Forming Efficient Agent Groups for Completing Complex Tasks

Efrat Manisterski [1], Esther David [2]
[1] Computer Science Department
Bar-Ilan University
Ramat-Gan 52900, Israel
{maniste,sarit}@cs.biu.ac.il

Sarit Kraus [1] and Nicholas R. Jennings [2]
[2] Electronics and Computer Science
University of Southampton
Southampton SO17 1BJ, UK
{ed,nrj}@ecs.soton.ac.uk

ABSTRACT

In this paper we produce complexity and impossibility results and develop algorithms for a task allocation problem that needs to be solved by a group of autonomous agents working together. In particular, each task is assumed to be composed of several subtasks and involves an associated predetermined and known overall payment (set by the task's owner) for its completion. However, the division of this payment among the corresponding contributors is not predefined. Now to accomplish a particular task, all its subtasks need to be allocated to agents with the necessary capabilities and the agents' corresponding costs need to fall within the preset overall task payment. For this scenario, we first provide a cooperative agent system designer with a practical solution that achieves an efficient allocation. However, this solution is not applicable for non-cooperative settings. Consequently, we go on to provide a detailed analysis where we prove that certain design goals cannot be achieved if the agents are self interested. Specifically, we prove that for the general case, no protocol achieving the efficient solution can exist that is individually rational and budget balanced. We show that although efficient protocols may exist in some settings, these will inevitably be setting-specific.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence —*Multiagent systems*

General Terms

Algorithms, Design, Economics

Keywords

Complex Task Allocation, Efficient Allocation

1. INTRODUCTION

Many task allocation problems require a number of autonomous agents to form groups in order to execute complex activities they can't perform individually [16]. In this work, we specifically consider the case in which tasks: (1) are composed of several subtasks and (2) have predetermined and known overall payments, set by the tasks' owners (i.e., a fixed price contract) [6]. To accomplish a particular task in this case, all its relevant subtasks need to be completed by agents with the necessary capabilities, and the agents' costs need to fall within the preset task payment. Therefore, in this scenario, allocated agents are likely to receive payments which are higher than their reported costs, to meet the requirement that the overall task payment need to be fully paid out.

This task allocation problem is common in a wide range of contexts, including both cooperative and non-cooperative systems. By means of illustration, for the former case consider a large organization composed of several departments, each with its own set of capabilities. Such organizations may receive offers to carry out multiple projects with known payments. In order to accomplish the specific goals of each project, several departments need to work together. In such cases, the role of the organization manager is to allocate the projects to the departments and to decide on the division of the complete available payments to the contributing departments, in order to maximize the organization's efficiency. Here it is assumed the departments cooperate by revealing their actual costs and are prepared to execute any given subtask within their capabilities. Similar scenarios also occur in Grid resource allocation problems where a controller faces demands to allocate an appropriate set of computational resources for the completion of a set of tasks within the fixed payments constraints of the tasks' owners. However, in contrast to the first example, here the controller neither owns the computation resources that may be deployed nor does it have full information about them. Therefore, this must be a non-cooperative designation. Each such resource has to explicitly declare which activities it is willing to perform and its corresponding costs. Only with this information can the controller make the allocation.

Given the ubiquity of this problem, it is surprising that it has received no significant attention to date. Specifically, the main work in this area is due to Kraus et al. [7, 6] who consider a non-cooperative agent system for executing such tasks. In particular they provide a protocol that achieves 80% efficiency, but which does not scale to a large number of subtasks and agents. In addition, they benchmark their protocol's efficiency against a solution achieved by a heuristic and not against the the most efficient solution (which makes their results even more fragile). To rectify these shortcomings, in this paper we first provide a cooperative agent system designer with a practical solution that achieves the efficient allocation. We do so by developing

AAMAS'06 May 8–12 2006, Hakodate, Hokkaido, Japan.

an algorithm which is exponential in the number of tasks, but polynomial in the number of agents and the number of subtasks. Since in many real world situations, the number of tasks is relatively small and can be bounded by a small constant number, the algorithm's complexity can be considered polynomial in the size of the input. We also prove that no polynomial algorithm exists when the number of tasks is unbounded (unless P=NP). This is therefore the best result that can be achieved.

Unfortunately, this solution is not applicable for non-cooperative settings, where more difficulties are raised (e.g., uncertainties about agent's cost, self interested nature). Consequently, in the second part of the paper we analyze what can and cannot be achieved when we consider self interested agents. Specifically, for the general problem we prove that no individually rational and budget balanced protocol can exist which achieves an efficient solution. However, there are some non-cooperative settings for which an efficient protocol does exist. The drawback is that such an efficient protocol for one such setting is likely to be inefficient for another. This result can then be used by the designers of agent systems, who are faced with this task allocation scenario, to direct them to develop appropriate protocols and strategies for particular settings of the problem, and to ensure they know what can and cannot be achieved. Moreover, the performance of any proposed protocol for the non-cooperative setting can in the future be benchmarked against our solution for the cooperative case. This will give more accurate results than benchmarking against heuristics as has been done in previous work.

The rest of the paper is organized as follows. Section 2 introduces our problem definition. In section 3 a cooperative setting for the problem is considered for which we develop an algorithm resulting in an efficient solution. Section 4 focuses on the non-cooperative setting where we show what can and cannot be achieved. In section 5 related work is described and, finally, we conclude in section 6.

2. PROBLEM FORMALIZATION

The general description of our task allocation problem assumes there are M candidate tasks $\Im = \{T_1, ..., T_M\}$, where each T_i is composed of several subtasks $\left\{ST_{i_1}, ..., ST_{i_{k_i}}\right\}$. The overall payment of a task, $p(T_i)$, is predefined and known to all and no inter-task payments transfer is allowed. Moreover, by definition, the task payments need to be fully paid out for their completion. However, the partitioning of the task payment among its subtasks is not predetermined, but will be set as part of the task allocation process. Here a task is considered to have been completed if all its subtasks have been executed.

For the completion of these tasks, a set of agents, $A = \{A_1, ..., A_N\}$, is given. Each of these agents has capabilities to perform multiple subtasks (where some overlapping among different agents' capabilities is assumed). Here, we assume these capabilities to be known to all participating agents and the controller. However, as the subtasks should be carried out simultaneously, we assume that each agent is capable of performing only one subtask at a time.[1] Note that given the dependency on the resource constraints and the reward structures, not all of the tasks may get executed.

In this paper we consider both cooperative and non-cooperative settings. For both cases we assume a central controller is responsible for the task allocation. In the cooperative case we assume the controller has full information about the agents' costs (as the capabilities are known to all) either because of the system's characteristics or because the agents are assumed to report their costs truthfully (since they have no incentive to manipulate this information). In addition, here an agent's cooperation also signals its agreement to complete any assigned subtask which it is capable of performing. Now in the non-cooperative setting, the agents' capabilities are still known to all including the controller, but in this case the controller can only assign subtasks to those agents that indicated their willingness to perform them. Therefore, an agent may strategize by choosing which of the subtasks it is willing to undertake and by manipulating their corresponding declared costs. Note that in doing so agents may use the information they have about the other agents' capabilities and their cost distributions as we define below.

Formally, a knowledge capability function (φ) that is known to all agents and the controller is assumed, where $\varphi(A_j, ST_{i_l})$ equals one if agent A_j is capable of performing subtask ST_{i_l} and otherwise zero. Using this function, we summarize each agent's capabilities by $\Psi_j = \{ST_{i_l} | \varphi(A_j, ST_{i_l}) = 1\}$. Conversely, for any subtask ST_{i_l} there is a set of agents capable of completing it, $\Omega_{i_l} = \{A_j | \varphi(A_j, ST_{i_l}) = 1\}$. Moreover, each agent A_j has a private cost for performing a given subtask ST_{i_l}, $c^j_{i_l}$, taken from a common distribution F_{i_l} in the range of $\left[\underline{\theta}_{i_l}, \overline{\theta}_{i_l}\right]$.

An *allocation* Φ is defined as a match between some of the agents and some of the subtasks, $\Phi = \left\{(A_{i_1}, ST_{i_{1_{l_1}}}), ..., (A_{i_n}, ST_{i_{n_{l_n}}})\right\}$. Here Φ_T indicates the set of tasks allocated by Φ ($\Phi_T \subseteq \Phi$) and an allocation Φ is *feasible*, if and only if it satisfies three conditions:

C1 Each agent in Φ is allocated to at most one subtask.

C2 Any task is either fully allocated (one agent per subtask) or it is unallocated.

C3 For each task $T_i \in \Phi_T$, the total cost for the agents to perform its constituent subtasks does not exceed the task's total payment $p(T_i)$.

As discussed in the introduction, we seek to find an efficient allocation. To this end, we define the **value of an allocation** Φ, $V(\Phi)$, to be the difference between the total payments of the allocated tasks and the total cost for the assigned agents to perform the corresponding subtasks, $V(\Phi) = \sum_{T \in \Phi_T} p(T) - \sum_{(A_j, ST_{i_l}) \in \Phi} c^j_{i_l}$. The feasible allocation with the highest value is considered to be the ***efficient allocation*** Φ_{eff}. Consequently, the efficiency of any proposed allocation Φ is measured relative to Φ_{eff} as: $efficiency(\Phi) = \frac{V(\Phi)}{V(\Phi_{eff})}$.

Next we present a formal description of our problem using an integer program. The variables y_i and $x^j_{i_l}$ are decision variables where $y_i = 1$ if task T_i is chosen to be allocated and otherwise zero, and $x^j_{i_l} = 1$ if agent A_j is chosen to perform subtask ST_{i_l} and otherwise zero:

$$\Phi_{eff} = \max_{y_i, x^j_{i_l}} \left(\sum_{i=1}^{M} (y_i \cdot p(T_i)) - \sum_{j=1}^{N} \sum_{ST_{i_l} \in \Psi_j} (x^j_{i_l} \cdot c^j_{i_l})\right) \quad (1)$$

[1]This is for simplicity of exposition and our results can be easily extended to the case where an agent is capable of performing any constant number of subtasks at a time.

such that:

$$\sum_{ST_{i_l} \in \Psi_j} (x_{i_l}^j) \leq 1, \forall j \in 1, ..., N \quad (2)$$

$$\sum_{j \in \Omega_{i_l}} (x_{i_l}^j) = y_i, \forall i \in 1, ..., M, \text{ and } l \in 1, ..., k_i \quad (3)$$

$$x_{i_l}^j, y_i \in \{0, 1\} \forall i \in 1, ..., M, \forall l \in 1, ..., k_i, \forall j \in \Omega_{i_l} \quad (4)$$

where equations 2 and 3 correspond to C1 and C2, respectively. Note that condition C3 is implicitly achieved by the definition of Φ_{eff}.[2]

3. THE COOPERATIVE SETTING

In this section, we develop a practical algorithm that can be used by the system's controller to achieve an efficient solution in a cooperative setting. Specifically, we analyze the problem's complexity, then develop an algorithm to find an efficient solution, and finally explore the algorithm's properties.

3.1 Problem Complexity

The complexity of our task allocation problem is given by the following lemma:

LEMMA 1. *Given the task allocation problem defined in the integer program (1-4), the computational complexity of finding an efficient solution is NP-hard. Moreover, it is hard to approximate within a factor of* $M^{1-\epsilon}$.

PROOF. We show this using a reduction from the winner determination problem in combinatorial auctions. In a winner determination problem, there is a set of n items and a set of M bids $B_j(S_j, p_j)$, where S_j may be any bundle of items and p_j is the price offered for S_j. The goal of winner determination is to find the subset of bids that maximizes the revenue with no conflicts (in the sense that the same item would not be included in more than one bid). Given an instance of the winner determination problem we present the items as the agents in our problem (we define a one to one correspondence, so that any item i is associated with an agent A_i). Each bid $B_j(S_j, p_j)$ is viewed as a task T_j for which its payment is p_j. T_j is composed of $|S_j|$ subtasks, where the agents that are associated with the items in S_j are the only agents that are capable of performing the subtasks in T_j. We assume that an agent can perform any of the subtasks of the task it is associated with. In addition, we assume that the cost of the agents performing any subtask is zero. Given this mapping, it is easy to see that there is a solution to the winner determination problem in combinatorial auctions that will achieve a certain profit P for the auctioneer, if and only if an allocation to our task allocation problem exists in which its value is P. Building on this, [14] proves that no polynomial time algorithm can guarantee a bound $k \leq M^{1-\epsilon}$ for any $\epsilon > 0$ for the winner determination problem and thus based on reduction preserving approximability, our task allocation problem is also hard to approximate within a factor of $M^{1-\epsilon}$. □

Having shown our task allocation problem is NP hard, we now go on to present an algorithm which is exponential solely in the number of tasks, but polynomial in the numbers of subtasks and agents. This is a significant improvement over the brute-force algorithm which exhaustively goes through all possible assignments and searches for the minimum cost one. Such an algorithm is also exponential in the number of subtasks.

[2]Assume, by way of contradiction, that an Φ_{eff} exists that doesn't satisfy C3 (i.e., some tasks exist for which the agent's costs exceed the tasks' payments). Omitting these tasks from the allocation results in a higher value allocation, which is a contradiction.

Agents	ST_{1_1}	ST_{1_2}	ST_{2_1}	ST_{2_2}
A_1	5	-	5	-
A_2	-	5	-	5
A_3	30	-	10	-
A_4	-	30	-	10

Table 1: Costs and capabilities for example 1

3.2 Finding an Efficient Allocation

Our solution is based on the algorithm for finding the minimum weighted perfect matching in a bipartite graph [1]. Before describing the algorithm, we first provide an interpretation of our problem in terms of a bipartite graph $G = (V, E)$. Each of the agents and subtasks are represented by a vertex in V, such that a capability of an agent to perform some subtask is indicated by an edge (in E) between the corresponding vertices. A graph G is considered a **bipartite** graph if its vertices, V, can be partitioned into two groups, V_1 and V_2, such that no edge in E exists between members of the same group. In our case, this partition can be viewed as the agents and the subtasks groups. Given a bipartite graph G, a **perfect matching** $M \subset E$ is defined to be a subset of edges that cover all vertices and for which no common vertex exists for any pair of edges in M. In terms of our problem, a perfect matching is an assignment of agents to subtasks such that all the agents and the subtasks in V are allocated. By representing an agent's cost to complete some subtask as a weight of the edge connecting the two corresponding vertices, we obtain a weighted graph. Thus a **minimum weighted** perfect matching is a perfect matching, such that the sum of its edges' weights is the minimum (which in our case is an efficient solution if it satisfies the feasibility conditions). However, in our problem, some of the tasks may not be completed, as explained in section 2, thus in order to find the efficient solution one must go through all the subsets of tasks as we formally describe below.

Given a set of M candidate tasks $\Im = \{T_1, ..., T_M\}$, we define the power set of $\Im$ to be $\Lambda = \{\lambda | \lambda \subseteq \Im\}$). For each λ in Λ, we construct a bipartite graph $G_\lambda = (V_1, V_2, E)$, where V_1 represents all the subtasks, ST, associated with one of the tasks in λ, and V_2 corresponds to the agents in A (if the number of vertices in V_2 is larger than the number of vertices in V_1, the algorithm extends V_1 to the size of V_2 by adding $|V_2| - |V_1|$ vertices that are connected to all vertices in V_2 with a weight of 0). An edge between ST in V_1 and A_j in V_2 exists in E if A_j is capable of performing ST and the edge's weight is A_j's cost to perform ST. Given this construction, the problem of finding the ***minimum cost assignment*** that allocates all tasks in λ to agents in A is equivalent to the problem of finding the ***minimum weighted perfect matching*** M_λ in G_λ. An additional step of the feasibility test is required to validate the solution. The Algorithm is described in figure 1.

To illustrate the proposed algorithm consider the following example:

EXAMPLE 1. *Assume there are four agents* $\{A_1, A_2, A_3, A_4\}$ *and two tasks, each of which includes two subtasks* $T_1 = \{ST_{1_1}, ST_{1_2}\}$, $T_2 = \{ST_{2_1}, ST_{2_2}\}$, *where the task payments*

1. For each λ in Λ do:
   (a) Build a bipartite graph, $G_\lambda = (V_1, V_2, E)$.
   (b) Find the minimum weighted perfect matching M_λ in G_λ (if no perfect matching exists go to 1).
   (c) Check whether M_λ is a feasible allocation.
   (d) If M_λ is feasible, compute its allocation value $V(M_\lambda)$, otherwise go to 1.
2. Return the most efficient allocation, M_λ, which is a feasible allocation with the highest value $V(M_\lambda)$.

Figure 1: Algorithm for finding an efficient allocation

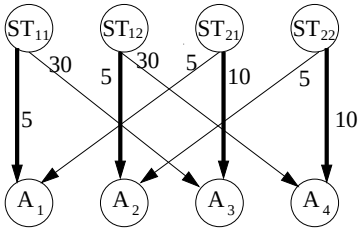

Figure 2: Graph for $\lambda_1 = \{T_1, T_2\}$

are $p(T_1) = 500$ and $p(T_2) = 12$. The agents' costs and capabilities are given in Table 1. The power set of the tasks includes three nonempty sets: $\lambda_1 = \{T_1, T_2\}$, $\lambda_2 = \{T_1\}$ and $\lambda_3 = \{T_2\}$. For each of them, the algorithm constructs the corresponding graph and computes the minimum cost allocation. Figure 2 shows the graph built for $\lambda_1 = \{T_1, T_2\}$.

Here the bold edges indicate the minimum cost allocation generated by the algorithm
$\Phi_{eff} = \{(A_1, ST_{1_1}), (A_2, ST_{1_2}), (A_3, ST_{2_1}), (A_4, ST_{2_2})\}$ for λ_1 whose value is 482. As this allocation is infeasible (since the total cost for T_2 is 20 whilst T_2's payment is 12) the algorithm ignores it and continues by considering λ_2. The efficient allocation of this example is to allocate only task T_1 to agents A_1 and A_2 for which the value is 490 (the highest possible).

According to the algorithm, given a certain subset of tasks, λ, first, a minimum weighted perfect matching is generated. Next its feasibility is verified[3]. Once it fails to satisfy the feasibility requirement, all other possible allocations of λ are ignored. It may seem possible that the efficient solution is not necessarily the minimum perfect matching for some λ, but, instead, is the minimum of all the *feasible weighted perfect matching*. For instance, in example 1, a feasible allocation exists where
$\Phi' = \{(A_1, ST_{2_1}), (A_2, ST_{2_2}), (A_3, ST_{1_1}), (A_4, ST_{1_2})\}$ that might have been the efficient allocation, but the algorithm ignores it. However, as we will prove in section 3.3.2, such a scenario cannot occur.

[3]Note that for each set of tasks $\lambda \in \Lambda$, the algorithm checks for the allocation that includes all tasks in λ, regardless of its feasibility, and computes the minimum allocation if such an allocation exists. However, if we change the algorithm to forbid infeasible allocations, the problem of checking for the *existence* of a *feasible* allocation that allocates all tasks in λ will become NP complete (a reduction can be made from the subset sum problem [2]).

3.3 The Algorithm's Properties

Here, we consider the time complexity and the correctness of our algorithm.

3.3.1 Time Complexity

The complexity of step 1 is exponential in the number of tasks in $\Im$, simply because it goes through all the $\Im$'s subsets. However, the complexity of the internal steps 1a-1d is polynomial, as the complexity of the matching problem is polynomial [1]. Therefore the total complexity of the algorithm is exponential in the number of tasks $|\Im|$ and polynomial in the numbers of subtasks and agents.

3.3.2 Correctness

Here, we prove that if the set of tasks allocated by an efficient allocation Φ is λ, then for G_λ the algorithm will return a feasible perfect minimum matching with the same value as Φ. This implies that the algorithm always returns an efficient allocation. Note that the algorithm can return an allocation different from Φ, but since it has the same value as Φ it is also considered an efficient allocation.

LEMMA 2. *Assume Φ_{eff} is an efficient allocation that allocates the set of tasks in λ_{eff}. Then the algorithm: (i) finds the minimum weighted perfect matching $M_{\lambda_{eff}}$ in graph $G_{\lambda_{eff}}$, (ii) $M_{\lambda_{eff}}$ is a feasible allocation, and (iii) $M_{\lambda_{eff}}$ has the same value as Φ_{eff}'s value.*

PROOF. Assume λ_{eff} is the set of tasks that are allocated by the efficient allocation Φ_{eff}. We begin by proving that a minimum weighted perfect matching $M_{\lambda_{eff}}$ exists for the corresponding graph $G_{\lambda_{eff}}$. As Φ_{eff} is feasible, by definition, it satisfies C1 and C2. That is, Φ_{eff} defines a perfect matching for $G_{\lambda_{eff}}$ which, in turn, means that a *minimum weighted* perfect matching exists. Therefore the algorithm returns $M_{\lambda_{eff}}$ (in the worst case it will be Φ_{eff}). Next, we prove that $M_{\lambda_{eff}}$ is a feasible allocation. Assume, by way of contradiction, that $M_{\lambda_{eff}}$ is an infeasible allocation. By definition $M_{\lambda_{eff}}$ satisfies C1 and C2. This implies that C3 is not satisfied. Let $\lambda_{infeasible}$ be the set of tasks in $M_{\lambda_{eff}}$ that do not satisfy C3 (i.e., $\lambda_{infeasible} = \left\{T_i | T_i \in \lambda_{eff} \text{ and } \sum_{(A_j, ST_{i_l}) \in M_{\lambda_{eff}}} c_{i_l}^j > p(T_i)\right\}$). Now let λ_{new} be the set of tasks that are allocated by $M_{\lambda_{eff}}$ that satisfy C3. That is, $\lambda_{new} = \lambda_{eff} - \lambda_{infeasible}$. As we assume $M_{\lambda_{eff}}$ is an infeasible allocation, $\lambda_{infeasible}$ must not be empty (therefore $\lambda_{eff} \neq \lambda_{new}$). Let $M_{\lambda_{new}} = \left\{(A_j, ST_{i_l}) | (A_j, ST_{i_l}) \in M_{\lambda_{eff}} \text{ and } T_i \in \lambda_{new}\right\}$. $M_{\lambda_{new}}$ is a perfect matching for the set λ_{new}. Consequently, $M_{\lambda_{new}}$ is a feasible allocation for λ_{new} and thus, from the feasibility definition, $M_{\lambda_{new}}$ is a feasible allocation for the entire set of tasks $\Im$. In conclusion, at this point, we have two feasible allocations for $\Im$, Φ_{eff} and $M_{\lambda_{new}}$. Next, we will calculate their efficiency values.

$V(\Phi_{eff}) = \sum_{T \in \lambda_{eff}} p(T) - \sum_{(A_j, ST_{l_i}) \in \Phi_{eff}} c_{i_l}^j$
$\leq$ [4] $\sum_{T \in \lambda_{eff}} P(T) - \sum_{(A_j, ST_{l_i}) \in M_{\lambda_{eff}}} c_{i_l}^j = V(M_{\lambda_{eff}})$
$= \sum_{T \in \lambda_{infeasible}} P(T) - \sum_{(A_j, ST_{i_l}) \in M_{\lambda_{eff}} - M_{\lambda_{new}}} c_{i_l}^j +$
$\sum_{T \in \lambda_{new}} P(T) - \sum_{(A_j, ST_{i_l}) \in M_{\lambda_{new}}} c_{i_l}^j$

[4]As the minimum perfect matching finds the lowest weight matching but not necessarily a feasible one, the efficiency value of $V(M_{\lambda_{eff}})$ is greater or equal to $V(\Phi_{eff})$.

$< \sum_{T \in \lambda_{new}} P(T) - \sum_{(A_j, ST_{i_l}) \in M_{\lambda_{new}}} c^j_{i_l} = V(M_{new})$.

To conclude, we prove that $V(\Phi_{eff}) < V(M_{new})$, which contradicts the fact that Φ_{eff} is the allocation with the maximum efficiency value. Therefore the efficiency value of $V(M_{\lambda_{eff}})$ is equal to $V(\Phi_{eff})$. Previously we proved that $M_{\lambda_{eff}}$ is a feasible allocation. That is, the value $V(M_{\lambda_{eff}})$ cannot be greater than $V(\Phi_{eff})$ since Φ_{eff} is *the* efficient allocation. On the other hand, $V(M_{\lambda_{eff}})$ cannot be less than $V(\Phi_{eff})$ as both allocate all tasks in λ_{eff} and $M_{\lambda_{eff}}$ is the minimum matching cost . □

Up to this point, we have dealt with a cooperative setting. Unfortunately, however, the algorithm developed for this setting cannot be simply applied to the non-cooperative case for the following reasons: (1) the controller in a non-cooperative setting cannot assign a subtask to an agent that has not indicated its willingness to take on the activity and (2) the agents in a non-cooperative case may manipulate their reported costs in order to increase their individual benefit. Moreover, while in a cooperative setting the division of a task's payment does not have any impact on the allocation outcome, in the non-cooperative case the payment division may well play an important role (as it directly affects the agents' behavior). Therefore, as part of any proposed protocol, a rule by which the task's payment is divided among the contributing agents needs to be designed. As a consequence, no guarantees about the proposed algorithm's efficiency exist when applied to the non-cooperative settings. Thus next we examine the non-cooperative setting in terms of desiderata, impossibility results, and potential solutions.

4. THE NON-COOPERATIVE SETTING

In a non-cooperative system, agents have the freedom to decide which subtasks they wish to carry out and which costs to report (since the controller has no information on the agents' actual costs). Thus the controller can only allocate the subtasks based on the agents' requests. Therefore, in order to achieve an efficient solution, the simple approach of asking agents to report their private information (as assumed by the algorithm for the cooperative case) needs to be replaced by a more sophisticated protocol. That protocol should still seek to maximize the overall efficiency, while taking into consideration that each individual agent may strategise (on the subset of its capabilities and their costs to report) in order to maximize its own utility.

In most task allocation scenarios, the strategy space for a buyer considers values that are lower than its actual valuation. Similarly, the strategy space for a seller considers values that are higher than its cost. In our case, the agents are selling their services to perform some subtasks. Given this, one might think that as part of our agents' strategies in the task allocation problem we need to consider only those costs that are higher than their actual costs. In contrast, however, we show that this is not the case in our setting and an agent designer needs to consider values that are both higher and lower than the actual agents' costs[5]. Consequently, given that in non-cooperative settings the agents may make strategy-based declarations, we would like to develop protocols for which dominant strategies (i.e., the best strategy for an agent regardless of the strategies chosen by other players), or at least Bayesian Nash equilibria [11] exist to free agents from the need to strategize.

In particular, we would like to find one general protocol for which agents will have equilibrium strategies (for the strongest concept possible) that achieves the ***efficient*** solution for all instances of our problem and, in addition, will satisfy the following desirable properties. First, as we have self interested agents, a primary desiderata of any proposed protocol is to be ***individually rational*** (i.e., the expected revenue for each agent from participating should be higher than or equal to the revenue achieved by not participating). This is essential to guarantee that no agent loses by participating in the allocation process. Second, the protocol should also be ***budget balanced*** to match the definition of our task allocation problem (i.e., any task's payment should be fully paid out to its contributing agents and no transfer is allowed between tasks). The motivation for such a requirement is the fact that the preset overall payment is known and that it has been allocated solely for the completion of the task. Third, we would like our protocol to be ***undiscriminating*** (i.e. it should not discriminate among the agents by utilizing the knowledge of the agents' capabilities. In this model the protocol is required to give the same expected payment to any two agents with the same declared capabilities and the same costs, regardless of the difference in their actual capabilities). Such a requirement is vital to ensure that the maximum number of agents are willing to engage in the pool of providers.

For the analysis that follows, we consider two classes of environment: one in which the agents may strategize about their costs and another in which they cannot. We term the latter a ***real cost environment*** and we consider it a means of factoring out any issues that follow from the information uncertainty, rather than the self-interested nature of the agents.

4.1 No Efficient, Individually Rational, and Budget Balanced Mechanism Exists

Here we prove that no protocol can exist that is efficient, individually rational and budget balanced, for which there is a dominant strategy or at least a Bayesian Nash equilibrium.

To prove this, we use a reduction to the bilateral trading problem. In the bilateral trading problem, a single buyer wants to buy a single item from a single seller. Both the seller and the buyer have private values for the item v_s and v_b, respectively. Both values are independently drawn from the same uniform distribution between 0 and 1. In such problems there are cases in which trading is possible ($v_s \leq v_b$), but due to the buyer's and seller's strategies, trading might not be achieved. Myerson and Satterthwaite prove that for this problem no efficient protocol exists which is individually rational and budget balanced even in a Bayesian Nash equilibrium [10].

LEMMA 3. *Given the general task allocation problem defined in the integer program (1-4) even for only one task $|\Im| = 1$, no efficient protocol exists that is individually ra-*

[5]As an illustration, consider the task allocation described in Example 1 with a protocol, that given the agents' declaration, calculates the efficient solution (using the algorithm presented in figure 1) and distributes the surplus of each task equally among its contributing agents. In this case, A_3 is incentivised to declare a cost lower than the real one for subtask ST_{1_1} (for which $p(T_1) = 500$); by doing this it increases its chance to win this subtask.

tional, and budget balanced, even in a Bayesian Nash equilibrium.

PROOF. Assume, by way of contradiction, that there is an efficient, budget balanced and individually rational protocol PR_{task} for our task allocation problem. We can then use PR_{task} in order to design a protocol $PR_{BiTrading}$ that solves the bilateral trading problem.

Given an instance of the bilateral trading problem, the protocol $PR_{BiTrading}$ is defined as follows:

- Both the seller and the buyer bid their values for the item b_s and b_b, to the controller, respectively.
- The controller builds the following instance I_{task} of our task allocation problem that considers two agents, $\{A_1, A_2\}$, and one task including two subtasks, $T_1 = \{ST_{1_1}, ST_{1_2}\}$ where $p(T_1) = 1$. The agents' costs for both subtasks are uniformly distributed over $[0, 1]$. Agent A_1 is capable of performing only subtask ST_{1_1} and its cost is b_s. Agent A_2 is capable of performing only subtask ST_{1_2} and its cost is $1 - b_b$.
- Run PR_{task} to solve the instance of our task allocation problem I_{task}.
- If T_1 is allocated and the payments p_1 and p_2 are made to A_1 and A_2 respectively, (since the fixed payment constraint is $p_1 = 1 - p_2$) the seller sells the item to the buyer and the buyer pays p_1 to the seller, otherwise no trade occurs.

Since we assume that PR_{task} is efficient, budget balanced and individually rational, $PR_{BiTrading}$ is as well. This is a contradiction to Myerson and Satterthwaite's work which proved that there is no efficient, individually rational and budget balanced protocol that exists for the bilateral trading problem which is in Bayesian Nash equilibrium [10]. □

Hence, as the property of being strategy proof (i.e. truth telling is a dominant strategy) is a strictly stronger concept than the Bayesian Nash equilibrium, from Lemma 3 we can conclude that no efficient, individually rational and budget balanced protocol which is strategy proof exists for our general problem.

In the next section, we consider real cost environments, removing the agents' ability to falsely report their costs for their own benefit and demonstrate that there is still no efficient protocol meeting our requirements.

4.2 No Efficient Protocol Exists for Real Cost Environments

There are some situations in which the real costs of the agents may be verified. One example is a scenario where service providers do not directly receive payment for the costs of their raw materials, but payments are given against original receipts. Notice that in this case, in addition to the cost reimbursement, the agents expect a supplementary payment from the surplus of the task overall payments, and this is their motivation for participating in the allocation.

In such cases an agent's strategy reduces to one dimension: its willingness to perform certain subtasks (it no longer incorporates the subtask cost dimension). Therefore, a truth telling agent in this case is one that declares all the subtasks it is able to perform. If in this scenario an agent is forced to declare all the subtasks it can perform, we are reduced to the problem which is solved in section 2. However, we wish to consider the case where the agent is free to choose the subtasks it declares.

Agents	ST_{1_1}	ST_{1_2}	ST_{2_1}	ST_{2_2}
A_1	1	1	-	-
A_2	1	1	-	-
A_3	1	1	2	-
A_4	-	-	-	3

Table 2: Agents' capabilities and costs of example 4.2

Against this background, we will now go on to prove that even for real cost environments, no protocol achieving the efficient solution can exist that is individually rational and budget balanced (except where the allocation problem consists of a single task).

LEMMA 4. *Given the general task allocation problem defined in the integer program (1-4) and assuming there are at least two tasks, no protocol achieving the efficient solution can exist for real cost environments that is individually rational and budget balanced, not even in a Bayesian Nash equilibrium.*[6]

PROOF. We demonstrate that there is no general protocol by exhibiting a particular setting for which no protocol can exist. Assume there are four agents $\{A_1, A_2, A_3, A_4\}$ and two tasks, each of which includes two subtasks $T_1 = \{ST_{1_1}, ST_{1_2}\}$, $T_2 = \{ST_{2_1}, ST_{2_2}\}$, where the task payments are $p(T_1) = 100$ and $p(T_2) = 8$. The costs of subtasks ST_{1_1} and ST_{1_2} are 1 for all agents, and the costs for ST_{2_1} and ST_{2_2} are uniformly taken from [1,4]. The specific agents' costs and capabilities are given in table 2.

By the revelation principle[11] it is sufficient to prove that no efficient protocol exists where an agent's best strategy is to agree to perform all the subtasks it is able to perform and declare its real costs assuming all other agents will do the same. If agent $A3$ declares its willingness to perform all the subtasks it is capable of performing $\{ST_{1_1}, ST_{1_2}, ST_{2_1}\}$, it will be assigned to ST_{2_1} and its utility will be less than 8 (as any payment can not exceed the task payment). However, if A_3 agrees to perform only $\{ST_{1_1}, ST_{1_2}\}$, it becomes similar to A_1 and A_2 [7]. Consequently, its expected utility is $\frac{98}{3}$ (as the expected payment for agents with identical costs and the same declared subtasks should be the same). Therefore, for every protocol, A_3 won't declare subtask ST_{2_1}, resulting

[6]Note that this lemma cannot be derived from lemma 3 since here we assume a real cost environment (whilst in lemma 3 the agents can lie about their costs). On the other hand, lemma 3 cannot also be derived from this lemma (since there we proved that no desired solution can be achieved for one task and, for real cost environments a desired solution does exist for one task).

[7]Note that since A_3 knows the other agents' capabilities and subtask's costs distribution it knows both A_1 and A_2's costs and capabilities (the costs of ST_{1_1} and ST_{1_2} are equal to 1 for all agents). By assuming that all other agents declare all their subtasks, A_3 knows that by agreeing to perform only subtasks $\{ST_{1_1}, ST_{1_2}\}$ and declaring the costs of 1, it agrees to perform the same subtasks and declare the same costs as A_1 and A_2. Therefore the protocol should yield A_3 the same expected payment.

in an inefficient allocation as only one of the tasks can be allocated. □

However, in contrast to our results in section 4.1, next we prove that in the case where there is only one task to allocate, an efficient protocol exists for real cost environments.

LEMMA 5. *Given a real cost environment, an efficient protocol exists for the task allocation problem defined in the integer program (1-4) where* ***there is only one task****.*

PROOF. Consider a protocol in which each agent is requested to declare the subtasks it is interested in and its costs for them. Based on this information, the controller calculates the efficient allocation using the perfect minimum weighted matching algorithm (section 3). According to this efficient allocation, the protocol pays the cost of each allocated agent and an extra equal share of the surplus from the task payment. Given this protocol and the fact that there is only one task to allocate, it is easy to see that a dominant strategy exists where each agent is motivated to declare all the subtasks it is able to perform with their real costs (as we assume the real cost environment). By assuming that all agents use this dominant strategy, we can deduce that this protocol inevitably derives the real efficient solution as it does in the full information case. □

So far, we have shown that no single efficient protocol exists for the general case that achieves the desirable properties. However, there are settings for which such efficient protocols exist. The next section explores these settings.

4.3 Only Setting-Specific Efficient Protocols Exist

In order to show that efficient solutions can be generated for specific settings, but the ensuing solutions are specific to these settings, we consider two reasonably common environments. Specifically, we consider a Different Capabilities Setting (DCS) in which the agents have different capabilities and all the tasks are associated with the same overall payment, and a Different Task Payment Setting (DTPS) in which all agents are capable of performing all the subtasks and the tasks are associated with different payments. Then, for each setting, we provide an efficient protocol, and prove that this protocol is not efficient for the other one. In both settings we consider that the number of subtasks is equal to the number of agents and that each task includes K subtasks (the same number for all the tasks). Finally, we assume the agents' cost for any subtask is equal to c.

Given these settings, we consider two protocols: (1) the sequential protocol ***SEQP*** and (2) the simultaneous protocol ***SIMP***. According to both protocols, the task's payments are divided equally among the contributing agents. Then as it is assumed that each agent's cost to perform any of the candidate subtasks is c, each agent has to declare which subtasks it wants to perform but not their costs.

In detail, Protocol **SIMP** allocates all tasks simultaneously. Therefore, each agent has to declare at once all the subtasks (of all the tasks) it wants to perform. Based on these declarations, the protocol finds the most efficient allocation (using the algorithm described in figure 1). In contrast, the **SEQP** protocol allocates the tasks sequentially according to a decreasing order of the tasks' payments (a random order is used to break ties). In each round the controller announces a task, T_i, and the agents declare which of its subtasks they wish to perform. Based on these declarations, in each round, the SEQP protocol finds the most efficient allocation for the considered task. Next we show that for each setting, only one of the protocols achieves an efficient solution.

We begin by considering the efficient protocol for the DCS setting. In this case all subtasks' payments are equal; thus any agent is indifferent to which subtask it is allocated. Accordingly, under the SIMP protocol, a rational agent will declare all the subtasks it can perform. Moreover, as the SIMP protocol searches for the overall efficient allocation, based on the reported capabilities, it achieves the same result as in the cooperative system case. However, using SEQP in this setting does not achieve an efficient solution. This is because the allocation is made separately for each task in a greedy manner (i.e. finding the most efficient allocation for a given task at a time while ignoring the rest of the tasks). For example, consider the case where two agents, A_1 and A_2, are capable of performing subtask ST of task T, while A_1 is the only agent capable of performing subtask $ST^{'}$ of task $T^{'}$. For this case, if the allocation is made simultaneously for all tasks, A_1 will be allocated to $ST^{'}$ and A_2 will be allocated to ST so that both tasks may be completed. However, for SEQP protocol, T can be allocated first (as a random order is used to break ties). Therefore in this case, A_1 may be chosen to perform ST and consequently, $T^{'}$ won't be allocated.

For the DTPS setting, SEQP achieves an efficient solution. This is because when it allocates a certain task, it is the currently most attractive one (as all tasks with higher payments have already been allocated). Thus, each agent is motivated to declare all subtasks of the announced tasks. As all agents are capable of performing any of the tasks, the concern that some particular agent will be needed for a later allocation does not exist. By contrast, in this case SIMP does not achieve an efficient solution. This is because there may be a case where some subtasks may not be declared by any agent. To understand why SIMP fails in this setting, consider the case of an agent that declares all the subtasks. This agent might be assigned to the lowest payment subtask, while the highest payment one is allocated to another agent who declared only the subtasks with the highest payments.

In conclusion, we have demonstrated that in the non-cooperative case, there is no general protocol which fulfils all our desiderata. Moreover we have proved that there exist settings for which protocols can be developed. However, each such settings requires individual treatment and a tailored protocol. Consequently, any further attempt to develop a protocol for this task allocation problem in the non-cooperative setting, will have to commence by clearly defining the setting in which it is to operate.

5. RELATED WORK

Task allocation and coalition formation in multi-agent environments have been widely studied in recent years. However, to date most of the works including classical solutions for coalition formation (such as Shaply value and Kernal [5, 4]) assume complete information about the agents is available and that they are self interested. Therefore these solutions are not applicable to our cooperative and non-cooperative settings, where, respectively, cooperative agents and incomplete information are assumed.

However, the coalition formation work by Li and Zhang does consider a cooperative setting for a task allocation problem that is similar to ours [8]. However, the authors assume that the agents may use all their resources simultaneously (i.e using one resource type doesn't prevent the agent from using its other available resource types). This contradicts our assumption that agents have finite capabilities and can perform only a constant number of subtasks simultaneously. Thus their results cannot be applied to our problem. Other works on coalition formation in a cooperative setting are [15, 3] in which the authors looked for the optimal coalition structure. However, in their works they considered environments in which only one value is associated with each coalition. This contrasts with our case, where a coalition may have different values for each task it can perform depending on the subtask allocation.

In our algorithm for the cooperative setting we use matching in order to find an efficient solution, similar to other works like [13] that use matching in order to solve various allocation and assignment problems. However none of these works have the same constraints as we have (e.g., constraints C1-C3 in section 2) . Therefore their solutions are inapplicable to our case. The only work that considers the same problem constraints as ours is [6, 7]. However, as explained in section 1, they use heuristics (e.g., hill climbing), while we find the exact efficient solution.

Another related research area is the cost sharing mechanism problem [9]. According to this work, incomplete information, and, typically, a fixed known cost per coalition (similar to our task's payment) is assumed. But, in contrast to our work, they consider a much simpler structure of the problem. Specifically, the allocator is responsible for forming only one coalition, while in our case the allocator faces the more complex decision as of which group of agents to assign a certain task to and how to allocate the subtasks to its coalition members.

Our problem also has similarities to a combinatorial market problem [12]. Each subtask can be treated as an item, each task as a bundle of complementary items, and each agent as a seller with substitutable valuations, since an agent can perform only one sub-task simultaneously (see section 2). However, in the combinatorial market the payment for each bundle is determined by the auction. By contrast, in our problem the payment for each task is predefined and known. In addition, in most work on combinatorial markets the surplus of the overall buyers' values and the overall sellers' values is divided among all participants. This is not appropriate in our model, where the task payment is divided only among the agents that perform the task.

6. CONCLUSIONS

In this paper we have considered a complex task allocation problem that needs to be solved by a group of autonomous agents working together. Given the ubiquity of this problem, it is surprising that to date it has not received significant attention. To rectify this omission, we consider both cooperative and non-cooperative settings of the problem. For the former setting we provide a practical algorithm for finding an efficient solution. The time complexity of this algorithm is polynomial in case the number of tasks is bounded by a small constant number (which is often the case in real world situations). In contrast, for the non-cooperative setting we prove that no general protocol achieving the efficient solution exists for our problem that satisfies our desired properties. However, we show that there exist some non-cooperative settings scenarios for which an efficient protocol does exist. Unfortunately, each such setting requires the design of a bespoke protocol, since the solution for one setting is likely to be inappropriate for another. In so doing, this study hopes to assist system designers confronting this problem to understand what can and cannot be achieved.

7. ACKNOWLEDGMENTS

This research was partially funded by the DIF-DTC project (8.6) on Agent-Based Control. This work was also supported in part by NSF No. IIS0208608 and ISF 1211. Kraus is also aliated with UMIACS. In memory of Matat Rosenfeld-Adler.

8. REFERENCES

[1] D. Avis and C. Lai. The probabilistic analysis of a heuristic for the assignment problem. *SIAM J. Comput.*, 17(4):732–741, 1988.

[2] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[3] V. Dang and N. Jennings. Generating coalition structures with finite bound from the optimal guarantees. In *AAMAS'04*, pages 564–571, 2004.

[4] J. Kahan and A. Rapoport. *Theories of coalition formation*. Lawrence Erlbaum Associates, 1984.

[5] S. Kraus and O. Shehory. Feasible formation of coalitions among autonomous agents in non-super additive environments. *Computational Intelligence*, 15(3):218–251, 1999.

[6] S. Kraus, O. Shehory, and G. Taase. Coalition formation with uncertain heterogeneous information. In *AAMAS03*, pages 1–8, 2003.

[7] S. Kraus, O. Shehory, and G. Taase. The advantages of compromising in coalition formation with incomplete information. In *AAMAS04*, pages 588–595, 2004.

[8] H. Lau and L. Zhang. Task allocation via multi-agent coalition formation: taxonomy, algorithms and complexity. In *ICTAI'03*, pages 346–350, 2003.

[9] H. Moulin and S. Shenker. Strategyproof sharing of submodular costs:budget balance versus efficiency. *Economic Theory*, 18(3):511–533, 2001.

[10] R. B. Myerson and M. A. Satterthwaite. Efficient mechanisms for bilateral trading. *Economic Theory*, 29(3):265–281, 1983.

[11] D. Parkes. Mechanism design chapter 2 in phd dissertation. 2001.

[12] A. Pekec and M. H. Rothkopf. Combinatorial auction design. *Management Science*, 49(11):1485–1503, 2003.

[13] M. Penn and M. Tennenholtz. Constrained multi-object auctions and b-matching. *Information Processing Letters*, 75(1–2):29–34, July 2000.

[14] T. Sandholm. An algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135:1–54, 2002.

[15] S. Sen and P. S. Dutta. Searching for optimal coalition structures. In *ICMAS00*, pages 287–292, 2000.

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