Distributed Task Allocation in Social Networks

Mathijs de Weerdt
Delft Technical University
Delft, The Netherlands
M.M.deWeerdt@tudelft.nl

Yingqian Zhang
Delft Technical University
Delft, The Netherlands
Yingqian.Zhang@tudelft.nl

Tomas Klos
Center for Mathematics and Computer Science (CWI)
Amsterdam, The Netherlands
tomas.klos@cwi.nl

ABSTRACT

This paper proposes a new variant of the task allocation problem, where the agents are connected in a social network and tasks arrive at the agents distributed over the network. We show that the complexity of this problem remains NP-hard. Moreover, it is not approximable within some factor. We develop an algorithm based on the contract-net protocol. Our algorithm is completely distributed, and it assumes that agents have only local knowledge about tasks and resources. We conduct a set of experiments to evaluate the performance and scalability of the proposed algorithm in terms of solution quality and computation time. Three different types of networks, namely small-world, random and scale-free networks, are used to represent various social relationships among agents in realistic applications. The results demonstrate that our algorithm works well and that it scales well to large-scale applications.

Categories and Subject Descriptors

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

General Terms

Algorithms, Experimentation

Keywords

Task Allocation, Social Networks, Agents, Resources, Computational Complexity

1. INTRODUCTION

Recent years have seen a lot of work on task and resource allocation methods, which can potentially be applied to many real-world applications. However, some interesting applications where relations between agents play a role require a slightly more general model. Such situations appear very frequently in real-world scenarios, and recent technological developments are bringing more of them within the range of task allocation methods. Especially in business applications, preferential partner selection and interaction is very common, and this aspect becomes more important for task allocation research, to the extent that technological developments need to be able to support it.

For example, the development of semantic web and grid technologies leads to increased and renewed attention for the potential of the web to support business processes [7, 15]. As an example, virtual organizations (VOs) are being re-invented in the context of the grid, where "they are composed of a number of autonomous entities (representing different individuals, departments and organizations), each of which has a range of problem-solving capabilities and resources at its disposal" [15, p. 237]. The question is how VOs are to be dynamically composed and re-composed from individual agents, when different tasks and subtasks need to be performed. This would be done by allocating them to different agents who may each be capable of performing different subsets of those tasks. Similarly, supply chain formation (SCF) is concerned with the, possibly ad-hoc, allocation of services to providers in the supply chain, in such a way that overall profit is optimized [6, 21].

Traditionally, such allocation decisions have been analyzed using transaction cost economics (TCE) [4], which takes the transaction between consecutive stages of development as its basic unit of analysis, and considers the firm and the market as alternative structural forms for organizing transactions. (Transaction cost) economics has traditionally built on analysis of comparative statics: the central problem of economic organization is considered to be the adaptation of organizational forms to the characteristics of transactions. More recently, TCE's founding father, Ronald Coase, acknowledged that this is too simplistic an approach [5, p. 245]: "The analysis cannot be confined to what happens within a single firm. (...) What we are dealing with is a complex interrelated structure."

In this paper, we study the problem of task allocation from the perspective of such a complex interrelated structure. In particular, 'the market' cannot be considered as an organizational form without considering *specific* partners to interact with on the market [11]. Specifically, therefore, we consider agents to be connected to each other in a social network. Furthermore, this network is not fully connected: as informed by the business literature, firms typically have established working relations with limited numbers of preferred partners [10]; these are the ones they consider when new tasks arrive and they have to form supply chains to allocate those tasks [19]. Other than modeling the interrelated

AAMAS'07 May 14–18 2007, Honolulu, Hawai'i, USA.

978-81-904262-7-5 (RPS)

structure between business partners, the social network introduced in this paper can also be used to represent other types of connections or constraints among autonomous entities that arise from other application domains.

The next section gives a formal description of the task allocation problem on social networks. In Section 3, we prove that the complexity of this problem remains NP-hard. We then proceed to develop a distributed algorithm in Section 4, and perform a series of experiments with this algorithm, as described in Section 5. Section 6 discusses related work, and Section 7 concludes.

2. PROBLEM DESCRIPTION

We formulate the social task allocation problem in this section. There is a set $\mathcal{A}$ of agents: $\mathcal{A} = \{a_1, \ldots, a_m\}$. Agents need resources to complete tasks. Let $R = \{r_1, \ldots, r_k\}$ denote the collection of the resource types available to the agents $\mathcal{A}$. Each agent $a \in \mathcal{A}$ controls a fixed amount of resources for each resource type in R, which is defined by a resource function: $rsc : \mathcal{A} \times R \to \mathbb{N}$. Moreover, we assume agents are connected by a *social network*.

Definition 1 (Social network). *An agent social network $SN = (\mathcal{A}, AE)$ is an undirected graph, where vertices $\mathcal{A}$ are agents, and each edge $(a_i, a_j) \in AE$ indicates the existence of a social connection between agents a_i and a_j.*

Suppose a set of tasks $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ arrives at such an agent social network. Each task $t \in \mathcal{T}$ is then defined by a tuple $\langle u(t), rsc(t), loc(t)\rangle$, where $u(t)$ is the utility gained if task t is accomplished, and the resource function $rsc : \mathcal{T} \times R \to \mathbb{N}$ specifies the amount of resources required for the accomplishment of task t. Furthermore, a location function $loc : \mathcal{T} \to \mathcal{A}$ defines the locations (i.e., agents) at which the tasks arrive in the social network. An agent a that is the location of a task t, i.e. $loc(t) = a$, is called the *manager* of this task.

Each task $t \in \mathcal{T}$ needs some specific resources from the agents in order to complete the task. The exact assignment of tasks to agents is defined by a *task allocation*.

Definition 2 (Task allocation). *Given a set of tasks $\mathcal{T} = \{t_1, \ldots, t_n\}$ and a set of agents $\mathcal{A} = \{a_1, \ldots, a_m\}$ in a social network SN, a **task allocation** is a mapping $\phi : \mathcal{T} \times \mathcal{A} \times R \to \mathbb{N}$. A **valid** task allocation in SN must satisfy the following constrains:*

- *A task allocation must be correct. Each agent $a \in \mathcal{A}$ cannot use more than its available resources, i.e. for each $r \in R$, $\sum_{t \in \mathcal{T}} \phi(t, a, r) \le rsc(a, r)$.*
- *A task allocation must be complete. For each task $t \in \mathcal{T}$, either all allocated agents' resources are sufficient, i.e. for each $r \in R$, $\sum_{a \in \mathcal{A}} \phi(t, a, r) \ge rsc(t, r)$, or t is not allocated, i.e. $\phi(t, \cdot, \cdot) = 0$.*
- *A task allocation must obey the social relationships. Each task $t \in \mathcal{T}$ can only be allocated to agents that are (direct) neighbors of agent $loc(t)$ in the social network SN. Each such agent that can contribute to a task is called a **contractor**.*

We write T_ϕ to represent the tasks that are fully allocated in ϕ. The utility of ϕ is then the summation of the utilities of each task in T_ϕ, i.e., $U_\phi = \sum_{t \in T_\phi} u(t)$. Using this notation, we define the *efficient task allocation* below.

Definition 3 (Efficient task allocation). *We say a task allocation ϕ is **efficient** if it is valid and U_ϕ is maximized, i.e., $U_\phi = \max(\sum_{t \in T_\phi} u(t))$.*

We are now ready to define the task allocation problem in social network that we study in this paper.

Definition 4 (Social task allocation problem). *Given a set of agents $\mathcal{A}$ connected by a social network $SN = (\mathcal{A}, AE)$, and a finite set of tasks $\mathcal{T}$, the **social task allocation problem** (or **STAP** for short) is the problem of finding the efficient task allocation ϕ, such that ϕ is valid and the social welfare U_ϕ is maximized.*

3. COMPLEXITY RESULTS

The traditional task allocation problem, TAP (without the condition of the social network SN), is NP-complete [18], and the complexity comes from the fact that we need to evaluate the exponential number of subsets of the task set. Although we may consider the TAP as a special case of the STAP by assuming agents are fully connected, we cannot directly use the complexity results from the TAP, since we study the STAP in an *arbitrary* social network, which, as we argued in the introduction, should be partially connected.

We now show that the TAP with an *arbitrary social network* is also NP-complete, even when the utility of each task is 1, and the quantity of all required and available resources is 1.

Theorem 1. *Given the social task allocation problem with an arbitrary social network, as defined in Definition 4, the problem of deciding whether a task allocation ϕ with utility more than k exists is NP-complete.*

Proof. We first show that the problem is in NP. Given an instance of the problem and an integer k, we can verify in polynomial time whether an allocation ϕ is a valid allocation and whether the utility of ϕ is greater than k.

We now prove that the STAP is NP-hard by showing that MAXIMUM INDEPENDENT SET $\le_P$ STAP. Given an undirected graph $G = (V, E)$ and an integer k, we construct a network $G' = (V', E')$ which has an efficient task allocation with k tasks of utility 1 allocated if and only if G has an independent set (IS) of size k.

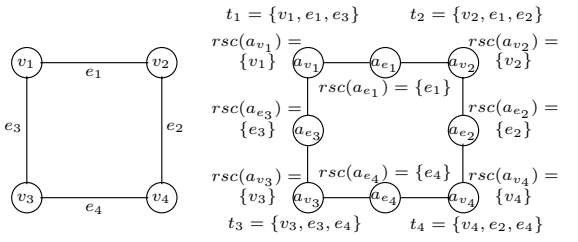

Figure 1: The MIS problem can be reduced to the STAP. The left figure is an undirected graph G, which has the optimal solution $\{v_1, v_4\}$ or $\{v_2, v_3\}$; the right figure is the constructed instance of the STAP, where the optimal allocation is $\{t_1, t_4\}$ or $\{t_2, t_3\}$.

An instance of the following construction is shown in Figure 1. For each node $v \in V$ and each edge $e \in E$ in the graph G, we create a vertex agent a_v and an edge agent a_e in G'.

When v was incident to e in G we correspondingly add an edge e' in G' between a_v and a_e. We assign each agent in G' one resource, which is related to the node or the edge in the graph G, i.e., for each $v \in V$, $rsc(a_v) = \{v\}$ (here we write $rsc(a)$ and $rsc(t)$ to represent the set of resources available to/required by a and t), and for each $e \in E$, $rsc(a_e) = \{e\}$. Each vertex agent a_{v_i} in G' has a task t_i that requires a set of neighboring resources $t_i = \{v_i\} \cup \{e | e = (u, v_i) \in E\}$. There is no task on the edge agents in G'. We define utility 1 for each task, and the quantity of all required and available resources to be 1.

Taken an instance of the IS problem, suppose there is a solution of size k, i.e., a subset $N \subseteq V$ such that no two vertices in N are joined by an edge in E and $|N| = k$. N specifies a set of vertex agents A_N in the corresponding graph G'. Given two agents $a_1, a_2 \in A_N$ we now know that there is no edge agent a_e connected to both a_1 and a_2. Thus, for each agent $a \in A_N$, a assigns its task to the edge agents which are connected to a. All other vertex agents $a' \notin A_N$ are not able to assign their tasks, since the required resources of the edge agents are already used by the agents $a \in A_N$. The set of tasks of the agents A_N ($|A_N| = k$) is thus the maximum set of tasks that can be allocated. The utility of this allocation is k.

Similarly, if there is a solution for the STAP with the utility value k, and the allocated task set is N, then for the IS problem, there exists a maximum independent set N of size k in G. An example can be found in Figure 1. □

We just proved that the STAP is NP-hard for an arbitrary graph. In our proof, the complexity comes from the introduction of a social network. One may expect that the complexity of this problem can be reduced for some networks where the number of neighbors of the agents is bounded by a fixed constant. We now give a complexity result on this class of networks as follows.

THEOREM 2. *Let the number of neighbors of each agent in the social network SN be bounded by Δ for $\Delta \geq 3$. Computing the efficient task allocation given such a network is* NP*-complete. In addition, it is not approximable within Δ^{ε} for some $\varepsilon > 0$.*

PROOF. It has been shown in [2] that the maximum independent set problem in the case of the degree bounded by Δ for $\Delta \geq 3$ is NP-complete and is not approximable within Δ^{ε} for some $\varepsilon > 0$. Using the similar reduction from the proof of Theorem 1, this result also holds for the STAP. □

Since our problem is as hard as MIS as shown in Theorem 1, it is not possible to give a worst case bound better than Δ^{ε} for any polynomial time algorithm, unless P = NP.

4. ALGORITHMS

To deal with the problem of allocating tasks in a social network, we present a distributed algorithm. We introduce this algorithm by describing the protocol for the agents. After that we give the optimal, centralized algorithm and an upper bound algorithm, which we use in Section 5 to benchmark the quality of our distributed algorithm.

4.1 Protocol for distributed task allocation

We can summarize the description of the task allocation problem in social networks from Section 2 as follows. We have a (social) network of agents. Each agent has a set of resources of different types at its disposal. We also have a set of tasks. Each task requires some resources, has a fixed benefit, and is located at a certain agent. This agent is called a manager. We only allow neighboring agents to help with a task. These agents are called contractors. Agents can fulfill the role of manager as well as contractor. The problem is to find out which tasks to execute, and which resources of which contractors to use for these tasks.

Algorithm 1 Greedy distributed allocation protocol (GDAP).

Each manager a calculates the efficiency $e(t)$ for each of their tasks $t \in T_a$, and then **while** $T_a \neq \emptyset$:

1. Each manager a selects the most efficient task $t \in T_a$ such that for each task $t' \in T_a$: $e(t') \leq e(t)$.
2. Each manager a requests help for t from all its neighbors (of a) by informing these neighbors of the efficiency $e(t)$ and the required resources for t.
3. Contractors receive and store all requests, and then offer all relevant resources to the manager for the task with the highest efficiency.
4. The managers that have received sufficient offers allocate their tasks, and inform each contractor which part of the offer is accepted. When a task is allocated, or when a manager has received offers from all neighbors, but still cannot satisfy its task, the task is removed from the task list T_a.
5. Contractors update their used resources.

The idea of the protocol is as follows. All manager agents $a \in \mathcal{A}$ try to find neighboring contractors to help them with their task(s) $T_a = \{t_i \in \mathcal{T} \mid loc(t_i) = a\}$. They start with offering the task that is most efficient in terms of the ratio between benefit and required resources. Out of all tasks offered, contractors select the task with the highest efficiency, and send a bid to the related manager. A bid consists of all the resources the agent is able to supply for this task. If sufficient resources have been offered, the manager selects the required resources and informs all contractors of its choice. The efficiency of a task is defined as follows:

DEFINITION 5. *The* efficiency e of a task $t \in \mathcal{T}$ *is defined by the utility of this task divided by the sum of all required resources:* $e(t) = \frac{u(t)}{\sum_{r \in R} rsc(t,r)}$.

A more detailed description of this protocol can be found in Algorithm 1. Here it is also defined how to determine when a task should not be offered anymore, because it is impossible to fulfill locally. Obviously, a task is also not offered anymore when it has been allocated. This protocol is such that, when no two tasks have exactly the same efficiency, in every iteration at least one task is removed from a task list.[1] From this the computation and communication property of the algorithm follows.

PROPOSITION 1. *For a STAP with n tasks and m agents, the run time of the distributed algorithm is $O(nm)$, and the number of communication messages is $O(n^2 m)$.*

[1]Even when some tasks have the same efficiency, it is straightforward to make this result work. For example, the implementation can ensure that the contractors choose the task with the lowest task-id.

Algorithm 2 Optimal social task allocation (OPT).

Repeat the following for each combination of tasks:

1. If the total reward for this combination is higher than any previous combination, test if this combination is feasible as follows:
2. Create a network flow problem for each resource type $r \in R$ (separately) as follows:
   (a) Create a source s and a sink s'.
   (b) For each agent $a \in \mathcal{A}$ create an agent node and an edge from s to this node with capacity equal to the amount of resources of type r agent a has.
   (c) For each task $t \in \mathcal{T}$ create a task node and an edge from this node to s' with capacity equal to the amount of resources of type r task T requires.
   (d) For each agent a connect the agent node to all task nodes of neighboring tasks, i.e., $t \in \{t \in \mathcal{T} \mid (a, loc(t)) \in AE\}$. Give this connection unlimited capacity.
3. Solve the maximum flow problem for the created flow networks. If the maximum flow in each network is equal to the total required resources of that type, the current combination of tasks is feasible. In that case, this is the current best combination of tasks.

Proof. In the worst case, in each iteration exactly one task is removed from a task list, so there are n iterations. In each iteration in the worst case (i.e., a fully connected network), for each of the $O(n)$ managers, $O(m)$ messages are sent. Next the task with the highest efficiency can be selected by each contractor in $O(n)$. Assigning an allocation can be done in $O(m)$. This leads to a total of $O(n+m)$ operations for each iteration, and thus $O(n^2+nm)$ operations in total. The number of messages sent is $O(n(nm+nm+nm)) = O(n^2m)$. □

We establish the quality of this protocol experimentally (in Section 5). Preferably, we compare the results to the optimal solution.

4.2 Optimal social task allocation

The optimal task allocation algorithm should deal with the restrictions posed by the social network. For this NP-complete problem we used an exponential brute-force algorithm to consider relevant combinations of tasks to execute. For each combination we use a maximum-flow algorithm to check whether the resources are sufficient for the selected subset of tasks. The flow network describes which resources can be used for which tasks, depending on the social network. If the maximum flow is equal to the sum of all resources required by the subset of tasks, we know that a feasible solution exists (see Algorithm 2). Clearly, we cannot expect this optimal algorithm to be able to find solutions for larger problem sizes. To establish the quality of our protocol for large instances, we use the following method to determine an upper bound.

4.3 Upper bound for social task allocation

Given a social task allocation problem, if the number of resource types for every task $t \in \mathcal{T}$ is bounded by 1, the problem is polynomially solvable by transforming it to a flow network problem. Our method for efficiently calculating an upper bound for STAP makes use of this special case by converting any given STAP instance P into a new problem P' where each task only has one resource type.

Algorithm 3 An upper bound for social task allocation (UB).

Create a network flow problem with costs as follows:

1. Create a source s and a sink s'.
2. For each agent $a \in \mathcal{A}$ and each resource type $r_i \in R$, create an agent-resource node a'_i, and an edge from s to this node with capacity equal to the amount of resources of type r agent a has available and with costs 0.
3. For each task $t \in \mathcal{T}$ and each resource type $r_i \in R$, create a task-resource node t'_i, and an edge from this node to s' with capacity equal to the amount of resources of type r task t requires and costs $-e(t)$.
4. For each resource type $r_i \in R$ and for each agent a connect the agent-resource node a'_i to all task-resource nodes t'_i for neighboring tasks $t \in \{t \in \mathcal{T} \mid (a, loc(t)) \in AE \text{ or } a = loc(t)\}$. Give this connection unlimited capacity and zero costs.
5. Create an edge directly from s to s' with unlimited capacity and zero costs.

Solve the minimum cost flow network problem for this network. The costs of the resulting network is an upper bound for the social task allocation problem.

More specifically, for every task $t \in \mathcal{T}$ with utility $u(t)$, we do the following. Let m be the number of resource types $\{r_1, \ldots, r_m\}$ required by t. We then split t into a set of m tasks $T' = \{t'_1, \ldots, t'_m\}$ where each task t'_i only has one unique resource type (of $\{r_1, \ldots, r_m\}$) and each task has a fair share of the utility, i.e., the efficiency of t from Definition 5 times the amount of this resource type $rsc(t'_i, r_i)$. After polynomially performing this conversion for every task in $\mathcal{T}$, the original problem P becomes the special case P'. Note that the set of valid allocations in P is only a subset of the set of valid allocations in P', because it is now possible to partially allocate a task. From this it is easy to see that the solution of P' gives an upper bound of the solution of the original problem P.

To compute the optimal solution for P', we transform it to a minimum cost flow problem. We model the "cost" in the flow network by the negation of the new task's utility. A polynomial-time implementation of a scaling minimum cost flow algorithm [9] is used for the computation. The resulting minimum cost flow represents a maximum allocation of the tasks for P'. The detailed modeling is described in Algorithm 3. In the next section, we use this upper bound to estimate the quality of the GDAP for large-scale instances.

5. EXPERIMENTS

We implemented the greedy distributed allocation protocol (GDAP), the optimal allocation algorithm (OPT), and the upper bound algorithm (UB) in Java, and tested them on a Linux PC. The purpose of these experiments is to study the performance of the distributed algorithm in different problem settings using different social networks. The per-

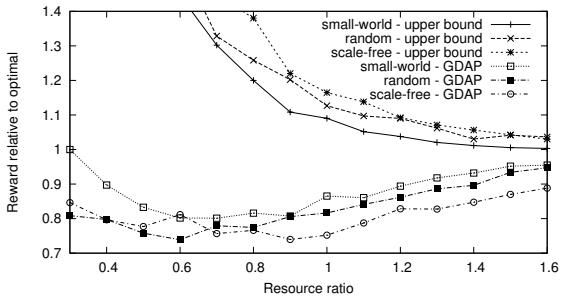

Figure 2: The solution qualities of the GDAP and the upper bound depend on the resource ratio.

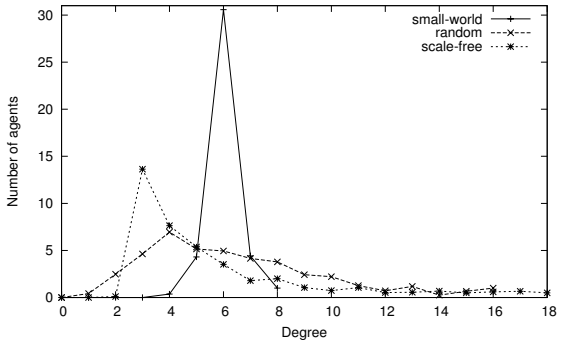

Figure 3: The histogram of the degrees.

formance measurements are the solution quality and computation time, where the solution quality (SQ) is computed as follows. When the number of tasks is small, we compare the output of the distributed algorithm with the optimal solution, i.e., $SQ = \frac{GDAP}{OPT}$, but if it is not feasible to compute the optimal solution, we use the value returned by the upper bound algorithm for evaluation, i.e., $SQ = \frac{GDAP}{UB}$.

To see whether the latter is a good measure, we also compare the quality of the upper bound to the optimal solution for smaller problems. In the following, we describe the setup of all experiments, and present the results.

5.1 Experimental settings

We consider several experimental environments. In all environments the agents are connected by a social network. In the experiments, three different networks are used to simulate the social relationships among agents in potential real-world problems.

Small-world networks are networks where most neighbors of an agent are also connected to each other. For the experiments we use a method for generating random small-world networks proposed by Watts et al. [22], with a fixed rewiring probability $p = 0.05$.

Scale-free networks have the property that a couple of nodes have many connections, and many nodes have only a small number of connections. To generate these we use the implementation in the JUNG library of the generator proposed by Barabási and Albert [3].

We also generate *random networks* as follows. First we connect each agent to another agent such that all agents are connected. Next, we randomly add connections until the desired average degree has been reached.

We now describe the different settings used in our experiments with both small and large-scale problems.

Setting 1. The number of agents is 40, and the number of tasks is 20. The number of different resource types is bounded by 5, and the average number of resources required by a task is 30. Consequently, the total number of resources required by the tasks is fixed. However, the resources available to the agents are varied. We define the *resource ratio* to refer to the ratio between the total number of available resources and the total number of required resources. Resources are allocated uniformly to the agents. The average degrees of the networks may also change. In this setting the task benefits are distributed normally around the number of resources required.

Setting 2. This setting is similar to Setting 1, but here we let the benefits of the tasks vary dramatically—40% of the tasks have around 10 times higher benefit than the other 60% of the tasks.

Setting 3. This setting is for large-scale problems. The ratio between the number of agents and the number of tasks is set to 5/3, and the number of agents varies from 100 to 2000. We also fix the resource ratio to 1.2 and the average degree to 6. The number of different resource types is 20, and the average resource requirement of a tasks is 100. The task benefits are again normally distributed.

5.2 Experimental results

The experiments are done with the three different settings in the three different networks mentioned before, where each recorded data is the average over 20 random instances.

5.2.1 Experiment 1

Experimental setting 1 is used for this set of experiments. We would like to see how the GDAP behaves in the different networks when the number of resources available to the agents is changing. We also study the behavior of our upper bound algorithm. For this experiment we fix the average number of neighbors (*degree*) in each network type to six.

In Figure 2 we see how the quality of both the upper bound and the GDAP algorithm depends on the resource ratio. Remarkably, for lower resource ratios our GDAP is much closer to the optimal allocation than the upper bound. When the resource ratio grows above 1.5, the graphs of the upper bound and the GDAP converge, meaning that both are really close to the optimal solution. This can be explained by the fact that when plenty of resources are available, all tasks can be allocated without any conflicts. However, when resources are very scarce, the upper bound is much too optimistic, because it is based on the allocation of sub-tasks per resource type, and does not reason about how many of the tasks can actually be allocated completely. We also notice from the graph that the solution quality of the GDAP on all three networks is quite high (over 0.8) when the available resource is very limited (0.3). It drops below 0.8 with the increased ratio and goes up again once there are plenty of resources available (resource ratio 0.9). Clearly, if

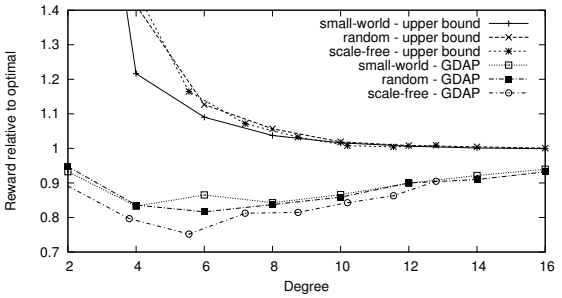

Figure 4: The quality of the GDAP and the upper bound depend on the network degree.

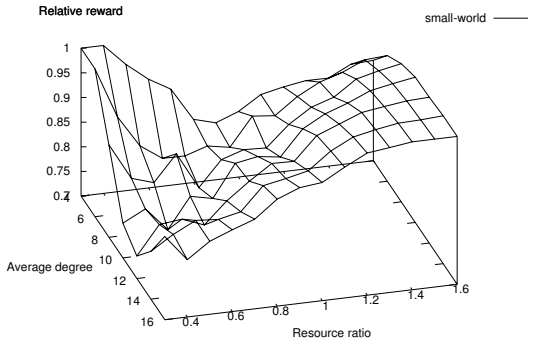

Figure 5: The quality of the GDAP depends on both the resource ratio and the network degree.

resources are really scarce, only a few tasks can be successfully allocated even by the optimal algorithm. Therefore, the GDAP is able to give quite a good allocation.

Although the differences are minor, it can also be seen that the results for the small-world network are consistently slightly better than those of random networks, which in turn outperform scale-free networks. This can be understood by looking at the distribution of the agents' degree, as shown in Figure 3. In this experiment, in the small-world network almost every manager has a degree of six. In random networks, the degree varies between one and about ten. However, in the scale-free network, most nodes have only three or four connections, and only a very few have up to twenty connections. As we will see in the next experiment, having more connections means getting better results.

For the next experiment we fix the resource ratio to 1.0 and study the quality of both the upper bound and the GDAP algorithm related to the degree of the social network. The result can be found in Figure 4. In this figure we can see that a high average degree also leads to convergence of the upper bound and the GDAP. Obviously, when managers have many connections, it becomes easier to allocate tasks. An exception is, similar to what we have seen in Figure 2, that the solution of the GDAP is also very good if the connections are extremely limited (degree 2), due to the fact that the number of possibly allocated tasks is very small. Again we see that the upper bound is not that good for problems where resources are hard to reach, i.e. in social networks with a low average degree.[2]

Since the solution quality clearly depends on the resource ratio as well as on the degree of the social network, we study the effect of changing both, to see whether they influence each other. Figure 5 shows how the solution quality depends on both the resource ratio and the network degree. This graph confirms the results that the GDAP performs better for problems with higher degree and higher resource ratio. However, it is now also more clear that it performs better for very low degree and resource availability. For this experiment with 40 agents and 20 tasks, the worst performance is met for instances with degree six and resource ratio 0.6 to instances with degree twelve and resource ratio 0.3. But even for those instances, the performance lies above 0.7.

[2]The consistent standard deviation of about 15% over the 20 problem instances is not displayed as error-bars in these first graphs, because it would obfuscate the interesting correlations that can now be seen.

5.2.2 *Experiment 2*

To study the robustness of the GDAP against different problem settings, we generate instances where the task benefit distribution is different: 40% of the tasks gets a 10 times higher benefit (as described in Setting 2). The effect of this different distribution can be seen in Figure 6. These two graphs show that the results for the "skewed" task benefit distribution are slightly better on average, both when varying the resource ratio, and when varying the average degree of the network. We argue that this can be explained by the greedy nature of GDAP, which causes the tasks with high efficiency to be allocated first, and makes the algorithm perform better in this heterogeneous setting.

5.2.3 *Experiment 3*

The purpose of this final experiment is to test whether the algorithm can be scaled to large problems, like applications running on the internet. We therefore generate instances where the number of agents varies from 100 to 2000, and simultaneously increase the number of tasks from 166 to 3333 (Setting 3). Figure 7 shows the run time for these instances on a Linux machine with an AMD Opteron 2.4 GHz processor. These graphs confirm the theoretical analysis from the previous section, saying that both the upper bound and the GDAP are polynomial. In fact, the graphs show that the GDAP almost behaves linearly. Here we see that the locality of the GDAP really helps in reducing the computation time. Also note that the GDAP requires even less computation time than the upper bound.

The quality of the GDAP for these large instances cannot be compared to the optimal solution. Therefore, in Figure 8 the upper bound is used instead. This result shows that the GDAP behaves stably and consistently well with the increasing problem size. It also shows once more that the GDAP performs better in a small-world network.

6. RELATED WORK

Task allocation in multiagent systems has been investigated by many researchers in recent years with different assumptions and emphases. However, most of the research to date on task allocation does not consider social connections among agents, and studies the problem in a centralized

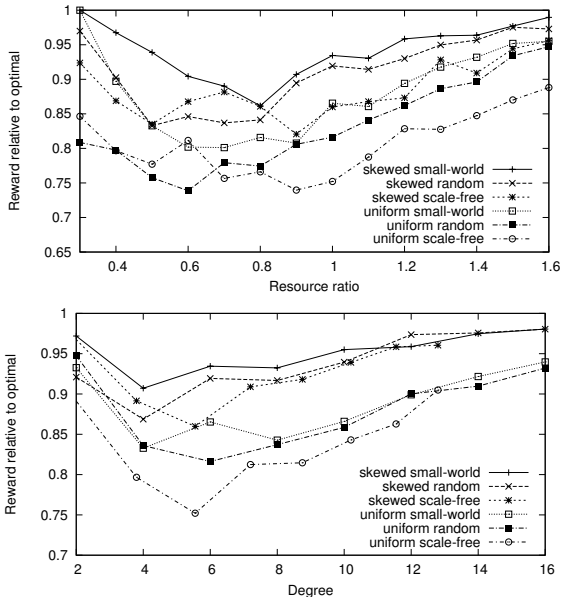

Figure 6: The quality of the GDAP algorithm for a uniform and a skewed task benefit distribution related to the resource ratio (the first graph), and the network degree (the second graph).

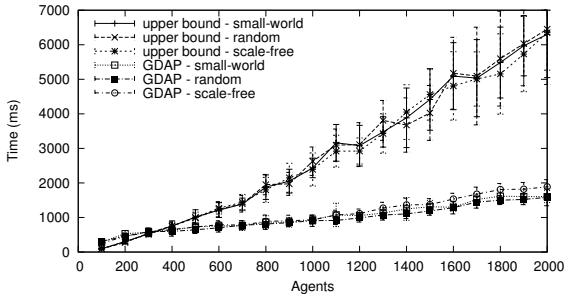

Figure 7: The run time of the GDAP algorithm.

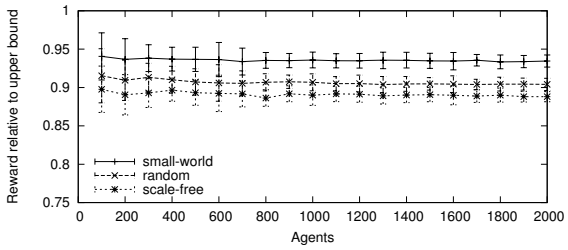

Figure 8: The quality of the GDAP algorithm compared to the upper bound.

setting. For example, Kraus et al. [12] develop an auction protocol that enables agents to form coalitions with time constraints. It assumes each agent knows the capabilities of all others. The proposed protocol is centralized, where one manager is responsible for allocating the tasks to all coalitions. Manisterski at al. [14] discuss the possibilities of achieving efficient allocations in both cooperative and non-cooperative settings. They propose a centralized algorithm to find the optimal solution. In contrast to this work, we introduce also an efficient completely distributed protocol that takes the social network into account.

Task allocation has also been studied in distributed settings by for example Shehory and Kraus [18] and by Lerman and Shehory [13]. They propose distributed algorithms with low communication complexity for forming coalitions in large-scale multiagent systems. However, they do not assume the existence of any agent network. The work of Sander et al. [16] introduces computational geometry-based algorithms for distributed task allocation in geographical domains. Agents are then allowed to move and actively search for tasks, and the capability of agents to perform tasks is homogeneous. In order to apply their approach, agents need to have some knowledge about the geographical positions of tasks and some other agents. Other work [17] proposes a location mechanism for open multiagent systems to allocate tasks to unknown agents. In this approach each agent caches a list of agents they know. The analysis of the communication complexity of this method is based on lattice-like graphs, while we investigate how to efficiently solve task allocation in a social network, whose topology can be arbitrary.

Networks have been employed in the context of task allocation in some other works as well, for example to limit the interactions between agents and mediators [1]. Mediators in this context are agents who receive the task and have connections to other agents. They break up the task into subtasks, and negotiate with other agents to obtain commitments to execute these subtasks. Their focus is on modeling the decision process of just a single mediator. Another approach is to partition the network into cliques of nodes, representing coalitions which the agents involved may use as a coordination mechanism [20]. The focus of that work is distributed coalition formation among agents, but in our approach, we do not need agents to form groups before allocating tasks.

Easwaran and Pitt [6] study 'complex tasks' that require 'services' for their accomplishment. The problem concerns the allocation of subtasks to service providers in a supply chain. Another study of task allocation in supply chains is [21], where it is argued that the defining characteristic of Supply Chain Formation is *hierarchical subtask decomposition* (HSD). HSD is implemented using task dependency networks (TDN), with agents and goods as nodes, and I/O relations between them as edges. Here, the network is given, and the problem is to select a subgraph, for which the authors propose a market-based algorithm, in particular, a series of auctions. Compared to these works, our approach is more general in the sense that we are able to model different types of connections or constraints among agents for different problem domains in addition to supply chain formation.

Finally, social networks have been used in the context of team formation. Previous work has shown how to learn which relations are more beneficial in the long run [8], and adapt the social network accordingly. We believe these results can be transferred to the domain of task allocation as well, leaving this as a topic for further study.

7. CONCLUSIONS

In this paper we studied the task allocation problem in a social network (STAP), which can be seen as a new, more general, variant of the TAP. We believe it has a great amount of potential for realistic problems. We provided complexity results on computing the efficient solution for the STAP, as well as a bound on possible approximation algorithms. Next, we presented a distributed protocol, related to the contract-net protocol. We also introduced an exponential algorithm to compute the optimal solution, as well as a fast upper-bound algorithm. Finally, we used the optimal solution and the upper-bound (for larger instances) to conduct an extensive set of experiments to assess the solution quality and the computational efficiency of the proposed distributed algorithm in different types of networks, namely, small-world networks, random networks, and scale-free networks.

The results presented in this paper show that the distributed algorithm performs well in small-world, scale-free, and random networks, and for many different settings. Also other experiments were done (e.g. on grid networks) and these results held up over a wider range of scenarios. Furthermore, we showed that it scales well to large networks, both in terms of quality and of required computation time. The results also suggest that small-world networks are slightly better suited for local task allocation, because there are no nodes with very few neighbors.

There are many interesting extensions to our current work. In this paper, we focus on the *computational aspect* in the design of the distributed algorithm. In our future work, we would also like to address some of the related issues in game theory, such as strategic agents, and show desirable properties of a distributed protocol in such a context.

In the current algorithm we assume that agents can only contact their neighbors to request resources, which may explain why our algorithm performs not as good in the scale-free networks as in the small-world networks. Our future work may allow agents to reallocate (sub)tasks. We are interested in seeing how such interactions will affect the performance of task allocation in different social networks.

A third interesting topic for further work is the addition of reputation information among the agents. This may help to model changing business relations and incentivize agents to follow the protocol.

Finally, it would be interesting to study real-life instances of the social task allocation problem, and see how they relate to the randomly generated networks of different types studied in this paper.

Acknowledgments. This work is supported by the Technology Foundation STW, applied science division of NWO, and the Ministry of Economic Affairs.

8. REFERENCES

[1] S. Abdallah and V. Lesser. Modeling Task Allocation Using a Decision Theoretic Model. In *Proc. AAMAS*, pages 719–726. ACM, 2005.

[2] N. Alon, U. Feige, A. Wigderson, and D. Zuckerman. Derandomized Graph Products. *Computational Complexity*, 5(1):60–75, 1995.

[3] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[4] R. H. Coase. The Nature of the Firm. *Economica NS*, 4(16):386–405, 1937.

[5] R. H. Coase. My Evolution as an Economist. In W. Breit and R. W. Spencer, editors, *Lives of the Laureates*, pages 227–249. MIT Press, 1995.

[6] A. M. Easwaran and J. Pitt. Supply Chain Formation in Open, Market-Based Multi-Agent Systems. *International J. of Computational Intelligence and Applications*, 2(3):349–363, 2002.

[7] I. Foster, N. R. Jennings, and C. Kesselman. Brain Meets Brawn: Why Grid and Agents Need Each Other. In *Proc. AAMAS*, pages 8–15, Washington, DC, USA, 2004. IEEE Computer Society.

[8] M. E. Gaston and M. desJardins. Agent-organized networks for dynamic team formation. In *Proc. AAMAS*, pages 230–237, New York, NY, USA, 2005. ACM Press.

[9] A. Goldberg. An Efficient Implementation of a Scaling Minimum-Cost Flow Algorithm. *J. of Algorithms*, 22:1–29, 1997.

[10] R. Gulati. Does Familiarity Breed Trust? The Implications of Repeated Ties for Contractual Choice in Alliances. *Academy of Management Journal*, 38(1):85–112, 1995.

[11] T. Klos and B. Nooteboom. Agent-based Computational Transaction Cost Economics. *Economic Dynamics and Control*, 25(3–4):503–526, 01.

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

[13] K. Lerman and O. Shehory. Coalition formation for large-scale electronic markets. In *Proc. ICMAS*, pages 167–174. IEEE Computer Society, 2000.

[14] E. Manisterski, E. David, S. Kraus, and N. Jennings. Forming Efficient Agent Groups for Completing Complex Tasks. In *Proc. AAMAS*, pages 257–264. ACM, 2006.

[15] J. Patel et al. Agent-Based Virtual Organizations for the Grid. *Multi-Agent and Grid Systems*, 1(4):237–249, 2005.

[16] P. V. Sander, D. Peleshchuk, and B. J. Grosz. A scalable, distributed algorithm for efficient task allocation. In *Proc. AAMAS*, pages 1191–1198, New York, NY, USA, 2002. ACM Press.

[17] O. Shehory. A scalable agent location mechanism. In *Proc. ATAL*, volume 1757 of *LNCS*, pages 162–172. Springer, 2000.

[18] O. Shehory and S. Kraus. Methods for Task Allocation via Agent Coalition Formation. *Artificial Intelligence*, 101(1-2):165–200, 1998.

[19] R. M. Sreenath and M. P. Singh. Agent-based service selection. *Web Semantics*, 1(3):261–279, 2004.

[20] P. T. Tošić and G. A. Agha. Maximal Clique Based Distributed Coalition Formation for Task Allocation in Large-Scale Multi-Agent Systems. In *Proc. MMAS*, volume 3446 of *LNAI*, pages 104–120. Springer, 2005.

[21] W. E. Walsh and M. P. Wellman. Modeling Supply Chain Formation in Multiagent Systems. In *Proc. AMEC II*, volume 1788 of *LNAI*, pages 94–101. Springer, 2000.

[22] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small world' networks. *Nature*, 393:440–442, 1998.