Auton Agent Multi-Agent Syst
DOI 10.1007/s10458-011-9168-3

Multiagent task allocation in social networks

Mathijs M. de Weerdt · Yingqian Zhang · Tomas Klos

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-complete. Moreover, it is not approximable within some factor. In contrast to this, we develop an efficient greedy algorithm for this problem. Our algorithm is completely distributed, and it assumes that agents have only local knowledge about tasks and resources. We conduct a broad 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 also that it scales well to large-scale applications. In addition we consider the same problem in a setting where the agents holding the resources are self-interested. For this, we show how the optimal algorithm can be used to incentivize these agents to be truthful. However, the efficient greedy algorithm cannot be used in a truthful mechanism, therefore an alternative, cluster-based algorithm is proposed and evaluated.

Keywords Task allocation · Social networks · Resource allocation · Distributed algorithm · Mechanism design

M. M. de Weerdt (✉) · T. Klos
Delft University of Technology, Delft, The Netherlands
e-mail: M.M.deWeerdt@tudelft.nl

T. Klos
e-mail: T.B.Klos@tudelft.nl

Y. Zhang
Department of Econometrics, Erasmus School of Economics, Rotterdam, The Netherlands
e-mail: yqzhang@ese.eur.nl

Published online: 26 February 2011

1 Introduction

Recent years have seen a significant amount of work on task and resource allocation methods, which can potentially be applied to many real-world applications. However, 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 [20,48]. 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" [48, 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 these subtasks to different agents who may each be capable of performing different subsets of these 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 [17,60].

Traditionally, such allocation decisions have been analyzed using transaction cost economics (TCE) [12], 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 [13, 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 [32]. 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 [27]; these are the ones they consider when new tasks arrive and they have to form supply chains to allocate those tasks [56]. Other than modeling the interrelated 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.

Moreover, each agent in our model has a limited amount of resources of different types at its disposal. Agents may also have tasks to be completed. Each task, with a specified value on completion, requires some resources for execution. An agent with a task is called a *manager*, and only its 'neighboring' agents are allowed to provide their resources to this task. These agents are called *contractors*. The social task allocation problem (STAP) is, given the set of tasks and the available resources of the agents, to decide which tasks to execute and which resources of which contractors to supply their resources, such that the total value of the allocated tasks is maximized.

This simple framework is able to capture a variety of applications. For example, when each agent has some reputation in the eyes of other agents, agents may prefer to deal only with others whose reputation is 'good enough.' Only these are then considered as neighbors in the agent network. Alternatively, consider a disaster rescue scenario, such as in RoboCup Rescue [19,43]. Emergency events occur in different parts of a city, and different types of emergency services can cooperate to perform rescue tasks. Geographical proximity determines which other agents are available for cooperation, while different types of equipment carried by these services are modeled by the resources in our model.

The results presented in this paper improve and extend upon earlier work by the same authors [62]. This paper first studies the social task allocation problem in a *cooperative* setting, where the agents reveal their information truthfully to their neighboring partners. The main research question in this cooperative setting is the development of a computational model and efficient algorithm, and the effect of the structure of a social network on its performance. In the next section, we give a formal description of this cooperative social task allocation problem. Section 3 shows that the complexity of this problem is NP-hard. An exact method is put forward in Sect. 4.1. Since any exact algorithm is too computationally expensive in practice, Sect. 4.2 proposes a greedy polynomial-time algorithm, which is extended to a distributed algorithm in Sect. 6. We perform a series of experiments with these algorithms in different network types in Sect. 5 and 7, respectively.

The experimental results demonstrate that the distributed algorithm works well in this setting where the information of the available resources among agents is correctly known. In some situations, especially where multiple organizations or companies are involved, however, agents are often self-interested. They may not act according to the designed algorithm or protocol. Such deviations could lead to a very bad performance of the proposed task allocation algorithm. Thus, in this paper, we also study the social task allocation problem among selfish agents. The main research problem of this *selfish* STAP is, how to incentivize self-interested agents to report their private information correctly, in order to sustain the performance of the proposed algorithms.

The private information in STAP is the set of available resources of each contractor. We assume in this paper that only the contractor agents strategize about their private information. To avoid manipulation by the contractors, we aim to design a truthful mechanism by adding a payment function to reward agents for the use of their resources. Nisan and Ronen [47] show that the truthfulness of agents can be guaranteed by so-called VCG mechanisms (see Definition 8) under the condition that the mechanism is able to compute the optimal solution. Since many interesting optimization problems are intractable, they showed an alternative way to achieve a truthful *VCG-based mechanism* by replacing the exact algorithm with an approximation [46]. However, they showed that for a certain class of minimization problems (*cost minimization allocation problem*), any truthful VCG-based mechanism is either optimal or can lead to degenerate results, i.e., for any approximation there are instances in which the result can be arbitrarily far from the optimal solution. Their result suggests that for many NP-hard problems, developing good polynomial-time VCG-based truthful mechanisms is not a trivial problem.

We model the STAP as a mechanism design problem in Sect. 8. Section 8.1 proposes an exact, truthful mechanism with the optimal allocation algorithms introduced in Sect. 4.1. We then show in Sect. 8.2 that the proposed greedy algorithm is not truthful, and then introduce a polynomial-time truthful mechanism, which relies on the idea of constructing clusters in the social network that are considered independently. The quality of this approximation algorithm is tested experimentally in Sect. 8.4.

Finally, we discuss our approaches by comparing with other existing work in Sect. 9, followed by our conclusions in Sect. 10.

2 Social task allocation problem

In our framework, we consider agents that need to complete tasks. The completion of a task yields a certain value, and it requires varying numbers of resources of different types. A task can be completed only if all required resources of all required types are allocated to that task. Agents are endowed with these resources. In addition, each agent is connected to a limited number of other agents, yielding a social network. For the completion of her task, an agent may enlist the resources of other agents, but only those she's connected to in the network. The existence of the network constrains the allocation of resources to tasks. The problem we consider is to determine which resources to assign to which tasks, in order to maximize the value of the allocated tasks.

More formally, let $\mathscr{A}$ denote a set of m agents that need resources to complete tasks. Let $R = \{r_1, \ldots, r_l\}$ denote the collection of the resource types available to the agents in $\mathscr{A}$. Each agent $i \in \mathscr{A}$ initially holds a fixed amount of resources for each resource type in R, which is defined by a resource function: $s_i : R \to \mathbb{N}$. Finally, we assume agents are connected by a *social network*.

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

Suppose a set of tasks $\mathscr{T} = \{t_1, t_2, \ldots, t_n\}$ arrives at such an agent social network. Each task $t \in \mathscr{T}$ is then defined by a tuple $\langle U(t), \text{req}(t), \text{loc}(t)\rangle$, where $U(t)$ is the value gained if task t is accomplished, and the function $\text{req}(t) : R \to \mathbb{N}$ specifies the amount of resources required for the accomplishment of task t, while other combinations of resources do not accomplish this task. Furthermore, a location function $\text{loc} : \mathscr{T} \to \mathscr{A}$ defines the locations (i.e., agents) of the tasks in the social network. An agent i that is the location of a task t, i.e., $\text{loc}(t) = i$, is called the *manager* of task t. The exact assignment of resources to tasks is defined by a *task allocation*.

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

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

The set of all valid task allocations is denoted by $\mathscr{O}$.

We write T_o to represent the tasks that are complete in o, that is, the tasks to which o assigns sufficient resources of each required type. The value of o is then the sum of the values of

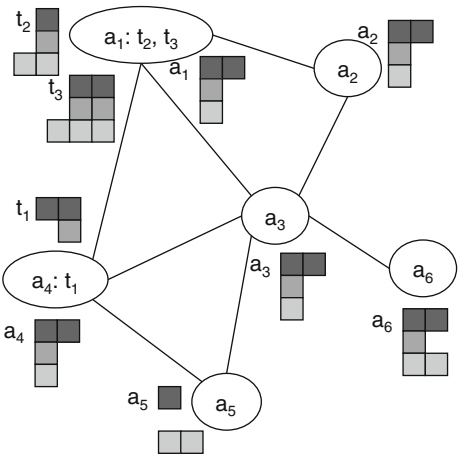

Fig. 1 An example instance of the Social Task Allocation Problem

each task in T_o, i.e., $U_o = \sum_{t \in T_o} U(t)$. Note that we do not include costs for resources, since we assume that the resources have already been paid for. Our only goal is to allocate these resources as efficiently as possible. We thus define the *efficient task allocation* as follows.

Definition 3 (Efficient task allocation) We say a task allocation o^* is *efficient* if it is valid and U_{o^*} is maximized, i.e., $o^* = \arg\max_{o \in \mathcal{O}} U_o$.

The social task allocation problem is then defined as follows.

Definition 4 (Social task allocation problem) Given a set of agents $\mathcal{A}$ that are connected by a social network $SN = (\mathcal{A}, AE)$, a finite set of tasks $\mathcal{T}$ need to be allocated to the agents. The goal of this *social task allocation problem* (STAP) is to find the efficient task allocation o^*.

Furthermore, in this paper we focus on a situation where all knowledge about the problem is local. In particular, initially a task and its value are only known to the task's manager, and each agent only knows the resources it has available itself.

Figure 1 shows an example instance of the STAP. In this figure, the oval nodes represent agents, while edges represent relations between pairs of agents. There are six agents, with a total of three tasks: task t_1 at agent a_4, and tasks t_2 and t_3 at agent a_1. This makes agents a_1 and a_4 *managers*, while all agents can act as *contractor*. The squares in the figure represent resources, with different shades of gray for different types. Resources required by tasks are stacked from right to left, while the resources agents are endowed with are stacked from left to right. In this instance, agent a_1 does not own all the resources of all types required for executing the two tasks assigned to it, so it will have to procure these from other agents. Candidate contractors for agent a_1 are all agents except a_5 and a_6, because agent a_1 is not connected to those agents. Agent a_1 might enlist the help of agents a_3 and a_4 to supply the two required resources of the medium-gray type. However, agent a_4 would then not anymore be able to obtain the resource of that type that it requires for executing its task t_1. In general situations, the number of agents, tasks, and resource types becomes larger, but the combinatorial and difficult nature of the problem is already intuitively recognizable from this example.

3 Complexity results

In a general task allocation problem (TAP), there are a center with multiple tasks and agents with resources. The center can allocate its tasks to every agent in the system as long as the agent has the required resources. Thus, we can view the TAP as a social task allocation problem with a fully connected network. Shehory and Kraus [52] argued (informally) that the TAP is NP-complete. The complexity comes from the fact that we need to evaluate an exponential number of subsets of the task set. When the center has only one task to allocate, the TAP becomes easy to solve [52]. We may consider the TAP as a special case of the STAP by assuming agents are fully connected, and then conclude that the STAP is also NP-complete. However, this does not give sufficient insights into the hardness of our problem, since in the STAP (i) typically several agents hold tasks (instead of only one in the TAP [52]); and (ii) agents are connected by some social network, which is never fully connected. In fact, because of the latter, the TAP is strictly speaking not a special case of the STAP.

Therefore, in the following, we show that in the presence of a social network with several agents holding tasks, the social task allocation problem STAP is NP-complete, even when each agent has no more than 1 task, the utility of each task is 1, and the quantity of all required and available resources is 1. We then show that the STAP remains hard for very restricted classes of networks, i.e., trees.

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 o with utility more than k exists is* NP*-complete even when each agent has no more than 1 task, the utility of each task is 1, and the quantity of all required and available resources is 1.*

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 o is a valid allocation and whether the utility of o is greater than k.

We now prove that the STAP is NP-hard by showing that MAXIMUM INDEPENDENT SET (MIS) $\leq_P$ 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 of size k.

An instance of the following construction is shown in Fig. 2. For each node $v_i \in V$ and each edge $e_j \in E$ in the graph G, we create a vertex agent a_i and an edge agent a'_j in G'. Both vertex agents and edge agents are nodes in G'. When v_i was incident to e_j in G we correspondingly add an edge e' in G' between a_i and a'_j. We assign each agent in G' one resource. This resource is related to the node or the edge in the graph G, i.e., for each $v_i \in V, s_{a_i}(v_i) = 1$, and for each $e_j \in E, s_{a'_j}(e_j) = 1$. Each vertex agent a_i in G' has a task t_i that requires a set of neighboring resources, i.e., $req(t_i)(v_i) = 1$ and for each e in $\{e \in E \mid \exists_{u \in G}(u, v_i) \in E\}, req(t_i)(e) = 1$. There is no task on the edge agents in G'. We define utility 1 for each task.

Taken an instance of MIS, 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 set of tasks that can be allocated. The utility of this allocation is k.

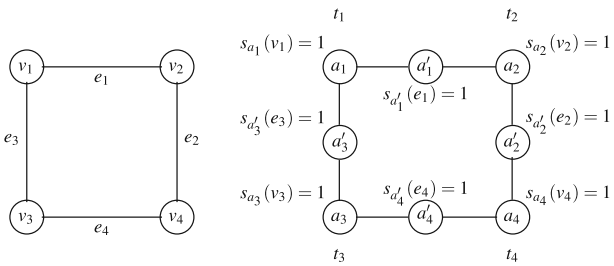

Fig. 2 MIS can be reduced to STAP. The left figure is an undirected graph G, which has the optimal solutions $\{v_1, v_4\}$ or $\{v_2, v_3\}$. The figure on the righthand side represents the constructed instance of the STAP, where the tasks t_i are managed by the agents a_i and require the following resources. Task t_1 requires v_1, e_1, and e_3, task t_2 requires v_2, e_1 and e_2, task t_3 requires v_3, e_3, and e_4, and task t_4 requires v_4, e_2, and e_4. The optimal allocation here is either $\{t_1, t_4\}$ or $\{t_2, t_3\}$

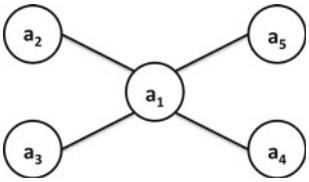

Fig. 3 The general task allocation problem with a fully connected network can be reduced to the social task allocation problem on a tree

If there is a solution for the STAP with the utility value k, and the allocated task set is N, then for MIS, there exists a maximum independent set N of size k in G. An example can be found in Fig. 2. □

The STAP is NP-complete for arbitrary graphs. In our proof, the complexity comes from the existence 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 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 the STAP is as hard as MIS (Theorem 1), it is not possible to give a worse case bound better than Δ^{ε} for any polynomial time algorithm, unless P = NP. □

One may wonder whether the above complexity result holds for some special graph structure, for instance, a tree. It is straightforward to see that any general task allocation problem can be reduced to the STAP on a tree. Given a task allocation problem, we construct a STAP instance with one agent who becomes the manager of all tasks but has no resources (agents a_1 in Fig. 3), and a number of neighboring agents (e.g., agents a_2, a_3, a_4, a_5) that own the resources but no tasks. In the example in Fig. 3 agent a_1 is the root of a tree. An optimal solution to the general task allocation problem is the solution to the constructed STAP, and vice versa. Since the general task allocation problem is NP-complete [52], we conclude that the social task allocation on a tree also remains NP-complete.

4 Algorithms for social task allocation

In this section, we first show how to solve STAP optimally. Due to the complexity of the problem, an optimal, exact algorithm unavoidably runs in exponential time. Therefore, exact algorithms may not always be suitable for large-scale problems. We thus also propose a heuristic that can be used to tackle such large real-world problems.

4.1 Optimal solution

An optimal task allocation should incorporate the restrictions posed by the social network. For this NP-complete problem we use a straightforward translation to an integer linear programming (ILP) problem to solve this problem. For the ILP formulation we introduce two types of variables: the binary variables $y_j \in \{0, 1\}$ for $1 \leq j \leq n$ describe whether or not task j is allocated, and the integer variables x_{ijk} ($\forall i, j, k$, where $1 \leq i \leq m, 1 \leq j \leq n, 1 \leq k \leq l$) denote the amount of resources of type k agent i supplies to task j. The ILP formulation then looks as follows.

$$\texttt{Maximize} \quad \sum_{j=1}^{n} y_j \cdot U(t_j)$$

subject to having sufficient resources of each type for each chosen task from the neighboring agents, i.e.,

$$\forall j, k (1 \leq j \leq n, 1 \leq k \leq l) \quad \sum_{\{i \in [1,m] | (i, loc(t_j)) \in AE\}} x_{ijk} \geq y_j \cdot req(t_j)(r_k),$$

and not using more resources than there are available, i.e.,

$$\forall i, k (1 \leq i \leq m, 1 \leq k \leq l) \quad \sum_{j=1}^{n} x_{ijk} \leq rsc(i)(r_k).$$

Solving this ILP results in an optimal solution, but we can only give a worst case upper bound on the run time of solving this ILP that is exponential in the number of variables, i.e., the number of tasks, agents, and the resource types.

Clearly, we cannot expect this ILP to always be able to find solutions for larger problem sizes. Hence, we now present a polynomial-time algorithm for the STAP.

4.2 Centralized greedy algorithm

A greedy algorithm ranks tasks with respect to some measure expressing how promising a task is, and then tries to allocate tasks in the order of their ranking. A decision to allocate a task is not reconsidered, which makes such an algorithm quite fast. If allocating the task is feasible, it is inserted; if not, it is removed from the current selection of tasks. Feasibility of a selection of tasks is checked by translating the problem to a (polynomially solvable) network flow instance (see Algorithm 1).

We consider the following rankings based on the social structure as well as on the value and the number of resources of a task.

Efficiency The first heuristic is based on the idea of a greedy approximation for 0–1 knapsack [14]. In such an approximation for knapsack all items are ranked on their relative value. This approximation even has a theoretical bound: the resulting allocation is never more than

Algorithm 1 Centralized greedy task allocation algorithm (GTA).

1. Sort all tasks from all managers according to some ranking heuristic. Denote the sorted tasks by $t'_1, t'_2, \ldots, t'_n$, and the current selection of tasks by $T' = \emptyset$.
2. For $i = 1, \ldots, n$ do:
   (a) $T' \leftarrow T' \cup \{t'_i\}$.
   (b) Test if T' is feasible as follows. Create a network flow problem:
      i. Create a source s and a sink s'.
      ii. For each agent $j \in \mathscr{A}$ and each resource type $r \in R$, if $s_j(r) > 0$, create an agent-resource node a_j^r, and an edge from s to this node with capacity $s_j(r)$.
      iii. For each task $t \in T'$ and each resource type $r \in R$, if $req(t)(r) > 0$, create a task-resource node t^r, and an edge from this node to s' with capacity $req(t)(r)$.
      iv. For each agent $j \in \mathscr{A}$ and each resource type r, connect the agent-resource nodes a_j^r to the task-resource nodes t^r if tasks t are direct neighbors of j, i.e., $\{t \in T' \mid (j, loc(t)) \in AE\}$. Give this connection unlimited capacity.
   (c) Solve the maximum flow problem for the created flow network. If the maximum flow is equal to $\sum_{t \in T'} \sum_{r \in R} req(t)(r)$, the current combination of tasks is feasible. Otherwise remove task t'_i from T'.
3. Output the task set T' and the current allocation.

twice as bad as the optimal allocation. For the STAP the relative value of a task is the ratio of its utility and its resources. We call this the efficiency of a task.

Definition 5 The *efficiency* e of a task $t \in \mathscr{T}$ is defined by the utility of this task divided by the sum of all required resources: $e(t) = \frac{U(t)}{\sum_{r \in R} req(t)(r)}$.

The heuristic then ranks the tasks in order of descending efficiency. With such a ranking on efficiency we can leverage the idea of the knapsack approximation also to give a guarantee on the quality of the greedy centralized allocation algorithm.

Proposition 1 *The greedy allocation algorithm (GTA) with the efficiency heuristic is a* $l \cdot K$*-approximation algorithm for STAP, where* K *is the maximum number of resources of one type a task can require. The run time of* GTA *is* $O(l^2 n^2 m(m+n))$.

Proof In the worst case, the greedy algorithm selects only the most efficient feasible task $t^* \in \mathscr{T}$ with value $U_{\mathsf{GTA}} = U(t^*)$, and the use of resources of this task t^* blocks all other feasible tasks being allocated. Assume the optimal solution in this case is to select a subset of all other feasible tasks. So, the value of the optimal solution U_{OPT} is always less or equal than the sum of the values of all other (feasible) tasks: $U_{\mathsf{OPT}} \leq \sum_{t \in \mathscr{T} \setminus \{t^*\}} U(t)$. Task t^* takes at least one resource from each task $t \in \mathscr{T} \setminus \{t^*\}$ in order to block t being selected, and so in the worst case its required resources $\sum_{t \in R} req(t^*, r)$ is at least $|\mathscr{T} - 1| = n - 1$. Thus its efficiency is at most $\frac{U(t^*)}{n-1}$. The resource requirement of other tasks $t \neq t^*$ is at most $K|R| = Kl$, and therefore their efficiency is at least $\frac{U(t)}{Kl}$. These tasks should have a lower efficiency than t^*, so $\frac{U(t)}{Kl} \leq \frac{U(t^*)}{n-1}$, and thus $U(t) \leq \frac{U(t^*) \cdot Kl}{n-1}$. Consequently the approximation factor is:

$$\frac{U_{\mathsf{OPT}}}{U_{\mathsf{GTA}}} \leq \frac{\sum_{t \in \mathscr{T} \setminus \{t^*\}} \frac{U(t^*) \cdot lK}{n-1}}{U(t^*)} = lK.$$

Concerning the computation time of GTA, sorting the tasks takes $O(n \log(n))$. To check each set of tasks, it takes $O((ml + nl)mnl)$ to create and solve the maximum flow problem. There are up to n such task sets to check. Hence, GTA takes $O(l^2 n^2 m(m+n))$. □

Betweenness Besides the value and the number of resources of a task, it is also important whether its neighbors can be expected to be able to provide the required resources. This second heuristic is taken from work on social networks and uses the number of shortest paths (between any two agents) through an edge or vertex, called the *betweenness* [24]. The intuition here is that tasks with a high betweenness must lie somewhere in the center of the graph and are therefore a good place to start with allocating. To validate this intuition, we also consider a ranking starting with the task with the lowest betweenness, which we denote by InvBetweenness.

Clustering Coefficient Another such measure on the neighborhood of a task is the clustering coefficient of an agent [61]. This is the fraction of the agent's neighbors that are also neighbors of each other. This fraction is computed by considering the subgraph of the agent's k neighbors, counting the number of connections, and dividing this by the maximal number of possible connections, i.e., $\frac{1}{2}k(k-1)$. Also based on this measure we consider two ranking heuristics: Cluster considers tasks in decreasing order of the clustering coefficient of their manager agent, and InvCluster considers the tasks in increasing order of the clustering coefficient.

5 Experiments with centralized heuristics

We implemented the centralized greedy heuristics in Java, and compared the results to those obtained solving the ILP formulation optimally using the GNU Linear Programming Kit [38]. For all experiments we used a PC with a 2.4 GHz AMD Opteron and we allowed the Java code to use at most 1 GB of memory. The purpose of these experiments is to study the performance of the different heuristics in different social networks. In the experiments, three different types of 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. [61]. According to their model, networks are generated starting from a regular ring lattice with m nodes and l edges per node. Each edge is then randomly rewired with probability p. The probability of rewiring p allows us to determine the amount of "chaos" and "regularity" in the network. In the experiments, we use a fixed rewiring probability $p = 0.05$.

Scale-free networks have the property that a few nodes have many connections, and many nodes have only a small number of connections. In other words, the degree distribution function $P(m)$ follows a power-law: $P(m) \sim m^{-\gamma}$, where γ is the scaling parameter. To generate these networks we use the implementation in the JUNG library of the generator proposed by Barabási and Albert [6]. The generation mechanism starts with a small number of nodes and a new node is added to the network at each time step. The new node connects preferentially to existing nodes with high degree. The preferential attachment is the probability Π that new nodes will be connected to the exiting node i : $\Pi(m_i) = \frac{m_i}{\sum_j m_j}$.

We also generate *random networks* as follows. First we connect each agent to a uniform randomly selected earlier connected agent such that all agents are connected. Next, we randomly add connections (drawing twice from a uniform distribution over agents) until the desired average degree has been reached.

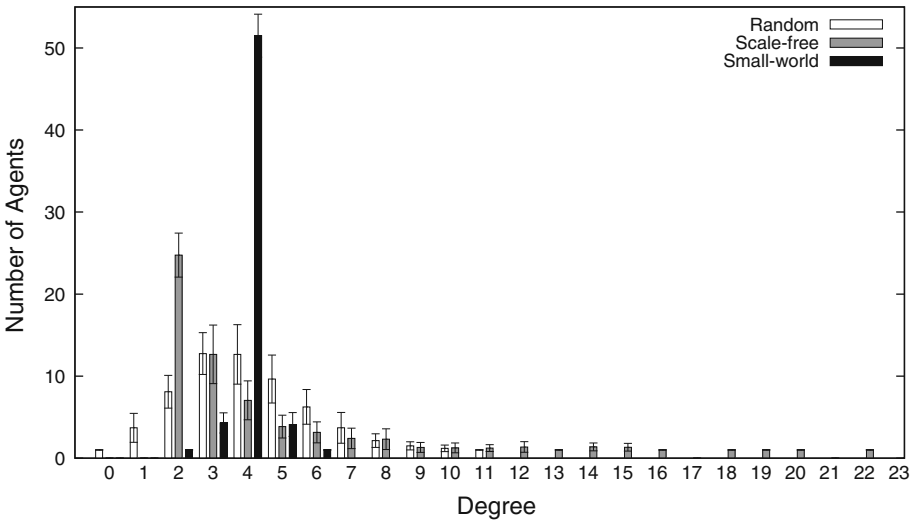

Fig. 4 The histogram of the degrees of agents in networks with 60 agents and an average degree of 4

These three algorithms result in networks with a different distribution of the connections over the agents. For example, these distributions for several randomly generated networks with 60 agents and an average degree of 4 are given in Fig. 4. As in this figure, we will often slightly horizontally offset different plots with respect to the x-axis to prevent overlap of different graphs' datapoints and error bars, which makes the plots clearer.

We limit these experiments to 5 resource types, 20 resources per task on average and $n = 80$ tasks to make it feasible to compute the optimal solution for every setting we consider within at most 15 min. Each of the required resources (i.e., in total always 20 times the number of tasks) is given one out of the five types at random, and is also randomly assigned to one of the tasks. In all experiments the value of a task is drawn uniformly from the interval between 0 and the number of resources assigned to that task, so as to create a correlation between the size of a task and its value, with some random noise. The number of resources available in the network is set by the resource ratio parameter. This parameter determines the ratio between the number of available resources and the number of required resources (for each type separately) and defaults to 0.5, allowing at most half of the tasks to be allocated in a fully connected network.

In the experiments with the centralized greedy algorithm, we compare the five heuristics (Efficiency, Betweenness, Cluster, InvBetweenness, and InvCluster) to each other and to the optimal solution in the three networks given above, in networks with sizes varying from 10 to 120 agents (with steps of 10). For each setting, we generate 20 problem instances and compute the average and the standard deviation over these 20 instances. Where we put 'Quality' on the y-axis, this is the value of the allocation given by a heuristic divided by the value of the optimal allocation. Sometimes we use the word 'Value,' which is the absolute value of an allocation. The results for small-world and scale-free networks can be found in Figs. 5 and 6. Here the number of agents is shown on the x-axis, and the y-axis shows the average value divided by the average optimal value, which we call the (relative) quality of the heuristics.

These figures show that the results on the different networks are quite similar. In particular the results on random networks are almost equivalent to those on the scale-free networks, and therefore not included here. An important observation is that most heuristics work

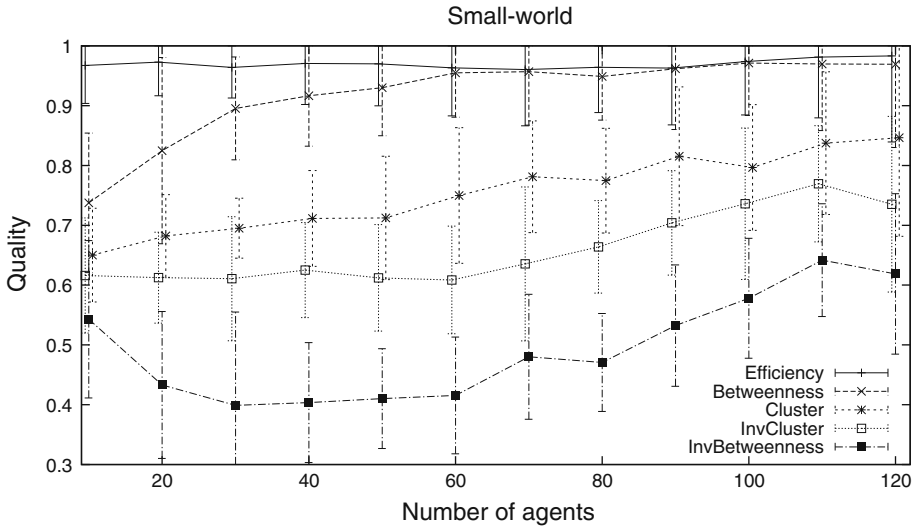

Fig. 5 The quality of the centralized greedy algorithm on small-world networks for the five heuristics

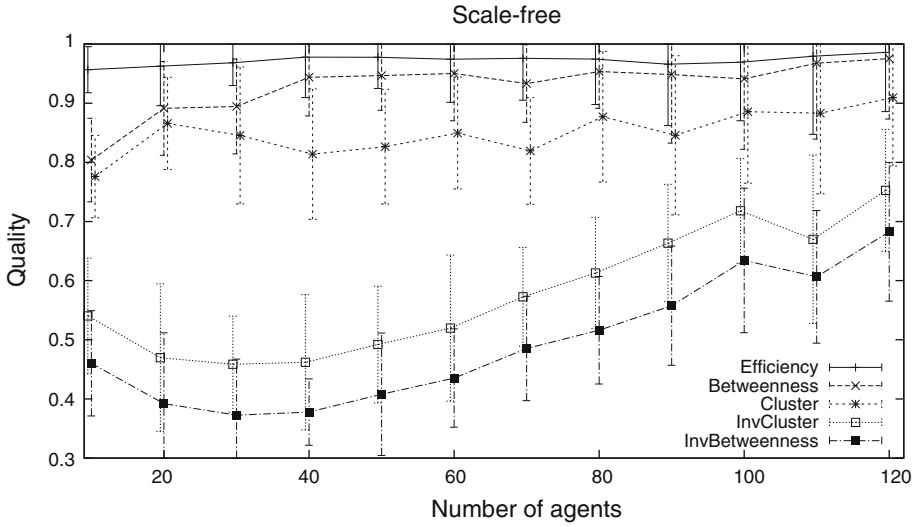

Fig. 6 The quality of the centralized greedy algorithm on scale-free networks for the five heuristics

surprisingly well with an average quality of at least 0.5. This is especially surprising, considering the theoretical hardness (and in particular the approximation-hardness) of the problem. We believe this can be attributed to the fact that all heuristics exploit the locality that is inherent in this problem. In most problem instances the consequences of a sub-optimal allocation of a single task does not influence decisions in another part of the network.

Regarding the four heuristics that use the structure of the network, we can see that this indeed has some effect. For example, starting with tasks that are central in the network (with a high betweennness or a high clustering coefficient) results in significantly higher quality solutions over all network types than when using the inverse ranking. However, for larger

Algorithm 2 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 (ties are broken on the task id). Other managers are informed that no bid will be made this round.
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. If all resources are used, the agent informs its neighbors and stops.

networks (with the same number of tasks), the difference is reduced, because the relative quality of the inverse heuristics is higher there. This effect is caused by the fact that fewer tasks can be allocated even in the optimal solution, simply because in the larger networks resources and tasks are spread out more. Because of that, the problem becomes easier, and the order in which the tasks are considered does not matter so much anymore. It can also be observed that the clustering coefficient works better on scale-free networks and the betweenness measure is almost as good as the efficiency measure on small-world networks. Overall, the efficiency heuristic significantly outperforms all other heuristics in all network types and reaches a very high average quality of above 0.95.

Besides the heuristics described and evaluated above, we have also experimented with heuristics such as the number of neighbors of an agent, and the average distance to all other agents (the bary center), but they did not do any better. Also, a different heuristic could be used altogether, for example where contractors bid on tasks maximizing their local efficiency, or a combinatorial variant where contractors can bid on multiple tasks simultaneously. We leave an investigation of such alternative settings (including their termination criteria) to future work and continue with the best heuristic found so far.

6 A distributed protocol

We now show how to transform the centralized greedy algorithm into a distributed protocol. Since the efficiency heuristic works very well, we concentrate on this heuristic. In the distributed protocol, information on a task is only given to the manager of that task, and resources are only known to the contractor who controls them. The idea of the protocol is as follows. All manager agents $a \in \mathscr{A}$ try to find neighboring contractors to help them with their task(s) $T_a = \{t_i \in \mathscr{T} \mid loc(t_i) = a\}$. They start with offering the task that is most efficient in terms of the ratio between value 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 choice for the set of contractors whose offers are accepted is made randomly.

A more detailed description of this protocol can be found in Algorithm 2. 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. In every iteration, the most efficient task among all managers' task lists will receive offers

from all its neighbors, and so this task will be either allocated or removed from the task list of the related manager, dependent on whether the received resources are sufficient or not. Therefore 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. This is ensured by having contractors choose the task with the lowest task-id in cases where tasks have the same efficiency. The computation and communication complexity of the algorithm then follow from this observation.

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

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 (i.e., to all m agents). 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 number of $O(nm)$ operations which can be done in $O(n+m)$ parallel steps, and $O(nm)$ messages for each iteration. This results in a total of $O(n^2m)$ operations which can be done in $O(n^2+nm)$ parallel steps. The total number of messages sent is $O(n(nm)) = O(n^2m)$. □

We establish the quality of the proposed centralized and distributed algorithms experimentally in next section.

7 Experiments with the distributed protocol

Theoretically it is impossible to establish a fixed worst-case approximation ratio for any polynomial algorithm for STAP, because of the inherent hardness of the problem. We are therefore interested in showing experimentally in which settings GDAP performs worst, compared to the optimal solution. The purpose of these experiments is to study the performance of the distributed algorithm in different problem settings using different social networks. In particular, we are interested in the influence of the size and degree of the network, the number of tasks, and the number of resources available on the quality of the solution and the run-time. Regarding these experiments, we have the following expectations regarding the quality of the results of GDAP.

1. When the network is better connected (fewer agents or a higher degree), the performance is better.
2. When there are more resources available, the performance is better.

The reason for these expectations is that when there are many resources, and many combinations of resources possible, the greedy method works fine, because an initial wrong choice cannot prevent many other tasks to be allocated.

We first ran several exploratory experiments (some of which are reported in [62]), varying the resource ratio, the degree, and the number of agents for each of the three network types. From this initial investigation it turned out that GDAP performs worst when only about half of the required resources is available, when the average degree is 4, and the 80 tasks and 1,600 resources are distributed over about $m = 60$ agents. We use this setting as the basis of all of the following experiments.

1. We vary the number of resources that are available in the network between 10 and 140% of the number of required resources in steps of 10% (i.e., the resource ratio from 0.1 to 1.4).
2. We vary the average degree of the agents in the generated network between 2 and 30 in steps of 2.
3. We vary the number of agents from 10 to 120 in steps of 5.
4. We vary the number of tasks from 10 to 120 in steps of 5. Since the number of resources is always the resource ratio (0.5) multiplied by 20 times the number of tasks, the number of available resources in this experiment vary from 100 to 1200.

For each of these experiments we measure the value of the computed allocation (the sum of the values of the completed tasks), as well as the total computation time. Again, 'Quality' on the y-axis means normalized value, whereas 'Value' means absolute value. To be able to compare the computation time to that of the optimal algorithm, we measure the total time required for running the distributed algorithm sequentially.

7.1 Resource ratio

In Fig. 7 the resource ratio is given on the x-axis, and the y-axis is used to express the relative quality of the GDAP algorithm. Again this quality is determined by dividing the reward obtained by GDAP by the value of the optimal solution. Again, we slightly offset the plots for the three networks with respect to the x-axis to prevent overlap and thus make the figure easier to read. From a ratio of about 0.4 and higher, the quality of GDAP increases for all three network types almost linearly up to a quality above 0.9. This can be explained by the fact that when there are more than sufficient resources available, all tasks can be allocated without any conflicts. For very low resource ratios the results are much less predictable. One thing we can say is that if resources are really scarce, only a few tasks can be successfully allocated even by the optimal algorithm and GDAP finds these as well. For example, of the 8 (out of 20) small-world instances with resource ratio of 0.1 where the optimal algorithm could allocate at least one task, GDAP finds this optimum in 7 cases. As for the different network topologies, a t-test for the difference between the means (significance level of 0.05) shows that the small-world instances yield significantly higher relative performance than both other network types up to a ratio of 0.6, mainly caused by the higher values of the optimal solutions in the other networks, as can be seen from Table 1.

Both situations with very few resources as well as with abundant resources are relatively easy to solve for both GDAP as well as the optimal algorithm, in a way reminiscent of the phase transition in many satisfiability problems like random k-SAT [40] or Distributed CSP [29], where instances are easy when they are either over- or underconstrained. This is illustrated by the run time of the optimal algorithm in Fig. 8. Note that in all graphs on run time we use a logarithmic scale on the y-axis, to be able to include the run time of the optimal algorithm. (This also explains why the errorbars for the standard deviations are asymmetrical.) In this setting with a varying resource ratio, we can see that for a resource ratio between 0.2 and 0.8, the ILP requires much more time than in the other settings. This can be explained by the fact that the ILP needs to use branch and bound and rounding techniques [57] to find the optimal solutions. Especially the small-world instances seem to pose a formidable challenge for the optimal algorithm, whereas for the GDAP algorithm the small-world instances are easiest: a t-test shows they require significantly less time across almost the entire range of values of the resource ratio (except for ratio 1.4 for random and 1.2 and 1.4 for scale-free). This can be explained by the fact that in the small-world networks we generated for our experiments

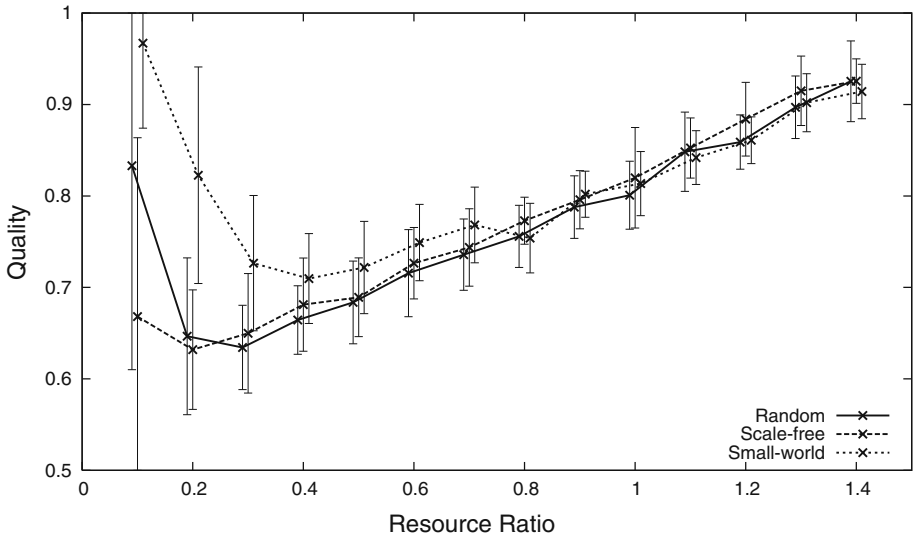

Fig. 7 The relative solution quality of GDAP is influenced by the resource ratio

Table 1 The average values of the computed task allocations for resource ratios 0.1–1.4 for GDAP (the first line) and Optimal (the second line), rounded to three digits

0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2	1.3	1.4
Random													
17.5	91.5	172	265	341	413	484	527	591	615	685	703	704	755
22.1	142	271	399	498	577	659	697	751	768	808	819	785	816
Scale-free													
29.4	111	190	270	340	426	493	553	606	617	670	710	729	718
45.0	176	294	396	495	587	663	716	761	752	787	803	798	776
Small-world													
13.2	65.2	169	259	333	413	475	505	583	634	639	685	713	751
13.6	79.8	233	365	462	551	618	671	728	779	760	796	790	821

(with a rewiring probability of 0.05), most agents have about four neighbors (see Fig. 4). Because of this, many managers can have (locally) the most efficient task according to all their neighbors and many of them may thus simultaneously (in the same iteration) conclude that it is impossible to allocate the task for which they have requested. In the other networks, even if there is only one agent with a high number of neighbors, it may take that many iterations for all its neighboring managers to give up.

7.2 Degree

For the next experiment we fix the resource ratio to 0.5 and study the quality of GDAP related to the degree of the social network. The result can be found in Fig. 9. In this figure we can see that a high average degree leads to better results for GDAP. Obviously, when managers have many connections, it becomes easier to allocate tasks. An exception is, similar to what

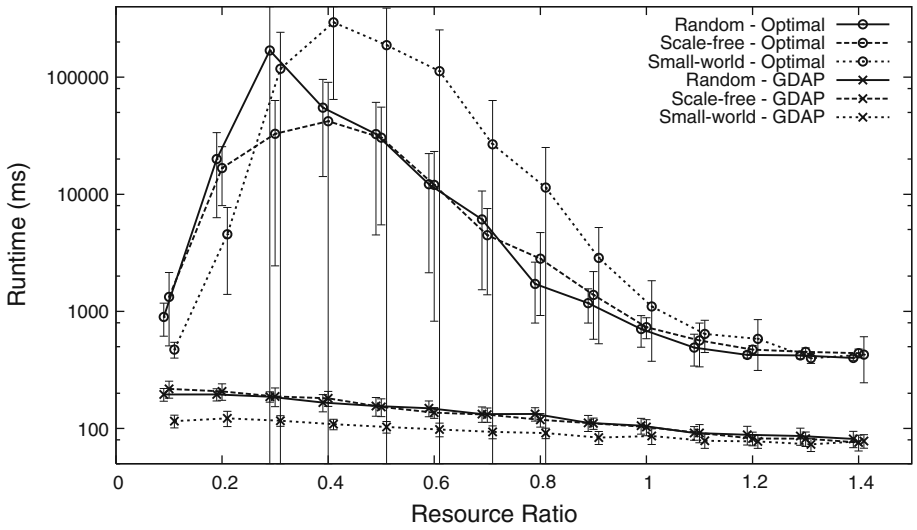

Fig. 8 The run time of the optimal algorithm is strongly affected by the resource ratio. A high resource ratio has relatively short run times for both GDAP and the optimal algorithm. A ratio around 0.5 takes the most time for the optimal algorithm

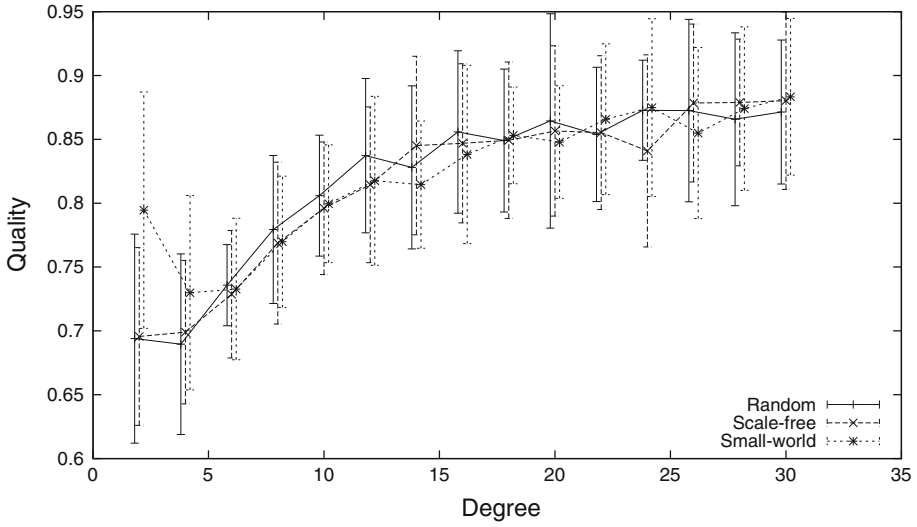

Fig. 9 The quality of GDAP varies with the network degree

we have seen in Fig. 7, that the solution of the GDAP is also relatively good in small-world instances where the connections are extremely limited (significantly better than both random and scale-free in instances with degrees 2 and 4). Again this is caused by the low values for the optimal solution under these conditions.

The results on the run time are not very interesting. GDAP takes about 200 ms on average and the optimal algorithm about 10,000 ms. The degree does not seriously influence either of these results.

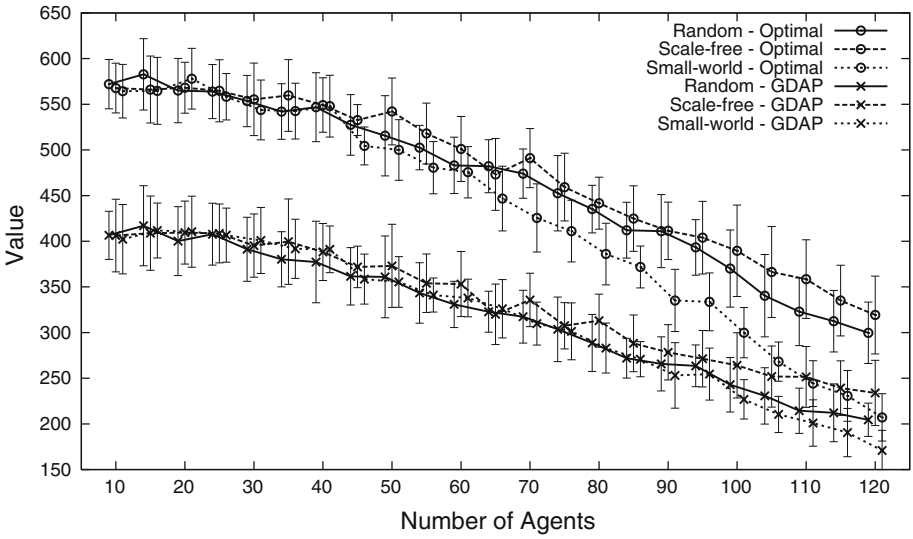

Fig. 10 The value of both the optimal allocation and the one produced by GDAP gradually decrease when the number of agents is increased

7.3 Agents

We are interested in the performance of GDAP when the number of agents m is increased. For this experiment we kept the average degree at 4. The run time of GDAP is not significantly influenced by the number of agents. There is, however, a noticeable effect on the quality of the resulting allocations.

Because all parameters except the number of agents are kept constant, the effect of increasing the number of agents is that the value of the optimal solution decreases (see Fig. 10). This can be explained by the fact that it becomes harder to assign resources to neighboring tasks when the agents are more dispersed. To be more precise, with an average degree of 4, the expected number of agents that can provide resources stays 5 for any number of agents, while the average number of available resources per agent decreases when we increase just the number of agents.

Regarding the quality of the allocations produced by GDAP, we see in Fig. 11 that this is consistently in the range from 0.6 to 0.8 when we vary the number of agents. The most remarkable observation here is that for larger numbers of agents, the differences between the network types become more pronounced. In particular, when the networks become large, GDAP performs significantly better in small-world networks: at 65 agents and above, small world networks enable significantly higher normalized performance than both random and (except at 80 agents) scale-free networks. From 110 agents onwards, scale-free networks also surpass random networks.

Actually, as can be seen in Fig. 10, what happens is that, when the number of agents is increased, the value of the optimal solution decreases more sharply in small-world networks. At 45 agents and higher, the value of the optimal solution in the small-world networks is significantly lower than in both other types of networks (except at 50 and 60 agents for random networks). This can be understood by comparing the distribution of the agents' degrees (Fig. 12) and the distribution of successfully allocated tasks by the optimal algorithm over

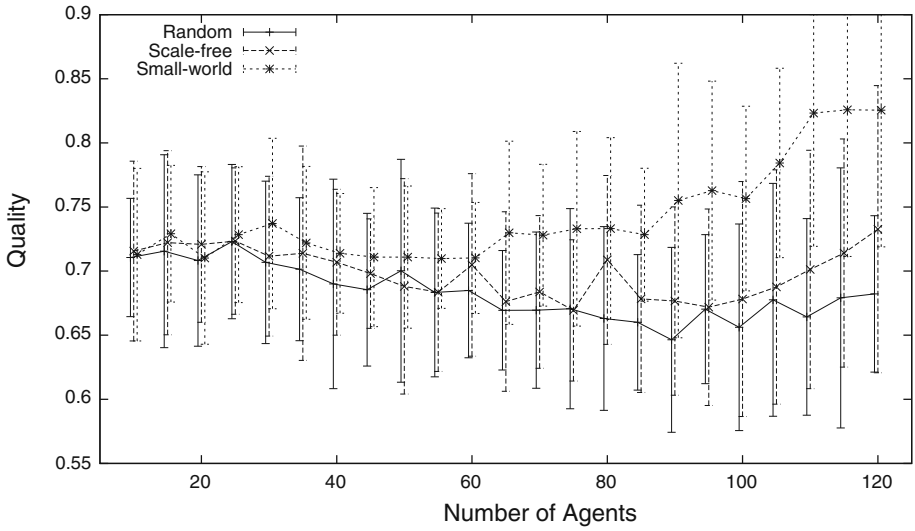

Fig. 11 The quality of GDAP is consistently in the range from 0.6 to 0.8 for varying numbers of agents

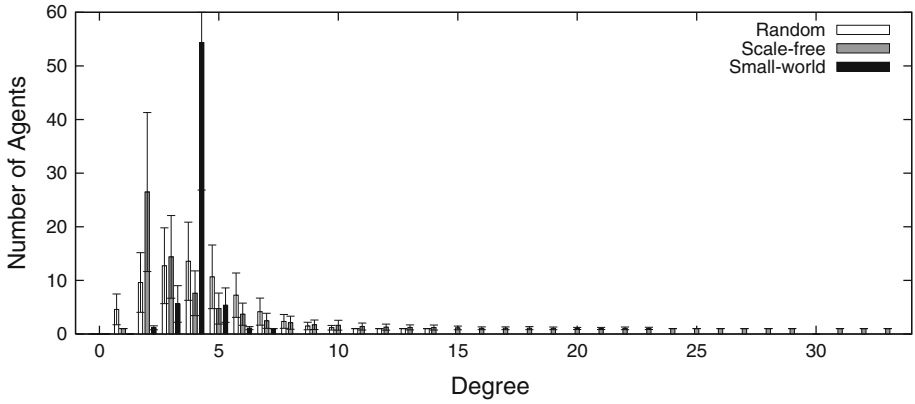

Fig. 12 The average degree distribution of agents in networks with 10–120 agents and an average degree of 4

agents with different degrees (Fig. 13). In random and especially scale-free networks the few agents with high degree can allocate almost all their tasks, whereas there are no agents with a high degree in the small-world network (see Fig. 12). This is further confirmed by Fig. 14. Interestingly, we can thus conclude that the agents in the scale-free network who are in the tail of the degree distribution (with a high degree), become more important when the network grows.

7.4 Tasks

The number of tasks mostly influences the run time (see Fig. 15), but does not seem to influence the quality of GDAP significantly (see Fig. 16). The ratio of the value of the solutions produced by GDAP to that of the optimal solutions quickly converges to around 0.7 for

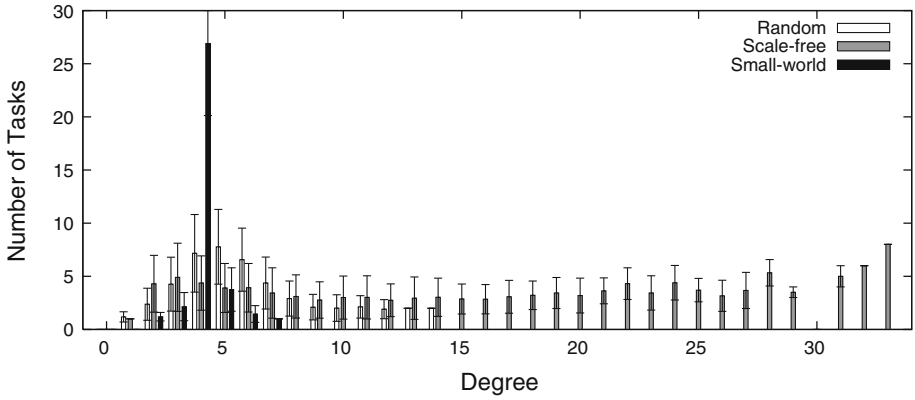

Fig. 13 The average number of successfully allocated tasks by *all agents* of a given degree in networks with 10–120 agents and an average degree of 4

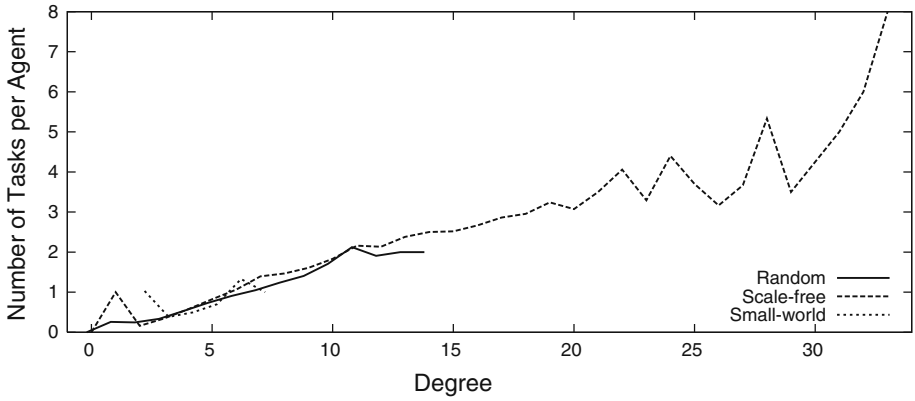

Fig. 14 The average number of tasks successfully allocated by a *single agent* of a given degree in networks with 10–120 agents and an average degree of 4

increasing number of tasks. Regarding the run time, we again see that the run time of the optimal algorithm has a very large standard deviation, which is caused by a few instances with comparatively long run times. We also see that the general trend is that run time depends exponentially on the number of tasks up to where the number of tasks is approximately equal to the number of agents. To the computations of the ILP it seems to matter whether an agent has any task at all, but the effect of having one or more tasks does not seem to make a difference. In contrast, the run time of GDAP is almost linear in the number of tasks.

7.5 Skewed value-distribution for tasks

Apart from the experiments discussed above, we have studied the performance of GDAP in many other settings, for example by varying the number of resources types, the number of resources per tasks or combinations of each of the discussed parameters. The results of such additional experiments could always be explained by a direct extrapolation of the trends observed above. Still, we present one additional setting in this section. From the way the algorithm works, i.e., preferring tasks with a high efficiency, we expect the value of tasks

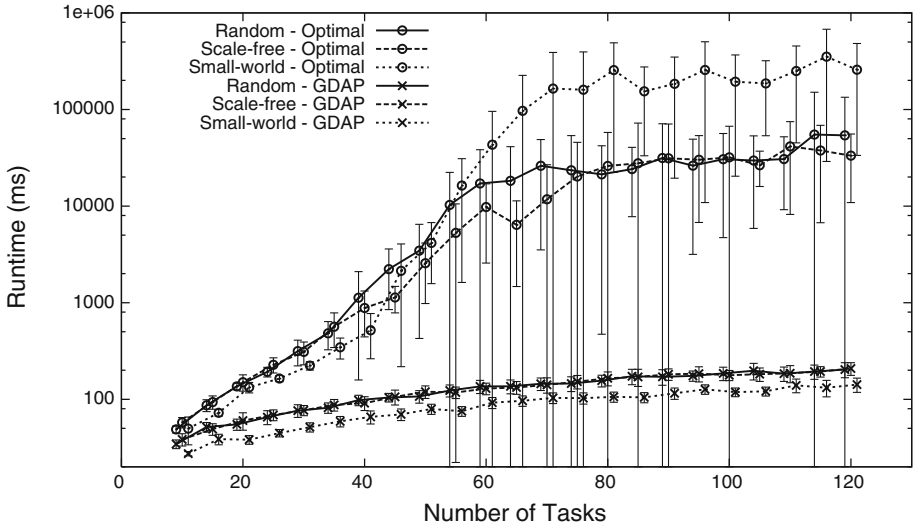

Fig. 15 The run time of the optimal algorithm and GDAP increases with the number of tasks

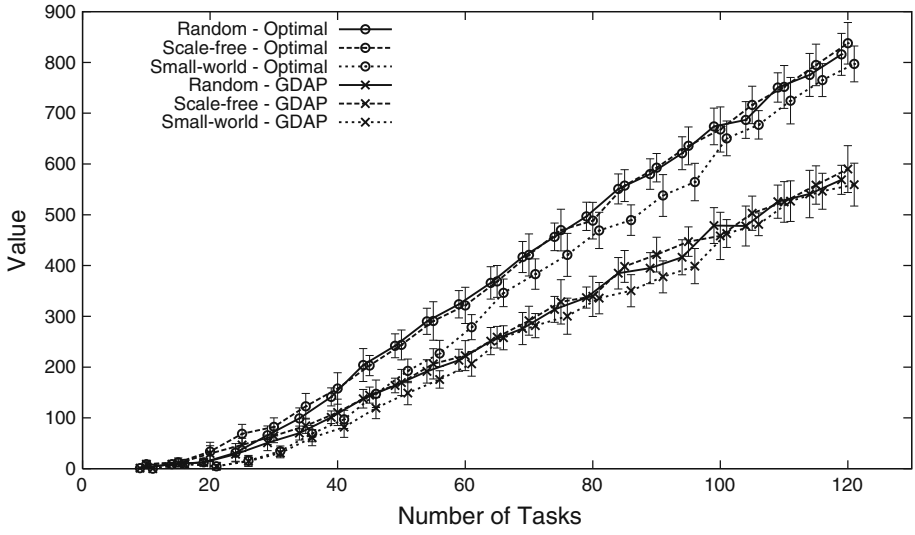

Fig. 16 The value of both GDAP and the optimal algorithm both increase with the number of tasks

to have a significant effect on the results. In all of the above experiments we have used a straightforward distribution of the value of all tasks. In this section we investigate the effect of dramatically altering this distribution.

To study the robustness against different distributions of the value of tasks, we generate instances where the task value distribution is different: 40% of the tasks gets a 10 times higher benefit. This appeared to have no significant effect on the run time altogether, so we focus on the effect on the quality of the solutions relative to the optimal solutions. We again varied the resource ratio and the degree (Fig. 17), the number of agents and the number of

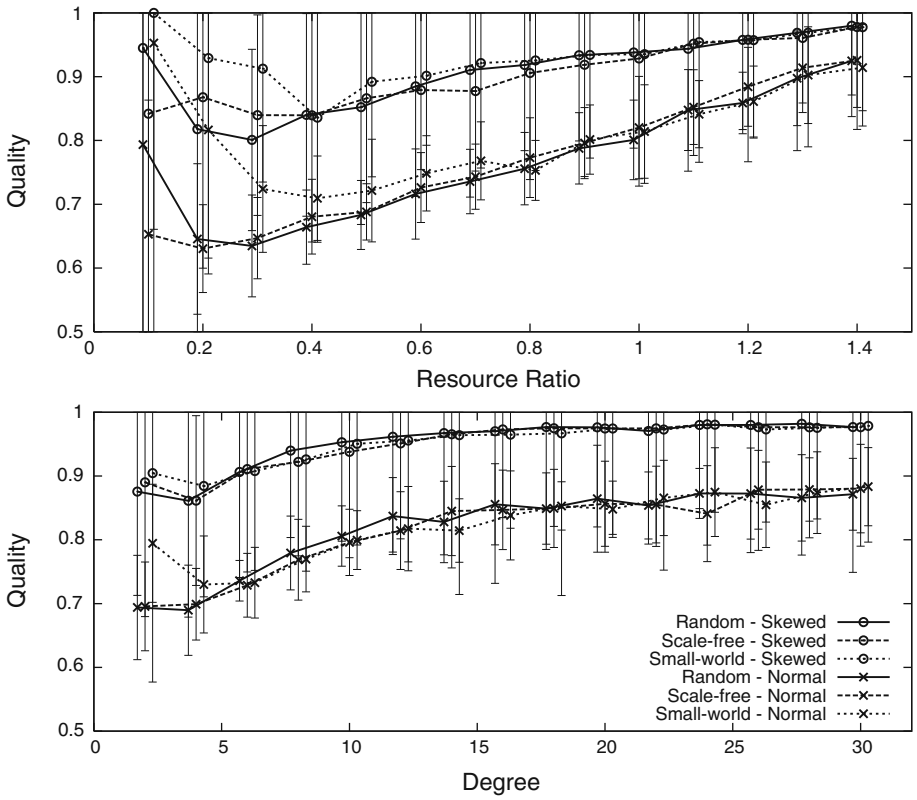

Fig. 17 The quality of GDAP for a normal and a skewed task value distribution related to the resource ratio and the degree

tasks (Fig. 18). From all of these graphs we can observe that the results for this "skewed" task value distribution are significantly better on average. We argue that this can be explained by the greedy nature of GDAP, which causes the tasks with high efficiency to be allocated first. In this heterogeneous setting, the differences between the efficiency of tasks are larger, and the greedy choice is therefore more often equal to the optimal choice.

7.6 Summary

We summarize the performance of the greedy distributed allocation protocol (GDAP) based on the experimental results.

- GDAP works well in all the settings we tested. In addition, as what we expected, its performance is improved when the network is better connected, or when there are sufficient resource available. More specifically, when the average network degree is greater than 10 (see Fig. 9), or the resource ratio is over 1 (see Fig. 7), GDAP consistently returned the solutions with a quality over 0.8, regardless of the network types.
- The worst-case performance ratio of GDAP is about 0.6, which occurs with the problem instances where the network degree is small (around 4) and only less than half of the required resources is available in the network.

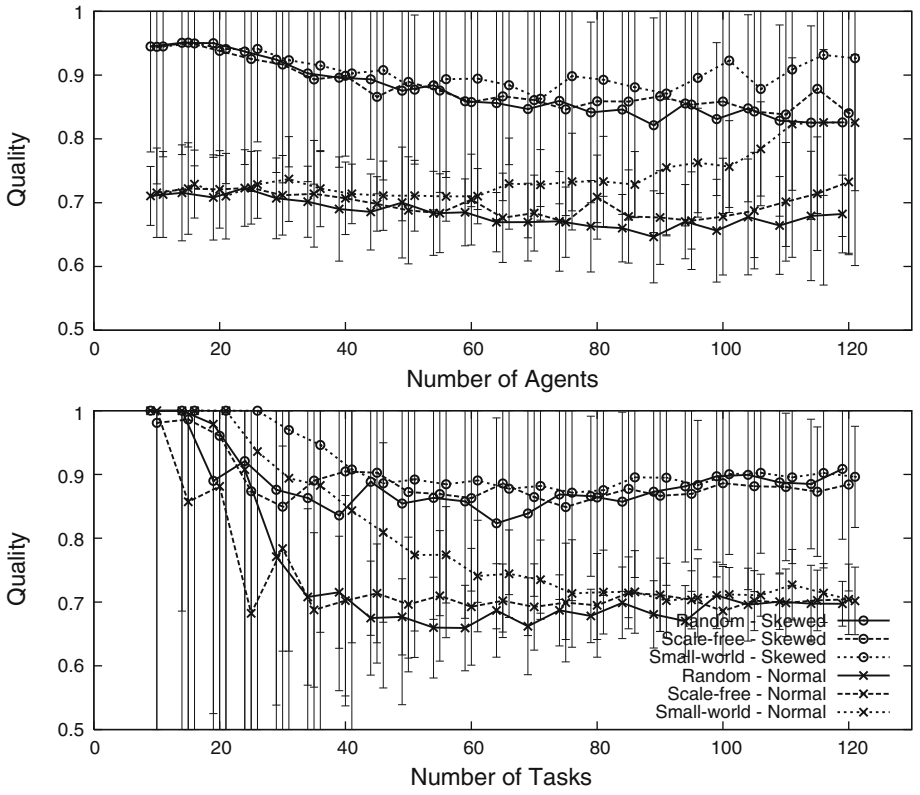

Fig. 18 The quality of GDAP for a normal and a skewed task value distribution related to the number of agents and the number of tasks

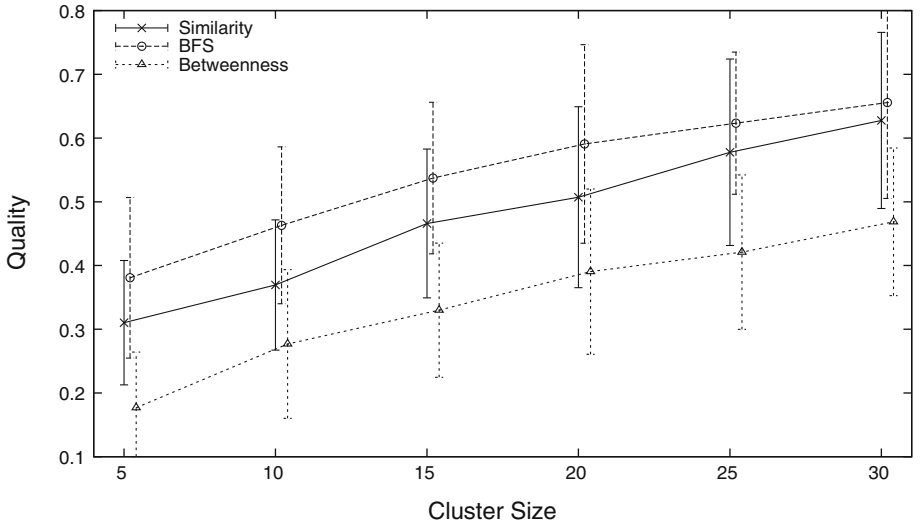

Fig. 19 The quality of the cluster-based algorithm increases for larger cluster sizes and a fixed number of tasks (80) and agents (60)

- Compared to the network degree and the resource ratio, the number of tasks or agents does not influence the quality of GDAP significantly. However, when resources are more sparsely distributed, GDAP works better with small-world networks relative to the optimal solution (Fig. 11, 18).
- In fact, we have seen that the optimal solution in general has a lower value for small-world networks than for random and especially scale-free networks when resources are scarce, because the few agents with a high degree can still allocate some of their tasks, while there are no such agents in the generated small-world networks.
- The experimental results confirm that the run-time of GDAP (Proposition 2) increases only slowly with the number of agents, and the number of tasks (Fig. 15).
- GDAP seems to work even better in more heterogeneous settings, such as when the values of the tasks differ a lot in the network (see Fig. 17, 18).

Based on these results, we expect that the proposed GDAP is able to give high quality solutions for a wide range task allocation problems and handle large problem instances efficiently.

8 Mechanism design for STAP

A high quality task allocation can only be found if the agents report their available resources truthfully. Since we cannot always rely on the agents to be honest, we treat the problem as a mechanism design problem so that every agent is incentivized to report its true resources, no matter what strategies other agents use.

We give a brief summary of the relevant mechanism design concepts below, but for a more elaborate introduction please see [45] or [54]. In a mechanism design setting, we provide a method that determines an outcome, in our case, a valid task allocation $o \in \mathcal{O}$, given the inputs (the strategies) from the contractor agents and all public knowledge. A typical assumption for a mechanism design problem is that some of the information is private. In this paper we consider the setting where the contractor agents have private information.

This private information or *type* s_i of a contractor agent is the description of the resources it has available, i.e., $s_i : R \to \mathbb{N}$. The set of all such functions is called its type space S. The type space of all m agents is defined by S^m. We use $\mathbf{s} = (s_1, \ldots, s_m) \in S^m$ to denote the *type profile* of the agents. We sometimes denote $\mathbf{s}$ by $(s_i, \mathbf{s}_{-i})$, where $\mathbf{s}_{-i}$ denotes the types of all contractors except i. Different from some other mechanism design problems, we assume in this problem setting that using a resources does not incur additional costs (compared to having them but not using them). This situation occurs in for example grid computing, but also in many companies, because the hardware and/or the employees need to be paid anyway.

The revelation principle [41] says that for any coordination mechanism any equilibrium can also be achieved by a truthful direct-revelation mechanism. A direct revelation mechanism is one where the strategy space of the agents is exactly their type space. In our study we therefore define the strategy space A to be equal to the type space S, and search for a mechanism such that none of the contractor agents can improve its utility by manipulating the report of its type, i.e, a truthful mechanism (see Definition 7).

Besides the strategies of the contractor agents, part of the input for the mechanism consists of public information. In our case, this is a social network and a set of tasks. We use Z to denote this public parameter space of the social task allocation problem. Each $z \in Z$ is a tuple $(SN, \mathcal{T})$.

When the mechanism receives inputs $\mathbf{a} = (a_1, \ldots, a_m) \in A^m$ and z, it selects an allocation $o = O(z, \mathbf{a})$ with some allocation algorithm O. In addition, assume the mechanism computes payments $(p_1(z, \mathbf{a}), \ldots, p_m(z, \mathbf{a}))$ for all contractor agents. The result for agent i, called its *utility*, is the sum of the *valuation* v_i that i gets from the resulting allocation o with its type s_i and the payment it receives from the mechanism:

$$u_i(\mathbf{a}) = v_i(s_i, o) + p_i(z, \mathbf{a}).$$

In the STAP, we define the valuation of agent i as its fair share of the utilities of the tasks it helps to fulfill.

$$v_i(s_i, o) = \sum_{t \in T_o} \sum_{r \in R} o(t, i, r) \cdot e(t). \tag{1}$$

For example, if an agent i contributes 3 out of 6 resources of different types for a task t with value 10, then $o(t, i, r) = 1$ whenever r is one of these types. The valuation of i in this case is then $3 \cdot e(t)$. This efficiency (Definition 3) is $e(t) = \frac{u(t)}{\sum_{r \in R} req(t,r)} = \frac{10}{6}$, so $v_i(s_i, o) = 5$.

The utility u_i is what agent i aims to maximize. The *social welfare* $W(o)$ of the system is the sum of the valuations of the contractors in the allocation o, i.e., $W(o) = \sum_{i=1}^{m} v_i(a_i, o)$. We use this to define the mechanism design problem for social task allocation formally.

Definition 6 (Mechanism design for STAP) Given the parameter space Z, the strategy space A and the social welfare function W, the *mechanism design problem for* STAP is to find a mechanism $\mathscr{M} = (O, p)$ that consists of an *allocation function* $O : Z \times A^m \to \mathscr{O}$, and a *payment function* $p_i : Z \times A^m \to \mathbb{R}$ such that the selected output $o \in \mathscr{O}$ maximizes the total social welfare $W(o)$.

A mechanism is *efficient* if it maximizes the social welfare. Such a mechanism design problem is called a *utilitarian mechanism design problem* [46]. We now show that the goal of this mechanism design problem is exactly the same as the optimization criterium for the STAP, i.e., the utility of all allocated tasks T_o.

$$\begin{aligned} U(T_o) &= \sum_{t \in T_o} U(t) = \sum_{t \in T_o} \sum_{r \in R} req(t)(r) \cdot e(t) \\ &= \sum_{i \in \mathscr{A}} \sum_{t \in T_o} \sum_{r \in R} o(t, i, r) \cdot e(t) \\ &= \sum_{i=1}^{m} v_i(a_i, o) = W(o) \end{aligned}$$

Thus, when an algorithm for STAP gives the optimal solution, it also outputs the optimal social welfare for the mechanism, as long as agents report their private information truthfully, i.e., if the mechanism is *truthful*.

Definition 7 (Truthful) Given an output algorithm O, a (direct) mechanism is *truthful* if $A = S$, and for any parameter $z \in Z$, for any strategy profile $\mathbf{a} \in A^m$, for any agent i with type $s_i \in S$ it holds that

$$\begin{aligned} u_i(s_i, \mathbf{a}_{-i}) &= v_i(s_i, O(z, s_i, \mathbf{a}_{-i})) + p_i(z, s_i, \mathbf{a}_{-i}) \\ \geq u_i(a_i, \mathbf{a}_{-i}) &= v_i(s_i, O(z, a_i, \mathbf{a}_{-i})) + p_i(z, a_i, \mathbf{a}_{-i}) \end{aligned}$$

Informally, agent i is never worse off by revealing its true private type s_i to the mechanism, no matter what strategies other agents play. Truthful mechanisms can be achieved with carefully designed payment functions, such as the Groves mechanism [26]. It has been shown that truthfulness can be guaranteed by a Groves mechanism if the mechanism is able to compute the optimal solution [46].

Definition 8 (Groves mechanism [46]) A mechanism $\mathscr{M} = (O, p)$ is a Groves mechanism if:

1. The allocation function $O(z, \mathbf{a})$ maximizes the total welfare according to $\mathbf{a}$, i.e. for all $\mathbf{a}$, $O(z, \mathbf{a}) \in \arg\max_{o \in O(z,\mathbf{a})} W(o)$.
2. The payment of agent i is calculated as follows: $p_i(z, \mathbf{a}) = -v_i(a_i, O(z, \mathbf{a})) + W(O(z, \mathbf{a})) + h(\mathbf{a}_{-i})$, where h is an arbitrary function of $\mathbf{a}_{-i}$.

In the following, we first discuss a Groves mechanism for the STAP allocation algorithm that maximizes the objective function.

8.1 Exact truthful mechanism

Consider the following Groves mechanism using an optimal algorithm.

Definition 9 ([M_{OPT} **for** STAP) The optimal task allocation mechanism consists of

- A *task allocation algorithm* OPT: Let $z = (SN, \mathscr{T})$ be an instance of STAP. First the mechanism center announces a set of tasks $\mathscr{T}$—required resources (type and demand), utilities and locations—that need to be allocated to all contractor agents. Next the contractors declare their types $\mathbf{a}$ to the center. The center then finds the efficient allocation $o = \mathsf{OPT}(z, \mathbf{a})$ using the ILP translation.
- A *payment function* p^{OPT}: The payment of an agent is its marginal contribution to the society, i.e.,

$$p_i^{\mathsf{OPT}}(z, a_i, \mathbf{a}_{-i}) = -v_i(a_i, o) + W(o) - W(o_{-i}) \tag{2}$$

  where $o_{-i} = \mathsf{OPT}(z, \mathbf{a}_{-i})$ is the efficient allocation computed by OPT without i's participation. This is also called the Clarke pivot payment or Clarke's tax [54].

Because this exact mechanism is based on the Groves mechanism, it almost automatically inherits properties such as being *efficient* and *truthful*. An agent has no incentive to not fully state its available resource types and amounts, since the utility of an agent is its marginal contribution to the society: suppose an agent states less resources, the total number of allocated tasks will be no more than when it fully states its available resources. Thus the marginal contribution of the agent to the social welfare would then be no more, since the resulting efficient allocation has lower utilities. Therefore the agent will derive a lower utility due to its incomplete report.[1]

Corollary 1 *The mechanism* $\mathscr{M}_{\mathsf{OPT}} = (\mathsf{OPT}, p^{\mathsf{OPT}})$ *is a truthful and efficient mechanism, where agents always receive non-negative utilities by participating in the game, i.e., agents are individually rational (IR). In addition, the mechanism gives no payment to agents that do not get any allocated tasks.*

[1] In this paper we do not address the case that an agent overstates its resources. Such misreports have been studied in for example [36].

This statement is a corollary of the truthfulness, IR, and optimality results of Groves mechanisms with a Clarke pivot payment (VCG mechanisms) that use an optimal allocation rule, see e.g., [54].

An additional desirable property would be that the payments for each task in total is exactly the same as the value of that task. This property is called budget balancedness. However, it follows directly from Myerson and Satterthwaite [42] that it is impossible to achieve budget balancedness together with efficiency and individual rationality for a truthful mechanism. In this paper we therefore do not place this additional requirement, but focus on how to obtain efficient solutions in polynomial time. Truthfulness is very important in this respect, because otherwise the mechanism may make decisions based on incorrect information.

8.2 A computationally efficient truthful mechanism

The exact mechanism for STAP, $\mathscr{M}_{\mathsf{OPT}}$, is truthful and efficient. However, it takes exponential time to compute the allocation and the payment. Obviously, this is not feasible when the problem size is large. In this section, we first show that a mechanism using the greedy algorithm cannot be truthful. We then develop a truthful polynomial-time mechanism by splitting the problem into sub-problems that each can be solved optimally in polynomial time.

Proposition 3 *No Groves mechanism is truthful with the greedy task allocation algorithm* GTA.

Proof We show that for a specific instance the Groves mechanism cannot incentivize a contractor to declare all its available resources truthfully if the greedy task allocation algorithm is used. Without loss of generality we assume that $h(\mathbf{a}_{-i}) = 0$. Thus, $p_i(z, \mathbf{a}) = -v_i(a_i, \mathsf{GTA}(z, \mathbf{a})) + W(\mathsf{GTA}(z, \mathbf{a}))$.

Consider a problem instance with tasks t_1, t_2 and t_3. Task t_1 requires resources $\{r_1, r_2, r_3\}$; task t_2 requires $\{r_2, r_4\}$; and t_3 requires $\{r_3, r_5\}$. All three tasks are connected to contractors i and j, where i has resources $\{r_1, r_4, r_5\}$, and j owns $\{r_2, r_3\}$. Let the utilities of the tasks be $U(t_1) = 15$, $U(t_2) = 8$, and $U(t_3) = 8$. Thus the efficiencies are 5, 4, and 4, respectively. Suppose that agent j is truthfully reporting its resources $\{r_2, r_3\}$. We now compare two situations. When i also declares its type truthfully to the mechanism, i.e. $\{r_1, r_4, r_5\}$, then according to the greedy algorithm, the resulting allocation is $o_1 = \mathsf{GTA}(z, s_j, s_i)$ with $T_{o_1} = \{t_1\}$, because t_1 has the highest efficiency. The payment then is $p_i(z, s_j, s_i) = -v_i(s_i, o_1) + W(o_1)$. So in this case, the utility that i receives by declaring truthfully is $u_i(s_j, s_i) = v_i(s_i, o_1) + p_i(z, s_j, s_i) = W(o_1) = 15$.

Consider now a case where i mis-reports ($a_i \neq s_i$) its resources, i.e., $\{r_4, r_5\}$. In this case t_1 cannot be allocated. The greedy algorithm then outputs the allocation $o_2 = \mathsf{GTA}(z, s_j, a_i)$ and $T_{o_2} = \{t_2, t_3\}$. The utility of i then becomes $u_i(s_j, a_i) = v_i(s_i, o_2) - v_i(a_i, o_2) + W(o_2) = (4 + 4) - (4 + 4) + 16$. Since $u_i(s_j, a_i) > u_i(s_j, s_i)$, agent i is better off by lying about its available resources. This mechanism is not truthful. □

Clearly, another approach is required to obtain a polynomial algorithm that is also truthful. The idea is to create sub-problems that can be solved in such a way that the resulting mechanism is truthful and the quality of the resulting solutions is still quite good. To achieve truthfulness, we should divide the problem in a way that does not depend on the declared types of the agents. In addition, we would like the algorithm to be computationally feasible. In most applications, this means that the algorithm should be polynomial in at least the number of agents m and the number of tasks n (for a fixed number of resource types).

Algorithm 3 Cluster-based algorithm CLS.

Input: a set of agents $\mathscr{A}$, tasks $\mathscr{T}$, a network $SN = (\mathscr{A}, AE)$ and a cluster size c.
Output: a task allocation and its value.
While there are tasks and agents left, execute the following lines.

1. Find a cluster C of agents and their tasks of size at most $\max\{c, \log n\}$ using a polynomial-time clustering method that does not use private information of the agents.
2. Compute the task allocation for C using the optimal algorithm OPT.
3. Remove C from the problem.

The idea of the cluster-based algorithm we present below is thus to find a *partitioning* of the given social network into several disjoint subnetworks (or clusters) so that each subnetwork is small enough to be solved by the optimal algorithm using ILP described in Sect. 4.1. Given a STAP with n tasks and m agents, we partition the graph so that each cluster C contains at most $\log n$ tasks and $\log m$ agents when n and m are large. For smaller numbers of tasks we limit the number of tasks per cluster by a fixed number c. Pseudocode of this algorithm is given in Algorithm 3.

Proposition 4 *Given a* STAP *with n tasks, m agents, l resource types, and the degree of the network bounded by Δ, the cluster-based algorithm* CLS *is polynomial in m, n, Δ, and exponential in l. When the number of resource types l is bounded by a constant,* CLS *is a polynomial-time algorithm.*

Proof In this proof we show that the asymptotic run-time is polynomial; we therefore ignore the case that $\log n < c$ in the following. Given that the clustering method is polynomial, the run time is mainly determined by the computation of the optimal solution for each cluster. Recall that the ILP can be solved in time exponential in n, m, and l. In each cluster, the size of the input for the ILP is reduced by restricting the number of tasks and agents. More precisely, in each cluster the number of variables in the ILP formulation is $\mathrm{O}(\log(n)\log(m)l)$, therefore, computing the solution for each cluster using ILP takes: $e^{\mathrm{O}(\log(n)\log(m)l)} = \mathrm{O}\left(m \cdot n \cdot 2^l\right)$.

This algorithm is thus fixed-parameter tractable in l [44]. In other words, when the number of resource types l is bounded by a constant, the computation time of CLS after clustering is $\mathrm{O}\left(\frac{n}{\log n}\right) \cdot \mathrm{O}(mn) = \mathrm{O}\left(\frac{n^2 m}{\log n}\right)$, which is polynomial. □

The cluster-based algorithm can be used to define a truthful mechanism as follows.

Definition 10 (A cluster-based mechanism $\mathscr{M}_{\mathsf{CLS}}$) A cluster-based mechanism $\mathscr{M}_{\mathsf{CLS}} = (\mathsf{CLS}, P_{\mathsf{CLS}})$ works as follows.

- Let $z = (SN, \mathscr{T})$ be an instance of STAP. First, the mechanism partitions the agent network according to Algorithm 3, Steps 1–4.
- After one cluster C_j is formed, the agents in this cluster are asked to submit their private types $\mathbf{a}$ to the mechanism. Based on the declared types, the mechanism uses the task allocation algorithm CLS to get the optimal allocation o^j of this cluster (Algorithm 3, Step 2).
- A payment function p^{CLS} calculates the payment to the agents in each cluster C_j based on the same payment function as used by the OPT mechanism (Definition 9), i.e., $p_i^{\mathsf{CLS}}(z, a_i, \mathbf{a}_{-i}) = -v_i(a_i, o^j) + W(o^j) - W(o^j_{-i})$.

The mechanism calculates the allocation and the payment for each cluster. Agents are asked to submit their types after the clusters are formed. Therefore, agents' private types will

not influence the partitioning process, i.e. they cannot manipulate in order to enter "good" clusters. When the mechanism divides the network into several clusters, it uses only the public information—the network structure and the information of the tasks, no private information of agents is involved in this stage, and each agent can only belong to one cluster. Indeed, we will show this is the key fact which ensures that the mechanism is truthful even when a sub-optimal allocation algorithm (CLS) is used.

Theorem 3 *The mechanism* $\mathscr{M}_{\mathsf{CLS}} = (\mathsf{CLS}, p^{\mathsf{CLS}})$ *is truthful, individually rational and gives no payment to agents that do not get any allocated tasks.*

Proof The cluster-based algorithm (Algorithm 3) removes some edges between agents and tasks, and divides the network into disjoint clusters. Given a problem instance z, as a result of partitioning, the set of "allowable" allocations given the type space of agents $\mathbf{a}$ is restricted to $\Omega = \omega_1 \cup \ldots \cup \omega_k$ where $\omega_j = \{o_1^j, o_2^j, \ldots\}$ denotes the set of allowable allocations in cluster C_j. Note that Ω is a subset of the allowable outputs $\mathscr{O}$ in the exact mechanism: $\Omega \subseteq \mathscr{O}$. Each agent is only in one cluster after partitioning.

The declarations of the agents will not influence the partitioning process. Next we prove that for each cluster, truth-telling is always in the best interests of all agents. For this we use a similar proof as for the exact mechanism (Theorem 1).

Given an agent i in the cluster C_j, we define the true type of i by s_i and any other type by a_i. Let $o^j = \mathsf{CLS}(z, s_i, \mathbf{a}_{-i})$, and $\hat{o}^j = \mathsf{CLS}(z, a_i, \mathbf{a}_{-i})$, $o^j, \hat{o}^j \in \omega_j$, respectively. The difference δ of the utility that i will receive by declaring s_i and a_i is:

$$\delta = u_i(s_i, a_{-i}) - u_i(a_i, a_{-i}) = W(o^j) - W(\hat{o}^j).$$

Since we use the optimal algorithm OPT to find the optimal allocation in every cluster, o^j is the best allocation over ω_j, i.e., $o^j = \arg\max_{o'^j \in \omega_j} W(o'^j)$. Therefore, $W(o^j) - W(\hat{o}^j) \geq 0$, i.e., agents are better off by report their types truthfully. The truthfulness result holds for all clusters. In addition, since the network is partitioned without the knowledge of the contractor agents' private information, the contractors cannot manipulate the mechanism in order to enter different clusters. Similarly, we can show the mechanism is individually rational. □

8.3 Clustering heuristics

Finding a graph partitioning where a minimal number of connections is broken is NP-hard for more than two clusters (so when the cluster size $c < \frac{1}{2}m$, where m is the number of agents) [21]. In this paper we therefore limit ourselves to evaluating the quality of the task allocation where the clusters are determined heuristically. We discuss three examples of such heuristics.

Breadth-first clustering The first heuristic repeatedly performs a breadth-first search in the (remaining) graph until as many agents are visited as are allowed in a cluster. A next cluster is formed starting from the agent that was about to be visited in the previous cluster. When there is no such unvisited agent, a starting agent is randomly selected from the remaining unvisited agents. This heuristic is very efficient and runs in $O(m^2)$ time, but can even be implemented to run in linear time.

Clustering using the betweenness of agents For the second heuristic we again use the betweenness measure described in Sect. 4.2. This measures the number of shortest paths (between any two agents) through an edge [24]. The intuition now is that when an edge connects two clusters, it usually has a high betweenness. By repeatedly removing the edge

with the highest betweenness, at some point the graph falls apart into separate components (clusters). This heuristic continues to do so for each cluster that is larger than the allowed maximum. The run time of this method requires m times computations of all-pair-shortest paths. Our implementation uses the betweenness implementation from the Jung graph library for each connection in AE, which in itself is fairly inefficient; the heuristic in total runs in $O(|AE|(m^2 + m|AE|))$ time.

Clustering using efficiency and the similarity of the neighborhood of tasks The final heuristic comes from the consideration that tasks which connect to the same contractor agents should end up in the same cluster, because these interactions between tasks are the core of our problem. We would like to keep as many of these interactions in the sub-problems to maintain a certain level of quality.

For this heuristic, we start with the most efficient task, which has the highest utility for using resources. The tasks are divided in such a way that those in one cluster are more related to each other than to tasks outside of the cluster. The similarity of a task t to a cluster C is measured by the number of mutual contractor agents they are connected to: $\text{sim}(t, C) = \sum_{t' \in C} |\text{mutual}(t, t')|$, where

$$\text{mutual}(t, t') = \left\{ i \in \mathscr{A} \,\middle|\, \begin{array}{l} ((i, loc(t)) \in AE \text{ or } i = loc(t)) \text{ and} \\ ((i, loc(t')) \in AE \text{ or } i = loc(t')) \end{array} \right\}$$

After we find a cluster C with at most $\max\{c, \log n\}$ tasks, we limit the number of contractor agents in C by keeping only the agents which have most connections to the tasks in C. To distribute the contractor agents evenly over the $\frac{n}{\log n}$ clusters, we add at most q contractor agents to each cluster, where $q = \min\left\{\log(m), \frac{m \cdot \max\{c, \log n\}}{n}\right\}$, so for small problems we just use $q = \frac{mc}{n}$. In this way, the size of each cluster k is bounded by $\max\{c, \log n\} + q$.

Sorting the set of tasks on efficiency takes $O(n \log n)$. For a cluster C, computing the similarity of one task to C is bounded by $O(\log n \Delta)$. This similarity is calculated for at most $O(n)$ tasks. This is repeated for each task to be added to the cluster. Thus, determining one cluster takes $O\left(n(\log n)^2 \Delta\right)$. As there are in total at most $O\left(\frac{n}{\log n}\right)$ clusters to be found, the computation time for this clustering heuristic is: $O\left(n \log n + n^2 \log n \Delta\right) = O\left(n^2 \log n \Delta\right)$.

These (truthful) cluster-based algorithms may bring some undesired performance losses, depending on the clustering used. As we cannot guarantee the performance theoretically, in the next section, we show the performance of this polynomial-time mechanism $\mathscr{M}_{\text{CLS}}$ experimentally on these three different clustering heuristics.

8.4 Experiments on selfish social task allocation

To evaluate both the quality and the run time of the cluster-based algorithm, we compare the clustering heuristics in two settings. First we study the effect of the cluster size (i.e., the number of tasks in one cluster) on quality and run time in the default setting of 60 agents and 80 tasks, and then we give some results on quality and run time for larger problem sizes.

8.4.1 Effect of cluster size

For the default setting with 80 tasks, half of the required resources available, an average degree of 4, 1,600 resources ratio, and 60 agents, we vary the cluster size from 5 to 30 in steps of 5 and measure both the runtime and the quality compared to the optimal solution.

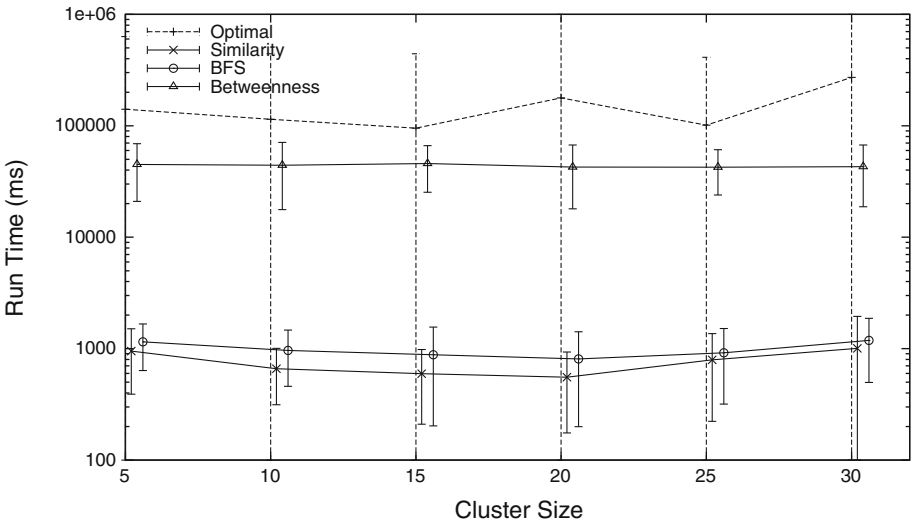

Fig. 20 The run time of the cluster-based algorithm stays approximately constant for increasing cluster sizes and a fixed number of tasks (80) and agents (60)

The results on the quality comparison averaged over all three network types can be found in Fig. 19. The quality of all three heuristics increases with the cluster size, which is not surprising, given that an optimal algorithm is used for each cluster. The figure also shows that the performance of the betweenness heuristic is significantly worse than the other two heuristics. For the other two heuristics the cluster-based algorithm performs reasonably well with a quality between 0.3 for small clusters up to 0.65 when the cluster size is around 30.

The computation time required for these results stays approximately constant for increasing cluster sizes and a fixed number of tasks and agents, as illustrated by Fig. 20. In addition, this figure illustrates the long computation time required for the betweenness heuristic; it almost takes as long as running the optimal algorithm.

8.4.2 Scalability

When the number of tasks grows and the cluster size stays the same, the cluster-based algorithm will create more clusters. To see the effect of this on quality and computation time, we run an experiment with 60 agents, a cluster size of 15, while varying the number of tasks from 10 to 120 (and thus the number of clusters from 1 to 8). The results of this experiment can be found in Figs. 21, 22, and 23. Surprisingly, when the number of tasks is at least 40, the relative quality is approximately constant between 0.3 and 0.4 for random and scale-free networks, and between 0.6 and 0.7 for small-world networks (see Fig. 21), which is all significantly lower than the performance of GDAP. The difference in performance of the cluster-based algorithm in small-world networks can be explained by the fact that a small-world network by construction is already a bit clustered, and therefore the loss of quality by ignoring some of the relations is less severe. Only for a small number of tasks, we see the (positive) effect of having fewer clusters.

This can be seen even more clearly from Fig. 22. When there are very few tasks and 60 agents, on average only one task can be allocated (in the optimal solution). Both the BFS and the similarity heuristic obtain this optimal result, because they create clusters around the few

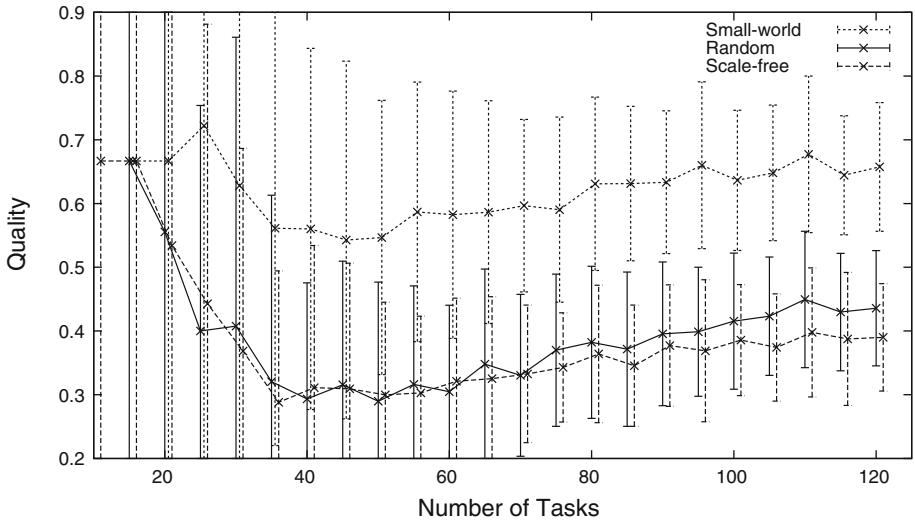

Fig. 21 On average over all heuristics the small-world networks are best suited for the clustering algorithm

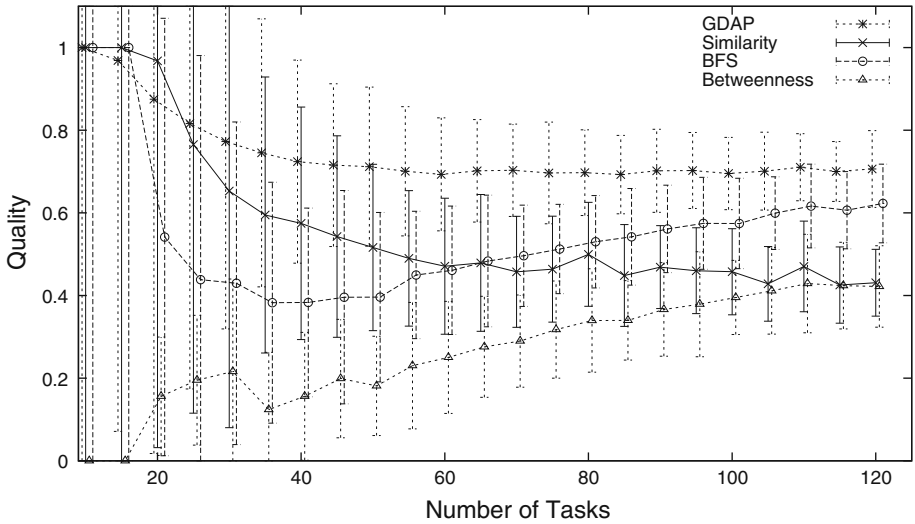

Fig. 22 The quality of the similarity heuristic stays constant around 0.4, while both BFS and betweenness slowly increase with the number of tasks (beyond 40 tasks)

available tasks. The betweenness heuristic only considers the agent network and therefore performs very bad here.

The results on the run time (Fig. 23) give another reason for not using betweenness in this way; for some problems the clustering heuristic uses more time than the optimal algorithm. Besides this, the run time results behave as expected, requiring significantly more time when there are more tasks.

To see the effect of an increasing network size, we also run the cluster-based algorithm on a setting with 80 tasks and with a maximum cluster size of 15, but varying the number of agents

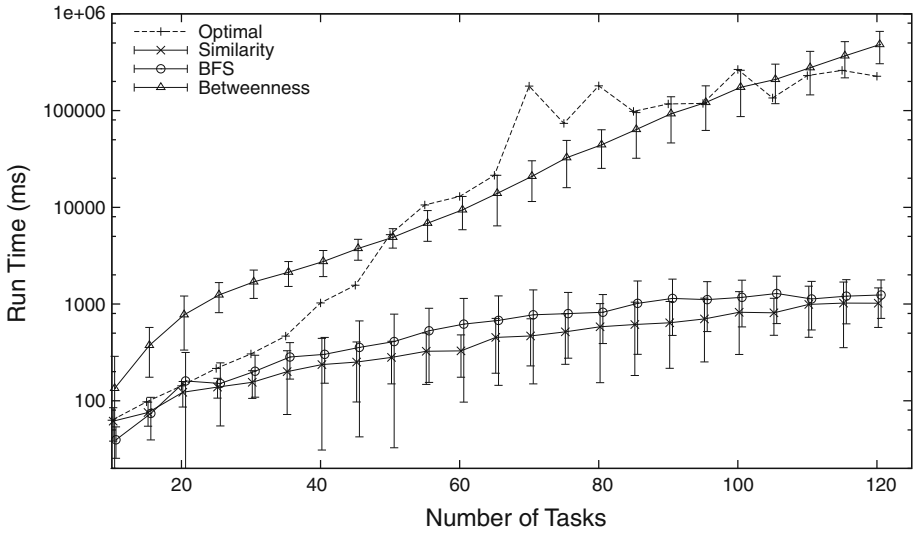

Fig. 23 The run time of the cluster-based algorithm increases with the number of tasks because the number of clusters increases

from 10 to 120. Changing the number of agents while keeping the number of tasks constant hardly has any interesting effect on the runtime, but regarding the quality we can make the following observations, based on Fig. 24. This figure contains the results for the similarity and BFS heuristics on all three networks (we left out betweenness for clarity and because it is dominated in both run time and quality). The figure shows that increasing the size of the network slightly reduces the quality of the clustering algorithm using the BFS heuristic. When these networks grow while keeping the number of tasks the same, the clusters constructed using BFS contain a decreasing portion of the available tasks and resources, making it hard to find good overall solutions. However, the similarity heuristic has a stable performance on both the scale-free and random networks, significantly outperforming the BFS heuristic in such networks with over 90 agents. Even more striking, however, is that the quality of the similarity heuristic in small-world networks keeps *increasing* with the size of the network. This increase can be partly explained by the decrease in the absolute optimal value for small-world networks with a large number of agents and a fixed (small) number of tasks, as we discussed in Sect. 7.3. In addition, the main difference between the similarity heuristic and BFS is that the similarity heuristic puts tasks with similar requirements together in the same cluster. This seems to pay off mostly in large networks where tasks and resources are scarce.

From these results we conclude that networks in which it is harder to find good clusters, such as random and scale-free networks, may need another type of heuristic to break the problem down into good independent and manageable parts. We leave finding such an alternative heuristic for future work.

9 Related work

Task allocation in multiagent systems has been investigated by many researchers in recent years with different assumptions and emphases, ranging from centralized methods to

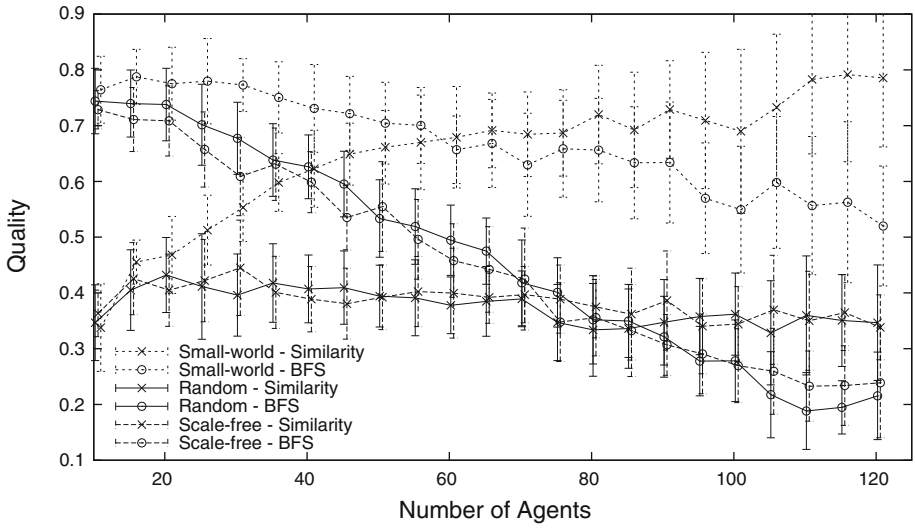

Fig. 24 The quality of the cluster-based algorithm (averaged over all three heuristics) for small-world networks is more or less stable, but decreases on random and scale-free networks with the number of agents

distributed ones, fully cooperative agents to self-interested settings, taking the relations between agents into account, or assuming full connectivity. In this section we aim to provide a guideline to such literature related to task allocation in social networks. First we discuss recent developments in computing task allocations when all agents are connected. Then we concentrate on the relation of these approaches to using auctions for similar allocation problems. We conclude this section with an overview of related work that takes relations between agents into account.

9.1 Allocations with fully connected agents

Shehory and Kraus [52] study how to allocate tasks that require cooperation among a set of agents. Such a set of agents is called a coalition, and in fact this is thus strongly related to the problem of coalition formation. In coalition formation usually stability is the primary goal, versus the problem of optimizing the value of the allocated tasks, as in our work. A recent paper by Dunne et al. [16] provides a thorough analysis of the computational complexity of this problem of finding stable coalitions, which they call coalitional resource games.

In our setting we concentrate on maximizing the value of the allocated tasks, in a way similar to work by Kraus et al. [34,35], where they study the case where the tasks need to be executed by self-interested agents under time constraints (i.e., a discount of the utility of a task per auction round). They focus on finding strategies for distributing revenues fairly among agents that are close to a notion of stability. There are no theoretical guarantees on stability or incentive compatibility provided, but they perform experiments with up to 15 agents and 5 tasks, that show that the strategies are close to stable. We believe their work provides an interesting alternative revenue division for the proportional revenue that we use in our work here. The theoretical counterpart of this work is provided by Manisterski et al. [39], who show that when budget balancedness is required, efficiency is only possible in specific settings (such as with only one task). In our setting we do not obtain budget balancedness, but we provide two generally applicable mechanisms: one that is guaranteed to be efficient, but

may require exponential running time, and one that is computationally efficient, but where the performance depends on the cluster size. However, the proposed task allocation protocol in [34,35] but also the Pareto-optimal approach in [53] is centralized, where one manager is responsible for the auction of all tasks: announcing the tasks, collecting the proposals, and allocating the tasks to the agent coalitions. We generalize that model by having each task managed by a different agent, distributed over the network. The other extreme is to let agents form and break coalitions completely distributedly, using simple local heuristics. Such an approach is for example used for forming buyer coalitions in large-scale multiagent systems by Lerman and Shehory [37] and can also be found in [53].

Focusing more on optimality than stability, a related line of work is on scheduling with self-interested agents [28,33,47]. The similarity between task allocation and scheduling jobs on machines is that in both cases resources (machines) are allocated to tasks (jobs). However, in scheduling the resources (machines) can be used as often as required, and usually all jobs are allocated. The goal is there to minimize e.g., makespan, lateness (in case of deadline), or weighted completion time of a schedule allocating all tasks. The difficulty in task allocation arises exactly from the fact that not all tasks can be allocated and that therefore an optimal selection of tasks needs to be chosen.

9.2 Auctions for allocation problems

Auctions have been a popular method for realizing distributed task or resource allocation in multiagent systems [5,25,15,55], among which the Contract Net (or CNP) [55] is probably the most well-known. In the CNP, the auctioneer sends out a call for proposals, the bidders submit their bids and the auctioneer then evaluates the bids and determines the winner. Finally the bidders receive or reject the offer. Many variants of the CNP have been developed for different problem domains, for instance, in multirobot coordination. Goldberg et al. [25] use auction-based methods for allocating the tasks to robots. In [5], the authors propose one-shot auction to tackle the changes in a dynamic environment when allocating tasks with time and dependency constraints. In their approach, any agent can become an auctioneer and call for proposals on the task it had accepted earlier in case this task fails (or might fail). The bids from the agents include a time window and the agents use combinatorial bids (i.e., combinations of tasks). The auctioneer selects the best bids which fit its overall schedule.

The winner determination for combinatorial auctions (CAs) is NP-complete [49]. CAs can be viewed as a "reverse" task allocation, where the agents with resources are the auctioneer(s) and the agents who want to obtain the resources (for completing their tasks) are bidders. The downside of CAs is that they often require an exponential number of task bundles to be considered, which results in an exponential number of communication messages. The difference from our approach is that we compute an allocation for each task (almost) independently, using locality of both tasks and resources. This can be done very efficiently, as shown in this paper. Inversely, a possible consequence of our work could be that we provide here some first insights in how to deal with combinatorial auction problems where resources and bidders are distributed and not every bidder is allowed to bid on every resource. More generally, none of the research mentioned above studies the effect of different social networks of self-interested agents on the run time and the quality of the allocations.

9.3 Allocations in networks of agents

There is some work where networks have been employed in the context of task allocation with a different focus or application. For example Gaston and DesJardins [23] study how

the topology of an organization's structure affects the performance of the teams forming in multiagent systems. They show that the agents' interaction topology is a key factor in effective team formation, mainly that scale-free networks support the *diversity* required for effective teams. Eberling and Kleine Büning [18] design a system where agents cooperate with neighbors to perform assigned tasks. They select others to cooperate with who are in some sense "like" themselves, whereas we study the problem of finding agents that have complimentary resources to allocate tasks. A very interesting aspect of their work is that they study the effect of adapting the structure of their local network. Such changes can for example be based on the long run benefit of relations, which could be learned from observing past behavior as has been put forward in earlier work [7,22]. The focus of the research discussed above is on distributed coalition or team formation among agents in a network. In our work we also investigate how the different network topologies affect the performance of the system; in our case a task allocation. However, the main difference from our approach is that we do not need agents to form groups *before* allocating tasks, but tailor the agents assigned to a task based on the specific problem instance, given existing social relations.

We study the effect of limiting cooperations exactly to such social relations, but networks of agents are also used to model the allowed communication between agents. For example, Kafalı and Yolum [30] investigate how different communication protocols improve the efficiency of the resource allocation when agents are not fully aware of all other agents and their demand for resources. Vidal [58] develops a protocol for agents to coordinate their actions such that communication allows them to align their individual search landscapes with the global fitness landscape. A minimum level of connectedness in the agents' communication network is required for this to work. Abdallah and Lesser [1] use networks to model limited interactions between agents and mediators. Mediators in this context are agents who receive the task and have connections to other agents. They break the task up 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. The work of [50] 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 be mobile, and they have some knowledge about the geographical positions of tasks and some other agents. Other work [51] 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.

Also extant work on resource allocation in social networks with self-interested agents in fact uses the network as a model for allowed communication in the sense of negotiation. In this model connected agents are allowed to negotiate about exchanging resources locally in a network, and as often as possible, as long as each exchange is individually rational [3,10,11]. The aim of this line of work is to show convergence to new allocations with properties such as envy-freeness. Tasks as such are not explicitly included, but could in principle be modeled by the utility functions. In our work we allow resources to change hands at most once, and the utility for resources is of a combinatorial nature, i.e., only certain sets of resources can achieve tasks and thus have any value.

Resource allocation in a social network has also been modeled by letting each pair of connected agent solve a bargaining problem [8,9,31]. In this abstract model each bargaining problem can be thought of as sharing a single pie among the two agents. Results show for example existence and computation of equilibria in specific graph structures, given efficiency

(and fairness) criteria such as the Nash bargaining solution. Although this work also takes social relations into account, the model is very different from the task allocation model commonly used in multiagent systems, and results do not seem to transfer. However, applying the game theoretical method used to obtain these results for our setting is a worthwhile direction for future work.

Another important direction for future work is to apply the developed algorithms to a real-world application. For example, a supply chain consists of a set of partially connected agents, and it is a natural application domain for task allocation in networks. Easwaran and Pitt [17] are concerned with the allocation of subtasks to service providers in a supply chain, where "complex tasks" require "services" for their accomplishment. Their algorithm breaks up the problem in two stages: (1) identifying which services need to be executed in order to solve the complex task by breaking the task into subtasks until there are only primitive tasks that can be solved by a combination of service agents, and (2) identifying an optimal set of service providers to provide the services identified in the first stage. For the first stage they use a hierarchical network planning and for the second stage a genetic algorithm. Both algorithms are run centrally. In contrast, Walsh and Wellman [59,60] develop a distributed approach for this problem, using a separate auction for each required resource. These separate auctions have the advantage of being computationally very efficient, but may also lead to managers that end up with insufficient resources to execute a task, while they still need to pay for these resources (the bidder exposure problem). To remedy this, they also include a mechanism for decommitment. This mechanism allows managers to return resources in case—after the auctions are concluded—the acquired set of resources turns out to be insufficient to complete the task. Such an approach works well when this does not happen too often. However, we expect that in situations where resources are scarce (for example with a resource ratio around 0.5 as in our experiments), few managers are able to acquire all resources required for their task when these need to be obtained in separate auctions. For such difficult instances, it may be better to consider the tasks in a greedy order such as in GDAP (if agents are not strategic), or to use an optimal combinatorial approach locally such as the cluster-based mechanism. Having said that, their model of hierarchical subtask decomposition is quite expressive, they provide a neat equilibrium analysis, and the application to the domain of supply chain formation is very inspiring.

10 Conclusions

In this paper we study the task allocation problem in a social network (STAP), which can be seen as a new, more general, variant of the task allocation problem. We believe this problem models many realistic task allocation problems. This paper includes the following contributions regarding this problem. We prove that computing the efficient solution for STAP is NP-complete and we give a bound on possible approximation algorithms. We present a computationally efficient distributed protocol, GDAP. We show that this protocol cannot be used as part of a truthful mechanism in case the agents who control the resources are self-interested, but we provide an alternative, cluster-based algorithm for such situations. Finally, we conduct an extensive set of experiments to assess the solution quality and the computational efficiency of the proposed algorithms in three types of social 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 all three types of networks, and for many different settings. Furthermore, the algorithm scales well to large networks, both in terms of quality and of required computation time. The

results also show that the value of an optimal task allocation in small-world networks where resources are scarce is relatively low compared to the same setting in the other network types, because there are no agents in these small-world networks with many connections that can still allocate tasks that require many resources. However, both the distributed algorithm as well as the cluster-based algorithm with the similarity heuristic perform relatively well in such small-world networks, because these algorithms exploit exactly such locality to reduce the computational burden. Overall as a price of truthfulness, the cluster-based algorithm performs a bit worse than GDAP, except when cluster size is large (i.e., less than five clusters for small-world networks or at most two for the others).

Because the quality of the allocations found by the (truthful) cluster-based algorithm is not as good as the quality of GDAP (except in some small-world networks), we aim in our future work to design an alternative truthful mechanism for networks in which clusters do not already occur naturally. But even for GDAP there may be room for different heuristics, perhaps attaining a better performance in specific settings. In addition, we believe the model can be extended to support use in other realistic situations. For example, in the current model we assume that agents can only contact their neighbors to request resources, but it would be interesting to see neighbors also as mediators in finding resources further away in the social network. This direction could benefit from the results on resource exchanges [3,11] discussed in the previous section. 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. For this we could look into taking some first results on dynamic social networks [4] into account.

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 small-world, scale-free, and uniformly random networks 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.

References

1. Abdallah, S., & Lesser, V. (2005). Modeling task allocation using a decision theoretic model. In *Proceedings of the 4th international conference on autonomous agents and multiagent systems (AAMAS 2005)* (pp. 719–726). New York: ACM.
2. Alon, N., Feige, U., Wigderson, A., & Zuckerman, D. (1995). Derandomized Graph Products. *Computational Complexity, 5*(1), 60–75.
3. Andersson, M. R., & Sandholm, T. W. (1998). Contract types for satisficing task allocation: II experimental results. In *AAAI spring symposium series: Satisficing models*. California: Stanford University
4. Assen, M. A. L. M., & van de Rijt, A. (2007). Dynamic exchange networks. *Social Networks, 29*(2), 266–278.
5. Babanov, A., & Gini, M. (2006). Deciding task schedules for temporal planning via auctions. In *AAAI Spring Symposium*.
6. Barabási, A. L., & Albert, R. (1999). Emergence of scaling in random networks. *Science, 286*(5439), 509–512.
7. Barton, L., & Allan, V. H. (2007). Methods for coalition formation in adaptation-based social networks. In *Cooperative information agents XI, LNAI* (vol. 4676, pp. 285–297). Heidelberg: Springer.

8. Braun, N., & Gautschi, T. (2006). A nash bargaining model for simple exchange networks. *Social Networks, 28*(1), 1–23.
9. Chakraborty, T., Kearns, M., & Khanna, S. (2009). Network bargaining: Algorithms and structural results. In *Proceedings of the tenth ACM conference on electronic commerce* (pp. 159–168).
10. Chevaleyre, Y., Dunne, P. E., Endriss, U., Lang, J., Lemaitre, M., Maudet, N., Padget, J., Phelps, S., Rodríguez-Aguilar, J. A., & Sousa, P. (2006). Issues in multi-agent resource allocation. *Informatica, 30*, 3–31.
11. Chevaleyre, Y., Endriss, U., & Maudet, N. (2007). Allocating goods on a graph to eliminate envy. In *Proceedings of the 22nd national conference on artificial intelligence (AAAI 2007)* (pp. 700–705).
12. Coase, R. H. (1937). The nature of the firm. *Economica NS, 4*(16), 386–405.
13. Coase, R. H. (1995). My evolution as an economist. In W. Breit & R. W. Spencer (Eds.), *Lives of the laureates* (pp. 227–249). Cambridge, MA: MIT Press.
14. Dantzig, G. (1957). Discrete variable problems. *Operations Research, 5*, 266–277.
15. Dias, M. B., Zlot, R. M., Kalra, N., & Stentz, A. T. (2005). *Market-based multirobot coordination: A survey and analysis. Technical report CMU-RI-TR-05-13*. Pittsburgh, PA: Robotics Institute, Carnegie Mellon University.
16. Dunne, P. E., Kraus, S., Manisterski, E., & Wooldridge, M. (2010). Solving coalitional resource games. *Artificial Intelligence, 174*(1), 20–50.
17. Easwaran, A. M., & Pitt, J. (2002). Supply chain formation in open, market-based multi-agent systems. *International Journal of Computational Intelligence and Applications, 2*(3), 349–363.
18. Eberling, M., & Kleine Büning, H. (2010). Self-adaptation strategies to favor cooperation. In *Agent and multi-agent systems: Technologies and Applications, LNAI* (vol. 6070, pp. 223–232). New York: Springer.
19. Ferreira, P., dos Santos, F., Bazzan, A., Epstein, D., & Waskow, S. (2010). Robocup rescue as multiagent task allocation among teams: Experiments with task interdependencies. *Autonomous Agents and Multi-Agent Systems, 20*, 421–443.
20. Foster, I., Jennings, N. R., & Kesselman, C. (2004). Brain meets Brawn: why grid and agents need each other. In *Proceedings of the 3rd international conference on autonomous agents and multiagent systems (AAMAS 2004)* (pp. 8–15). Washington, DC: IEEE Computer Society.
21. Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability—a guide to the theory of NP-completeness*. New York: W.H. Freeman and company.
22. Gaston, M. E., & DesJardins, M. (2005). Agent-organized networks for dynamic team formation. In *Proceedings of the 4th interence confere on autonomous agents and multiagent systems (AAMAS 2005)* (pp. 230–237). New York: ACM Press.
23. Gaston, M. E., & DesJardins, M. (2008). The effect of network structure on dynamic team formation in multi-agent systems. *Computational Intelligence, 24*(2), 122–157.
24. Girvan, M., & Newman, M. E. J. (2002). Community structure in social and biological networks. *Proceedings of the National Academy of Sciences of the United States of America, 99*(12), 7821–7826.
25. Goldberg, D., Cicirello, V., Dias, M. B., Simmons, R., Smith, S., & Stentz, A. T. (2003). Market-based multi-robot planning in a distributed layered architecture. In *Multi-robot systems: From swarms to intelligent automata: Proceedings from the 2003 international workshop on multi-robot systems* (Vol. 2, pp. 27–38). Dordrecht: Kluwer Academic Publishers.
26. Groves, T. (1973). Incentives in teams. *Econometrica, 41*(4), 617–631.
27. Gulati, R. (1995). Does familiarity breed trust? The implications of repeated ties for contractual choice in alliances. *Academy of Management Journal, 38*(1), 85–112.
28. Heydenreich, B., Muller, R., & Uetz, M. (2007). Games and mechanism design in machine scheduling—an introduction. *Production and Operations Management, 16*(4), 437–454.
29. Hirayama, K., Yokoo, M., & Sycara, K. (2004). An easy-hard-easy cost profile in distributed constraint satisfaction. *Transactions of the Information Processing Society of Japan, 45*(9), 2217–2225.
30. Kafalı, Ö., & Yolum, P. (2008). Improving self-organized resource allocation with effective communication. In *Seventh international workshop on agents and peer-to-peer computing, AAMAS* (pp. 7–18).
31. Kleinberg, J., & Tardos, E. (2008). Balanced outcomes in social exchange networks. In *Proceedings of the 40th annual ACM symposium on theory of computing* (pp. 295–304).
32. Klos, T., & Nooteboom, B. (2001). Agent-based computational transaction cost economics. *Economic Dynamics and Control, 25*(3–4), 503–526.
33. Koutsoupias, E. (2003). Selfish task allocation. *Bulletin of EATCS, 81*, 79–88.
34. Kraus, S., Shehory, O., & Taase, G. (2003). Coalition formation with uncertain heterogeneous information. In *Proceedings of the 2nd international conference on autonomous agents and multiagent systems (AAMAS 2003)* (pp. 1–8). New York: ACM.

35. Kraus, S., Shehory, O., & Taase, G. (2004). The advantages of compromising in coalition formation with incomplete information. In *Proceedings of the 3rd international conference on autonomous agents and multiagent systems (AAMAS 2004)* (pp. 588–595).
36. van der Krogt, R., de Weerdt, M., & Zhang, Y. (2008). Of mechanism design and multiagent planning. In *ECAI* (pp. 423–427). Amsterdam: IOS Press.
37. Lerman, K., & Shehory, O. (2000). Coalition formation for large-scale electronic markets. In *Proceedings of 4th international conference on multi-agent systems (ICMAS 2000)* (pp. 167–174). Boston, MA: IEEE Computer Society.
38. Makhorin, A. (2004). GLPK. GNU linear programming kit http://www.gnu.org/software/glpk/.
39. Manisterski, E., David, E., Kraus, S., & Jennings, N. (2006). Forming efficient agent groups for completing complex tasks. In H. Nakashima, M. P. Wellman, G. Weiss& P. Stone (Eds.),*Proceedings of the 5th international conference on autonomous agents and multiagent systems (AAMAS 2006)* (pp. 257–264). Hakodate: ACM.
40. Mitchell, D. G., Selman, B., & Levesque, H. J. (1992). Hard and easy distributions of SAT problems. In *Proceedings of the national conference on artificial intelligence (AAAI 1992)* (pp. 459–465).
41. Myerson, R. (1979). Incentive-compatibility and the bargaining problem. *Econometrica, 47*, 61–73.
42. Myerson, R. B., & Satterthwaite, M. A. (1983). Efficient mechanisms for bilateral trading. *Journal of Economic Theory, 29*(2), 265–281.
43. Nair, R., Ito, T., Tambe & Marsella, S. (2002). Task allocation in the robocup rescue simulation domain: A short note. In A. Birk, S. Coradeschi & S. Tadokoro (Eds.), RoboCup 2001: Robot Soccer world cup V, lecture notes in computer science (Vol.2377 , pp. 1–22). Springer Berlin, Heidelberg.
44. Niedermeier, R. (2006). *Invitation to fixed-parameter algorithms, Oxford lecture series in mathematics*. Oxford: Oxford University Press.
45. Nisan, N. (2007). Introduction to mechanism design (for computer scientists). In N. Nisan, T. Roughgarden, E. Tardos, & V. Vazirani (Eds.), *Algorithmic game theory* (pp. 209–242). Cambridge: Cambridge University Press.
46. Nisan, N., & Ronen, A. (2000). Computationally feasible VCG mechanisms. In *Proceedings of the 2nd ACM conference on electronic commerce* (pp. 242–252). New York: ACM.
47. Nisan, N., & Ronen, A. (2001). Algorithmic mechanism design. *Games and Economic Behavior, 35*(1–2), 166–196.
48. Patel, J., Teacy, W. L., Jennings, N. R., Luck, M., Chalmers, S., Oren, N., Norman, T. J., Preece, A., Gray, P. M., Shercliff, G., Stockreisser, P. J., Shao, J., Gray, W. A., Fiddian, N. J., & Thompson, S. (2005). Agent-based virtual organizations for the grid. *Multi-Agent and Grid Systems, 1*(4), 237–249.
49. Rothkopf, M. H., Pekeč, A., & Harstad, R. M. (1998). Computationally manageable combinational auctions. *Management Science, 44*, 1131–1147.
50. Sander, P. V., Peleshchuk, D., & Grosz, B. J. (2002). A scalable, distributed algorithm for efficient task allocation. In *Proceedings of the 1st international conference on autonomous agents and multiagent systems (AAMAS 2002)* (pp. 1191–1198). New York: ACM Press.
51. Shehory, O. (2000). A scalable agent location mechanism. In N. R. Jennings & Y. Lespérance (Eds.), *Proceedings of intelligent agents VI, agent theories, architectures, and languages (ATAL), LNCS* (Vol. 1757, pp. 162–172). Heidelberg: Springer
52. Shehory, O., & Kraus, S. (1998). Methods for task allocation via agent coalition formation. *Artificial Intelligence, 101*(1–2), 165–200.
53. Shehory, O., & Kraus, S. (1999). Feasible formation of coalitions among autonomous agents in nonsuperadditive environments. *Computational Intelligence, 15*(3), 218–251.
54. Shoham, Y., & Leyton-Brown, K. (2008). *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge: Cambridge University Press.
55. Smith, R. G. (1981). The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers, C-29*(12), 1104–1113.
56. Sreenath, R. M., & Singh, M. P. (2004). Agent-based service selection. *Web Semantics, 1*(3), 261–279.
57. Vazirani, V. V. (2001). *Approximation algorithms*. New York: Springer-Verlag New York Inc.
58. Vidal, J. M. (2003). A method for solving distributed service allocation problems. *Web Intelligence and Agent Systems, 1*(2), 139–146.
59. Walsh, W. E., & Wellman, M. P. (1999). Efficiency and equilibrium in task allocation economies with hierarchical dependencies. In *International joint conference on artificial intelligence (IJCAI)* (pp. 520–526)
60. Walsh, W. E., & Wellman, M. P. (2000). Modeling supply chain formation in multiagent systems. In *Agent-mediated electronic commerce II, LNAI* (Vol. 1788, pp. 94–101). Heidelberg: Springer.
61. Watts, D. J., & Strogatz, S. H. (1998). Collective dynamics of "small world" networks. *Nature, 393*, 440–442.

62. de Weerdt, M., Zhang, Y., & Klos, T. B. (2007). Distributed task allocation in social networks. In *Proceedings of the 6th international conference on autonomous agents and multiagent systems (AAMAS 2007)* (pp. 17–24). New York: ACM.