# Task Reallocation in Multi-Robot Formations

Noa Agmon, Gal A Kaminka, Sarit Kraus and Meytal Traub

***Abstract*—This paper considers the task reallocation problem, where $k$ robots are to be extracted from a coordinated group of $N$ robots in order to perform a new task. The interaction between the team members and the cost associated with this interaction are represented by a directed weighted graph. Consider a group of $N$ robots organized in a formation. The graph is the monitoring graph which represents the sensorial capabilities of the robots, i.e., which robot can sense the other and at what cost. The team member reallocation problem with which we deal, is the extraction of $k$ robots from the group in order to acquire a new target while minimizing the cost of the interaction of the remaining group, i.e., the cost of sensing amongst the remaining robots. In general, the method proposed in our work shifts the utility from the team member itself to the interaction between the members, and calculates the reallocation according to this interaction cost. We found that this can be done optimally by a deterministic algorithm, while reducing the time complexity from $\mathcal{O}(N^k)$ to $\mathcal{O}(2^k)$, thus resulting in a polynomial time complexity in the common case where a small number of robots is extracted, i.e., when $k = \mathcal{O}(N)$. We show that our basic algorithm creates a framework that can be extended for use in more complicated cases, where more than one component should be taken into consideration when calculating the robots' cost of interaction. We describe two such extensions: one that handles prioritized components and one that handles weighted components. We describe several other non-robotic domains in which our basic method is applicable, and conclude by providing an empirical evaluation of our algorithm in a robotic simulation.**

***Index Terms*—multi-robot systems, multi-robot formation, multi-robot task reallocation**

## I. INTRODUCTION

This paper discusses a team of $N$ robots engaged in a cooperative task. Specifically, we consider the problem of choosing $k$ team members in order to reassign them to a new task. We assume that all members are capable of participating in the execution of the new task, and the cost of the new task does not depend on the identity of the robots chosen to perform it. Therefore this work concentrates on the problem of choosing $k$ robots such that the performance of the existing task, performed by the remaining group members, will be as efficient as possible.

The general problem of choosing $k$ out of $N$ team members in order to perform a new task is an important problem [6], [9], [10], [13], [14], [18], [30]. Previous work in task reallocation in teams of agents usually concentrate on optimizing some group-objective function that corresponds to the abilities or characteristics of the agents (e.g. [11]). The general problem was proven to be $\mathcal{NP}$-hard as a special case of the Set Partitioning Problem [12].

In this work we suggest a new method for choosing team members for task reallocation that is based on the interaction between team members. Our approach concentrates on the minimization of the cost of interaction between the entities, rather than on the capabilities of the agents. Under this model, the problem remains $\mathcal{NP}$-hard, however we manage to substantially reduce the number of possibilities for optimal assignments, thus we reduce the search domain from order $\binom{N}{k}$ to order $2^k$. This reduction makes it possible to solve the problem faster even using a brute-force algorithm, and in common cases where $k$ is relatively small it is possible to solve the problem both optimally and efficiently (i.e., in polynomial time).

In the model we propose, the set of team members and the interaction between them is represented by a weighted directed graph, where the vertices represent the members, the directed edges represent the interaction (interaction of a vertex $a$ to vertex $b$ could be different from interaction of $b$ to $a$, thus the edges are directed) and edge weights represent the cost of the interaction between them. We assume the team has a leader, which can correspond to a formation leader in the example accompanying us throughout the work. In the general case, this leader is a team member that has only incoming edges and no outgoing edges, i.e., it does not depend on other team member's input in its interaction with them.

The problem used to illustrate methods is the fundamental problem of maintaining a formation by a team of robots; this problem has received considerable attention in the literature (e.g. [2], [3], [16], [21]). In order to maintain the formation, a robot has to monitor one or more robots in the formation. Several common methods for choosing the identity of the robot(s) to be monitored are known [2]. We choose to focus on the method proposed in [16], in which the identity is driven by minimization of the monitoring cost of the robots. Therefore the graph is the monitoring graph which represents the sensorial capabilities of the robots, i.e., which robot can sense the other and at what cost. The problem thus involves extracting $k$ robots in order to perform a new task (for example acquire a new target) while minimizing the monitoring cost of the remaining group.

In particular, in this work we make the following contributions.

1) We introduce a new method in which the problem of choosing $k$ out of $N$ team members is modeled by a graph, and the decision is taken while emphasizing the cost of interaction between the members.
2) We describe a deterministic algorithm for choosing $k$

Preliminary results appeared in Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI'05) [1]

This research is supported by NSF Grant #0705587 and ISF Grant #1685

Noa Agmon is with Department of Computer Science and Applied Mathematics. Weizmann Institute of Science, Israel.
E-mail: noa.agmon@weizmann.ac.il

Gal A Kaminka, Sarit Kraus (also affiliated with UMIACS) and Meytal Traub are with Computer Science Department. Bar Ilan University, Israel.
E-mail: {galk, sarit, traubm}@cs.biu.ac.il

team members while minimizing the cost of interaction between the members of the remaining group assuming that the group has one possible leader. The algorithm reduces the trivial time complexity of $\mathcal{O}(N^k)$ to $\mathcal{O}(2^k)$, thus is polynomial if we assume that $k = \mathcal{O}(\log N)$.

3) We demonstrate the generality of the basic algorithm by showing how it extends to deal with the following cases:
   a) The group has more than one member which can potentially act as the leader. These cases are significantly more complex, as they require that we check possible leader replacements; yet we show that they can still be done optimally by applying an algorithm that works in reduced time (exponential in $k$, hence polynomial for $k = \mathcal{O}(\log N)$).
   b) Not all team members can be chosen for the new task. We show that in some cases the basic algorithm can still be used under these circumstances.
   c) The cost function of the robots is composed of more than one component. In particular, we consider weighted components where we each component is associated with a relative weight, and prioritized components of the cost function, where the components are considered according to their priority.
4) We show that the method we propose, and the basic algorithm within it, is a general method that can be used in several other domains.

We then present an empirical evaluation of the algorithm, using the Player/Stage simulated environment [15]. This implementation illustrates the use of three different task reallocation algorithms—the basic algorithm, and two algorithms that take other considerations into account (weighted and prioritized components)—in different formations.

## II. Related Work

Our problem, task reallocation in multi-robot formation, is on the line between two canonical problems in multi-robot systems, multi-robot pattern formation and maintenance [2], [8], [16], [17], and multi-robot task allocation [9], [10], [14], [20], [22], [28].

The only work, to our knowledge, that considers the problem of task allocation in multi-robot formation is the work by Michael et al. [19]. In their work, they use combinatorial auctions in order to assign $n$ robots to $m$ tasks, i.e., splitting the robots into $m$ groups. They implement their proposed solution in a team of mobile robots in a formation under the constraint of collision avoidance. In our work we also provide an empirical evaluation of the case in which it is required to avoid collisions, however our main general method concentrates on optimize the cost of interaction (sensing) amongst the remaining team. Note that our work does not require communication, as the deterministic algorithm can be executed by each robot individually.

The class of pattern formation and maintenance of a team of mobile robots received considerable attention in the literature [2], [8], [16], [17]. Balch and Arkin [2] discussed the reactive behavior-based techniques for formation control, motivated by nature phenomena. They describe three possible basic techniques a robot can use in order to maintain its position in a formation: leader referenced, neighbor referenced and unit-center referenced.

Several works used the neighbor referenced technique, and examined it in different aspects. Desai [8] offers a graph modeling of a formation using *control graphs*, based on the neighbor referenced technique, where the robots can change their formation by switching between control graphs. Lemay et al. [17] consider the problem of initially assigning robots to specified locations in the formation. Kaminka et al. [16] assume that a robot follows not necessarily the closest neighbor, but the one that the cost of sensing it is minimal, and represent the formation in a sensing graph, i.e., a graph that represent the cost of sensing. We use the work of Kaminka et al. as baseline for our work, thus choose to reallocate $k$ robots to a new task based on the sensing graph.

Balch and Hybinette [3] presented a new behavior based approach for formation control, based on potential functions that attract robots, in designated positions, from their team members in the formation. The advantage of this new approach is that it is easily scalable to large formations, it assumes only local sensing and it is flexible in the sense that it allows to work in various formations and avoid obstacles. This work concentrates on determining the proper formation position for each team member.

Our problem belongs to the ***multi robot task allocation*** (MRTA) domain. Following the taxonomy for MRTA systems given by Gerkey and Mataric [14], our work deals with instantaneous assignments of single-task robots performing multi-robot tasks.

Parker [20] presents a behavior based architecture ALLIANCE for cooperative multi-robot control. This architecture deals with missions that are composed of loosely coupled subtasks that are independent. The task achieving behaviors are divided into lower-lever behaviors that correspond to individual ad-hoc primitives such as obstacle avoidance, and high-level behaviors that corresponds to the team-mission actions. This work, however, does not deal with task reallocation within the team of robots but in the individual behavior of the robot that assures completion of team tasks.

Studies that discuss the problem of choosing $k$ out of $N$ agents in order to perform a new task mostly concentrate on maximizing the profit gained by the optimal performance of the new task. For example in [9], [10] Dias et al. consider market-based coordination methods for assigning tasks to robots. In such systems, tasks are allocated between robots using auctions. Finding the optimal assignment using combinatorial auctions, where bidders can bid on combinations of items, has been proven to be $\mathcal{NP}$-hard [23]. We, on the other hand, concentrate on maximizing the benefit (minimizing the cost) gained by the optimal execution of the original task. The problem can be considered equivalent, if it is perceived as choosing $N-k$ team members to perform the original task. Nevertheless, we aim to find the exact optimal solution where in other studies the problem is handled heuristically.

Other studies discuss allocation of agents to several given tasks. These studies mostly provide heuristic algorithms for

efficient allocation of agents to tasks. In [22], Sander et al. describe task allocation heuristic algorithms for settings in which the agents and tasks are geographically dispersed in the plane.

The problem of designing and forming groups of agents while maximizing some mutual objective is usually referred to as coalition formation. The work of Shehory and Kraus [28] is close to the scenario discussed in this paper. They suggest algorithms for coalition formation among agents, where an agent can be either a member of only one coalition (similar to our case), or coalitions may overlap. They provide heuristic algorithms, rather than an exact optimal solution.

Sandholm and Lesser [25] also dealt with coalition formation, but with self interested agents. Our problem can be perceived as a private case of the general coalition formation studies (forming two coalitions - one for the old task and one for the new).

Tosic and Agha describe a distributed algorithm for generating coalitions based on the current physical configuration of the agents, using maximal cliques [29]. They show that the agents convert to the same coalitions, but their work does not refer to any kind of group utility, as opposed to our work, which maximizes the joint utility of the agents performing the original task.

Another subject in coalition formation which is closely related to the problem discussed here is the problem of coalition structure generation [7], [24], [26]. In this problem, which has been shown to be $\mathcal{NP}$-hard, a division of the agents into coalitions is searched such that the group utility (payoff) is maximized. Sandholm et al. [24] and Dung and Jennings [7] provide algorithms for coalition formation with a guaranteed lower bound from the optimal solution. Sen and Dutta describe a stochastic search approach for searching for optimal coalitions [26]. As opposed to our scenario, whereby an optimal solution is guaranteed, in these studies a sub optimal solution is given.

## III. Problem definition

The motivation for the following representation of the problem comes from the world of multi robot formation maintenance. In this domain, the robots monitor one another in order to maintain a formation. While in the formation, $k$ of the robots are required to leave the team in order to acquire a new target. We wish to extract the robots in such a way that will minimize the disruption to the original formation. A detailed description of the multi robot problem follows the general definition of the problem, which is applicable to other domains (as seen in Section VII).

Let $G = \{P_1, \ldots, P_N\}$ be a group of $N$ homogenous team members. The interaction between team members can be represented by a cost function. Note that this cost function can be generalized into an interaction-based *utility* function. The group has one root - a team member that acts as the leader. The set of members and the interaction between them can be presented as a directed weighted graph $G = (V, E)$, where $v \in V$ are the team members, and the edges represent the interactions. Namely, $(v_i, v_j) \in E$ if an interaction exists between $v_i$ and $v_j$ with a $\text{cost}(v_i, v_j)$, which is the weight of the edge $(v_i, v_j)$. An *Optimal Interaction Tree* (OIT) is built on this graph by simply running Dijkstra's shortest path algorithm between all vertices of the graph and the leader (similar to [16]).

One main constraint is required on this graph as follows:

**Constraint A:** If $(v_1, v_2) \in E(\mathsf{OIT})$ and $(v_2, v_3) \in E(\mathsf{OIT})$, then $\text{cost}(v_1, v_2) + \text{cost}(v_2, v_3) < \text{cost}(v_1, v_3)$.

This means that the triangle inequality exists on the *optimal* tree of $G$, $\mathsf{OIT}(G)$ (but not necessarily on $G$). Intuitively, if a path is chosen to be in $\mathsf{OIT}(G)$, then there is no shorter path between the two edges of the path. Note that this does not mean that the triangle inequality holds for general edges of $G$, as depicted in Figure 1. In this example, the triangle inequality does not hold in $G$, for example $\text{cost}(a, c) + \text{cost}(c, e) \leq \text{cost}(a, e)$, thus clearly the edge $(a, e)$ will not be in $\mathsf{OIT}(G)$. The triangle inequality can still exist between some vertices, for example $\text{cost}(a, b) + \text{cost}(b, c) > \text{cost}(a, c)$.

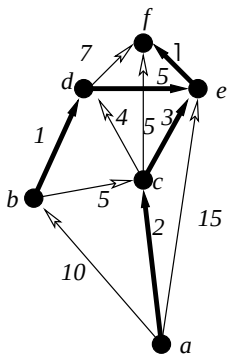


Fig. 1. An example of a case in which the triangle inequality exists in $\mathsf{OIT}(G)$, yet it does not exist in $G$.

The motivation behind this constraint is as follows. In order to achieve a relatively lower time complexity, we would like to restrict the options of the removal of nodes. Specifically, this constraint allows us to consider only leaves or subtrees of the tree. The constraint is necessary in order to avoid cases like the one depicted in Figure 2, in which the structure of $G$ does not follow Constraint A. In this case, illustrated in our robotic domain, the cost of the sensing robot $r_1$ by $r_3$ is 7, and lower than the cost of sensing $r_1$ by $r_3$ via robot $r_2$ ($10 + 10 = 20$). However, the edge $(r_3, r_1)$ does not appear in the graph since $r_2$ lies exactly between $r_3$ and $r_1$, thereby blocking the sensing capabilities of $r_3$. This example conflicts with constraint A since edge $(r_1, r_3)$ is not in the tree, whereas $\text{cost}(r_3, r_1) < \text{cost}(r_3, r_2) + \text{cost}(r_2, r_1)$, and indeed it would have been more profitable to remove $r_2$ and not $r_3$ which is the leaf.

In many cases the leader of the team has unique characteristics that distinguish it from the other team members. Consequently there is no point in discussing the removal of the team leader. However, in some cases all robots are homogenous, including the leader, thus an additional property can be added to the problem, as given in the following

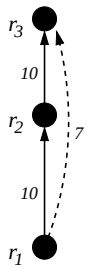


Fig. 2. An example of a case where Constraint A is not fulfilled.

definition.

**Definition 1:** In a graph $G = (V, E), V = \{v_1, \ldots, v_N\}$, a *potential leader* group $L = \{\tilde{v}_1, \ldots, \tilde{v}_M\}$, $1 \leq M \leq N$ is a subset of $V$ containing possible leaders from the group. We denote the size of the potential leader group by $M$.

Having the OIT of the group, $k < N$ vertices should be extracted from the graph. The extraction should be done while satisfying the following basic objectives.

1) The cost of the remaining OIT should be minimal.
2) At least one of the vertices from the potential leader group of $G$ should remain in the graph (this requirement is necessary due to the fact that we assume the remaining formation must have a leader, and only one of the potential leader group members can act as a leader).

It is assumed that all $N$ team members can theoretically be extracted, hence if we are dealing with acquiring a new target or performing a new task, then all members are compatible for the mission (see an exception to this assumption in Section IV-C). Therefore, potentially, the number of different possibilities for extracting the $k$ team members from the group is $\binom{N}{k}$. The algorithms described later in this paper, which work in our settings and under our constraints substantially reduce the complexity to $\mathcal{O}(2^{k/2})$.

Returning to the multi-robot domain, the root of the tree is the formation leader, the original graph is the monitoring multigraph where each vertex represents the location of a robot, and the edges represent the monitoring capabilities and cost of each robot of its peers. As proposed by Kaminka et al. in [16], an Optimal Monitoring Tree, OMT, which is a special case of the OIT, is built on the monitoring multigraph. The OMT describes who each entity should monitor in order to minimize the cost of the sensing path from itself to the leader. The value of monitoring multigraphs and specifically OMTs, is its compatibility with real world scenarios, i.e., in the real world robots usually have limited sensing capabilities and the cost of sensing varies from one sensed object to another—depending on its distance and angle with respect to the sensing robot. When extracting $k$ robots, the objective is to minimize the cost of sensing of the remaining group.

## IV. Reallocation while minimizing costs

In this section we describe three variants of the team reallocation algorithm. In the first and basic algorithm, $k$ robots are extracted from a team of $N$ robots, where the formation has one specified leader that cannot be extracted. In the second algorithm, the formation has a set of possible leaders, where the only restriction in the extraction is that one of the possible leaders will remain in the formation as the leader. In the final variant of the algorithm described in this section we discuss the case in which some robots cannot be extracted from the formation.

### A. Team member reallocation with one possible leader

In this section we describe an algorithm that finds the optimal $k$ vertices that should be extracted from the graph such that the cost of the remaining OIT is minimized. The Tree_Pruning algorithm described in this section, finds the optimal $k$ vertices to be extracted, assuming that the leader cannot be changed.

The following lemma and its corollary provides the motivation behind the algorithm. The lemma proves that the algorithm should concentrate on removing the leaves or subtrees from the $OIT$ rather than arbitrary vertices without all vertices that are in the subtree rooted in that vertex. The reason lies in the fact that the OIT is built in an optimal manner, hence any removal of a vertex without all the subtrees rooted in it will force the vertices of those subtrees to find an alternative path towards the leader. Based on the optimality of the tree, this alternative either will incur the same cost or will be more expensive than the original one.

**Lemma 1.** *Consider an* OIT$(G)$*, satisfying Constraint A. If vertex $v$ that is not a leaf nor the leader is removed, then the sum of the weights of the edges of* OIT$(G \setminus v)$ *will not decrease.*

*Proof:* In a DAG, every vertex that is not a leaf is an articulation vertex, meaning, removing it will disconnect the graph. Therefore all vertices connected to $v$ should find another node to connect to, i.e., all $u_i \in V$ such that $(u_i, v) \in E$ and $(v, u) \in E$, should create a new edge $(u_i, v_j)$ such that the cost of the DAG is minimized.

If $v_j = u$ then the proof is completed, since by Constraint A $\text{cost}(u_i, u) > \text{cost}(u_i, v) + \text{cost}(v, u)$. If $v_j \neq u$ then by minimality of the OIT it follows that if $\text{cost}(u_i, v_j) < \text{cost}(u_i, v)$ then the algorithm would have chosen $u_i$ to point to $v_j$ in the first place, contradicting the minimality of the OIT. ■

**Corollary 2.** *In an* OIT$(G)$ *satisfying Constraint A, the benefit gained from removing a leaf is greater than the benefit gained from removing one of its ancestors.*

The following definitions are used throughout the description of the algorithm.

**Definition 2:**

2.1 A *palindromic composition* of a number $k$ is a collection of one or more positive integers whose sum is $k$. The number of palindromic compositions of a number $k$ is $2^{\frac{k}{2}}$ [4].

2.2 A *bundle* originating in vertex $v$ is the subtree rooted in vertex $v$. Vertex $v$ *nests a bundle* of size $t$ if the bundle originating in it has $t$ vertices, including $v$.

2.3 In a directed tree $G = (V, E)$ where all paths are directed to the root of the tree, if $(v, u) \in E$ then $u$ is called $v$'s *pivot*.

From Corollary 2 we can assert that the gain from removing bundles or leaves from OIT$(G)$ will be greater than the gain from removing arbitrary vertices from the graph. This fact motivates the algorithm Tree_Pruning, which exhaustively searches all possible combinations of leaves and bundles and picks the combination which results in the highest reduction in cost to the remaining OIT$(G)$.

The Tree_Pruning algorithm (1) works as follows. First, it creates a table in which it stores the vertices in levels $1, \ldots, k$, where each level $i$, $1 \leq i \leq k$, contains vertices that nest a bundle of size $i$. For each of these elements it indicates the gain from removing the bundle originating in that vertex. This gain is simply the sum of all the costs of edges in this subtree, including the cost of the edge going from the root of the subtree to its pivot. After the table is created, the algorithm starts checking all palindromic compositions of the number $k$. For each composition $\alpha_1 + \alpha_2 + \ldots + \alpha_t$ the algorithm first checks whether it is feasible, i.e., whether there are components of sizes $\alpha_1, \ldots, \alpha_k$. If so - for each $\alpha_i$ it checks for the element with maximal gain in level $\alpha_i$ of the table. If the algorithm picks up non-disjoint bundles, then it checks the gain of removing each element of the non-disjoint set alone. Summing up the gains from removing the composition is compared to the current maximal gain, and the set resulting in the higher gain is saved. Finally, after all palindromic compositions of $k$ are examined, the algorithm returns the set with the highest cost reduction upon its removal.

**Algorithm 1** Algorithm Team$_k$ = Tree_Pruning$(G = (V, E), k)$

1: **for** each leaf $v \in V$ **do**
2: Start building a $k$-bundle bottom-up:
3: $C_{best} \leftarrow 0$ and Team$_k \leftarrow \emptyset$.
4: Add each subtree of size $1 \leq t \leq k$ to the table in row $t$ and calculate its cost.
5: Sort all elements in each row according to their cost.
6: Generate a palindromic composition of the number $k$ and sort each composition from left to right in decreasing order.
7: **for** each possible composition $C_j$ of $k = \alpha_1 + \alpha_2 + \ldots + \alpha_t$ **do**
8: Check whether the composition is feasible.
9: **for** each $\alpha_i$ in the composition, $i = 1, 2, \ldots, t$ **do**
10: Pick highest order unmarked element from row $t_i$ and mark it.
11: **if** the elements are not disjoint, then check all possibilities: **then**
12: Pick element with highest $\alpha_i$, next don't pick it and pick element with next highest $\alpha_i$ and so on.
13: Calculate the cost of the composition $C_j$.
14: **if** $\text{cost}(C_j) \geq C_{best}$ **then**
15: $C_{best} \leftarrow C_j$ and Team$_k \leftarrow$ current composition.
16: Return Team$_k$

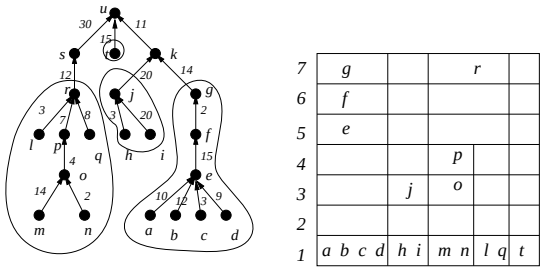


Fig. 3. An example of 7-bundling of a tree.

**Theorem 3.** *The* Tree_Pruning *algorithm finds the optimal $k$ vertices to be removed from the group such that the cost of the remaining* OIT *is minimized.*

*Proof:* As seen in Corollary 2, the optimal benefit for the remaining tree is obtained by removing vertices that are not articulation points in the graph. Therefore the examination of all removal possibilities of leaves and bundles, as done by the algorithm, assures that the optimal group of $k$ vertices will indeed be removed. ■

**Time Complexity:** The time complexity of the preliminary work of building the table is $\mathcal{O}(N)$, as each vertex is visited once. The sorting of the rows will cost additional $\mathcal{O}(N \log N)$ time. The number of palindromic compositions of a number $k$ is $2^{\frac{k}{2}}$ [4]. The algorithm might check each composition (the worst case) $N$ times, if the chosen elements are not disjoint. Assuming that each approach to an element takes $\mathcal{O}(1)$ steps (depending on the data structure used), each composition is calculated in (the worst case) $\mathcal{O}(N)$ steps.

Therefore the total time complexity of the algorithm is $N \log N + N 2^{\frac{k}{2}}$. Assuming that $k$ is in order $\log N$, then the time complexity of the algorithm is $\mathcal{O}(N \log N + N\sqrt{N}) = \mathcal{O}(N^{1.5})$.

### *B. Team member reallocation with multiple possible leaders*

Removing the leader $v_{lead}$ can significantly decrease the total cost of the graph in cases where the weights of edges entering $v_{lead}$ are considerably higher than the other weights in the graph. Therefore when examining the vertices, it can be highly profitable to examine removal of both leaves (or bundles) and the leader. The removal of the leader is possible only in cases where the potential leader group of the formation is of a size greater than one. Consider the case in which robots move in a formation. In this case, it is possible that several robots in the formation can lead the group. This is obvious in cases in which all robots are homogenous, whereby, theoretically, the number of robots in the potential leader group is $N$.

The order of extraction of the leading vertex is important, since changing the leader might result in a different structure of the OIT. In particular, it might result in changing the identity of the leaves and bundles. If we wish, for example, to extract three vertices from the graph, the result may vary if we pick a leaf, leader and a leaf from the new tree, as opposed to picking, say, two leaves and then the leader. Note that in some cases removal of a leaf might result in changing the leader,

depending on the leader election algorithm. However, in our case we assume that this does not happen.

In the extreme case in which $M = N$, the number of different possibilities for choosing $k$ vertices - a combination of leaves and leaders, is $\mathcal{O}(2^k)$. See Figure 4 for an example.

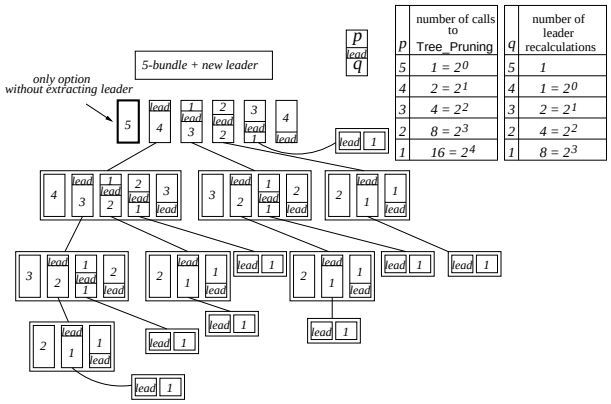


Fig. 4. An example of the search tree of all possibilities of extracting $k = 5$ vertices from the graph where the leader can be elected as well in each step.

In the first level, $l = 1$, we can either pick $k$ vertices using the Tree_Pruning algorithm (meaning, we do not change the leader), or change the leader first, second and so on until the $k$'th vertex is selected. Each of these options except the former branches out similarly in the next level ($l = 2$) where $k$ decreases by one. If $M \geq k - 1$ then the formal time complexity analysis is as follows.

Assume that a leader is chosen in a round where $t$ vertices remain to be chosen, $1 \leq t \leq k$. It is possible to pick $p$ vertices before the leader is replaced and $q$ afterwards, $0 \leq p, q \leq t - 1$ and $p + q = t - 1$. Therefore there are $t$ different choices of the order in which the leader is extracted, in addition to the choice where the leader is not replaced. When a leader is replaced, the calculation of the $p$ vertices chosen prior to it is obtained simply by running Tree_Pruning for $p$. The remaining $q$ launch an additional level where, again, they branch into $q + 1$ new options. In order to calculate the complexity of finding the best allocation, we need to calculate the number of times each $p$ appears (the $q$ is calculated in the next level). As demonstrated in the table below, each number $i = 1, \ldots, k$ appears as $p$, i.e., above the leader line, $2^{k-i}$ times. Using Tree_Pruning, the complexity of each extraction of size $i$ is $N \log N + N2^{\frac{i}{2}}$. Therefore the total complexity is $\sum_{i=1}^{k} 2^{k-i} \cdot (N \log N + 2^{\frac{i}{2}}) = \mathcal{O}(2^k N \log N) + N \sum_{i=1}^{k} 2^{k-\frac{i}{2}} = \mathcal{O}(2^k N \log N) + \mathcal{O}(2^k N) = \mathcal{O}(2^k N \log N)$. If $k = \mathcal{O}(\log N)$, then the complexity of choosing the members is $\mathcal{O}(N^2 \log N)$.

To this complexity we need to add the cost of recalculating the graph after a leader is extracted. As shown in [5], the complexity of calculating the OIT of a graph of size $N$ given the identity of the leader vertex is simply running Dijkstra's shortest paths algorithm that takes $\mathcal{O}(N^3)$. If there are $\tilde{M}$ potential leaders, then the complexity is $\mathcal{O}(N^3 M)$. Hence here the time complexity can be bounded from above by the total number of $q$s, times $\mathcal{O}(N^3 M)$. The total number of $q$s, as demonstrated in Figure 4, is $\sum_{i=1}^{k-1} 2^{(k-1)-i}$, therefore the total time complexity is $N^3 M \sum_{i=1}^{k-1} 2^{(k-1)-i} = \mathcal{O}(N^3 M 2^k)$. Again, in our case $k = \mathcal{O}(\log N)$, hence the final complexity of the algorithm is $\mathcal{O}(N^4 M)$.

### C. Addressing non-removable robots

Sometimes, there may be a requirement that some team members must not be extracted from the group. For example, if the robots moving in the formation are heterogenous, some might be incapable of performing the new task and thus cannot be chosen to do it. In another example, if the formation itself defines critical positions that cannot be deserted, it will dictate the identity of the robots that cannot be extracted. In both cases—driven by the old or new task requirements—some robots must remain in the original team. Converting it to our graph problem, if vertices should be extracted only from a subgroup $G_1$ of $G$, $G_1 \subseteq G$ and $|G_1| \geq k$, then a small variation of the basic algorithm can be used in some cases. First, if $|G_1|$ is exactly $k$, then the only option is that all vertices in $G_1$ be extracted.

**Definition 3:** In an OIT graph $G$, a vertex $v \in V(G)$ is called a *bundle blocker* if it cannot be extracted from the graph and the bundle above it stops spreading.

In our case, the bundle blockers are all vertices $u_i$ such that $u_i \notin G_1$. If the bundle blockers lie in accumulating levels of depth $k$ or more, then a simple variation of Tree_Pruning can be used in order to find the optimal $k$ vertices to be extracted. In this variation of Tree_Pruning, the algorithm should be run on the OIT graph $G$, but should stop at bundle blockers or at depth $k$, whichever comes first.

## V. Multiple components in the cost function

When establishing the nature of the cost function, it is possible that more than one property will be taken into consideration. For example, one might want to consider both the cost of the remaining formation and the cost of the new formation. We consider two manners in which more than one consideration can be incorporated in the cost function: prioritized components and weighted components. We show examples in which the Tree_Pruning algorithm can be used in both cases.

Note that the ground rule which stands at the base of the use of the Tree_Pruning algorithm is that it is more profitable to remove leaves and bundles of the OIT. Hence any scenario in which we can incorporate the use of Tree_Pruning must apply to this rule. In the following examples we were motivated by the stability of the formation, whereby any change in the pivot of robots might cause instability of the formation. Therefore the following case suits the requirement that only bundles or leaves be removed, and thus the Tree_Pruning algorithm perfectly suits this problem.

### A. Prioritized cost components

In prioritized cost components, the components are presented in a prioritized list, namely, component one is more important than component two and so on. Therefore we first minimize the cost according to the first priority. In case of a

tie, we examine the component listed second in the priorities, and if that results in a tie we move on to the third priority component and so on.

A good example of this would be when we wish to minimize events that might undermine the remaining group's formation stability as well as minimize the monitoring cost. For instance, we assume that robots leaving the formation in order to acquire a new target continue in a straight line from their current location towards the new target. First, we would like to cause minimal changes to the current OIT, so that robots will not have to switch pivots and thereby cause temporary instability. Thus only leaves or bundles should be removed, and the leader should remain intact. Second, we wish to minimize collisions between the robots leaving for the new target and the ones remaining in the formation. As seen in Figure 5, if $v_2$ moves straight towards $t_G$ and maintains its predefined velocity, it will intersect with $v_3$. In addition, if $v_5$ moves towards $t_G$ then at some point it will block $v_4$'s view of its pivot, $v_3$. Third, we wish to minimize the monitoring cost of the remaining formation. Hence the prioritized components list is as follows.

1) Minimize collisions between the robots leaving for the new target and the ones remaining in the formation
2) Minimize the incidents of robots leaving the formation while, at some point, crossing an OIT edge, thus hiding the pivot of some robot remaining in the formation and potentially causing it to divert from the group formation.
3) Minimize the remaining group's monitoring cost.

The Prioritized_Pruning algorithm works as follows. First, assuming that the robots are homogeneous, it is simple to calculate the expected intersections between paths of robots leaving the formation and the remaining OIT vertices (robots) and edges. For each robot $r_i$ (vertex $v_i$) that leaves towards goal point $p_G$, the algorithm checks against all robots $r_j \neq r_i, r_j \neq r_{lead}$ whether the path of $r_i$ hides the outgoing edge from robot $r_j$ (vertex $v_j$). If so, it adds $t_j$ to the entry of $r_i$ in a pre-specified Table under column $E$ (see for example Figure 5). If the robots themselves intersect, then $r_j$ is added to Table under column $I$. After creating this table, Tree_Pruning is run on the graph where three features are examined in each step: $I$ intersections, $E$ intersections which are extracted from the Table, and the cost incurred by the remaining OIT (in this order).

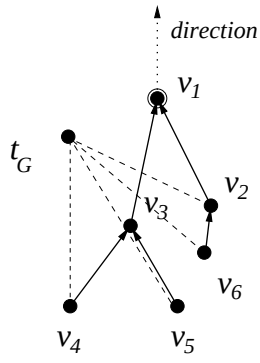


Table:

| | *E intersection* | *I intersection* |
|---|---|---|
| $v_2$ | Ø | $v_3$ |
| $v_3$ | Ø | Ø |
| $v_4$ | Ø | Ø |
| $v_5$ | $v_4$ | Ø |
| $v_6$ | $v_4$ $v_5$ | Ø |

Fig. 5. An example of a path/edge intersection.

The Prioritized_Pruning algorithm is guaranteed to find the $k$ robots that will minimize the potential disturbance to the formation satisfying the criterions we defined. The time complexity of the algorithm is composed of two steps. In stage 1, a simple brute force algorithm that finds the intersections will take $\mathcal{O}(N^2)$ steps by simply comparing each pair of robots. Stage 2 is similar to the Tree_Pruning algorithm with an addition of maximum $\mathcal{O}(k)$ comparisons at each step, hence the complexity is $\mathcal{O}(N^{1.5}\log N)$ (assuming that $k = \mathcal{O}(\log N)$), and altogether the complexity is $\mathcal{O}(N^2)$.

**Algorithm 2** Algorithm $\mathsf{Team}_k = \mathsf{Prioritized\_Pruning}(G = (V, E), k, t_G)$

**for** each vertex $v_i \in V$ such that $v_i$ is not the leader **do**
  Go over all vertices of the graph except for vertices in the bundle originating in $v_i$.
  **if** the outgoing edge of vertex $v_j$ intersects the path of $v_i$ on its course towards $t_G$ **then**
    Add $v_j$ to $\mathsf{Table}(v_i, E)$.
  **if** $v_i$ on its course towards $t_G$ intersects $v_j$ **then**
    Add $v_j$ to $\mathsf{Table}(v_i, I)$.
Run procedure $\mathsf{Tree\_Pruning}(G, k)$ with the following modifications.
Set $E_{best} \leftarrow \infty$, $I_{best} \leftarrow \infty$ and $\mathsf{Team}_k \leftarrow \emptyset$.
Check the number of $E$ and $I$ intersections between members of the current chosen elements and the remaining ones and store them in $E_{cur}$ and $I_{cur}$, respectively.
**if** $I_{cur} < I_{best}$ **then**
  $I_{best} \leftarrow I_{cur}$ and
  $\mathsf{Team}_k \leftarrow$ current composition.
**if** $I_{cur} = I_{best}$ and $E_{cur} < E_{best}$ **then**
  $I_{best} \leftarrow I_{cur}$ and $\mathsf{Team}_k \leftarrow$ current composition.
**if** $I_{cur} = I_{best}$ and $E_{cur} = E_{best}$ **then**
  Check the difference between the cost of the composition and save the best of two choices, as done in Tree_Pruning .
Return $\mathsf{Team}_k$.

### *B. Weighted cost components*

Formally, weighted cost components allow us to assign an accumulating percentage for each component, and choose the option resulting in the minimized cost value $U$. Each of these cost values is composed of $l$ different components $\{u_1, \ldots, u_l\}$, and $w_t$ is the weight of the cost component $u_t$, where $\sum_{t=1}^{l} w_t = 1$. Therefore the weighted cost value $U$ is calculated as: $U = \sum_{t=1}^{l} w_t^i u_t^i$

In the following example we assume that robots leaving the formation remain in their current location, i.e., they will change their relative position to other robots. Our objective is to minimize both the remaining group's monitoring cost and the monitoring cost of the robots leaving the formation, according to their location at the moment they leave the formation. This is applicable in cases where a subgroup of robots is required to leave the formation and create a formation of its own with minimal sensorial cost of the new formation. Our original assumption is that we wish to minimize disruptions of the original formation, thus we will examine the removal of only leaves and bundles, and leave the leader

intact. This property will allow us to use the Tree_Pruning algorithm in this case. Note that if we consider only the monitoring cost of the leaving robots, we might have needed to remove vertices that are not necessarily leaves or bundles. For example, if $k = N - 2$, then the optimal choice of removal would be the removal of the minimal edge in the OIT, regardless of its location if only the monitoring cost of the extracted group is considered. Therefore, as we have shown in the previous subsection, the initial assumption that we remove only leaves and bundles is crucial for the use of the Tree_Pruning algorithm and ensures its complexity.

Minimizing the remaining group's sensorial cost (first component) contradicts, in most cases, the minimization of the extracted group's sensorial cost. This contradiction is exemplified in Figure 6, and results straightforward from the fact that the first minimization requirement would cause the removal of the most expensive edges, while the second component would require the removal of the least expensive edges from the graph.

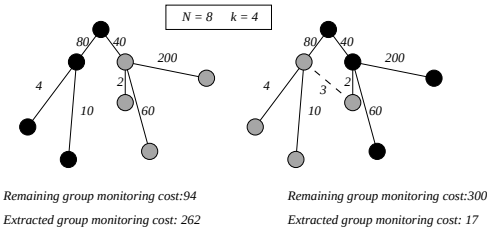


Fig. 6. An example of a contradiction between the two cost components: the remaining group's OIT cost and the extracted group's OIT cost. Here $N = 8, k = 4$ and the optimal choice of nodes for removal is colored in gray, while the remaining nodes are colored in black.

Use $w_1$ to denote the weight of the cost component of the remaining formation's monitoring cost , and $w_2$ to denote the weight of the cost component of the extracted formation's monitoring cost. The Weighted_Pruning algorithm, then, works as follows. For each possible choice of $k$ robots, the algorithm calculates the cost of the remaining OIT multiplied by $w_1$, the cost of the extracted OIT multiplied by $w_2$, and it sums the two values. If the resulting value is lower than the lowest value obtained so far, this is saved as the potential optimal choice. After all choices have been checked, the choice with the optimal cost is reported.

**Algorithm 3** Algorithm $\mathsf{Team}_k = \mathsf{Weighted\_Pruning}(G = (V, E), k, w_1, w_2)$

1: Run procedure $\mathsf{Tree\_Pruning}(G, k)$ with the following modifications.
2: Set $E_{best} \leftarrow \infty$, $I_{best} \leftarrow \infty$ and $\mathsf{Team}_k \leftarrow \emptyset$.
3: $C_j \leftarrow$ current composition.
4: $C_R \leftarrow$ cost of $\mathsf{OIT}(G \setminus C_j)$, and $C_E \leftarrow$ cost of $\mathsf{OIT}(C_j)$
5: $C_{cur} \leftarrow w_1 \times C_R + w_2 \times C_E$.
6: **if** $C_{cur} < C_{best}$ **then**
7: $C_{best} \leftarrow C_{cur}$ and $\mathsf{Team}_k \leftarrow C_j$.
8: Return $\mathsf{Team}_k$.

Algorithm Weighted_Pruning is guaranteed to find the $k$ robots that will minimize the total cost according to the weights we received from the user. The time complexity of the algorithm is identical to the time complexity of the Tree_Pruning algorithm, since we go over the entire graph only once for each possible composition,, which leaves us with a time complexity of $N2^{\frac{k}{2}}$ needed to check all compositions.

## VI. Empirical Evaluation

We implemented the three algorithms described herein, Tree_Pruning, Prioritized_Pruning and Weighted_Pruning, in order to perform an empirical evaluation of the algorithms. The implementation was done using the Player/Stage simulation package [15], a practical and popular development tool for both simulated and real robots.

We simulated 16 robots, traveling in one of three formations commonly tested in general multi-robot formation problems (e.g. [16]) - see Figure 7: *A.* Diamond *B.* Triangle *C.* Arrowhead. The edges between the robots in each formation were given weights according to the cost of sensing, similar to the weights given in [16]. In the first step, the robots built a spanning tree, instructing each robot which other robot to monitor in order to minimize the cost of sensing inside the formation. The Spanning trees are indicated by bold arcs in Figure 7.

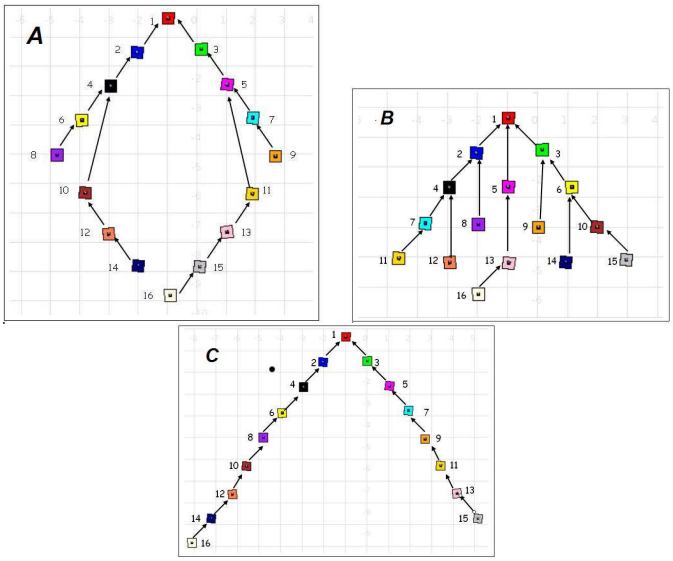


Fig. 7. Three different formations tested in our Player/Stage simulation. The arcs represent the edges of the minimal sensing tree from all robots to the leader (robot 1).

Following the first phase of constructing the minimal spanning tree, we executed the three algorithms on the formation: Tree_Pruning, Prioritized_Pruning and Weighted_Pruning. We were interested in revealing which robots were extracted as the output of each algorithm, or more specifically what different outputs would be returned by each algorithm for the same formation. We wanted to test whether the Prioritized_Pruning algorithm would indeed answer possible problems raised by the use of Tree_Pruning in the multi-robot formation domain. We continue with a

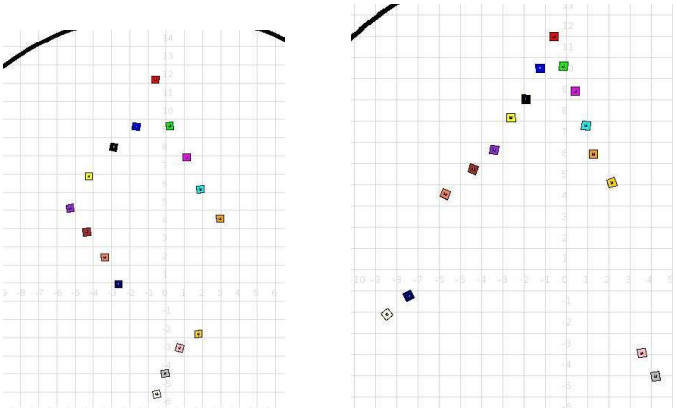

Fig. 8. Execution of the Tree_Pruning algorithm on formation A (left) and C (right). The snapshot was taken approximately $15$ seconds after the extraction.

description of the results of the algorithms' performance on the chosen formations.

First, the robots executed Tree_Pruning in order to reallocate $4$ out of the $16$ team members to a new task. The extracted robots were instructed to remain in their position while the remaining formation continued their movement in their initial direction. We obtained the following results. In formation A, robots $11, 13, 15$ and $16$ were extracted from the team (Figure 8). In formation C, robots $13, 14, 15$ and $16$ were extracted (Figure 8). The more interesting results were obtained in formation B, where robots $8, 9, 12$ and $14$ were extracted. Since the extracted robots remained in place and the remaining formation continued straight, robot $8$ collided with robot $16$ (see Figure 9).

The choice of robots to be extracted in formation $B$ using Tree_Pruning strengthens the motivation to use the Prioritized_Pruning algorithm. Indeed, when executing the Prioritized_Pruning algorithm for formation $B$, the extracted set of robots was $9, 11, 14$ and $15$. In this case, the target point of the robots was behind the formation, simulating a similar behavior given to algorithm Tree_Pruning, and thus emphasizing how the Prioritized_Pruning algorithm is able to solve the problem evolving from the use of the Tree_Pruning algorithm (see Figure 9).

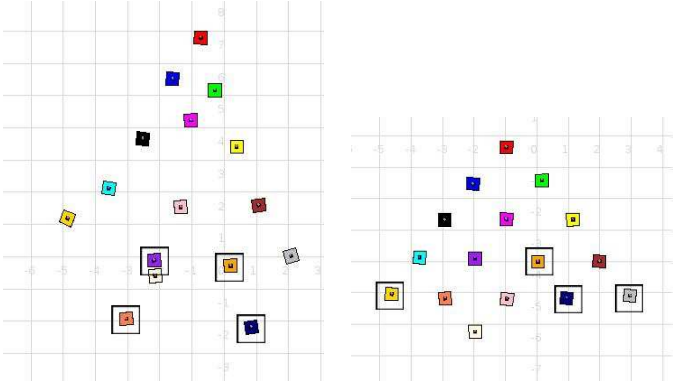

Fig. 9. Execution of the Tree_Pruning (left) and the Prioritized_Pruning (right) algorithms on formation B. The extracted robots are denoted by a surrounding square.

When we used the Weighted_Pruning algorithm with $w = 0.5$, the extracted robots were $11, 13, 15$ and $16$ in formation A, which is the same set that was extracted by Tree_Pruning. However, in formations $B$ and $C$ when adding the consideration of the weight of the extracted team of robots, the set of extracted robots was different. In formation $B$ the set of extracted robots was $4, 7, 11, 12$, and in formation C the set was $10, 12, 14$ and $16$. We checked the output also given other weights ($w = 0.2$ and $w = 0.8$), however in both formations the resulted chosen team was similar. This can be explained by the fact that the weight of edges between the robots that cannot sense one another is $\infty$. Consequently any choice of a set of robots that would be extracted such that even one robot would remain disconnected from the other team members would receive a weight of $\infty$, and therefore that set would not chosen.

## VII. Applicability in additional domains

The general representation of the problem as *team member* reallocation, makes it applicable in other domains, in addition to the multi-robot formation domain, which motivated the current study.

One example is a variation of the ***dependency tree*** [27]. A dependency tree $G = (V, E)$ describes a group of $N$ tasks (vertices) with prerequisite relation, i.e., an edge $(u, v) \in E$ exists if $u$ has to be executed before $v$, and $\mathrm{cost}(u, v)$ is the cost of executing $v$ after $u$. The root of the tree is, then, the task that has to be executed last. In our case, we use a slight variation of the dependency tree. Here, we are given one task that should be conducted last and the interaction between all other tasks. If two tasks $v_1$ and $v_2$ are independent, then if $v_1$ is executed before or after $v_2$ their cost will be the same. If $v_1$ and $v_2$ are dependent, then without loss of generality, $v_1$ can rely on the fact that $v_2$ will perform a part of its task, thus $\mathrm{cost}(v_1, v_2)$ in this case will be smaller than the cost in the independent case. The OIT describes the optimal tree of execution of the tasks. The requirement is to remove $k$ tasks from the group such that the cost of the remaining execution tree is minimized.

The ***warehouse assembling*** problem presents an additional example in which the OIT is applicable. In the warehouse assembling problem we are given a set of $N$ warehouses located in $N$ distinct positions, and all the trucks are heading towards one main warehouse. The vertices of the graph represent the warehouses, and the edges represent the distances between two warehouses (note that triangle inequality does not apply). The objective is for any number of trucks to visit all warehouses in minimal time. The OIT represents the optimal tree of paths from all warehouses to the main warehouse. The number of trucks $t$ is, then, the number of leaves in the OIT. The requirement is to close $k$ warehouses in order to cut back expenses while not increasing the value $t$, and thus remain with the assembling tree with the lowest cost.

The last problem is the ***network broadcast*** problem, in which we are given a network with one source vertex that should constantly broadcast messages to the rest of the network. The edges represent the cost of the link between every

two vertices, and the OIT is the optimal broadcast tree. Once again in this case we are required to remove $k$ vertices in order to cut back expenses and remain with a broadcast tree with the lowest cost.

## VIII. Conclusions

We considered the task reallocation problem in multi-robot formation. In this problem, a team of $N$ robots move in a formation, and $k$ of them need to be extracted from the group. The extraction is done considering the interaction cost between the team members—in our case the cost of sensing inside the formation—where the goal is to minimize the interaction cost between the remaining team members (and thus maximizing the utility function of the remaining group). A summary of our contributions in this paper is as follows.

- We introduce a new method in which the problem of reallocating $k$ out of $N$ team members to a new task is modeled by a graph, and the utility function is based on the interaction cost between the team members.
- We describe a deterministic algorithm for the reallocation problem which reduces the time complexity of the solution from $\mathcal{O}(N^k)$ to $\mathcal{O}(2^k)$. This result is shown for both cases in which the formation can have either one or more possible leaders.
- We generalize the use of the basic reallocation algorithm for cases in which the cost function has more than one component. In particular, we consider weighted components and prioritized components of the cost function.
- We describe an empirical evaluation of the algorithm and its variations using the Player/Stage simulated environment.
- We show that the method we propose that focuses on the interaction between team members, and the basic algorithm within it, is a general method that can be used in several other domains different from the multi-robot formation domain.

There are several areas we plan to pursue in our future work, which include considering the cost/utility of the robots leaving the formation and uncertainty in the actual cost of interaction.

## References


[1] N. Agmon, G. A. Kaminka, and S. Kraus. Team member-reallocation via tree pruning. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI)*, pages 35–40, 2005.

[2] T. Balch and R. Arkin. Behavior based formation control for mutlirobot systems. *IEEE Transactions on Robotics and Automation*, 14(12):926 – 939, 1998.

[3] T. Balch and M. Hybinette. Social potentials for scalable multi-robot formations. In *IEEE International Conference on Robotics and Automation (ICRA '00)*, volume 1, pages 73 – 80, 2000.

[4] P. Chinn, R. Grimaldi, and S. Heubach. The frequency of summands of a particular size in palindromic compositions. *Ars Comb.*, 69, 2003.

[5] T.H. Corman, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[6] T. S. Dahl, M. Matarić, and G. S. Sukhatme. Multi-robot task allocation through vacancy chain scheduling. *Journal of Robotics and Autonomous Systems*, 57(6):674–687, 2009.

[7] V. D. Dang and N. R. Jenings. Generating coalition structures with finite bound from the optimal guarantees. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-02)*, pages 564–571, 2004.

[8] J. P. Desai. A graph theoretic approach for modeling mobile robot team formation. *Journal of Robotic Systems*, 19(11):511–525, 2001.

[9] M. B. Dias and A. Stentz. A free market architecture for distributed control of a multirobot system. In *Proceedings of the Sixth Conference on Intelligent Autonomous Systems (IAS-6)*, pages 115–122, 2000.

[10] M. B. Dias, R. Zlot, M. Zinck, J. P. Gonzalez, and A. Stentz. A versatile implementation of the traderbots approach for multirobot coordination. In *Proceedings of the Eighth Conference on Intelligent Autonomous Systems (IAS-8)*, 2004.

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

[12] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

[13] B. P. Gerkey and M. J. Matarić. Murdoch: publish/subscribe task allocation for heterogeneous agents. In *Proceedings of the fourth international conference on Autonomous agents (AGENTS-00)*, pages 203–204, 2000.

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

[15] Brian P. Gerkey, Richard T. Vaughan, and Andrew Howard. The player/stage project - tools for multi-robot and distributed sensor systems. In *Proceedings of the International Conference on Advanced Robotics*, pages 317–323, Coimbra, Portugal, Jul 2003.

[16] G. A. Kaminka, R. Schechter-Glick, and V. Sadov. Using sensor morphology for multirobot formations. *IEEE Transactions on Robotics*, 24(2):271–282, 2008.

[17] M. Lemay, F. Michaud, D. Letourneau, and J. M. Valin. Autonomous initialization of robot formation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2004.

[18] K. Lerman, C. Jones, A. Galstyan, and M. Matarıc. Analysis of dynamic task allocation in multi-robot systems. *Inernational Journal of Robotics Research*, 25(3):225–241, 2006.

[19] N. Michael, M. M. Zavlanos, V. Kumar, and G. J. Pappas. Distributed multi-robot task assignment and formation control. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pages 128–133, Pasadena, CA, 2008.

[20] L. E. Parker. On the design of behavior-based multi-robot teams. *Advanced Robotics*, 10(6):547–578, 1996.

[21] W. Ren and N. Sorensen. Distributed coordination architecture for multi-robot formation control. *Robotics and Autonomous Systems*, 56(4):324–333, 2008.

[22] P. V. Sander, D. Peleshchuk, and B. J. Grosz. A scalable, distributed algorithm for efficient task allocation. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-02)*, pages 1191–1198, 2002.

[23] T. Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135(1-2):1–54, 2002.

[24] T. Sandholm, K. Larson, M. Andersson, O. Shehory, and F. Tohme. Coalition structure generation with worst case guarantees. *Artificial Intelligence*, pages 209–238, 1999.

[25] T. Sandholm and V. Lesser. Coalitions among computationally bounded agents. *Artificial Intelligence*, 94(1):99–137, 1997.

[26] S. Sen and S. Dutta. Searching for optimal coalition structures. In *Proceedings of the Fourth International Conference on Multiagent Systems*, pages 287 – 292, 2000.

[27] K. C. Sevcik. Characterizations of parallelism in applications and their use in scheduling. In *Proceedings of ACM Conference on Measurement and Modeling of Computation Systems*, pages 171–180, May 1989.

[28] O. Shehory and S. Kraus. Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101(1-2):165–200, 1998.

[29] P. Tosic and G. Agha. Maximal clique based distributed group formation for autonomous agent coalitions. In *Coalitions and Teams Workshop, AAMAS*, 2004.

[30] R. M. Zlot and A. Stentz. Market-based multirobot coordination for complex tasks. *International Journal of Robotics Research, Special Issue on the 4th International Conference on Field and Service Robotics*, 25(1):73–101, 2006.