Team Member Reallocation via Tree Pruning

Noa Agmon and **Gal A Kaminka** and **Sarit Kraus** *
Department of Computer Science
Bar Ilan University
Ramat Gan, Israel
{segaln, galk, sarit}@cs.biu.ac.il

Abstract

This paper considers the task reallocation problem, where k agents are to be extracted from a coordinated group of N agents in order to perform a new task. The interaction between the team members and the cost associated with this interaction are represented by a 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. Following this example, the team member reallocation problem this paper deals with 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. In general, the method proposed here shifts the utility from the team member itself to the interaction between the members, and calculates the reallocation according to this interaction utility. We found that this can be done optimally by a deterministic polynomial time algorithm under several constraints, the first constraint is that $k = \mathcal{O}(\log N)$. We describe several other domains in which this method is applicable.

Introduction

This article discusses a team of N agents engaged in a cooperative behavior (Kraus 1997). Specifically, we consider the problem of choosing k team members in order to assign them to a new task. We assume that all members are capable of performing the new task, and the cost of the new task does not depend on the identity of the agents chosen to perform it. Therefore this paper concentrates on the problem of choosing k agents such that the performance of the existing task, performed by the remaining group members, will be as efficient as possible. The measure according to which the reallocation is done is based on the interaction between team members. Therefore the efficiency in this context refers to the cost of interaction, i.e., as the team spends less resources on interaction, the execution of the task is more efficient. The set of team members and the interaction between them is represented by a weighted directed graph, where the vertices represent the members, and the edges represent the interaction and the cost of the interaction between them. As shown later in this paper, this representation yields an algorithm for choosing k agents while minimizing the interaction cost of the remaining group. The algorithm is exponential in k, and under the common assumption that $k = \mathcal{O}(\log N)$, the algorithm is polynomial in N.

For example, consider a group of N robots organized in a formation (Naffin & Sukhatme 2004). The graph is the monitoring graph which represents the sensorial capabilities of the robots, i.e., which robot can sense the other and on what cost. The problem discussed here is, then, extracting k robots in order to acquire a new target while minimizing the monitoring cost of the remaining group. Another example in which the method is applicable is the warehouse assembling problem. Here, given a system of N warehouses, trucks should pass through all warehouses on their way to the main warehouse for storage. The interaction graph represents the cost of transportation between every two warehouses, and the problem is to remove k of the warehouses in order to cut back expenses, hence the cost of the remaining assembling tree ought to be minimal.

The general problem of choosing k out of N team members to perform a new task such that some mutual group-objective function is optimized is an important problem. However, it was shown to be $\mathcal{NP}$-hard as a special case of the Set Partitioning Problem (Garey & Johnson 1979; Shehory & Kraus 1998). Therefore investigations dealing with related problems typically focus on theoretical approximation algorithms and heuristic algorithms, e.g. (Shehory & Kraus 1998; Sander, Peleshchuk, & Grosz 2002).

Our approach, which concentrates on the minimization of the cost of interaction between the entities, makes it feasible to solve the problem both optimally and efficiently. In particular, this paper makes the following four contributions. *First*, we introduce a new method in which the problem of choosing k out of N agents is modeled by a graph, and the decision is taken while emphasizing the cost of interaction between the agents. *Second*, we describe a deterministic polynomial time algorithm for choosing k agents while minimizing the cost of interaction between the members of the remaining group, assuming that $k = \mathcal{O}(\log N)$ and that the group has one possible leader. *Third*, we later broaden the algorithm for the case in which the group has more than one leader. This case is significantly more complex, as it re-

*Gal Kaminka is also affiliated with CMU and Sarit Kraus is also affiliated with UMIACS. This research was supported in part by NSF grant # IIS−0222914, ISF Grant # 1211/04 and by Israel's MoST.

quires checking possible leader replacements; yet we show that it can still be done optimally by a polynomial time algorithm. *Finally*, we show that the basic algorithm can be used as a base for cases in which not all chosen agents are equally fit, either for the mission or for the original task (remaining group). If not all agents can be chosen for the new task, we show that in some cases the basic algorithm can still be used. In the case where choosing some robots may undermine the execution of the original task, we give an example from the multi robot domain, for which we describe an extension of the original algorithm that takes into account the stability of the remaining formation. This algorithm works in polynomial time as well, again under the assumption that $k = \mathcal{O}(\log N)$.

Related work

The class of agreement and pattern formation problems of multi robot systems is discussed by (Suzuki & Yamashita 1999; Balch & Arkin 1998; Naffin & Sukhatme 2004). They describe possible creation protocols of formations and maintenance and control schemes of robots in a formation. In our work, we consider the situation in which the formation already exists, and we need to extract from it k team members. Our problem does not resemble the problem in which agents fail to operate (Kumar & Cohen 2000), as we are able to choose the k agents, an act that results in the maximization of the actions of the remaining team members.

Studies that discussed the problem of choosing k out of N agents in order to perform a new task concentrated on maximizing the profit gained by the optimal performance of the new task, for example (Gerkey & Matarić 2000) use publish/subscribe messaging paradigm for suitably and efficiently assigning robots to a given task. We, on the other hand, concentrate in maximizing the benefit (minimizing the cost) gained by the optimal execution of the original task.

Other studies discussed allocation of agents between several given tasks. These studies mostly provide heuristic algorithms for efficient allocation of agents to tasks. (Sander, Peleshchuk, & Grosz 2002) describe task allocation heuristic algorithms for settings in which the agents and tasks are geographically dispersed in the plane. Work by (Dias *et al.* 2004; Dias & Stentz 2000) discusses task allocations between robots using auctions. Finding the optimal assignment using combinatorial auctions, where bidders can bid on combinations of items, was proven to be $\mathcal{NP}$-hard as well (Sandholm 2002). The problem of designing and forming groups of agents while maximizing some mutual objective is usually referred to as coalition formation. (Tosic & Agha 2004) describe a distributed algorithm for generating coalitions based on the current physical configuration of the agents, using maximal cliques. They show that the agents convert to the same coalitions, but their work do not refer to any kind of group utility, as opposed to our work that maximizes the joint utility of the agents performing the original task. Close to the scenario discussed in this paper is the work by (Shehory & Kraus 1998) that suggests algorithms for coalition formation among agents, where an agent can be either a member of only one coalition (similar to the case discussed here), or coalitions may overlap. (Sandholm & Lesser 1997) also dealt with coalition formation, but of self interested agents. Our problem can be thought of as a private case of those studies, for example two tasks - new and old or two coalitions. Although the problem is still hard, we are able to provide an optimal polynomial time solution under several constraints, the first constraint is that $k = \mathcal{O}(\log N)$.

Problem definition

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. 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 graph $G = (V, E)$, where $v \in V$ are the team members, and the edges represent the interaction. Meaning, $(v_i, v_j) \in E$ if there exists an interaction between v_i and v_j with $\mathrm{cost}(v_1, v_2)$, which is the weight of the edge (v_i, v_j). If an edge between v_a and v_b does not exist, i.e., there is no interaction between agents P_a and P_b, then $\mathrm{cost}(v_a, v_b) = \infty$. Upon this graph an *Optimal Interaction Tree* (OIT) is built by simply running Dijkstra's shortest path algorithm between all vertices of the graph and the leader. The one main constraint required from this graph is as follows.

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

Note that the edge (v_1, v_3) does not necessarily exist, i.e., $\mathrm{cost}(v_1, v_3) = \infty$ and in such case the constraint comes straightforward. An additional property can be added to the problem, given in the following definition.

Definition: 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 further 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 is minimal.
2. At least one of the vertices from the potential leader group of G will remain in the graph.

It is assumed that all N team members can be extracted, hence if dealing with acquiring a new target or performing a new task, then it means that all are compatible for the mission. Therefore, potentially, the number of different possibilities for extracting the k team members from the group is $\binom{N}{k}$. The algorithms described later on substantially reduces the complexity under several assumptions, the first assumption is that $k = \mathcal{O}(\log N)$.

The first example for a problem that matches the description above comes from the ***multi robot task allocation*** (MRTA) world. Following the taxonomy for MRTA systems given by (Gerkey & Matarić 2004), this paper deals with instantaneous assignments of single-task robots performing multi-robot tasks. Given a group of N robots that move in a specified formation, k of them should be extracted from the group in order to acquire a new target. Here, 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 & Glick 2005), an Optimal Monitoring Tree (OMT, which

is equivalent to our OIT) is built upon the monitoring multigraph. The OMT describes for each entity whom it 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 that it is compatible 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 the k robots, it is clear that the objective is to minimize the cost of sensing of the remaining group.

As we are not interested in the utility of the new task but only in improving the utility of the original task by the extraction of k members, problems concerning removal of agents while optimizing the utility of the remaining group are considered. An additional example is a variation of the ***dependency tree*** (Sevcik 1989). 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. 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 $\text{cost}(v_1, v_2)$ in this case is 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, with one main warehouse towards which all trucks are heading. The vertices of the graph represent the warehouses, and edges represent the distances between two warehouses (note that triangle inequality does not apply). The objective is to visit all warehouses in the fastest way by any number of trucks. The OIT represent 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, thus remain with the lowest cost assembling tree. The last problem is the ***broadcast*** problem, in which we are given a network with one source vertex that should constantly broadcast messages to the rest of the group. The edges represent the cost of the link between every two vertices, and the OIT is the optimal broadcast tree. Again here, we are required to remove k vertices in order to cut back expenses, thus remain with the lowest cost broadcast tree.

Team member reallocation focused on minimizing the cost of remaining OIT

Member reallocation with a single possible leader

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

Lemma 1. *Consider an* $\mathsf{OIT}(G)$*, satisfying Constraint A. If a vertex* v *that is not a leaf nor the leader is removed, then the sum of weights of edges of* $\mathsf{OIT}(G \setminus v)$ *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. Let u be the vertex that v was connected to, i.e., $(v, u) \in \mathsf{OIT}(G)$. Therefore all vertices connected to v should find another node to connect to, i.e., all $u_i \in V$ such that $(u_i, v) \in \mathsf{OIT}(G)$ should find a new vertex v_j to connect to, thus creating a new edge $(u_i, v_j) \in \mathsf{OIT}(G \setminus v)$ such that the cost of $\mathsf{OIT}(G \setminus v)$ is minimized.

If $v_j = u$ then we are done, as by Constraint A $\text{cost}(u_i, u) > \text{cost}(u_i, v) + \text{cost}(v, u)$. If $v_j \neq u$ then by the minimality of $\mathsf{OIT}(G)$ it follows that if $\text{cost}(u_i, v_j) < \text{cost}(u_i, v)$ then it would have chosen to point to v_j in the first place, contradicting the minimality of $\mathsf{OIT}(G)$. If by removing v vertex u_j remains disconnected in $G \setminus v$, then $\forall v_j \in V$, $\text{cost}(u_j, v_j) = \infty$, hence $\text{cost}(u_j, v_j) < \text{cost}(u_i, v)$. ∎

Corollary 2. *In an* $\mathsf{OIT}(G)$*, 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 later throughout the algorithm description.

Definitions:

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 $2^{\frac{k}{2}}$ (Chinn, Grimaldi, & Heubach 2003).
2. A *bundle* originated in vertex v is the subtree rooted in vertex v. Vertex v *nests a bundle* of size t if the bundle originated in it has t vertices, including v. See example in Figure 1. The bundle of size t is built bottom-up, i.e., from the leaves up.
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*.

We introduce an algorithm that finds the optimal k vertices to be removed from the tree. Following Corollary 2 of Lemma 1, this algorithm has to choose a subgroup of k vertices that include either leaves or bundles of the OIT. Each choice of vertices results in differently-structured trees, and thus results in a different utility value. Algorithm Tree_Pruning first 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 (see example in Figure 1). For each such element it indicates the gain from removing the bundle originating in that vertex. This gain is simply the sum of all 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 α_i it checks for the element with maximal gain in level α_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 resulted in higher gain is saved. Finally, after all palindromic compositions of k are examined, then the set whose removal results with the highest cost reduction is returned from the algorithm.

Algorithm $\mathsf{Team}_k = \mathsf{Tree\_Pruning}(G = (V, E), k)$

For *each leaf* $v \in V$*, start building a* k*-bundle bottom-up:*

1. $C_{best} \leftarrow \infty$ *and* $\mathsf{Team}_k \leftarrow \emptyset$.
2. *Add each subtree of size* $1 \leq t \leq k$ *to the table in row* t *and calculate its cost.*
3. *Sort all elements in each row according to their cost.*
4. *Generate a palindromic composition of the number* k *and sort each composition from left to right in decreasing order.*
5. **For** *each possible composition* C_j *of* $k = \alpha_1 + \alpha_2 + \ldots + \alpha_t$ **do:**
   (a) *Check whether the composition is feasible.*
   (b) *For each* α_i *in the composition,* $i = 1, 2, \ldots, t$ *pick highest order unmarked element from row* t_i *and mark it.*
   (c) *If the elements are not disjoint, then check all possibilities: First pick element with highest* α_i*, next don't pick it and pick element with next highest* α_i *and so on.*
   (d) *Calculate the cost of the composition* C_j*. If* $\mathrm{cost}(C_j) \geq C_{best}$*, then* $C_{best} \leftarrow C_j$ *and* $\mathsf{Team}_k \leftarrow$ *current composition.*
6. *Return* Team_k

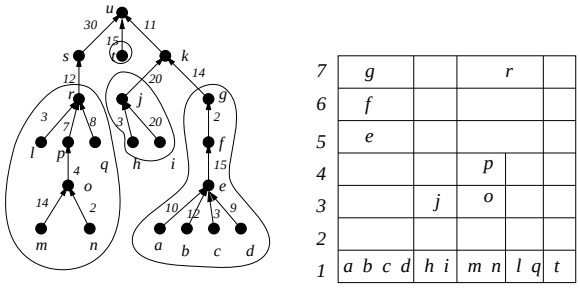

Figure 1: An example for 7-bundling of a tree.

Theorem 3. *Algorithm* Tree_Pruning *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 to 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 preliminary work of building the table will cost up to $\mathcal{O}(N)$ time, 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 $2^{\frac{k}{2}}$ (Chinn, Grimaldi, & Heubach 2003). The algorithm might check each composition (worst case) N times, in case that the chosen elements are not disjoint. Assuming that each approach to an element takes $\mathcal{O}(1)$ steps (depends on the data structure used), each composition is calculated in (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 $\mathcal{O}(N \log N + N\sqrt{N}) = \mathcal{O}(N^{1.5})$.

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 size greater than one. Such cases can be considered in all the problems mentioned earlier. In the problem involving robots, it is possible that several robots in the formation can act as the leader. In the warehouse assembling problem, several warehouses can be the target of the trucks and in the broadcast problem there can be more than one main source of broadcast information. The order of extraction is important. 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. The reason lies in the fact that the optimal tree might have different edges and leaves when the leader is different. Note that removal of a leaf might result in changing the leader, depending on the leader election algorithm. However, we assume that this does not happen here.

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

In the first level, $l = 1$, we can either pick k vertices using algorithm Tree_Pruning (meaning, we do not change the leader), or change the leader first, second and so on until picking it as the k'th vertex. Each such option but the former branches out similarly in the next level ($l = 2$) where the 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 are remained 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, plus the one where the leader is not replaced. When a leader is replaced, the calculation of the p vertices chosen prior to it is simply obtained by running Tree_Pruning for p. The remaining q launch an additional

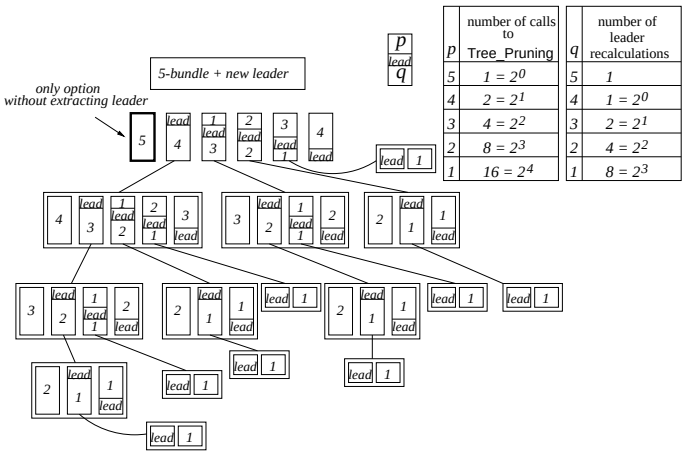

Figure 2: An example for the search tree of all possibilities of extracting $k = 5$ vertices from the graph where the leader can be elected as well in each step.

level where, again, it branches 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 (Cormen, Leiserson, & Rivest 1990), the complexity of calculating the OIT of a graph of size $\tilde{N}$ given the identity of the leader vertex is simply running Dijkstra's shortest paths algorithm that takes $\mathcal{O}(\tilde{N}^3)$. If there are $\tilde{M}$ potential leaders, then the complexity is $\mathcal{O}(\tilde{N}^3 \tilde{M})$. Hence here the time complexity can be bounded from above by the total number of qs, times $\mathcal{O}(N^3 M)$. The total number of qs, as demonstrated in Figure 2, 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 $\mathcal{O}(N^4 M)$.

An algorithm variation, in which not all vertices can be extracted

In some cases, either some team members are required ro remain in the group and cannot be extracted from it. For example, in a formation of robots in which k robots are required to acquire a new target, it is possible that not all robots are compatible for that mission, hence only the ones who are compatible can be extracted from the group. Converting it to our graph problem, if it is required to extract nodes (processors/robots/agents) with specific capabilities such that only a subgroup $G_1 \subseteq G$ satisfy those demands 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 there is no option but that all vertices in G_1 will be extracted.

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

In the case discussed here, 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 ran on the OIT graph G, but should stop at bundle blockers or on depth k, whichever comes first.

Domain specific Team member reallocation: example on robot formation

In this section we show that the basic Tree_Pruning algorithm can be used as a base for team member reallocation problems involving other considerations. We illustrate this proposition on the example from the robotic world, in which the OIT represents the optimal sensorial and monitoring capabilities of the robots towards the leader. We assume that the robots leaving the formation move in a straight line towards the new target. Although this assumption is not necessarily required, it allows us to nicely illustrate the idea of additional considerations incorporated in the algorithm. In this variation of the algorithm we wish to extract robots from the group while minimizing events that might undermine formation stability in the following two ways. First, we want to cause minimal changes to the current OIT. For that it is required that only leaves or bundles will be removed, and the leader will remain intact. Second, we wish to minimize collisions (actual robot intersections) between the robots leaving for the new target and the ones remaining in the formation. Moreover, we want to 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.

Algorithm Stable_Pruning 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) the algorithm checks against all other vertices but the leader whether, if r_i leaves towards the goal point p_G, its path hides the outgoing edge from the vertex v_j. If so, it adds t_j to the entry of v_i in a prespecified table Table with the mark E (see for example Figure 3). If the robots themselves intersect, then v_j is added to Table with the mark I. After creating this table, Tree_Pruning is ran on the graph where three features are examined in each step: I intersections, E intersections which are extracted from Table, and the remaining OMT cost (in this order).

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

1. **For** *each vertex* $v_i \in V$ *such that* v_i *is not the leader, do:*
   (a) *Go over all vertices of the graph except for vertices in the bundle originated in* v_i.
   (b) **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* $\mathsf{Table}(v_i, E)$.

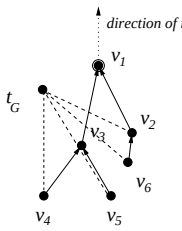

Table:

	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

Figure 3: An example for path/edge intersection.

(c) **If** v_i *on its course towards* t_G *intersects* v_j, **then** *add* v_j *to* Table(v_i, I).

2. *Run procedure* Tree_Pruning(G, k) *with the following modifications.*
   - *Set* $E_{best} \leftarrow \infty$, $I_{best} \leftarrow \infty$ *and* 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* Team$_k \leftarrow$ *current composition.*
   - **If** $I_{cur} = I_{best}$ *and if* $E_{cur} < E_{best}$ **then** $I_{best} \leftarrow I_{cur}$ *and* 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 .
3. *Return* Team$_k$.

Algorithm Stable_Pruning 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 the 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 $\mathcal{O}(N^2)$.

Conclusions and future work

We have presented algorithms for optimal team-member reallocation based on optimal interaction graphs. We have shown that these graphs are applicable in several different domains. We further expanded the algorithms to cases where the graphs can have more than one leader, and (under some conditions) cases with heterogenous entities. Finally, we have shown that the team-member reallocation algorithm can be be used as a base for reallocation problems that can consider, in addition to the optimality of the cost of remaining group, also other domain-specific constraints. Thus, our main contributions are the introduction of a new method for modeling and solving the problem of choosing k out of N agents by a polynomial time algorithm on graphs, assuming that $k = \mathcal{O}(\log N)$ and that constraint A applies, and that the group has either one or more than one possible leaders.

There are several areas we plan to pursue in future work.

- Examine empirically the case in which weighted components of the utility function according to which the agents are chosen are considered.
- Examine the case of greater k, for example $k = \frac{N}{2}$, and see whether a polynomial time algorithm exists for finding an optimal allocation.
- Consider the case in which Assumption A is removed, where probably a polynomial time algorithm does not exist, hence approximations should be given and tested empirically.
- Thoroughly investigate domain-specific problems based on this algorithm, and further adapt it to those scenarios. For example, in the robotic problem we can try making the algorithm robust to sensory failures.

References

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

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

Cormen, T. H.; Leiserson, C. E.; and Rivest, R. L. 1990. *Introduction to Algorithms*. MIT Press.

Dias, M. B., and Stentz, A. 2000. A free market architecture for distributed control of a multirobot system. In *Proc. IAS-6*, 115–122.

Dias, M. B.; Zlot, R.; Zinck, M.; Gonzalez, J. P.; and Stentz, A. 2004. A versatile implementation of the traderbots approach for multirobot coordination. In *Proc. IAS-8*.

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

Gerkey, B. P., and Matarić, M. J. 2000. Murdoch: publish/subscribe task allocation for heterogeneous agents. In *AGENTS*, 203–204.

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

Kaminka, G. A., and Glick, R. 2005. Reasoning about sensors in selective montoring. *Bar Ilan University Tech. Report*.

Kraus, S. 1997. Negotiation and cooperation in multi-agent environments. *Artif. Intell.* 94(1-2):79–97.

Kumar, S., and Cohen, P. R. 2000. Towards a fault-tolerant multi-agent system architecture. In *AGENTS '00*, 459–466.

Naffin, D. J., and Sukhatme, G. S. 2004. Negotiated formations. In *Int. Conf. on Intelligent Autonomous Systems*, 181–190.

Sander, P. V.; Peleshchuk, D.; and Grosz, B. J. 2002. A scalable, distributed algorithm for efficient task allocation. In *AAMAS*, 1191–1198.

Sandholm, T., and Lesser, V. 1997. Coalitions among computationally bounded agents. *Artif. Intell.* 94(1):99–137.

Sandholm, T. 2002. Algorithm for optimal winner determination in combinatorial auctions. *Artif. Intell.* 135(1-2):1–54.

Sevcik, K. C. 1989. Characterizations of parallelism in applications and their use in scheduling. In *Proc. of ACM Conf. on Measurement and Modeling of Comp. Sys.*, 171–180.

Shehory, O., and Kraus, S. 1998. Methods for task allocation via agent coalition formation. *Artif. Intell.* 101(1-2):165–200.

Suzuki, I., and Yamashita, M. 1999. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM J. Comput.* 28(4):1347–1363.

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