Methods for task allocation via agent coalition formation ⋆

Onn Shehory

The Robotics Institute, Carnegie-Mellon University,
5000 Forbes Ave, Pittsburgh, PA 15213, USA

Sarit Kraus

Dept. of Math and Computer Science, Bar Ilan University,
Ramat Gan, 52900 Israel,
and
Institute for Advanced Computer Studies, University of Maryland,
College Park, MD 20742, USA

Abstract

Task execution in multi-agent environments may require cooperation among agents. Given a set of agents and a set of tasks which they have to satisfy, we consider situations where each task should be attached to a group of agents that will perform the task. Task allocation to groups of agents is necessary when tasks cannot be performed by a single agent. However it may also be beneficial when groups perform more efficiently with respect to the single agents' performance. In this paper we present several solutions to the problem of task allocation among autonomous agents, and suggest that the agents form coalitions in order to perform tasks or improve the efficiency of their performance. We present efficient distributed algorithms with low ratio bounds and with low computational complexities. These properties are proven theoretically and supported by simulations and an implementation in an agent system. Our methods are based on both the algorithmic aspects of combinatorics and approximation algorithms for NP-hard problems. We first present an approach to agent coalition formation where each agent must be a member of only one coalition. Next, we present the domain of overlapping coalitions. We proceed with a discussion of the domain where tasks may have a precedence order. Finally, we discuss the case of implementation in an open, dynamic agent system. For each case we provide an algorithm that will lead agents to the formation of coalitions, where each coalition is assigned a task. Our algorithms are any-time algorithms, they are simple, efficient and easy to implement.

⋆ This material is based upon work supported in part by the NSF under grant

1 Introduction

Autonomous agents in multi-agent environments may need to cooperate in order to fulfill tasks. Given a set of tasks to be satisfied, we consider situations where each task is assigned a group of agents to perform it. We address cases in which dependencies among tasks, if such exist, are due to competing resources' requirements or execution precedence order. The allocation of tasks to groups of agents is necessary when tasks cannot be performed by single agents or when single agents perform them inefficiently. Various groups of agents may have different degrees of efficiency in task performance due to differing capabilities of their members. Task allocation should be done with respect to these differences.

The purpose of the allocation of tasks to groups of agents is to maximize benefits via their performance. We seek an algorithm that will enable a distributed task allocation, i.e., without a central authority. A low computational complexity shall be considered an important property for such a solution. Hence, in this paper we present algorithms that enable the agents to form groups and assign a task to each group. We call these groups *coalitions*. For the development of the required algorithms, we combine a combinatorial algorithmic approach and concepts from operations research, with autonomous agents' methods and distributed computing systems methods. The coalitions the agents form when using these algorithms are beneficial for systems of cooperative agents, as we show in this paper. We will concentrate on coalition formation in environments which are not necessarily super-additive [1] [11,21].

Distributed artificial intelligence (DAI) is concerned with problem solving in which several agents interact in order to execute tasks. During the past few years, several solutions to the coalition formation problem have been suggested by researchers in the field of DAI. These solutions concentrate on the special case of autonomous agents in a super-additive environment [28,44,64]. Most of these solutions are given for coalition formation in Multi-Agent Systems (MAS), where each agent tries to increase its own personal utility via cooperation. One of the main problems of coalition formation in the case of MAS is how to distribute the common outcome of a coalition among its members. We present coalition formation algorithms which are appropriate for Distributed Problem Solving [2] (DPS) [6] cases where agents cooperate in order to increase the overall outcome of the system [45] and are not concerned with their per-

No. IRI-9423967 and Army Research Lab under contract No. DAAL0197K0135. Preliminary results of this research were published in the proceedings of IJCAI-95 and ICMAS-96.

[1] In a super-additive environment any combination of two groups of agents into a new group is beneficial. For details see section 3.

[2] Recently, DPS agent systems are referred to as cooperative MAS [52].

sonal payoffs as they are in MAS. In such cases, the disbursements to the agents are not as important as they are in MAS. In addition, our solution is not restricted to the super-additive environment. Since in the case of a super-additive environment the grand-coalition is expected [3], a coalition formation process for DPS systems in super-additive environments shall be very simple. Conversely, the non-super-additive environment case is much more challenging and may be very realistic. In cases where the addition of every new agent to the coalition is costly [4], as the size of the coalition increases, it may becomes non-beneficial to form it. We suggest a solution which is most appropriate for such non-super-additive cases.

We begin by illustrating the problem we intend to solve (section 2). Then, we give a brief description of the environment with which we deal in section 3. We also briefly present the basic definitions and assumptions in section 3.1 and the set-covering and set-partitioning problems in 3.2. The algorithms are described in sections 4 and 5, their ratio-bounds are discussed in section 6, and their complexity is analyzed in section 7. Section 8 contains the simulation results and their analysis, and in section 9 we discuss the requirements and properties of an implementation in an open MAS. Related research is referred to in section 10, and section 11 ends our paper with a discussion and conclusions.

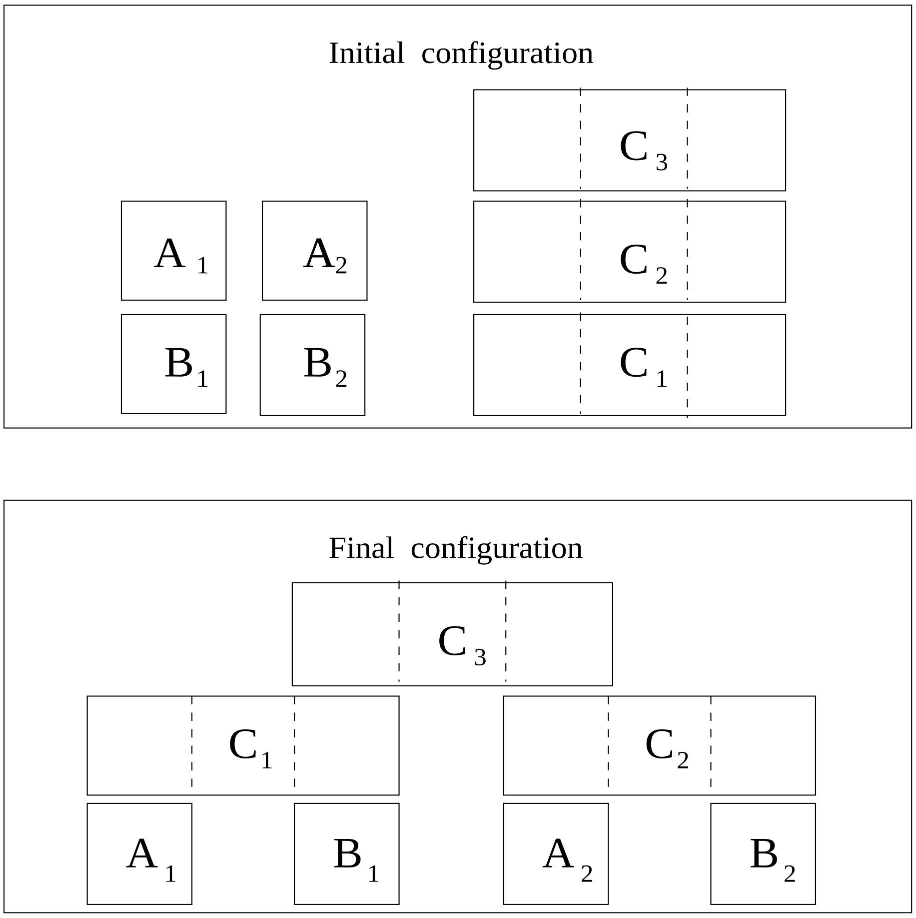
2 Illustration of the problem

The problem we solve in this paper is that of task allocation among groups of autonomous agents in a DPS system. Given a set of tasks, the system as a whole must seek a maximization of its benefits by satisfying tasks [5]. We consider cases where tasks may have a precedence order. We assume no central authority that distributes the tasks among the agents. Therefore, they shall reach an efficient task allocation by themselves, seeking a maximal outcome. This is achieved via the formation of coalitions, possibly overlapping ones.

[3] A grand coalition is a coalition that includes all of the agents [43].

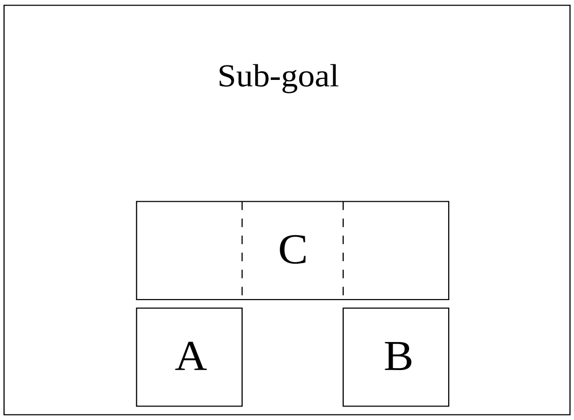
[4] Such costs may arise from the intra-coalition coordination and communication costs; these increase with the size of the coalition.

[5] Ideally, all of the task should be satisfied, however when attempting to maximize benefits, the execution of some tasks may be hindered.



We demonstrate the problem using a Blocks World domain as in the figure above. The blocks are of various sizes (for simplicity, we consider the case of a unit block and a row of attached unit-blocks). Each unit-block weighs one weight-unit. The blocks should be moved from the initial configuration to the final configuration. This should be done by a group of agents, each capable of lifting a limited weight (e.g., 2 weight-units) and moving it aside as much as necessary. Each block can be carried by a limited number of agents (due to physical limitations). We do not discuss the planning problem but assume a previously provided plan. This plan shall divide the goal into subgoals with a precedence order. Such subgoals, which are part of the global plan, are presented in the figure below. In such a subgoal, one cannot place block C before blocks A and B are properly located. In addition, cooperation among agents is necessary. Block C cannot be lifted by one agent. However, 4 agents can do so but less efficiently, due to coordination costs, and 20 of them will be far too many. Obviously, any member of the group that places C may be a member of a group that places A or B. However, if the agents have a limited amount of fuel, they may cease functioning after performing a set of tasks, and therefore they cannot be part of all working groups.

The global goal may be partitioned into several subgoals. For instance, it may be partition to three subgoals: subgoal I – locate blocks A_1, B_1, C_1; subgoal II – locate blocks A_2, B_2, C_2; subgoal III – locate block C_3. Each of these subgoals can be performed by a coalition of agents, and subgoal III can be performed only after subgoals I and II have been performed. Members of the coalitions that perform either subgoal I (C_I) or subgoal II (C_{II}) may be members of the coalition that performs subgoal III. It may also be possible that members of C_I be members of C_{II} and vice versa. However, to increase the efficiency of the performance of subgoals I and II, C_I and C_{II} should not have any common members. This will enable the performance of subgoals I and II simultaneously, thus saving time and avoiding an intra-coalitional coordination overhead that arises from larger coalitions.



Another example of our problem may be a transportation company[6]. The company supplies transportation services via a system of autonomous and automated trucks, lift trucks, cranes, boats and planes which we refer to as agents. This system is usually arranged in a distributed manner, since every single agent may perform limited tasks by itself. The agents differ in their capabilities. That is, they differ in the type of actions that they can perform, in the size, volume and weight of goods that they can carry at one time, in the transportation speed, its costs and the method by which it is performed. There may be occasions where agents cannot perform a given transportation task by themselves. In such cases, cooperation is necessary. Therefore, the agents shall form groups, and each group of agents will cooperatively fulfill a transportation task. We call such cooperating groups *coalitions*. Some coalitions may be unable to perform some tasks. Some tasks cannot be performed unless other tasks have been satisfied prior to their performance. Among the coalitions that are able to perform a given task, the efficiency and the costs may be completely different.

[6] Transportation systems have been extensively used as examples for DPS systems (e.g., [41]). We do not intend to solve a specific transportation problem. However, we provide this example because the reader may be familiar with it.

For example, suppose that a task of taking 10 passengers from Mirabel airport to Dorval airport (both in Montréal) has been ordered. This may be performed by several private cars or by a single helicopter. However, using a helicopter in such a case will probably cost much more than using private cars and take approximately the same time (due to flight constraints and regulations). Therefore, the most appropriate coalition in such a case is a coalition of private cars. This coalition shall be such that it consists of the optimal number of cars as per number of passengers. An excess of agents in the coalition may be costly either for the coalition members or for the system as a whole, or both. This is due to the communication, coordination and internal organization costs, both for the formation of the coalition and for its maintenance. These costs are an increasing function of the size of the coalition. In the case of disjoint coalitions, an overhead of agents in a coalition may prevent the formation of other beneficial coalitions and may therefore reduce the total outcome of the system.

The transportation company may have many agents, and therefore a central authority for coalition formation may be too costly in time and computational efforts. Using a distributed task allocation mechanism rather than a centralized one may be advantageous (as discussed in section 7). In such cases task allocation and cooperation shall be decided upon locally. However, such a company seeks the maximization of its benefits. Therefore, the company shall attempt to satisfy beneficial transportation orders, mainly to bring an immediate profit, but also to satisfy clients (which is important for the company's future). Since a single agent cannot always satisfy a client's order, close cooperation is necessary. For such cases, the company shall provide the agents with a simple but also efficient algorithm that will enable the formation of coalitions of agents.

The problem of the transportation company is generalized in this paper. We provide algorithms which enable the allocation of tasks among a system of agents via the formation of coalitions. We show that the algorithms are simple to implement, have a short run-time (hence can be used as a real-time method), and yield results which are close to the optimal results.

3 Environment description

In order to elucidate the problem and its solution, we briefly present some general notations and definitions of concepts. We assume that agents can communicate, negotiate and make agreements [59]. Communications require time and effort on the part of the agents. We also assume that resources can be transferred between agents. This ability may help the agents form more beneficial coalitions. Agreement on cooperation may be reached even if the last assumption is not valid (e.g., [1,2,29,62]). However, the possibility of goods transferability (or alternatively, side-payments) may help the agents form more beneficial coalitions. To emphasize the non-super-additivity property of the environment with which we deal, we assume that the addition of agents to a coalition is costly, and therefore expanding coalitions may be non-beneficial. Without this assumption, the grand coalition, as a coalition that consists of all of the formed coalitions within it, would be the most beneficial.

3.1 Definitions

We recall the following definitions as presented in [45,46]. There is a set of n agents, $N = \{A_1, A_2, \ldots, A_n\}$. Each agent A_i has a vector of real non-negative capabilities $B_i = \langle b_1^i, \ldots, b_r^i \rangle$. Each capability is a property of an agent that quantifies its ability to perform a specific action. These capabilities are either expendable or non-expendable (e.g., the carrying capability in the Blocks world). The expendability of the capabilities affects the overall task-execution of the agent-system. Recalling the transportation company, the volume and weight that can be transported by an agent is its transportation-volume capability. In order to enable the assessment of coalitions and of task-execution, an evaluation function shall be attached to each type of capability. Such a function shall transform the capability units into monetary units. In the transportation case, this function may be the income from performing a task. This may be a linear function of the number of passengers. We also assume that there is a set of m independent tasks [7] $T = \{t_1, t_2, \ldots, t_m\}$. For the satisfaction of each task t_l, a vector of capabilities $B_l = \langle b_1^l, \ldots, b_r^l \rangle$ is necessary. The utility gained from performing the task depends on the capabilities that are required for its performance. In our solution we assume, to simplify the calculations, that the utility gained is a linear function of the resource amount.

There may be occasions where a specific task t_j cannot be performed unless another specific task t_i has already been satisfied. This is generalized by a partial precedence order between the tasks, $t_{1_1} \preceq t_{1_2} \preceq \ldots \preceq t_{1_{n_1}}, \ldots, t_{k_1} \preceq t_{k_2} \preceq \ldots \preceq t_{k_{n_k}}$, where $t_i \preceq t_j$ means that t_i is the predecessor of t_j and t_j is the successor of t_i in the performance order. The precedence order and the resource consumption are the only dependencies that we assume. This restricts our solution to cases where no other explicit dependencies exist (this

[7] The partition of a single task into subtasks is beyond the scope of this paper. However, if these subtasks are independent, then the algorithms that we suggest can be used recursively to allocate sub-groups of agents to the subtasks. In cases where there are interdependencies (which are not precedence dependencies) among tasks, we suggest to combine dependent tasks into unified tasks. The recognition of such dependencies can be performed by implementing techniques from constraint satisfaction research (see [35,37]).

is appropriate for various domains, including the Blocks world and several transportation domains).

A coalition can be defined as a group of agents who have decided to cooperate in order to achieve a common task. We assume that a coalition can work on a single task at a time, and that each agent may sometimes be a member of more than one coalition. The latter assumption can increase an agent's ability to use its resources for task execution. A coalition C has a vector of capabilities B_c which is the sum of the capabilities that the coalition members contribute to this specific coalition. Note that in the case of overlapping coalitions this sum is not the sum of all of the capabilities of the members, because the agents may be members of more than one coalition, and can contribute part of their capabilities to one coalition and part of them to another. A coalition C can perform a task t only if the vector of capabilities necessary for its fulfillment B_t satisfies $\forall\ 0 \leq i \leq r\ , b_i^t \leq b_i^C$ and, in the case of precedence order, t has no unsatisfied predecessors. For each coalition C a value V can be calculated [8] which is the joint utility that the members of C can reach by cooperating via coalitional activity for satisfying a specific task [9]. The coalitional value V is directly affected by the capabilities that the members of the coalition contribute to it, the precedence order of the tasks and the number of coalition members. In the case of overlapping coalitions, a method is required according to which the agents decide how to partition their capabilities between coalitions in which they are members. This method is provided in the corresponding coalition formation algorithm, in section 4. Recall the transportation company: given a specific transportation task, if the sum of transportation-volumes of a coalition is less than the volume necessary for the task, then the value of the coalition for this specific task is zero. If the coalitional transportation volume is much greater than necessary for satisfying the task, then the value is positive, but relatively small compared to the case of having the exact volume. Actually, the transportation-volume is not the only capability of the agents and therefore does not affect the coalitional value exclusively.

To conform with a common representation, we may employ the notion of coalitional cost c instead of coalitional value. This cost may be calculated as the reciprocal of the coalitional value or as its negation. Such a calculation attaches a low cost to a high-valued coalition and vice versa. The coalitional rationality (as described below), which leads agents to try to increase the

[8] The calculation of coalitional values is, in the general case, rather complex. Nevertheless, low complexity approximation methods for calculating values of agreements among multiple agents have been recently developed [26], and these can be utilized for our case.

[9] This notion of coalitional value is different from the notion commonly used for coalitional values in game theory, since here the value depends on the coalitional configuration and on the task allocation.

coalitional value, likewise leads them to try to reduce the coalitional cost.

We assume that the agents are group-rational. That is, they form a coalition only if they benefit as a coalition at least as much as the sum of their personal benefits outside of it [22,33,39]. The agents benefit if they fulfill tasks. Group rationality is necessary to ensure that whenever agents form a coalition, they increase the system's common outcome, which is the sum of the coalitional outcomes. We also assume that each agent tries to maximize the common utility; among all of the possibilities that an agent has, it will choose the one that will lead to the maximum common utility. However, group rationality of the agents does not necessarily entail a super-additive environment. In order to emphasize the difference between the super-additive environment and the ones with which we deal, we describe the super-additive environment below.

A super-additive environment is such that the set of the possible coalitions satisfies the following rule: for each pair of coalitions C_1,C_2 in the set, $C_1 \cap C_2 = \emptyset$, if C_1,C_2 join together, then the newly formed coalition will have a new value $V_C^{new} \geq V_C^1 + V_C^2$, where V_C^1,V_C^2,V_C^{new} are the values of the coalitions C_1,C_2 and C_{new}, respectively [10]. Rational agents in a super-additive environment will prefer the grand coalition over all other coalitions. Hence, in a super-additive environment, when the grand coalition is formed, the only problem that remains is the utility distribution among its members. This is a major problem in MAS, however, in DPS systems such a problem is of lesser importance or may not exist at all.

More assumptions are as follows: we assume that the agent-population does not change during the coalition formation process. We *do not* assume complete information. That is, all of the agents must know about all of the goals and the existence of other agents, however only coalition members must know all of the details required for satisfying a specific goal. Note that for implementation in a dynamic agent system, the assumptions of a fixed agent population and full knowledge about goal and agent existence are later relaxed (in section 9). Additional assumption are as follows: agents outside a coalition C need not know the details of the intra-coalitional activity within C. Our algorithm should work regardless of clock synchronization among the agents [11]. That is, in order to decide which agent will perform what action, the agents do not have to know precisely when other agents will act. Nevertheless, significant time differences between the agents' clocks may reduce the efficiency of the overall goal satisfaction.

---

[10] Note that in our case there may be coalitions C_1,C_2 and C_{new} such that $V_C^{new} < V_C^1 + V_C^2$, however the group rationality prohibits the formation of such C_{new}s.

[11] This assumption is only with regards to the requirements of our algorithm. The properties of the tasks of the multi-agent system may require such a synchronization, independently of our algorithm requirements.

Now that we have made the necessary assumptions and definitions, we can formally present the problem. Given a set of m tasks $T = \{t_1, \ldots, t_m\}$, with an (optional) precedence order and a set of n agents $N = \{A_1, \ldots, A_n\}$ with their capabilities, the problem we solve is of assigning tasks $t_i \in T$ to coalitions of agents $C_i \subseteq N$ such that $\sum_i V_i$ (the total outcome) is maximal and the precedence order is respected as well. Note that while we initially refer to T and N as fixed, we later allow them to change dynamically. We solve the problem for the general case of non-super-additive environments. However, even in the case of a super-additive environment, the solution will consist of coalitions C_i that do not have null [12] members. This is because the membership of null members in a coalition will probably increase the coalitional costs thus reducing the overall outcome.

3.2 Set Covering and Set Partitioning

Since task allocation among agents may be approached as a problem of assigning groups of agents to tasks, the partition of the agents into subgroups becomes the main issue. Therefore, the task allocation problem becomes similar to the set partitioning and the set covering problems.

Below we formulate the two problems: SCP and SPP. Given a set $N = \{A_1, \ldots, A_n\}$ and a set of subsets of N, $S = \{C_1, \ldots, C_m\}$, such that $C_j \subseteq N$ and $S \subseteq 2^N$; a set-cover is any $S' \subseteq S$, such that $\bigcup_{C_j \in S'} C_j = N$. The members of S' are the covering sets. If the members of S' are also pairwise disjoint (i.e., $\forall\ C_i, C_j \in S',\ i \neq j \quad C_i \cap C_j = \emptyset$), then S' is a set-partitioning of N. We assume that each $C_j \in S$ has a positive cost c_j. The cost of a cover S' is $\sum_{C_j \in S'} c_j$. The set covering problem entails finding the cover with the minimum cost, and the set partitioning problem is defined correspondingly [3,4,20].

The set covering problem is NP-complete [12]. Therefore, the optimal solution implies an exponential computational complexity, which is too high for practical use. However, a variety of algorithms for solving this problem sub-optimally have been suggested, e.g. in [4,8,9]. Among them we can find the algorithm of Chvatal [9], which has a logarithmic ratio bound [13] . Given its low ratio bound, it is very tempting to adopt and adapt this algorithm for the case of multi-agent coalition formation. This cannot be done for the case of disjoint coalitions, since the algorithm provides a solution to the set covering problem, in which subgroups may overlap. Furthermore, even when overlapping coalitions are allowed other difficulties are still present. These include the

[12] Null members are agents that contribute nothing to the coalitional performance.
[13] An approximation algorithm for a problem has a ratio bound $\rho(n)$ if $\rho(n)$ is smaller than the ratio between the optimal cost and the approximated cost.

following: the set covering problem deals only with a small given set of subsets, and in the case of agents, the number of possible coalitions is 2^n (hence, we need heuristics for reducing this number); the algorithms for SCP and SPP are centralized and, since we deal with autonomous, distributed agents, we seek a distributed algorithm; the SCP and SPP algorithms do not refer to cases where there is a precedence order between the chosen subgroups. Despite the deficiencies indicated above, we attempt to borrow some of the properties of the Chvatal's algorithm, thus constructing coalition formation algorithms with a low ratio bound.

4 The algorithms

The algorithms we present below are greedy distributed set-partitioning and set-covering algorithms with low ratio bounds. They were designed for the special case of autonomous agents in an environment which is not necessarily super-additive, and that work as a DPS system (i.e., they try to act in order to increase the performance and benefits of the group as a whole). The algorithms are any-time algorithms, that is, if the execution is stopped before an algorithm would have normally terminated, it still provides the agents with a solution which is better than their initial state or other preceding states.

The algorithms consist of two main stages (presented later in detail):

(i) In the preliminary stage of each algorithm, all possible coalitions are distributively calculated and their initial values are computed.
(ii) The main stage of the algorithms consists of an iterative distributed greedy procedure in which two sub-stages occur:
   - The coalitional values are re-calculated. This is done because an efficient coalition formation algorithm requires up-to-date information about the values of the possible coalitions, to enable the choice of the preferred ones.
   - The agents decide upon the preferred coalitions and form them.

As previously stated, the solution of the SPP and the SCP in the case of autonomous agents are of an exponential complexity, since the number of the possible coalitions is exponential (2^n). A reduction in this number is possible by limitations on the permitted coalitions. This can be done via the constraints of the specific problem under investigation, e.g., it may happen that all of the tasks must be performed by the same number of agents. In case no specific limitations accrue from the properties of the specific problem, we recommend preferring small-sized coalitions. We justify such heuristics by the associated cost-estimation: since communication and computation-time are costly, and the agents seek cost-reduction, they should try to avoid unnecessary communication and computation activities. In this sense, small-sized coalitions are more economical to design than larger coalitions and therefore shall be preferred [14]. This is the case since the calculations and communication operations are exponentially dependent on the numbers of members in a coalition. These heuristics will be implemented in our algorithms by presenting an integer k which will denote the highest coalitional size allowed. This restriction will limit the number of coalitions to $O(n^k)$, which is a polynomial number in n. However, this does not trivialize the problem since even with such restrictions, the problem remains NP-complete [15]. In the transportation case, a limitation on the size of coalitions may be even more reasonable than in the general case. Here, it would be very convenient to assume that the volume of a single task never exceeds a given size. Such a restriction affects the number of coalitions in the same way as in the case of communication and computation restrictions. Additional information about the properties of the tasks and the coalitional values may enable the calculation of the expectation values of the outcome of different coalitions. This shall improve the heuristics we employ, thus reducing the number of coalitions and the complexity of the algorithm.

The initial coalitional state consists of n, single agents. The agents then begin negotiating [31] and, step by step, form coalitions. In the case of disjoint coalitions, agents that join coalitions quit the coalition-formation process, and only the remaining single agents will continue negotiations. The reduction in the number of agents that continue negotiating reduces the computational and communication costs. In the case of overlapping coalitions, agents that join coalitions do not quit the coalition-formation process unless their resources are depleted. However, the overlapping property increases the efficiency of task execution and thus the benefits of the system.

4.1 Preliminary agreement on calculation-distribution

Prior to forming the coalitions, the agents must agree on the distribution of value calculations among themselves. In order to achieve this distribution, each agent A_i should perform the following steps:

[14] Note that this seems to contradict the economies of scale principle. However, our coalitions are created for performing a single task, and they afterwards decompose, whereas the principle of economies of scale refer to long-lasting organizations which perform multiple tasks over time. Hence we find this concept inapplicable to our case. There may be, however, cases in which only large coalitions can satisfy the tasks. In such cases different heuristics must be adopted.

[15] This is because the number of combinations of coalitions may still be exponential. This is similar to the NP completeness of SPP and SCP which, too, refer to a small number of subsets.

(i) Calculate all of the permutations that include up to k agents including A_i and put in $\mathcal{P}_i$, the set of the potential coalitions of agent A_i.
(ii) While $\mathcal{P}_i$ is not empty, do:
   - Contact an agent A_j that is a member of a potential coalition in $\mathcal{P}_i$.
   - If this is the first contact with A_j, locate information about its capabilities (i.e., retrieve B_j).
   - Commit to the calculation of the values of a subset [16] S_{ij} of the common potential coalitions (i.e., a subset of the coalitions in $\mathcal{P}_i$ in which both A_i and A_j are members).
   - Subtract S_{ij} from $\mathcal{P}_i$. Add S_{ij} to your long-term commitment list L_i.
   - For each agent A_k that has contacted you, subtract from $\mathcal{P}_i$ the set S_{ki} of the potential coalitions it had committed to compute values for.
   - Compute values for the coalitions you have committed to [17] (S_{ij}), as detailed below (section 4.2, (i)-(iv)). If some capabilities are unknown, contact the relevant agents [18].
   - Repeat contacting other agents until $\mathcal{P}_i = \{A_i\}$ (i.e., no more agents to contact).

After the preliminary stage, each agent A_i has a list L_i of potential coalitions for which it had committed to repeatedly calculate the values, and their preliminary values. In addition, A_i has all of the necessary information about the capabilities of the members of these coalitions, and it updates the information if necessary [19].

4.2 Distributed, repeated calculation of coalitional values

The coalition formation algorithms require repeated calculation of the coalitional values. As opposed to the common approach to coalitional values, the values in our model vary continuously. Therefore, the calculations should be repeated iteratively. However, the calculation in each iteration follows the

[16] The size of this subset may vary from a single coalition to all of the common coalitions. The agents should make this decision with respect to their relative computation speed.
[17] Note that since agents may have different computation capabilities some will finish faster than others. These will re-enter the loop and commit to another subset of potential coalitions. By this, the computations will be distributed evenly with respect to computing speed.
[18] Instead, as done in the implementation (section 9), agents can query a matchmaker for this information.
[19] Note that this does not require that each agent know all of the capabilities of all the others because they do not necessarily compute coalitions where these agents are members. The necessity of agents' knowledge about others is further relaxed in the implementation in an open system, section 9.

same procedure, as described below. Note that this procedure is appropriate for either the overlapping or the non-overlapping cases. For the case of tasks with precedence order, some modifications will be necessary. The basic procedure for calculating the coalitional values is presented below. In addition to L_i, each agent A_i should maintain a list of the coalitions for which it should currently compute values, denoted L_i^{cr}, initially set [20] to $\emptyset$. Given these lists, agent A_i should perform the following steps:
Loop and for each coalition C on list L_i^{cr}, perform:

(i) Calculate the coalitional potential capabilities vector B_c^{pc}, by summing up the unused capabilities of the members of the coalition [21]. Formally, $B_c^{pc} = \sum_{A_i \in C} B_i$.
(ii) Form a list E_c of the expected outcomes of the tasks in T when coalition C performs them. For each task $t_j \in T$, perform:
  - Check what capabilities B_j are necessary for the satisfaction of t_j.
  - Compare B_j to the sum of the unused capabilities of the members of the coalition B_c^{pc}, thus finding the tasks that can be satisfied by coalition C.
  - If $\forall i, b_j^i \in B_j \leq b_{pc}^i \in B_c^{pc}$, (that is, t_j can be satisfied by C), calculate t_j's expected net outcome e_j with respect to $|C|$. This shall be done by calculating the gross benefit acquired from the task execution. The latter is the sum of the market-values of the capabilities necessary for the execution of t_j as expressed in B_j. From the gross benefit, the sum of capabilities costs and internal coordination costs must be subtracted. This will be the expected net outcome e_j of task t_j when coalition C performs it. Put e_j in E_c.
(iii) Among the expected outcomes on list E_c, choose the maximal. This will be the coalitional value V_c.
(iv) Calculate the coalitional cost which is $c_c = \frac{1}{V_c}$.

In cases where computation is the bottleneck of the agent-system, additional action on the part of A_i may be necessary. If A_i has finished computing values for the coalitions in L_i^{cr} but another agent, for instance A_j, has not finished its computations, A_i should contact A_j and commit to the computation of the values of a subset S_{ij}^{cr} of L_j^{cr}. Then, A_i should add S_{ij}^{cr} to L_i^{cr} and repeat the iterative process above.

Having calculated the coalitional values and costs, the agents can continue to the proceeding steps of the coalition formation iteration.

[20] L_i^{cr} is initially $\emptyset$ because all of the values were calculated in the preliminary stage.
[21] Note that in the case of overlapping coalitions, this sum is not B_c, the coalitional vector of capabilities.

4.3 Choosing coalitions

In this sub-stage, in each iteration of the algorithms, the agents decide step-by-step which coalition should be preferred and formed, and the coalitional configuration is gradually achieved. At the end of the coalition-value calculation sub-stage, each agent A_i will have a list L_i of coalitions and their values and costs which it had calculated. In this sub-stage, the agents shall form coalitions. Each agent A_i shall iteratively perform a sequence of steps. The overlapping coalitions' case will require a sequence different from the disjoint coalitions' case.

4.3.1 Disjoint coalitions

In order to simplify the representation of the algorithm, we denote the ratio between the cost of the coalition and the coalition's size by $w_i = c_i / \mid C_i \mid$ and call it the coalitional weight. At the end of the first sub-stage of each iteration of the algorithm, each agent will have calculated a list of coalitions and coalitional values and weights.

Each agent A_i shall iteratively perform the following:

(i) Locate in L_i the coalition C_j with the smallest w_j.
(ii) Announce the coalitional weight w_j that it has located.
(iii) Choose the lowest among all of the announced coalitional weights. This w_{low} will be chosen by all agents. Choose the corresponding coalition C_{low} and task t_{low} as well.
(iv) Delete the members of the chosen coalition C_{low} from the list of candidates for new coalitions.
(v) If you are a member of the chosen coalition C_{low}, join the other members and form C_{low}.
(vi) Delete from L_i the possible coalitions that include deleted agents.
(vii) Delete from T the chosen task t_{low}.
(viii) Assign to L_i^{cr} the coalitions in L_i for which values should be re-calculated (see details below).

The above procedures of calculating coalitional values and choosing the preferred coalitions will be repeated until all agents are deleted (that is, until all are assigned to coalitions), or until there are no more tasks to be allocated, or none of the possible coalitions is beneficial. Some of the coalitional values shall be re-calculated repeatedly since values may be affected by variations in the coalitional configuration. This is because each value is calculated subject to the tasks that should be performed. A change in the coalitional configuration corresponds to an assignment of a task to a coalition. This specific task no longer affects the coalitional values which it may have previously affected.

Therefore, the coalitional values that have been calculated with respect to a task that has just been allocated must be re-calculated. All other values remain unchanged.

4.3.2 Overlapping coalitions

Each agent A_i shall iteratively perform the following:

(i) Locate in L_i the coalition C_j that has the smallest cost c_j.
(ii) Announce the coalitional cost c_j that it has located.
(iii) Choose the lowest among all of the announced coalitional costs. This c_{low} will be chosen by all agents. The corresponding coalition C_{low} and task t_{low} shall be selected as well.
(iv) If you are a member of the chosen coalition C_{low}, join the other members of C_{low} and form the selected coalition.
(v) Erase from T the task according to which the value of the newly-formed C_{low} has been calculated.
(vi) Update the capability-vectors of all of the members of C_{low} according to their contribution to the task-execution.
(vii) Assign to L_i^{cr} the coalitions in L_i for which values should be re-calculated.

As in the case of disjoint coalitions, the iterative procedures of calculating coalitional values and costs, selecting the preferred coalitions and forming them, will be repeated until there are no more tasks to be performed or none of the possible coalitions is beneficial. Here, some coalitional values must be re-calculated in every iteration because each value is calculated with respect to the unused capabilities, and these may change due to the formation of a coalition. Note that in the disjoint case we use the coalitional weight whereas in the overlapping case we use the coalition cost. This difference is necessary since in the disjoint case agents can only contribute once to the total outcome, therefore their individual weight is important, while in the overlapping case agents may contribute to the total outcome several times, and therefore the contribution of whole coalitions is important. This difference causes a different choice of agents for the coalitions to be formed.

5 Tasks with precedence order

In a case where tasks have predecessors, each task can be satisfied only if all of its predecessors have been performed previously. Hence, our precedence-order algorithm requires that the choice of a task implies the choice of all of its predecessors (however it does not require that the same coalition will perform all the precedent tasks). For each task t, we denote its set of predecessors, including t itself, by P_t. The choice of t for coalition formation will depend on the costs of, and the benefits from, the formation of coalitions that perform *all* of the tasks in P_t.

In the original algorithms above, the agents form coalitions iteratively – one coalition in each iteration. They greedily satisfy tasks, performing the single task t that currently appears most beneficial. In the precedence-order case, the agents shall form several coalitions in each iteration, to perform all of the tasks in the specific set P_t which appears most beneficial among all such possible sets. For the evaluation of these sets of tasks, we introduce the concept of precedence value pV_t (p-value), which is the sum of the values of all of the tasks in the precedence set P_t.

As in the previous cases, where the value of a task was re-calculated in each iteration until it was chosen, pV_t will be re-calculated in each iteration as well. Each such calculation of pV_t requires the calculation of the values of all of the tasks in P_t. For this we employ the calculation methods of sections 5.2 and 5.3, where P_t is substituted into T from 5.3. The internal precedence order within P_t is not considered as to this value-calculation, since all of the tasks within P_t must be satisfied, if t is chosen. The distribution of the calculations among the agents will be performed as suggested in section 5.2. In each iteration, the best pV is chosen from among all of the current pV's. This implies that all of the tasks in P_t will be satisfied as well. Since we introduced the notion P_t, we must emphasize its difference from T, which denotes the set of all of the tasks that were not yet performed. Initially, T includes all of the tasks that must be satisfied by the agent-system.

The agents may either each perform all of the calculations concerning each task $t \in T$, or they may distribute these calculations, but in each step of the algorithm (when necessary) agree upon the P_t to be calculated. Recall that only part of the values corresponding to tasks in P_t should be re-calculated.

All of the agents, simultaneously, should iteratively perform the following:

- Given the current status of capabilities; for each $t \in T$ do
  (i) Compute greedily the values of all of the tasks in P_t using the distributed methods of section 5.3, where the input set of tasks for the greedy calculation is P_t.
  (ii) If, in step (i), all of the tasks in P_t were assigned coalitions to perform them, then set pV_t to be the sum of the values of the tasks in P_t.
  (iii) Otherwise, remove t from T [22].
- Choose the task t^* with the maximal pV_t to be performed together with all of its predecessors. Form the required coalitions as in 5.3.

[22] This is since t cannot be performed.

– Remove t^* and all of its predecessors from T.
– If an overlapping coalitions case, update the capability-vectors of the associated coalitions' members. Else, (disjoint coalitions), delete the members of the coalitions from the list of candidates for coalition formation.

The above iterative procedure of calculating p-values and costs, selecting the preferred coalitions for task execution and forming them will be repeated until there are no more tasks to be performed in T, or no more agents free to perform them. Note that as before, the values are dynamically changing as a result of the change in the accessibility of resources, and therefore shall be distributively re-calculated after each coalition formation.

6 Quality assessment

The algorithms presented above have several advantages. An important property of the algorithms is their low logarithmic ratio bounds. Denote a coalition configuration by $\mathcal{C}$, its total cost by $c_{tot} = \sum_{C_j \in \mathcal{C}} c_j$, and superscribe by $*$ for the optimal case.

6.1 The overlapping case

To express the ratio bound by the same notions both for the disjoint and the overlapping algorithms, we must define the agent weight w_i for the case of the overlapping coalitions. Denote the number of *new members* in a coalition C_i by z_i; the agent's weight is the ratio between the coalition cost c_i and z_i, i.e., $w_i = \frac{c_i}{z_i}$. Here, only agents who are *new members*, i.e., agents who join a new coalition for the first time, are considered for weight calculation (whereas in the disjoint case such a distinction is not necessary).

Theorem 1 *The ratio bound ρ of the algorithm for the overlapping case is given by*

$$\rho = \frac{c_{tot}}{c^*_{tot}} \leq \sum_{i=1}^{max(|C_j|)} \frac{1}{i} \qquad (1)$$

Proof: Each coalition C_j contributes its cost c_j to the cost c_{tot} of the coalition configuration $\mathcal{C}$ of the solution. Since each agent A_i receives some weight w_i, c_{tot}, which is the sum of the weights of all of the agents in the final coalitional configuration $\mathcal{C}_{final}$, is given by:

$$c_{tot} = \sum_{A_i \in N} w_i \leq \sum_{C_j \in \mathcal{C}_{final}} \sum_{A_i \in C_j} w_i \tag{2}$$

The inequality results from the fact that each agent contributes only once to the total cost (as expressed in the left side sum), and the right side sum allows multiple contributions by a single agent. For any final coalition configuration (and in particular for the optimal one):

$$\sum_{C_j \in \mathcal{C}_{final}} \sum_{A_i \in C_j} w_i \leq \sum_{C_j \in \mathcal{C}_{final}} \sum_{A_i \in C_j} \frac{c_j}{|C_j|} = \sum_{C_j \in \mathcal{C}_{final}} c_j \sum_{A_i \in C_j} \frac{1}{|C_j|} \tag{3}$$

Since the following relation holds in general,

$$\sum_{C_j \in \mathcal{C}_{final}} c_j \sum_{A_i \in C_j} \frac{1}{|C_j|} \leq \sum_{C_j \in \mathcal{C}_{final}} c_j \sum_{i=1}^{max(|C_j|)} \frac{1}{i} = c_{tot} \sum_{i=1}^{max(|C_j|)} \frac{1}{i} \tag{4}$$

it in particular holds for c^*_{tot}, and the conclusion from (2), (3), and (4) is that

$$c_{tot} \leq \sum_{C_j \in \mathcal{C}_{final}} c_j \sum_{i=1}^{|C_j|} \frac{1}{i} \leq c^*_{tot} \sum_{i=1}^{max(|C_j|)} \frac{1}{i} \tag{5}$$

Hence, we derive the ratio bound

$$\rho = \frac{c_{tot}}{c^*_{tot}} \leq \sum_{i=1}^{max(|C_j|)} \frac{1}{i} \tag{6}$$

□

Since the disjoint case algorithm is similar in its nature to the algorithm of the overlapping case, the quality assessment is similar as well. Despite minor modifications in its derivation, the ratio bound of the disjoint algorithm is similar, and therefore not presented separately. Note that $\lim_{n\to\infty} \sum_{i=1}^{n} \frac{1}{i} = \gamma + \ln n$, where γ is Euler's constant ($= 0.5772\ldots$). Therefore, this ratio bound grows logarithmically with the size of the coalitions to which the algorithm refers. Since we previously suggested presenting a constant to determine the maximum permitted coalitional size, we simultaneously limited the ratio bound to being a constant that can be calculated simply using (6). One must recall that the ratio bound is the worst case bound.

6.2 The precedence order case

As a result of the modified coalitional values, the costs and the weights are changed.

Proposition 1 *The ratio bound in the precedence order case is given by*

$$\rho = \frac{c_{tot}}{c^*_{tot}} \leq \ln max(|C_j|) \ln max(|p\text{-}value|) \tag{7}$$

Proof: The calculation of p-values in the precedence order case is similar to the calculation of values in both of the non-precedence-order cases. This is because the calculations in all cases are based on the concept of choosing the lowest cost each time and adding it to the total cost. Therefore, according to theorem 1, the ratio bound of the p-value within any precedence set P_t is

$$\rho_{\text{p-value}} = \frac{c_{tot}(P_t)}{c^*_{tot}(P_t)} \leq \sum_{i=1}^{max(|C_j \in P_t|)} \frac{1}{i} \leq \ln max(|C_j \in P_t|) \tag{8}$$

The total value is

$$c_{tot} = \sum_{p_t \subseteq \mathcal{C}} \text{p-value} \tag{9}$$

and therefore c_{tot} is bounded by $c^*_{tot} \ln max(|\text{p-value}|)$. As a result, the ratio-bound of the p-values contributes another logarithm into the ratio-bound, thus yielding:

$$\rho = \frac{c_{tot}}{c^*_{tot}} \leq \ln max(|C_j|) \ln max(|\text{p-value}|) \tag{10}$$

□

7 Complexity of the algorithms

In addition to the quality of the solution with respect to the optimal solution (given k), the efficiency of the algorithms should be judged from two main perspectives: that of computations and that of communications. At the preliminary stage, where the calculations are distributed, all of the relevant permutations of agents are calculated; this requires $\sum_{i=1}^{k}(n-1)!/((n-i-1)!i!)$ computation operations, which is of order $O(n^{k-1})$ per agent. During this stage the agents contact one another; each of them conducts up to $\frac{n^{k-1}}{min(|S_{ij}|)}$ contacts with the other agents (where $|S_{ij}|$ is the size of the subset of potential coalitions considered in each communication operation). Since the choice of $|S_{ij}|$ may be any integer in the range $[1, C^n_{k-2}]$, the communication complexity varies correspondingly. While in the best case (communication-wise), where $|S_{ij}| = C^n_{k-2}$ ($O(n^{k-2})$), the average number of communication operations per agent is $O(1)$, in the worst case, where $|S_{ij}| = 1$, the communication complexity per agent at the calculations-distribution stage is $O(n^{k-1})$.

Below we prove and discuss the complexity of the non-precedence case.

Proposition 2 *For the value calculation and task assignment, the computational complexity per agent is of order $O(n^k \cdot |T|)$.*

Proof: The value calculation process consists of the assignment of tasks to coalitions [23]. Such an assignment is performed for all of the tasks, hence $|T|$ assignments are necessary. The number of value-calculation operations per agent is of order [24] $O(n^{k-1} \cdot |T|)$. Given the assumption that the number of capabilities depends neither upon the number of agents nor upon the number of tasks, each assignment operation requires $O(1)$ operations. Choosing the largest value is of the order of the number of possible coalitions, i.e., $O(n^{k-1})$ computations per agent. The two processes of calculating coalitional values and choosing coalitions may be repeated up to n times (however, if $|T| < n$, n will be replaced by $|T|$. Therefore, the computational complexity per agent is of order $O(n^k \cdot |T|)$. □

This complexity can be compared to a centralized case, where a single agent performs all of the operations. In such a case, this single agent will experience $O(n^{k+1} \cdot |T|)$ computations and $O(n)$ communications. Therefore, the computational complexity is higher than in the average distributed case (the speed-up is of order $O(n)$), but the communicational complexity is lower [25]. The overlapping coalitions' case is similar to the disjoint coalitions' case. However, if $|T| > n$ this case still allows for $O(|T|)$ coalitions to be formed, whereas in the disjoint case n is the upper limit.

Following we present the complexity of the precedence order case. We do not provide an explicit proof, however justify it informally.

Proposition 3 *The computational complexity in the precedence order case is $O(n^k \cdot |T|^5)$.*

Proposition 3 is justified as follows. The change in the calculation of coalitional values due to the precedence order yields an increase in the computational complexity. In addition to the original calculations, $O(|T|^3)$ operations are necessary for the calculation of each task's p-value. This shall be performed up to $|T|$ times, and therefore the overall additional complexity is $O(|T|^4)$, by

[23] We assume that the planning for partitioning goals into sub-goals is given, and therefore the calculation of coalitional values is not as complicated as in [42].

[24] Note that, while some agents perform $O(n^{k-1} \cdot |T|)$ value-calculations, others may happen to perform less than $O(n^{k-1} \cdot |T|)$ calculations. This property of the process is advantageous because it occurs when there are differences in computational capabilities among the agents. In such cases the non-equal partition of the calculations moderates the differences in the calculation-time of the agents, thus reducing the average time of calculation completion.

[25] If, for all agents, there exists a communication channel between every pair of agents, then the computational overhead of the distributed case will not affect the performance of the algorithm.

which the complexity of the original algorithm shall be multiplied. Thus, the new complexity is $O(n^k \cdot |T|^5)$, which is a rough estimation of its complexity. A deliberate analysis and the introduction of some heuristics reduce the complexity.

8 Simulation results

In order to strengthen the validity and to demonstrate the quality of our approach, we have simulated the algorithm of overlapping coalitions with precedence order among the tasks. The simulation consists of a centralized program which calculates the task allocation and the coalition formation which would have resulted from the distributed algorithm. The centralization is aimed at simplifying both the code and the running process, however all of the properties of the original algorithm, except for its distribution, are kept. Separately, in the next section, we present a fully distributed implementation of the algorithm for a domain specific agent system (with some modifications to the algorithm).

We are especially interested in the ratio between the total outcome of agent-systems that act according to our algorithm and the optimal outcome of such systems. We are also interested in the number of tasks executed with respect to the number of tasks executed in the optimal case. Another important factor would be the run-time consumption with respect to the number of agents and tasks.

The simulation system was tested for different ratios between the average amounts of resources per agent and the average resource amount necessary for task execution, as we elaborate below. We have conducted several hundreds of runs of the simulation for various numbers of tasks, and the number of agents varying from 4 to 22. We have also performed few simulations for 50 to 60 agents to check the functionality of the algorithm for larger numbers of agents and tasks. There were 4 different resources, randomly partitioned among the agents. As a result of this non-equal partition, agents could have amounts of some resources which are significantly different from the average. This affected their ability to perform tasks and has driven agents to coalition formation.

The figures below present the results of the simulation, divided into three categories: figures 1 and 2 present the results in the case where the average amount of resources per agent is approximately equal to the average amount of resources necessary for each task execution; figures 3 and 4 present the results in the case where the average amount of resources per agent is significantly greater than the average amount of resources necessary for each task execution; figures 5 and 6 present the results in the case where the average amount fig. 1

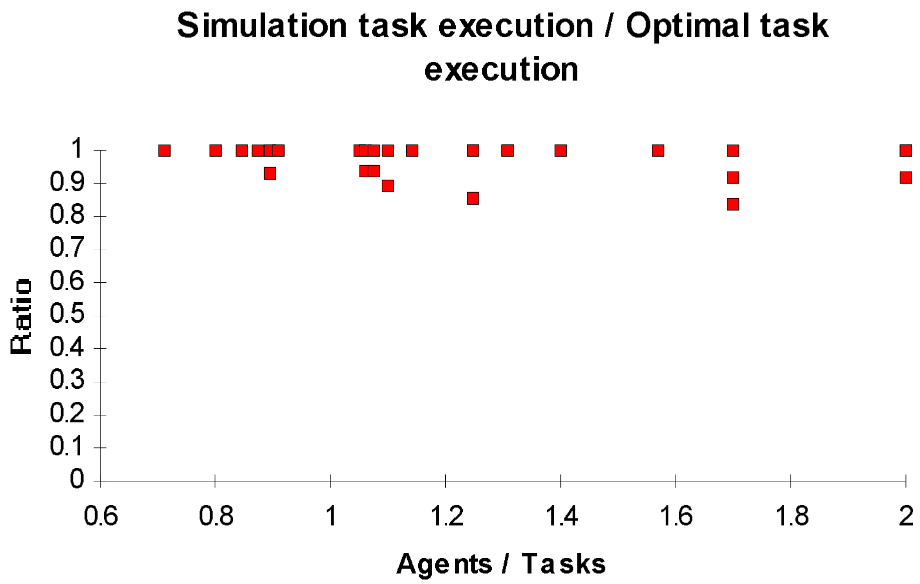


fig. 2

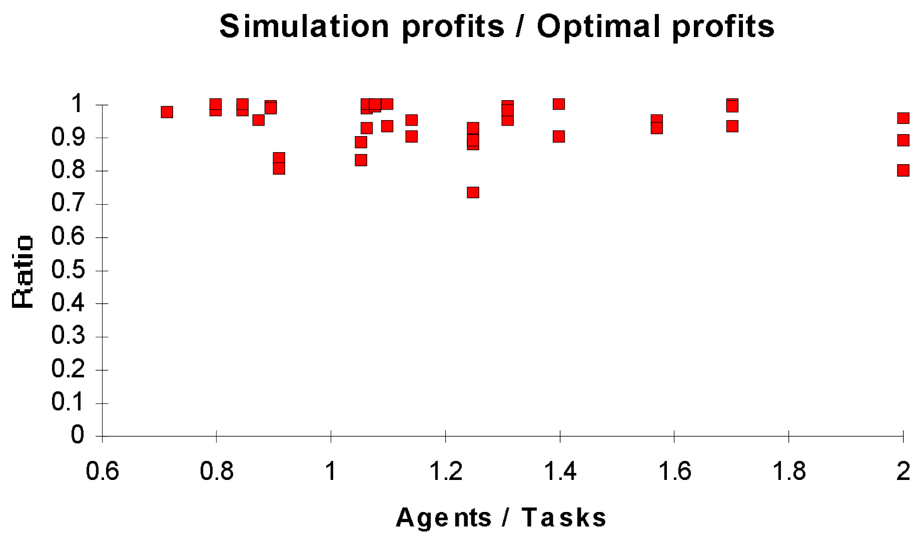


fig. 3

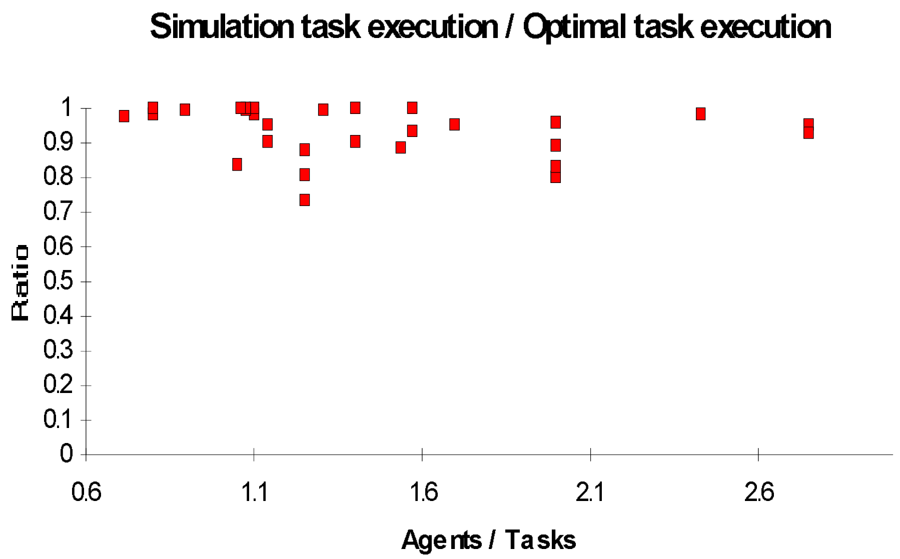


fig. 4

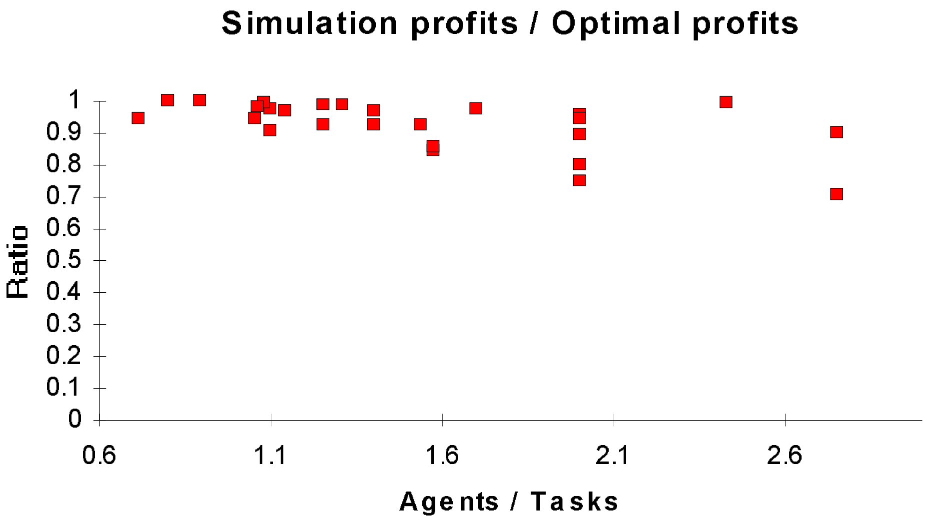


fig. 5

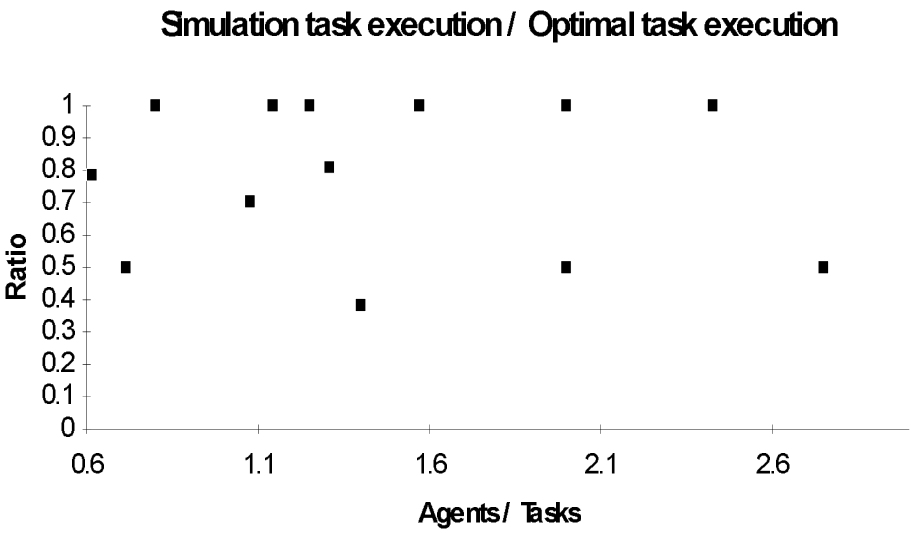
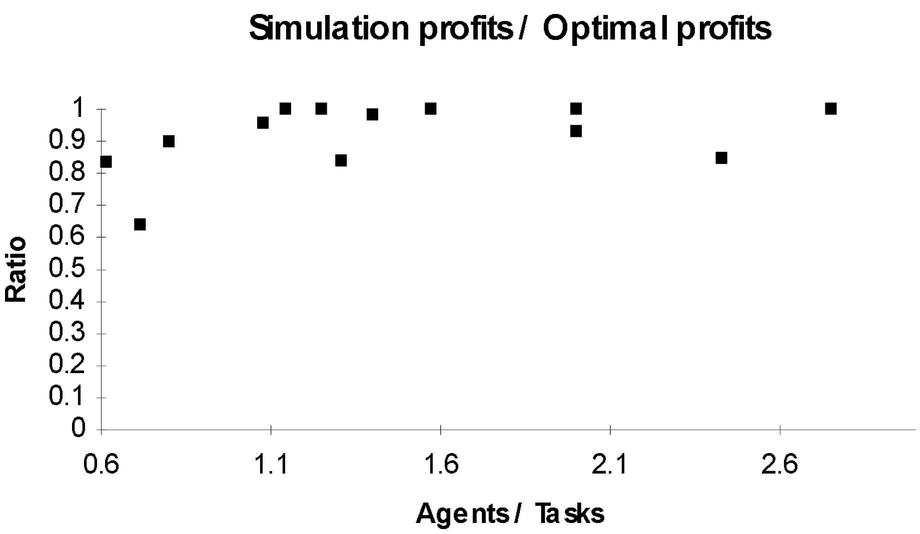


fig. 6



of resources per agent is significantly smaller than the average amount of resources necessary for each task execution. In each graph the x axis represents the ratio between the number of agents and the number of tasks, and the y axis represents the ratio between the simulation results and the optimal results [26]. Figures 1,3,5 measure this ratio of task execution and figures 2,4,6 measure this ratio of the global system profits. In all six graphs, each square repersents the result of a single simulation (however some squares overlap due to similar results of different experiments).

Via the simulations, we have examined the number of executed tasks with respect to the number of tasks that would have been executed in the optimal case, given the same settings (figures 1,3,5). In addition, we have tested the ratio between the profits gained by the agents (as a result of task execution via coalition formation) and profits that would have been gained in the optimal case (figures 2,4,6). The results in both cases are very close to optimal, as can be seen in the figures.

The main results of the simulations performed are as follows:

(i) The ratio between the simulation task-fulfillment and the optimal task-fulfillment is, in most cases, close to one, which means that the simulation task-fulfillment is near-optimal (figures 1,3). Only when the ratio between the agents' resources and the tasks' resources is significantly smaller than 1, does the task execution ratio decrease (figure 5). Nevertheless, the profit ratio remains sub-optimal. The reason for the decrement in the task-fulfillment is that the algorithm leads to the choice of the most beneficial tasks. Thus, it may perform fewer tasks, yet reach a sub-optimal profit.
(ii) The ratio between the profits from task execution via our algorithm and the optimal case is close to one (figures 2,4,6). Note that this ratio is significantly better than the theoretical ratio bound.
(iii) The run time for all of the simulations was usually a few seconds (running on a Sparc station) and always less than a minute, even when dealing with 50 to 60 agents and tasks (and running on a PC).
(iv) During the simulation, coalitions of various sizes have formed.

9 Implementation in a dynamic, open MAS

In this section we report an implementation of the coalition formation methods presented previously in a dynamic, open MAS. For this implementation

[26] We have computed an upper bound on the optimal results and not the optimal results themselves (due to complexity).

we have chosen an existing framework, the RETSINA agent system, which is a web-based system where tasks arrive dynamically to agents and agents may appear and disappear. A brief description of this system is provided below (the reader is referred to [52] for details). Implementing our algorithms in a RETSINA framework required some adjustments to the openness and dynamism of the system, however proves the applicability and usefulness of these algorithms to real-world MAS.

9.1 The multi-agent system

RETSINA (REusable Task-based System of Intelligent Networked Agents) [51–53] is a cooperative multi-agent system that was developed to integrate information gathering from web-based sources and perform decision support tasks. The agents in RETSINA compartmentalize specialized task knowledge, organize themselves to avoid processing bottlenecks, and can be constructed specifically to deal with dynamic changes in information, tasks, number of agents and their capabilities.

In RETSINA, the agents are distributed and run across different machines. Based on models of users, agents and tasks, the agents decide how to decompose tasks and whether to pass them to others, what information is needed at each decision point, and when to cooperate with other agents. The agents communicate with each other to delegate tasks, request or provide information, find information sources, filter or integrate information, and negotiate to resolve inconsistencies in information and task models. The system consists of three classes of agents: *interface* agents, *task* agents and *information* agents. Note that a similar infrastructure is used in InfoSleuth [36]. Interface agents interact with users receiving their specifications and delivering results. They acquire, model and utilize user preferences. Task agents formulate plans and carry them out. They have knowledge of the task domain, and which other types of task agents or information agents are relevant to performing various parts of the task. In addition, task agents have strategies for resolving conflicts and fusing information retrieved by information agents. They decompose plans and cooperate with appropriate task agents or information agents for plan execution, monitoring and results composition. Information agents provide intelligent access to a heterogeneous collection of information sources. They have models of the information resources and strategies for source selection, information access, conflict resolution and information fusion. Several typical tasks cannot be executed by a single agent and agents form teams, on demand, for such tasks.

For instance, suppose a user has requested an interface agent to monitor its investment portfolio and maintain a given level of risk. The interface agent must involve in this task information agents that can provide various types of information with regards to the assets in the portfolio (e.g. price, relevant news). It also needs task agents that can analyze the information (e.g. compute risk) and critique it and make decisions and suggestions with regards to the portfolio. There may be several agents that can provide such services, and selecting the most appropriate ones is part of the coalition formation process. Note that there may be multiple tasks arriving simultaneously at different agents in the system, therefore there may be several attempts to form such coalitions. The coalition formation algorithm serves for this purpose.

A basic design problem of cooperative, *open*, multi-agent systems is the connection problem [13]. That is, each agent must be able to locate the other agents who might have capabilities which are necessary for the execution of tasks, either locally or via coalition formation. In an open system where participating agents may dynamically enter and leave, which is distributed over the Internet, broadcast communication solutions are precluded. The solution provided in RETSINA relies on some well-known agents and some basic interactions with them – matchmaking [14,32]. In general, the process of matchmaking allows an agent with some tasks, the requester, to learn the contact information and capabilities of another agent, the server, who may be able to execute part of the requester's tasks via a matchmaker which is an agent that maintains the contact information, capabilities, and other service characteristics (e.g. cost, availability, reliability) of other agents.

During the operation of the multi-agent system, agents that join the system advertise themselves and their capabilities to a matchmaker, and when they leave the agent society, they un-advertise (for more details, see [51]). In search of agents with which they may possibly form coalitions, agents approach a matchmaker and ask for names of relevant agents. After having acquired the information about other agents they can directly contact these agents and initiate cooperation as needed. Note that there may be several matchmaker agents to relax the problem of unavailable or overwhelmed single matchmaker and to avoid a centralized solution.

9.2 Cooperation via coalition formation

The RETSINA system can receive several tasks from several users. Incorporation of the coalition formation mechanism increases the efficiency of groupwise task execution, resulting in near-optimal task performance. We report such results in section 9.3. In addition, this mechanism enables agents to decide upon the importance (and possibly – the order [27]) of tasks to be performed. Such

[27] A RETSINA agent has an internal scheduling mechanism, but our algorithm provides priority generation, which is not part of the scheduler.

decision making is important in real-world domains, where there may be situations in which a system cannot fulfill all of its tasks. A RETSINA agent considers cooperation if one of the following holds: the agent cannot perform a specific task by itself; the agent can perform a specific task, but other agents are more efficient in performing this task (e.g., they require less resources or perform faster); the agent can perform a specific task, but working on it collaboratively will increase the benefits from the task (or reduce the costs).

The implementation of the coalition formation mechanisms required that we modify them due to several differences between the assumed agent-systems and the RETSINA infrastructure, as discussed below:

- The number of goals and agents assumed in previous sections is fixed, while RETSINA is a dynamic system where agents appear and disappear and tasks vary constantly.
- The method in which the information with regards to the existence of tasks and their details is distributed is not discussed in the original algorithms.

To resolve the above restrictions, we made various modifications to the algorithms [28]. We describe below how agents behave within the RETSINA framework when coalition formation is applied, with the addition of the required modifications. Note that some functional issues of the RETSINA agents are described in an extremely simplified manner (elaborate details can be found in [51]). The activity of the system entails tasks which dynamically arrive from users or agents. The activity of a single agent for each arriving task is as follows.

(i) When a task t_j arrives at agent A_i (either from a user or from another agent) it performs:
   (a) Adds t_j to T_i (its task list).
   (b) Finds possible decompositions of t_j to subtasks $t_j^1, \ldots, t_j^l$ using its *tasks reduction library* which includes pre-given possible decompositions [29].
   (c) According to the possible decompositions, A_i determines the capabilities needed for performing t_j (e.g. information gathering) and contacts a matchmaker to find agents that have those capabilities and may be able to work on the task. It adds these agents to N_i (its agent

[28] Note that the original algorithms do not discuss capability reuse and tasks with complex time dependencies, such as partial overlapping use of a resource, which are typical in the RETSINA dynamic system. RETSINA agents take care of these even without the coalition formation (e.g., the agent's scheduling resolves time dependencies). The implemented coalition formation algorithm inherits these properties.
[29] For example, checking whether a portfolio is balanced requires (at least) a subtask of finding the prices of assets in the portfolio and a subtask of computing balance according to some criteria.

list) and sends them each the relevant subtasks of t_j (or the whole t_j, if it is not decomposable). If the sender of t_j to A_i is an agent, it is added to N_i as well.

(d) To avoid cyclic task delegation and to allow propagation of results back to the requester, the origin of the task (user or agent) as well as the delegation path (i.e., the agents through which the task has passed) are added to each delegated task.

(ii) When A_i's local schedule allows, it computes coalitions and their values and forms coalitions as follows:

(iii) **Computing coalitions values**[30]: for each task t_j in T_i it considers the possible coalitions that can be formed to perform it (based on its task reduction library) and computes the coalition value for each such possibility. The value calculation is performed by evaluating linear functions attached to each capability which takes into account the typical amount of computational resources (e.g. cpu, memory, disk space) necessary for such a task, as studied from past behavior of the RETSINA system and the time interval for execution.

(iv) **Choosing Coalitions**[31]:

(a) Among the values computed in step (iii), A_i chooses the one with the highest value (correspondingly, the lowest cost c_j), denotes it C_j, and registers C_j and its value at the matchmaker.

(b) A_i checks whether the value it registered is the highest registered at the matchmaker. If so it forms with the members of C_j the coalition and performs the associated task and subtasks. If no other values are registered A_i re-checks for values after a waiting period[32].

(c) Upon completion (on which the members of the coalition report to A_i), A_i removes t_j from T_i. In case of task execution failure (and if t_j's deadline has not passed) t_j is kept in the list for future handling. Note that updates to capabilities are performed individually by agents at their matchmaker.

(d) Concurrently, if T_i is not empty, A_i performs (as its local schedule allows) occasional polling of the matchmaker for changes in coalitions, values, agent availability, and continually performs the coalition formation algorithm by skipping to step (ii) (and possibly being a member of several coalitions).

[30] This step corresponds to the algorithm in section 4.2. However, while in the original algorithm tasks are associated with coalitions, here coalitions are associated with tasks. That is, for each task the best coalition is found. This is simpler to implement in an open system, where tasks are not known to all agents.

[31] This step corresponds to the algorithm in section 4.3.

[32] Such situations may occur due to asynchrony in the system as well as differences in agent computation time and network latency. Waiting for additional values may be beneficial if the task's deadline allows.

In RETSINA the communication- and computation-time for value calculation and coalition design are significantly small as compared to the task execution time and tasks can dynamically appear. Therefore, the modified algorithm includes a re-design process. When a new task is received by an agent, we require:

- If tasks that were assigned to coalitions have not been performed yet within the current iteration of the coalition formation algorithm, agents who are in these coalitions and are informed about the new task will re-calculate the coalitional values to take into consideration the arrival of the new task.
  - if inclusion of the newly arrived task in coalition recalculations in the current iteration introduces a new coalition value, greater than currently known values, then the agents re-design the coalitions, selecting again the best among the actual, re-designed, coalitions [33].
  - otherwise, the agents avoid coalition re-design, and consider the new task for inclusion in coalitions at the next iteration.
- If all previous tasks are in process, the new task, has it arrived at A_i, will be added to the group of tasks T_i and be dealt with in the next coalition formation iteration.
- In case of a rapid high-frequency stream of new tasks, the re-design process may be dis-enabled. If such a rate of new tasks is expected in advance, or the agents statistically infer such a rate by sampling the task stream and interpolating the statistical data, the re-design process shall be avoided.

9.3 Analysis of the modifications

The dynamic addition of tasks to the agent system may change the overall order of complexity of the algorithm. In the worst case, the re-design process will be performed for every new task. This will result in performing the whole process $|T_d|$ times, where $|T_d|$ is the number of dynamically received tasks. Therefore, the original complexity analysis should be multiplied by $|T_d|$. Since $T_d \subseteq T$, the re-design process implies an increase in complexity by an order of magnitude with respect to the number of tasks.

The communication requirements of the original algorithms are, in the worst case, rather large. We designed the implemented algorithm to avoid the worst case of communication complexity. This was performed by the requirement that on each contact of A_i with A_j it implicitly commits to compute the maximal S_{ij}. This results in a communication complexity for commitment which is $O(n \cdot T_d)$ per agent (however the average is $O(T_d)$). In the modified algorithm the communication complexity in the value calculation and task assignment

[33] Note that re-design may not be allowed if the expected task flow is rapid, lest the system will constantly re-design and not perform its tasks.

stages is similar to the original (i.e., $O(n \cdot |T|)$ per agent). However, additional communication is required for informing other agents about dynamically arriving tasks. Therefore, while the worst case remains unchanged, the average case becomes $O(|T_d| \cdot n)$. Yet this is a low linear complexity. Nevertheless, in web information gathering (in which RETSINA operates), the network latency causes the computation time for coalition formation to be dwarfed by comparison. This was observed in the course of our experiments.

Originally, the number of agents n was assumed to be constant. However, in RETSINA, n may dynamically change, hence the complexity analysis is slightly modified. Let $\nu = max(n)$, where this maximum may be decided upon by designers according to their expectations with regards to the agent system. Then, substitute n in the original complexity expression by ν, resulting in $O(\nu^{k-1} \cdot |T| \cdot |T_d|)$.

RETSINA agents are able to perform task reductions [60]. In practice, the internal complexity of a sub-task is determined within the task reduction and task schema libraries (in general, these libraries provide details for planning and executing tasks). The libraries are domain-specific, hence the designers of the domain-specific components have control over the complexity of sub-tasks. In the information domain of RETSINA sub-tasks, each sub-task can typically be performed by a small number of agents. This implies that the coalition formation procedure will concentrate on the formation of small coalitions of agents with particular expertise to perform a task.

For example, one of the domains in which RETSINA was implemented is satellite tracking. One of the tasks that the agents can cooperatively perform is finding if and when a specific satellite will be observable in a specific location. For this, up to 4 information agents are involved in the information gathering, and up to 4 other agents are involved in other related tasks. This means that the maximal coalition size for this task type is 8. Since other tasks of the system are of the same order of complexity, the sizes of coalitions are limited as well. The system may include other active agents, however these will be involved in other tasks or be idle.

The specialization of agents in RETSINA results in the incorporation of agents into coalitions according to their specialty/capability (when more than one agent with the same specialty are present, their utility functions enables comparison which results in the choice of the one with the highest payoff). Thus, a coalition size is limited to the number of different specialties which are necessary for the execution of the task that this coalition performs. In current RETSINA implementations, the number of specialties which are necessary for a given task execution is small and hence such is also the size of coalitions. The maximal number of specialties necessary for sub-task execution complies with the k restriction on the size of coalitions. However, since different decompositions of tasks along different specialty dimensions are possible, a specific agent system may have several k's. Among them, the maximal will determine the worst case complexity.

9.4 Implementation settings

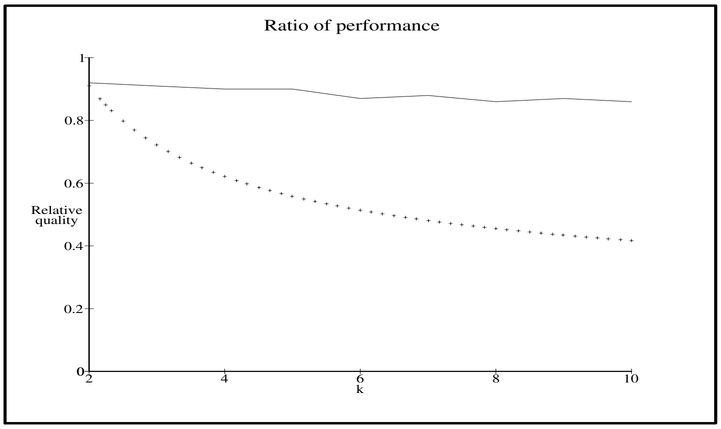


Fig. 1. The average performance (solid line) and the worst case (broken line), expressed as a fraction of the optimal performance, as a function of the maximal coalition size permitted k.

To apply the modified coalition formation algorithm to a RETSINA framework we had to simulate some of the RETSINA properties, while maintaining its openness and distribution as well as interaction mechanisms, including matchmaking and advertisement. This was performed by running degenerated RETSINA agents that were able to register at a matchmaker and query it for other agents, and could receive tasks as RETSINA agents do. The degenerated agents did not perform the tasks but only simulated the task performance. However, the simulated computational resources availability and consumption in the simulated task performance were similar to these values in the original system, as studied from the performance of RETSINA.

The implementation was performed by running up to 20 agents on several machines, and a stream of tasks was supplied to the agents, totalling to 1000 tasks. Each agent was represented by an agent-thread that simulated the resource-consumption and the task-queue of a real agent. The agents were running on several Sun workstations. Each agent-thread received an initial set of tasks, and additional tasks arrived at each agent dynamically in a random manner (i.e., the frequency of tasks, their type and the required capabilities were chosen in a random manner, according to the typical distributions as studied from RETSINA). Each task was associated with a vector of capabilities necessary for its execution and a payoff function for calculating the value of the task. For each task the agent creates a task-object that consumes time and memory and requires communication. The agent informs other agents about these tasks by passing references to task-objects. The CPU, communication and memory consumption are similar to those consumed by RETSINA agents. This resource consumption was previously measured as presented in [47].

During the simulation, coalitions of agents were formed, where a task was allocated to each, and the value of its execution by this coalition was calculated. The sum of these values was calculated to find the total payoff. When new tasks arrived, the re-design procedure was followed. We have performed this simulation several hundreds of times, and compared the total payoffs to the optimal payoffs (calculated off-line) and to the theoretical ratio bound, as depicted above.

In section 6 we have shown that the ratio bound increases logarithmically with respect to the number of coalition members. However, by implementing the algorithm in a simulated RETSINA agent system we show that the average case (for this specific domain) is close to the optimal case (see figure 1). The figure shows that the average performance (in terms of task allocation and execution) reached in the simulated RETSINA agents, depicted by the solid line, is around 0.9 of the optimal performance [34], while the worst case (the ratio bound), depicted by the broken line, declines fast to less than 0.5 of the optimal performance. Note that although in this implementation the asynchrony of the system prohibits the choice of the best value each time (instead, the best currently known value is selected), we still get near-optimal results.

10 Related work

Distributed task allocation has been discussed in the context of DPS systems. A well-known example of such a task allocation mechanism is the Contract Net Protocol (CNP)[49]. The CNP discusses cases in which an agent that attempts to satisfy a task may divide it into several sub-tasks and then sub-contract each sub-task to another agent via a bidding mechanism. In the CNP, tasks are allocated to single agents and a procedure for task-partitioning is necessary. The CNP allows single agents to perform more than one sub-task. This is similar to our approach as we, too, allow that agents be involved in the performance of more than one task in some of our algorithms. However, we do not deal with the case of single agents. Rather, we solve the problem of assigning tasks to groups of agents. In cases where single agents cannot perform tasks by themselves and tasks cannot be partitioned, or the partition is computationally too complex, close cooperation (within coalitions) is

[34] We calculated the optimal performance explicitly, off line, when a small number of agents are involved in coalition formation. Small here means up to 20.

required. In the CNP the efficiency of the solution was evaluated through simulations. However, since the issue of efficiency and complexity is crucial for the implementation of a solution, we provide a formal analysis of the quality and the complexity of our solutions, in addition to a simulation.

The allocation of tasks to teams of agents in order that they perform these tasks was discussed by several DAI researchers, e.g., [25,38,55]. Although discussing the assignment of roles to groups of agents, the process of designing and forming these groups, and the emphasis on seeking near-optimal performance, are not addressed there. Jennings [25] presents a system that handles task allocation by having a central organizer that uses information it has about the abilities of all other agents to assign tasks. Recipe selection for the complex group action is also managed by this central organizer. The team members select their own recipes for the single-agent constituent actions; there is no description of the mechanism according to which agents will form teams. In addition, the solution requires a central agent. Both of these limitations are avoided in our solution. Jennings discusses the joint intentions and the shared plans, whereas such planning issues are beyond the scope of our research.

As mentioned in the introduction, one can find two main extreme approaches in DAI: one is Distributed Problem Solving (DPS), which is concerned with cases where a group of agents faces satisfaction of a common task. DPS deals with the ways in which the agents distribute the task among them and individually or cooperatively fulfill the resulting sub-tasks [15,17,48]; the other extreme is Multi-Agent Systems (MAS), which focuses on systems of autonomous agents who are self-motivated and act in order to achieve their own personal tasks and increase their own personal utility. However, DPS and MAS are the two extremes in a variety of multi-agent situations lying between those two confines. In our paper we deal with autonomous agents' systems which are closer to the DPS type, although they may be acting as individuals and who try, under certain restrictions of the system, to increase their own benefits. Cooperation among autonomous agents may be mutually beneficial even if the agents are selfish and try to maximize their own expected utilities [30,50,63], as they do in MAS. In the case of DPS, the benefits of cooperation may be even greater than in the MAS case, since competition among the agents is restricted. In both cases, benefits may arise from cooperation via resource sharing and task redistribution.

DAI research has previously addressed the problem of tasks with precedence order and of overlapping problem solvers, as in [16]. There, coordination is based on an organizational view of node activity, where each node acts subject to its local control, solving a sub-goal of the global goal. The organizational view implicitly resembles the idea of a coalition. However, Durfee et al do not discuss coalitions nor do they address the problem of forming groups of nodes to improve the overall performance of goals.

Coalition formation is an important method of cooperation in multi-agent environments. Game theory provides an analysis of the possible coalitions that shall form as a result of a coalition formation process, and the resulting disbursements to the agents, assuming that agents do not have multiple memberships in coalitions. For example, see [18,27,40,61]. However, game theory does not provide algorithms which agents can use in order to form coalitions. Given a previously formed coalitional configuration (that is, a partition of the agents to subsets), game theory usually concentrates on checking its stability or its fairness [35] and on the calculation of the corresponding payments. Game theory rarely takes into consideration the special properties of a multi-agent environment. That is, the communication costs and limited computation time are seldom considered, and the solutions are not distributed. In our case, we are particularly interested in the coalition formation mechanism and how to distribute it among the agents. We also seek a dynamic evaluation of the coalitions [36] where game theory usually provides a static evaluation, and we allow agents to be members of more than one coalition. Therefore, the game-theoretic coalition formation theories are not appropriate for the multi-agent situation with which we are concerned.

Sandholm and Lesser [42] present a coalition formation model for bounded-rational agents and a general classification of coalition games. As in [42], we also allow for varying coalitional values. In [42], the value of a coalition depends on the computation time. However, we consider cases in which the time for computing the coalition values is polynomial, and the values vary with respect to the resource-consumption by previously-formed coalitions. The model presented by Sandholm and Lesser refers to a non-restricted number of configurations, whereas we present greedy algorithms that bound the number of configurations to be considered.

The problem of coalition formation can be approached as a Set Partitioning Problem (SPP). Coalition formation where coalitions may overlap can be approached as a Set Covering Problem (SCP) [37]. Set partitioning entails the partition of a set into subsets, and the set partitioning problem is finding a partition with the least cost. Since coalition formation of agents results in the partition of the agents into subgroups, SPP and SCP may be appropriate for determining which coalitions will form as a result of coalition formation algorithms. The SPP and the SCP have been dealt with widely in the context of NP-hard problems [19]. Exact solutions and approximations to SPP and

[35] Stability and fairness have several different definitions in the context of game theory.

[36] The values of the coalitions change dynamically due to rapid changes in the tasks and resource-availability, and therefore relying on the initial values is misleading.

[37] The SCP, which is very similar to the SPP. Both are described in in detail in section 3.2.

SCP have been proposed in the fields of operations research, combinatorial algorithms, and graph theory [3,4,8,9,20]. However, the solutions that have been proposed do not provide an appropriate solution to the problem of coalition formation among agents, due to three main deficiencies: 1) the exact and optimal solutions are solutions for NP-hard problems. That is, the complexity of the solution is exponential in the number of agents. Such a solution cannot be applied in cases where there are many agents in the environment, since the agents will be unable to calculate it; 2) the approximation algorithms, which lead to near-optimal solutions [38] are of polynomial complexity, but they deal with a tightly restricted number of pre-defined possible subgroups of a given set. This may be overly restrictive for the case of agent coalition formation, where there are 2^n subsets of a given set of n agents. This limitation may be resolved by artificially limiting the possible coalitions (as we do in our solution); 3) all of the present solutions are centralized. That is, the solutions can be calculated and implemented only by a central agent which dominates the coalition formation process and supervises the other agents. Such a situation is not typical in distributed agents' environments, and is inappropriate for our case, since it may result in a single point of failure as well as a computational bottleneck. In contrast MAS, among other goals, aim at reducing such problems and increasing robustness.

Distributed computing systems (DCS) research has dealt with problems of task allocation with precedence order. Optimal solutions were provided only for strongly constrained cases of two-processor systems (e.g. [10]). Sub-optimal solutions were also presented, (see e.g. [24]), yet these are constrained as well. The general problem is NP-complete, but some approximation algorithms [57] provide good solutions for the multi-processor system. In [7] an algorithm for such a task assignment was presented. The proposed solution is aimed at reducing the task turnaround time. The minimization of the task turnaround time is the main objective of task assignment within distributed computing systems. In our work, the main issue is the development of a task allocation that will increase the benefits of the agent-system, and not necessarily reduce the execution time. While DCS research usually discusses a fully-connected multi-processor system, we assume only a multi-agent system in which agents try to maximize the outcome of the system as a whole. Agents may be of various types, belong to different owners, be located in remote places and be indirectly connected. We discuss the specific case where single agents cannot perform the tasks alone, and they must join together in order to do so.

Research on the Loading Time Scheduling Problem (LTSP) [5] presents approximation algorithms for precedence-ordered task execution in cases where

[38] Near-optimal solutions are such that the ratio between them and the optimal solution is not large. In our case the available solutions have such a ratio for a small number of agents, but this ratio is growing logarithmically with in number of agents.

tasks may be performed either by a single machine or by several machines, but each task can be performed only by a subset of the machines. This appears to be similar to our research however, there are several differences, as follows: (i) our research concentrates on cases in which part of the tasks can only be performed by groups of agents and not by single agents; (ii) the main factor in the LTSP is the loading time of the first task in a sequence of tasks on a specific machine, while we discuss neither the loading time nor the effect of task-sequences on a single agent; (iii) we discuss the utility that arises from task execution given the precedence order and subject to the resource-distribution among the agents. The differences of our case from the DCS research requires another approach, as presented above.

A study of planning in MAS has been presented by Wellman [58]. In this research, the general-equilibrium approach from economics serves as the theoretical basis for the planning mechanism. Mechanisms in which competition is applied are used to construct a market-oriented programming environment, which is employed as a means for the construction and analysis of distributed planning systems. In his research, Wellman concentrates on competitive agents whereas we discuss cooperative agents. In addition, the formation of coalitions is not addressed there. The theoretic model underlying market-oriented programming is more suitable for large-scale systems of agents, although under some restriction may provide good results even for small number of agents, as proved by simulations in [34].

A Market-oriented approach was also utilized for task allocation, as presented in [56]. There, Walsh and Wellman present a simple auctioning protocol for the pricing and allocation of tasks among self-interested agents. In contrast, we address the case of cooperative agents. As a result, while Walsh can only prove convergence to a solution, we can, in addition, prove to that our protocols lead to near-optimal solutions. One important issue which is not addressed by Walsh and Wellman is the effects of the formation of groups agents. Such groups may allow for executing tasks which single agents cannot perform. Task execution by groups may also increase the efficiency of the execution. These have a major effect on the overall benefits of the agent system, hence we focus on groups (in our terms – coalitions) in our work.

Hard computational problems, including the class of NP-complete problems, are addressed Huberman et al in [23]. This research suggests, instead of employing a single approximation algorithm, using a portfolio of such algorithms where the algorithms are performed interleavingly. As proved in that research, such an approach may result in a solution that is preferable to any of the single component algorithm results. As the authors state, this economics-based approach can be used for the distribution of computational resources thus increase the efficiency of task performance. This might have been useful to address the problems discussed here, however no method is provided for distribution of the proposed approach and, more importantly, the issue of forming groups of agents as a means of increasing performance is not the goal of Huberman et al, therefore not addressed in that research.

Cooperation among agents for the solution of complex optimization problems is discussed in [54]. There, a team of multiple asynchronous programs (A-Team) works cooperatively to solve a problem. Although A-Teams improve upon other imperfect algorithms, the cooperation among the team members is very simplistic, as it refers to "non-zero intersection of output-space" of the cooperating agents. According to this approach, it is sufficient for the agents in an A-Team to have some common information with respect to the tasks or their solutions to claim cooperation among them. We refer to cases in which a higher level of cooperation is applied and these allows not only for improvement in solutions, but near-optimality. In addition, we deal with the formation of multiple groups of cooperating agents, not a single team. One interesting property of A-Teams is their scale-effectiveness, that is, the performance is better for greater teams. As opposed, we consider a comparatively small number of agents. This A-Team property is similar to the market-based systems as described above.

11 Discussion

In this paper we presented algorithms for task allocation among computational agents via coalition formation in a non-super-additive environment. The algorithms are suitable for cases where agents are motivated to act in order to maximize the benefits of the system as a whole. They are most appropriate for the incidents in which the agents cannot perform the tasks by themselves. However, they may also improve the efficiency of task execution when the performance of single agents is worse than their performance within groups. The algorithms are adjusted to cases in which the tasks have a precedence order as well, and still yield sub-optimal results.

Although general task allocation problems are computationally exponential (they are at least NP-complete), we present polynomial-complexity algorithms that yield results which are close to the optimal results and, as we have proven, are bounded by a logarithmic ratio bound. Our distribution method prevents most of the possibly overlapping calculations, thus saving unnecessary computational operations and leads to an even distribution of computation that takes into account the individual computational capabilities of the agents. In addition, the distribution of calculations is an outcome of the algorithm characteristics, since each agent performs primarily those calculations that are required for its own actions during the process. In cases with no precedence order, this distribution method prevents most of the calculations that may have been repeated by individual agents, thus saving unnecessary computational operations. However, this last property is less significant in the case of precedence-ordered tasks.

The algorithms are any-time algorithms. If halted before normally terminated, they still provide the system with several coalitions that have already formed and task allocated to them. In the non-precedence case the results, when halting, are of good quality. In the precedence-order case, better subgroups of tasks and coalitions are formed prior to others. However, if the algorithm is halted before the performance of such a subgroup has been completed, the accumulative value of the tasks within the subgroup is not necessarily the best. This means that although the any-time property of the algorithm exists for both cases, the case of precedence order may yield less beneficial intervention results. The any-time property of an algorithm is important for dynamic environments, wherein the time-period for negotiation and coalition-formation processes may be changed during the process.

The coalition formation methods were implemented in a real-world, information multi-agent system, to improve multi-agent cooperation (in terms of the joint payoff). To enable the implementation we had to relax several binding assumptions and limitations, taking into account requirements and constraints arising from the dynamism and openness of the system. We have shown that the incorporation of the coalition formation method induces a near-optimal task allocation while not significantly increasing the execution time.

To summarize, the algorithms presented in this paper can be used for task allocation and execution in dynamic, open systems of distributed computational agents, as we have demonstrated. They are not a general solution to the distributed task allocation problem. Some restrictive assumptions do apply. Nevertheless, the algorithms are appropriate for several realistic cases, where tasks may have a precedence order due to interdependencies and where agents can be involved in the performance of more than one task. The algorithms may be used in distributed cooperative agent-systems (in which tasks shall be assigned to agents in order that the agents will perform them) to maximize the expected outcome of the system as a whole and minimize the computational overhead.

References

[1] R. J. Aumann. The core of a cooperative game without side-payments. *Transactions of the American Mathematical Society*, 98:539–552, 1961.

[2] R. J. Aumann and B. Peleg. Von Neumann-Morgenstern solutions to cooperative games without side-payments. *Bulletin of the American*

Mathematical Society, 66:173–179, 1960.

[3] E. Balas and M. Padberg. On the set covering problem. *Operations Research*, 20:1152–1161, 1972.

[4] E. Balas and M. Padberg. On the set covering problem: An algorithm for set partitioning. *Operations Research*, 23:74–90, 1975.

[5] R. Bhatia, S. Khuller, and J. Naor. The loading time scheduling problem. In *Proc. of the 36th Annual IEEE Conf. of Computer Science (FOCS-95)*, pages 72–81, Wisconsin, 1995.

[6] A. H. Bond and L. Gasser. An analysis of problems and research in DAI. In A. H. Bond and L. Gasser, editors, *Readings in Distributed Artificial Intelligence*, pages 3–35. Morgan Kaufmann Publishers, Inc., San Mateo, California, 1988.

[7] G. H. Chen and J. S. Yur. A branch-and-bound-with-underestimates algorithm for the task assignment problem with precedence constraint. In *Proc. of the 10th international conference on Distributed Computing Systems*, pages 494–501, France, 1990. IEEE Computer Society.

[8] N. Christofides and S. Korman. A computational survey of methods for the set covering problem. *Mathematics of Operations Research*, 21(5):591–599, 1975.

[9] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[10] E. G. Coffman and L. R. Graham. Optimal scheduling for two-processor systems. *Acta Informatica*, 1:200–213, 1972.

[11] R. Conte, M. Miceli, and C. Castelfranchi. Limits and levels of cooperation: Disentangling various types of prosocial interaction. In Y. Demazeau and J. P. Muller, editors, *Decentralized A.I. - 2*, pages 147–157. Elsevier Science Publishers, 1991.

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

[13] R. Davis and R. G. Smith. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20(1):63–109, January 1983.

[14] K. Decker, K. Sycara, and M. Williamson. Middle-agents for the internet. In *Proceeding of IJCAI-97*, pages 578–583, Nagoya, Japan, 1997.

[15] E. H. Durfee and V. R. Lesser. Negotiating Task Decomposition and Allocation Using Partial Global Planning. In L. Gasser and M. N. Huhns, editors, *Distributed Artificial Intelligence, Volume II*, pages 229–244. Pitman/Morgan Kaufmann, London, 1989.

[16] E. H. Durfee, V. R. Lesser, and D. D. Corkill. Coherent cooperation among communicating problem solvers. In A. H. Bond and L. Gasser, editors, *Readings in Distributed Artificial Intelligence*, pages 268–284. Morgan Kaufmann Publishers, Inc., California, 1988.

[17] E. H. Durfee and V.R. Lesser. Global plans to coordinate distributed problem solvers. In *Proc. of IJCAI87*, pages 875–883, 1987.

[18] D. Fudenberg and J. Tirole. *Game Theory.* MIT Press, 1991.

[19] M. R. Garey and D. S. Johnson. *Computers and Intractability: a Guide to the Theory of NP-completeness.* W. H. Freedman and Company, New York, 1979.

[20] R. S. Garfinkel and G. L. Nemhouser. The set-partitioning problem: set covering with equality constraints. *Operations Research*, 17:848–856, 1969.

[21] J. C. Harsanyi. A simplified bargaining model for n-person cooperative game. *International Economic Review*, 4:194–220, 1963.

[22] J. C. Harsanyi. *Rational Behavior and Bargaining Equilibrium in Games and Social Situations.* Cambridge University Press, 1977.

[23] B. A. Huberman, R. M. Lukose, and T. Hogg. An economic approach to hard computational problems. *Science*, 275:51–54, 1997.

[24] O. H. Ibarra and C. E. Kim. On two-processor scheduling of one- or two-unit time tasks with precedence constraints. *Journal of Cybernetics*, 5(3):87–109, 1976.

[25] N. R. Jennings. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial Intelligence Journal*, 75(2):1–46, 1995.

[26] S. Jha, O. Shehory, and K. Sycara. Viewing contingent contracts between agents as options. Unpublished manuscript.

[27] J. P. Kahan and A. Rapoport. *Theories of coalition formation.* Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1984.

[28] S. P. Ketchpel. Forming coalitions in the face of uncertain rewards. In *Proc. of AAAI94*, pages 414–419, Seattle, Washington, 1994.

[29] S. Kraus. An overview of incentive contracting. *Artificial Intelligence*, 83(2):297–346, 1996.

[30] S. Kraus, J. Wilkenfeld, and G. Zlotkin. Multiagent negotiation under time constraints. *Artificial Intelligence*, 75(2):297–345, 1995.

[31] T. Kreifelts and F. Von Martial. A negotiation framework for autonomous agents. In *Proc. of the Second European Workshop on Modeling Autonomous Agents in a Multi Agent World*, pages 169–182, France, 1990.

[32] D. Kuokka and L. Harada. On using KQML for matchmaking. In *Proceedings of the First International Conference on Multi-Agent Systems*, pages 239–245, San Francisco, June 1995. AAAI Press.

[33] R. D. Luce and H. Raiffa. *Games and Decisions.* John Wiley and Sons, Inc, 1957.

[34] T. Mullen and M. Wellman. A simple computational market for network information services. In *Proc. of the First International Conference on Multiagent Systems*, pages 283–289, California, USA, 1995.

[35] B. Nadel. Constraint satisfaction algorithms. *Computational Intelligence*, 5:188–224, 1989.

[36] M. Nodine and A. Unruh. Facilitating open communication in agent systems: the infosleuth infrastructure. In M. Singh, A. Rao, and M. Wooldridge, editors, *Intelligent Agents 4*, Lecture Notes in Artificial Intelligence No. 1365, pages 281–296. Berlin:Springer-Verlag, 1997.

[37] P. Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9:268–299, 1993.

[38] A. S. Rao, M. P. Georgeff, and E. A. Sonenberg. Social plans: A preliminary report. In *Decentralized Artificial Intelligence, Volume 3*, pages 57–76. Elsevier Science Publishers, 1992.

[39] A. Rapoport. *N-Person Game Theory.* University of Michigan, 1970.

[40] E. Rasmusen. *Games and Information.* Basil Blackwell Ltd., Cambridge, Ma, 1989.

[41] T. W. Sandholm. An implementation of the contract net protocol based on marginal cost calculations. In *Proc. of AAAI-93*, pages 256–262, Washington D.C., 1993.

[42] T. W. Sandholm and V. R. Lesser. Coalitions among computationally bounded agents. *Artificial Intelligence*, 94:99–137, 1997.

[43] L. S. Shapley. A value for n-person game. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games.* Princeton University Press, 1953.

[44] O. Shehory and S. Kraus. Coalition formation among autonomous agents: Strategies and complexity. In C. Castelfranchi and J. P. Muller, editors, *Lecture Notes in Artificial Intelligence No. 957, From Reaction to Cognition*, pages 57–72. Berlin:Springer-Verlag, 1993.

[45] O. Shehory and S. Kraus. Task allocation via coalition formation among autonomous agents. In *Proc. of IJCAI-95*, pages 655–661, Montrèal, 1995.

[46] O. Shehory and S. Kraus. Formation of overlapping coalitions for precedence-ordered task-execution among autonomous agents. In *Proc. of ICMAS-96*, pages 330–337, Kyoto, Japan, 1996.

[47] O. Shehory, K. Sycara, and S. Jha. Multi-agent coordination through coalition formation. In M. Singh, A. Rao, and M. Wooldridge, editors, *Intelligent Agents 4*, Lecture Notes in Artificial Intelligence No. 1365, pages 143–154. Berlin:Springer-Verlag, 1997.

[48] R. G. Smith. A framework for distributed problem solving. In *Proc. of IJCAI-79*, pages 836–841, 1979.

[49] R. G. Smith. The contract net protocol: high-level communication and control in a distributed problem solver. *IEEE Transaction on Computers*, 29(12):1104–1113, 1980.

[50] K. Sycara. Persuasive argumentation in negotiation. *Theory and Decision*, 28:203–242, 1990.

[51] K. Sycara, K. Decker, A. Pannu, and M. Williamson. Designing behaviors for information agents. In *Proceeding of Agents-97*, pages 404–412, Los Angeles, 1997.

[52] K. Sycara, K. Decker, A. Pannu, M. Williamson, and D. Zeng. Distributed intelligent agents. *IEEE Expert – Inteligent Systems and Their Applications*, 11(6):36–45, 1996.

[53] K. Sycara and D. Zeng. Coordination of multiple intelligent software agents. *International Journal of Intelligent and Cooperative Information Systems*, 5(2 & 3):181–211, 1996.

[54] S. N. Talukdar, L. Baerentzen, A. Gove, and P. S. deSouza. Asynchronous teams: Cooperation schemes for autonomous, computer-based agents. Technical Report EDRC-18-59-96, Carnegie Mellon University, Engineering Design Research Center, 1996.

[55] G. Tidhar, A. S. Rao, and E. A. Sonenberg. Guided team selection. In *Proc. of ICMAS-96*, pages 369–376, Kyoto, Japan, 1996.

[56] W. E. Walsh and M. P. Wellman. A market protocol for distributed task allocation. In *Proc. of ICMAS-98*, Paris, France, 1998.

[57] L. Wang and W. Tsai. Optimal assignment of task modules with precedence for distributed processing by graph matching and state-space search. *Bit*, 28:54–68, 1988.

[58] M. P. Wellman. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research*, 1:1–23, 1993.

[59] E. Werner. Toward a theory of communication and cooperation for multiagent planning. In *Proc. of the Second Conference on Theoretical Aspects of Reasoning about Knowledge*, pages 129–143, Pacific Grove, California, March 1988.

[60] M. Williamson, K. Decker, and K. Sycara. Unified information and control flow in hierarchical task networks. In *Proceedings of the AAAI-96 workshop on Theories of Planning, Action, and Control*, 1996.

[61] L. Zhou. A new bargaining set of an n-person game and endogenous coalition formation. *Games and Economic Behavior*, 6:512–526, 1994.

[62] G. Zlotkin and J. Rosenschein. Negotiation and task sharing among autonomous agents in cooperative domain. In *Proc. of the 11th International Joint Conference on Artificial Intelligence*, pages 912–917, Detroit, MI, 1989.

[63] G. Zlotkin and J. Rosenschein. Cooperation and conflict resolution via negotiation among autonomous agents in noncooperative domains. *IEEE Transactions on Systems, Man, and Cybernetics, Special Issue on Distributed Artificial Intelligence*, 21(6):1317–1324, December 1991.

[64] G. Zlotkin and J. Rosenschein. Coalition, cryptography, and stability: Mechanisms for coalition formation in task oriented domains. In *Proc. of AAAI94*, pages 432–437, Seattle, Washington, 1994.