling probability. Simulation results for the monotone and non-monotone cases are demonstrated in Figs. 7 and 8, respectively.

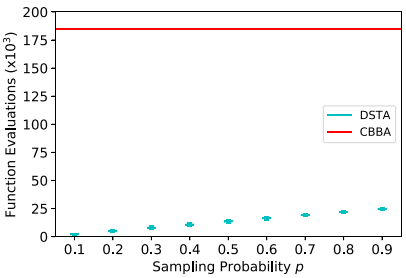
Figure 7 shows that the average function utility, computational complexity, and communication complexity increase as the sampling probability p increases in the monotone case. As shown in Fig. 7a and c, the increment rates of the functional utility and the number of consensus steps decrease for the fixed increase in the sampling probability. This feature is attributed to the fact that the utility function given in Eq. (16) and the task-robot pair coverage by random sampling are submodular. Figure 7a shows that the variation of the functional utility decreases as p increases, which confirms the analysis results in Theorem 3 and Remark 5. Figure 7b indicates that the average computational complexity is linearly increasing, which is coherent with the relevant theoretical result described in Theorem 2.

Simulation results in the non-monotone case are reported in Fig. 8. As shown in Fig. 8a, the average function utility achieved by DSTA initially increases and then decreases as the sampling probability increases, which demonstrates the non-monotone submodularity. Similarly, the average communication complexity shown in Fig. 8c also first increases and then decreases as p goes up from 0.1 to 0.9. This is because when $p \leq 0.5$, robots can cover more tasks with larger sampling probability, which means that they require more consensus steps. However, with larger sampling probability when $p > 0.5$, the task allocation process can get trapped easier by the special tasks. Figure 8b shows the tendency of the average computational complexity which is linearly increasing.

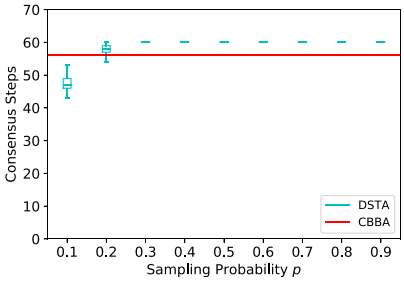
Now, let us examine the variation of the functional utilities in the non-monotone case. As shown in Fig. 8a, the spread of the error bars is bigger than that in the monotone case



(a) Solution quality represented by objective function utilities.



(b) Computational complexity measured by numbers of function evaluations.



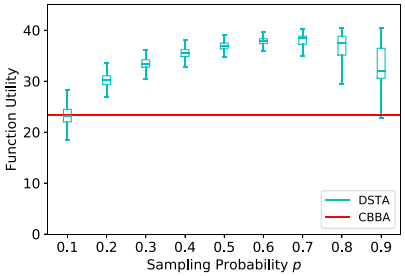
(c) Communication complexity measured by numbers of consensus steps.

Fig. 7 Performance of the DSTA algorithm with respect to different sampling probabilities in the monotone case $N_a = 15$ (red lines denote the values of CBBA, which are invariant with respect to the sampling probability)

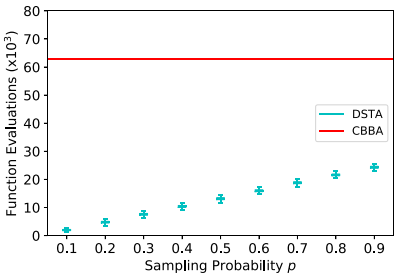
depicted in Fig. 7a. This is attributed to the difference in the nature of marginal gain values. In the monotone case, considering more tasks is always beneficial with respect to the functional utility as *mgv*s are always non-negative. In the non-monotone case, adding more tasks is not always beneficial since it might produce negative *mgv*s. Therefore, the task allocation process could provide poor solution quality when increasing the sampling probability. The global consensus stops once the task allocation process gets trapped. Therefore, the large spreads shown in Fig. 8a and c are resulted from the fact that the task allocation algorithm is likely to be trapped in local optimal solutions. Nevertheless, as demonstrated in Fig. 8a, DSTA provides better functional utility values in most of the cases and comparable values in the very worst case, compared with CBBA, unless p is significantly small.

It is worthwhile to note that the variation of the functional values is small around 0.5 and 0.6. Then, the variation becomes larger as p further increases. Beyond certain probability, as more task-robot pairs are sampled, the chance to be trapped in poor solutions increases. The variation of the functional values becomes even larger than lower probability cases as p gets closer to 1 in the non-monotone case. This is aligned with the analysis results of Theorem 3 and Remark 5.

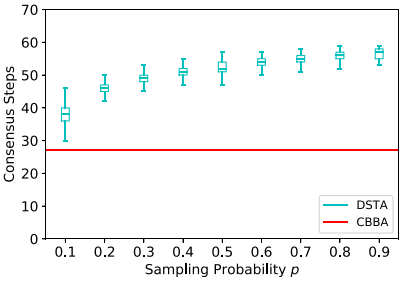
It is expected that the more significant the importance of special tasks becomes, the bigger the variation might become. This is because more significant importance implies more significant *mgv*, compared with other tasks. When *mgv*s of tasks are much bigger than the others, the greedy selection procedure in the algorithm could increase the chance of being



(a) Solution quality represented by objective function utilities.



(b) Computational complexity measured by numbers of function evaluations.



(c) Communication complexity measured by numbers of consensus steps.

Fig. 8 Performance of the DSTA algorithm with respect to different sampling probabilities with special tasks in the non-monotone case $N_a = 15$ (red lines denote the values of CBBA, which are invariant with respect to the sampling probability)

trapped in local solutions. For validation of the argument, we removed the special tasks from the non-monotone case and conducted Monte-Calro simulations. Figure 9 shows the results on the functional utility. As shown in the figure, median functional values keep increasing and the increment decreases as the sampling probability increases. Decreases in the functional values are not present and thus it is expected that the chance of being trapped in poor quality solutions is low. This implies that the variance of the functional values will decrease as p increases, which is confirmed in the simulation results illustrated in Fig. 9.

5.2 Scenario 2: Surveillance mission using satellites

We conducted a further comparison between DSTA and a state-of-the-art algorithm, DGA (Qu et al., 2019). The work in Qu et al. (2019) uses the concept of *admissible task sets* which means that the tasks are constrained to different admissible task sets for each robot. For a fair comparison, the proposed DSTA also adopts the concept of admissible task sets. Thus, the main difference between DSTA and DGA in the comparison is the task sampling procedure. We denote the admissible task set for Robot a as $\mathcal{N}_a$. DGA was developed and proved for stationary robots, unlike CBBA and DSTA. Hence, this paper adopts a new scenario with satellites from (Qu et al., 2019) in which the benefits of DGA can be well presented.

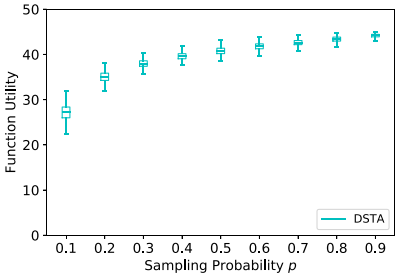


Fig. 9 Function utility of the DSTA algorithm with respect to different sampling probabilities without special tasks in the non-monotone case $N_a = 15$

Like in Qu et al. (2019), it is assumed that all satellites face vertically downwards to the ground. The operation range of each robot is a cone with a circle of radius $r_o = 20$ km on the ground. The mission area is assumed to be the plane of a local approximation of observation of the earth. Each task is a way-point ground target to be observed by satellites. The vertical projections of robots and the positions of targets are stationary and randomly located in the mission area which is a $W \times W$ 2D plane ($W = 60$ km). The tasks that are located in the operation circle of Robot a are contained in the admissible task set $\mathcal{N}_a$. We set the number of targets as 100 and increase the number of satellites from 4 to 20 which is denoted as N_a. The sampling probability for DSTA is fixed to 0.5.

We model the objective functions for the monotone and non-monotone cases as similar functions in the first scenario. The variables in this scenario have the same meanings as those described in Scenario 1, except that the robots are satellites instead of UAVs.

Case 1: Monotone submodular objective function

The overall objective function in the monotone case is modelled as

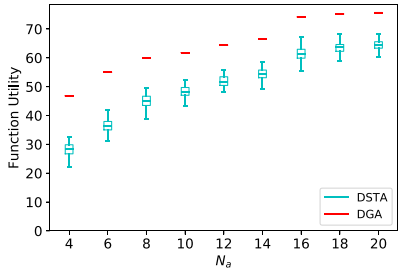
$$f(S) = \sum_{a \in \mathcal{A}} \left[\sum_{j \in \mathcal{T}_a} m_{aj} v_j + \sum_{j \in \mathcal{N}_a \setminus \mathcal{T}_a} m_{aj} v_j e^{-\frac{d_{\min}(j, \mathcal{T}_a)}{d_0}} \right]. \tag{18}$$

The parameters have the same settings as those in Scenario 1 Case 1. The simulation results are reported in Fig. 10. DSTA achieves lower function utilities and displays lower computational and communication complexity compared to DGA, as shown in Fig. 10 a, b, and c. According to Fig. 10 d, the quality of the DSTA solution increases from 60.3% to 85.1%, and the running time of DSTA increases from 31.7% to around 44.9% on average with DGA as a baseline. In other words, when $N_a = 20$, DSTA achieves 55.1% of computational complexity reduction by sacrificing 14.9% of function utility on average compared with DGA. The reason why the number of consensus steps of DSTA is smaller than that of DGA is that DSTA has selected fewer tasks due to random sampling.

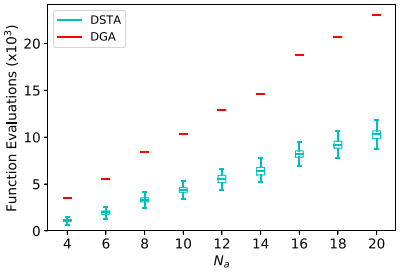
Case 2: Non-monotone submodular objective function

Similarly to Scenario 1 Case 2, the objective function in the non-monotone case in Scenario 2 is modelled as

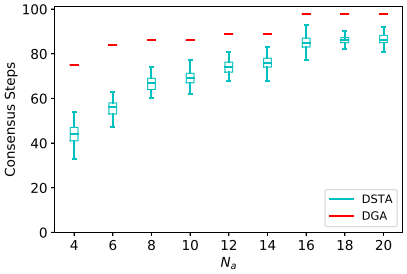
$$f(S) = \sum_{a \in \mathcal{A}} \left[\sum_{j \in \mathcal{T}_a} m_{aj} v_j - \lambda_x \sum_{i,j \in \mathcal{T}_a, i \neq j} x_{ij} \right]. \tag{19}$$



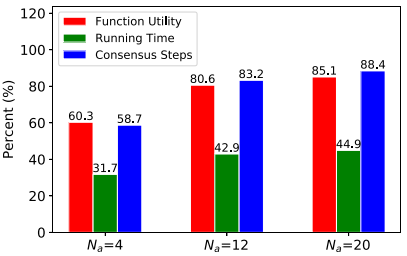
(a) Solution quality represented by objective function utilities.



(b) Computational complexity measured by numbers of function evaluations.



(c) Communication complexity measured by numbers of consensus steps.



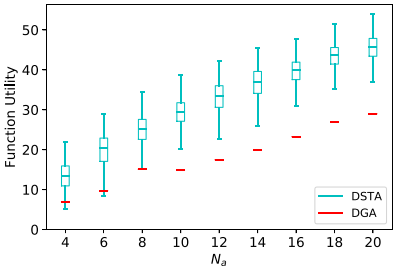
(d) Percentages of performance of DSTA with DGA as a baseline.

Fig. 10 Performance comparison of DSTA and DGA in the monotone case

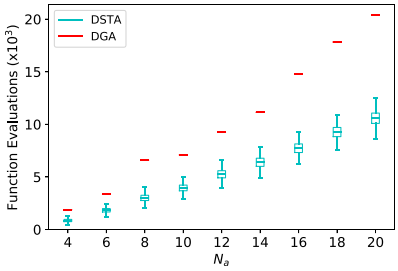
We use the same parameter settings as those in Scenario 1 Case 2, except the number of tasks and the range of mission area. The simulation results are reported in Fig. 11. As shown in Fig. 11a and b, DSTA's runtimes are approximately half with respect to DGA's, and DSTA achieves significantly better solution quality. The reason for the relatively poor performance of DGA in terms of solution quality is that DGA greedily selects those important special tasks at early iterations. As robots select more tasks, the *mgv*s could become negative. Hence, DGA gets trapped in local optima with only a few tasks selected. With the help of the sampling procedure, DSTA avoids those special tasks with a certain probability. Hence, robots cover more tasks and achieve better function values without getting trapped in local optima. This is also the reason why DSTA requires more consensus steps than DGA as shown in Fig. 11c.

6 Conclusions and future work

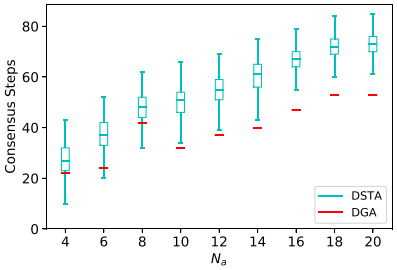
This paper presents an efficient decentralised task allocation algorithm for MRS. Since task allocation problems can be considered as optimisation of a set function subject to a matroid constraint, we have leveraged the submodular maximisation method to solve the task allocation problems for theoretical tractability. The CBBA algorithm (Choi et al., 2009), which is one of the most applied and practical task allocation algorithms, also utilises the submodularity concept and hence provides an approximation guarantee. The issue is that it provides an approximation guarantee only for monotone submodular utility functions.



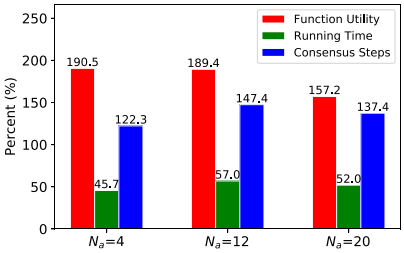
(a) Solution quality represented by objective function utilities.



(b) Computational complexity measured by numbers of function evaluations.



(c) Communication complexity measured by numbers of consensus steps.



(d) Percentages of performance of DSTA with DGA as a baseline.

Fig. 11 Performance comparison of DSTA and DGA in the non-monotone case

To overcome this issue, this paper utilises a sampling process, i.e., drawing task samples from the set of all tasks. Consequently, the proposed task allocation algorithm achieves an expected approximation guarantee not only for monotone submodular utility functions but also for general non-monotone submodular utility functions. Moreover, the computational complexity of the proposed algorithm can be further reduced and adjusted as the introduction of the sampling process allows reduction of function evaluations: it requires to evaluate function values for only task samples, not for all the tasks. The performance of the proposed task allocation algorithm is investigated through theoretical analysis. The results of numerical simulations confirm the validity of the theoretical analysis.

A future research direction would be improving the approximation ratio for both monotone and non-monotone submodular utility functions. There are some gaps between the theoretically achievable approximation guarantee and the actual performance of our algorithm. The key challenge will be how to improve the approximation guarantee for both monotone and non-monotone cases while maintaining reasonable computational complexity. Applying different sampling probabilities for different tasks could better overcome the local minima issue which is subject to our future study. Also, it would be worth to modify CBBA with a random sampling procedure and analyse its performance improvement in the non-monotone case. Another research direction would be further reducing the computational complexity while achieving the same or similar approximation guarantee. In our opinion, this could be achieved by introducing the lazy greedy concept (Minoux, 1978) to the proposed algorithm.

Appendix A Proof of Lemma 1

For the proof, we have three cases to analyse, depending on whether the current pair u is considered at some point of iteration, i.e., $u \in C$, and whether u is already in Q at the beginning of the iteration in Algorithm 4. Note that the size of K_u is kept as small as possible.

i. If $u \notin C$ for whole iterations, $K_u = \emptyset$ and thus the expectation is obtained as:

$$\mathbb{E}[|K_u|] = 0$$

ii. If $u \in C$ and $u \in Q$ at the beginning of the iteration, then $K_u = \emptyset$ for $u \in \mathcal{N}_s$ and $K_u = \{u\}$ for $u \notin \mathcal{N}_s$. Since u is sampled in $\mathcal{N}_s$ with probability p, the expectation is obtained as:

$$\mathbb{E}[|K_u|] = p \cdot |\emptyset| + (1-p)|\{u\}| = 1 - p.$$

iii. If $u \in C$ and $u \notin Q$ at the beginning of the iteration, then K_u contains at most one element for $u \in \mathcal{N}_s$, and $K_u = \emptyset$ for $u \notin \mathcal{N}_s$. According to the properties of partition matroid, if Q becomes dependent after adding u, then Q can remove one element that is in the same partition with u to remain independence. If Q is still independent after adding u, then $K_u = \emptyset$. Therefore, we have

$$\mathbb{E}[|K_u|] \leq p \cdot 1 + (1-p)|\emptyset| = p.$$

In summary, $\mathbb{E}[|K_u|] \leq \max(p, 1-p)$. □

Appendix B Proof of Lemma 2

Let us define a random variable $\mathcal{G}_u$ such that its value is equal to the increase of $f(S)$ when $u \in \mathcal{N}$ is considered, i.e.,

$$f(S) = f(\emptyset) + \sum_{u \in \mathcal{N}} \mathcal{G}_u.$$

Note that since f is assumed to be normalised, $f(\emptyset) = 0$. Given the event $\mathcal{E}_u$, the conditional expectation of $\mathcal{G}_u$ is obtained as

$$\mathbb{E}[\mathcal{G}_u|\mathcal{E}_u] = \sum_{\mathcal{G}_u} P(\mathcal{G}_u|\mathcal{E}_u)\mathcal{G}_u. \tag{20}$$

Here, if u is sampled, $\mathcal{G}_u$ is equal to $\Delta f(u|S'_u)$ with the probability of $P(\mathcal{G}_u|\mathcal{E}_u) = p$, where S'_u is defined as S_u given event $\mathcal{E}_u$. Note that if u is sampled but not in C, $\Delta f(u|S'_u)$ is defined as 0 by convention. Otherwise if u is not sampled, $\mathcal{G}_u$ is zero. Hence, the conditional expectation of $\mathcal{G}_u$ is:

$$\begin{aligned} \mathbb{E}[\mathcal{G}_u|\mathcal{E}_u] &= p\Delta f(u|S'_u) \\ &= p\mathbb{E}[\Delta f(u|S_u)|\mathcal{E}_u] \end{aligned} \tag{21}$$

By the law of total expectation, expectation of $\mathcal{G}_u$ is obtained as:

$$\mathbb{E}[\mathcal{G}_u] = \mathbb{E}[\mathbb{E}[\mathcal{G}_u|\mathcal{E}_u)]] = \sum_{\mathcal{E}_u} P(\mathcal{E}_u)\mathbb{E}[\mathcal{G}_u|\mathcal{E}_u]$$
$$= p\mathbb{E}\left[\Delta f(u|S_u)\right]$$

Hence, the expectation of $f(S)$ is obtained as:

$$\mathbb{E}[f(S)] = \sum_{u\in\mathcal{N}} p\mathbb{E}\left[\Delta f(u|S_u)\right]$$

□

Acknowledgements This material is based upon work supported by the Air Force Office of Scientific Research under award number FA9550-19-1-7032. Any opinions finding and conclusions or recommendations expressed in this material are those of the author(s) and donot necessarily reflect the views of the United States Air Force.

References

Badanidiyuru, A., & Vondrák, J. (2014). Fast algorithms for maximizing submodular functions. In Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), SIAM, pp. 1497–1514.

Buchbinder, N., Feldman, M., Naor, J. S., & Schwartz, R. (2014). Submodular maximization with cardinality constraints. In Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), SIAM, pp. 1433–1452.

Choi, H.-L., Brunet, L., & How, J. P. (2009). Consensus-based decentralized auctions for robust task allocation. *IEEE Transactions on Robotics, 25*(4), 912–926.

Corah, M., & Michael, N. (2018). Distributed submodular maximization on partition matroids for planning on large sensor networks. In 2018 IEEE Conference on Decision and Control (CDC), IEEE, pp. 6792–6799.

Cortés, J. (2008). Distributed algorithms for reaching consensus on general functions. *Automatica, 44*(3), 726–737.

Dias, M. B., Zlot, R., Kalra, N., & Stentz, A. (2006). Market-based multirobot coordination: A survey and analysis. *Proceedings of the IEEE, 94*(7), 1257–1270.

Ding, H., & Castanón, D. (2017). Multi-agent discrete search with limited visibility. In 2017 IEEE 56th Annual Conference on Decision and Control (CDC), IEEE, pp. 108–113.

Dolhansky, B. W., & Bilmes, J. A. Deep submodular functions: Definitions and learning, Advances in Neural Information Processing Systems 29.

Feldman, M., Harshaw, C., & Karbasi, A. (2017). Greed is good: Near-optimal submodular maximization via greedy optimization. In Proceedings of the 2017 Conference on Learning Theory (COLT), Vol. 65, PMLR, pp. 1–27.

Gerkey, B. P., & Matarić, M. J. (2004). A formal analysis and taxonomy of task allocation in multi-robot systems. *The International journal of robotics research, 23*(9), 939–954.

Giannini, S., Petitti, A., Di Paola, D., & Rizzo, A. (2016). Asynchronous max-consensus protocol with time delays: Convergence results and applications. *IEEE Transactions on Circuits and Systems I: Regular Papers, 63*(2), 256–264.

Iutzeler, F., Ciblat, P., & Jakubowicz, J. (2012). Analysis of max-consensus algorithms in wireless channels. *IEEE Transactions on Signal Processing, 60*(11), 6103–6107.

Korsah, G. A., Stentz, A., & Dias, M. B. (2013). A comprehensive taxonomy for multi-robot task allocation. *The International Journal of Robotics Research, 32*(12), 1495–1512.

Kotwal, J. G., & Dhope, T. S. (2015). Solving task allocation to the worker using genetic algorithm. *International Journal of Computer Science and Information Technologies, 6*(4), 3736–3741.

Krause, A., & Golovin, D. (2014). Submodular function maximization. *Tractability, 3,* 71–104.

Kumar, R. R., Varakantham, P., & Kumar, A. (2017). Decentralized planning in stochastic environments with submodular rewards. In AAAI, pp. 3021–3028.

Li, T., Shin, H.-S., & Tsourdos, A. (2019). Efficient decentralized task allocation for uav swarms in multi-target surveillance missions. In 2019 International Conference on Unmanned Aircraft Systems (ICUAS), IEEE, pp. 61–68.

Macal, C. M., & North, M. J. (2010). Tutorial on agent-based modelling and simulation, Journal of. *Simulation, 4,* 151–162.

Minoux, M. (1978). Accelerated greedy algorithms for maximizing submodular set functions. In Optimization Techniques, Springer, pp. 234–243.

Mirzasoleiman, B., Badanidiyuru, A., Karbasi, A. (2016). Fast constrained submodular maximization: Personalized data summarization. In Proceedings of the 33rd International Conference on Machine Learning (ICML), Vol. 48, pp. 1358–1367.

Olfati-Saber, R., & Murray, R. M. (2004). Consensus problems in networks of agents with switching topology and time-delays. *IEEE Transactions on Automatic Control, 49*(9), 1520–1533.

Qu, G., Brown, D., & Li, N. (2015). Distributed greedy algorithm for satellite assignment problem with submodular utility function. *IFAC-PapersOnLine, 48*(22), 258–263.

Qu, G., Brown, D., & Li, N. (2019). Distributed greedy algorithm for multi-agent task assignment problem with submodular utility functions. *Automatica, 105,* 206–215.

Segui-Gasco, P., Shin, H.-S., Tsourdos, A., & Segui, V. (2015). Decentralised submodular multi-robot task allocation. In 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, pp. 2829–2834.

Shin, H.-S., He, S., & Tsourdos, A. (2020). Sample greedy gossip distributed kalman filter. *Information Fusion, 64,* 259–269.

Song, H. O., Lee, Y. J., Jegelka, S., & Darrell, T. (2014). Weakly-supervised discovery of visual pattern configurations. In Advances in Neural Information Processing Systems, pp. 1637–1645.

Sun, X., Cassandras, C. G., & Meng, X. (2019). Exploiting submodularity to quantify near-optimality in multi-agent coverage problems. *Automatica, 100,* 349–359.

Williams, R. K., Gasparri, A., & Ulivi, G. (2017). Decentralized matroid optimization for topology constraints in multi-robot allocation problems. In 2017 IEEE International Conference on Robotics and Automation (ICRA), IEEE, pp. 293–300.