Overlay Networks for Task Allocation and Coordination in Large-scale Networks of Cooperative Agents

Panagiotis Karagiannis, George Vouros, Kostas Stergiou, Nikolaos Samaras

Abstract. This paper proposes a novel method for scheduling and allocating atomic and complex tasks in large-scale networks of homogeneous or heterogeneous cooperative agents. Our method encapsulates the concepts of searching, task allocation and scheduling seamlessly in a decentralized process where no accumulated or centralized knowledge or coordination is necessary. Efficient searching for agent groups that can facilitate the scheduling of tasks is accomplished through the use of a dynamic overlay structure of gateway agents and the exploitation of routing indices. The task allocation and the scheduling of complex tasks are accomplished by combining dynamic reorganization of agent groups and distributed constraint optimization methods. Experimental results display the efficiency of the proposed method.

Keywords: Task and resource allocation, coordination, cooperation, distributed constraint processing, cooperative agents.

Panagiotis Karagiannis, Nikolaos Samaras

Department of Applied Informatics, University of Macedonia, Greece

email: pkar@uom.gr samaras@uom.gr

George Vouros, Kostas Stergiou

Department of Information and Communication Systems Engineering, University of the Aegean, Greece

email: georgev@aegean.gr konsterg@aegean.gr

1 Introduction

This paper is concerned with the problem of distributed task allocation and coordination in large-scale networks of homogeneous or heterogeneous cooperative agents. Such a problem typically arises in peer-to-peer systems, grids and virtual organizations, in crisis management scenarios, etc. Decentralized control of large-scale systems of cooperative agents is a hard problem in the general case: The computation of an optimal control policy when each agent possesses an approximate partial view of the state of the environment (which is the case for large-scale distributed systems) and agents' observations are interdependent (i.e. one agent's actions affect the observations of the other) is very hard even if agents' activities are independent (i.e. the state of one agent does not depend on the activities of the other) [9]. Decentralized control of such a system in cases where agents have to act jointly is even more complex. In the case of *joint* activity, subsets of agents have to form teams in order to perform tasks subject to constraints: Acting as a team, each group has to be coordinated by scheduling subsidiary tasks with respect to temporal and possibly other constrains, as well as other tasks that team members aim to perform (e.g. as members of other teams).

Let us for instance consider a scenario (e.g. a terrorist attack) where crisis-management agents (policemen, fire brigades, health professionals, army forces) need to jointly perform numerous tasks within a certain time interval (i.e. agents have a limited time horizon due to the emergency of the situation). Each agent has its own capabilities and they all form an acquaintance network depending on the physical communication means available. While requests for joint activities arrive (e.g. new explosions, civilians requesting for help etc.) agents need to form teams to handle each individual situation. This comprises of three interleaved sub-problems: searching for the appropriate agents, allocating tasks to them according to their capabilities, and scheduling these tasks subject to temporal and other constraints. In contrast to previous works where these sub-problems are largely tackled independently, in this paper we view and solve the three-step process as a single problem.

The method we propose facilitates effective and efficient searching through the dynamic construction of overlay networks of gateway agents and the exploitation of routing indices. The search for the appropriate agents is done via the "heads" of services (gateway agents), which best know the availability and capabilities of subsidiaries (exploiting routing indices). However, since heads have to manage numerous incoming requests (e.g. emergent situations in the above scenario), they can propagate such requests to subsidiaries, which act so as to form task-specific teams depending on the requirements of each situation. After a team of agents is formed, the members of the team need to jointly schedule their activities taking into account interdependencies, as well as their other scheduled activities. This is in itself a hard problem which in our framework is tackled using a combination of dynamic team reorganization and distributed constraint optimization methods.

Being interested in the development of decentralized methods for the efficient task allocation and coordination in large multi-agent systems, we addressed the problem by building on self-organization approaches for ad-hoc networks, token-based approaches for coordination in large-scale systems, and distributed constraint satisfaction/optimization. In summary, we make the following contributions:

a. We propose a method that addresses searching, task allocation and scheduling in large-scale systems of cooperative agents. Specifically, we demonstrate how the interplay of simple searching, task allocation and scheduling techniques using routing indices, with the dynamic self-organization of the acquaintance network to an overlay network of gateway agents, can contribute to solving a complex problem in multi-agent systems efficiently. The applicability of our method is demonstrated by allocating temporally interdependent tasks with specific capability requirements to time-bounded agents.

b. We propose a generic method for task allocation and scheduling that combines dynamic reorganization of agent teams with distributed constraint optimization. Several versions of this method, which may differ either in the way task allocation or/and scheduling is performed, are implemented and compared experimentally.

c. We provide experimental results from simulated settings that demonstrate the efficiency of the proposed overall method and different variations of it. We compare two different configurations with distinct distributed constraint optimization methods and evaluate the efficiency of the task allocation and scheduling mechanism. Results show that the task allocation methods we propose coupled with the complete constraint optimization algorithm Adopt achieve quite promising results.

The rest of the paper is structured as follows: In Section 2 we discuss related work. In Section 3 we formally state the problem we deal with. Sections 4 and 5 describe the individual techniques employed for searching, task allocation and scheduling. In particular, Section 4 discusses the formation of the overlay network of gateways and the construction of routing indices, while Section 5 is concerned with the alternative methods for distributed constraint optimization used during the task allocation/scheduling phase. In Section 6 we present the overall method for integrating searching, task allocation and scheduling, outlining the different approaches to searching and task allocation. Experimental results are given in Section 7. Finally, Section 8 concludes the paper.

2 Related Work

In a large-scale system with decentralized control it is very hard for agents to possess accurate partial views of the environment, and it is even harder for agents to possess a global view of the environment. Furthermore, the agents' observations can not be independent, as one agent's actions can affect the observations of the others: for instance, when one agent leaves the system, then this may affect those agents that are waiting for a response; or when an agent schedules a task, then this must be made known to its team-mates who need to schedule temporally dependent tasks appropriately. This last example shows that transitions are interdependent too, which further complicates the computations.

Even if agents' activities are independent (i.e. the state of one agent does not depend on the activities of the other) [9] the overall complexity is exceedingly higher compared to centralized systems. Moreover, decentralized control of cooperative agent systems in cases where agents have to act jointly adds an even higher amount of complexity. In the case of joint activity, subsets of agents have to form teams in order to perform tasks, subject to ordering constraints. Acting as a team, each group has to be coordinated by scheduling subsidiary tasks with respect to temporal and possibly other constrains, as well as other tasks that team members aim to perform (e.g. as members of other teams).

Generally, in cases where the system has to allocate and schedule joint activities that involve temporally interdependent subsidiary tasks, the process involves at least (a) *searching*: finding agents that have the required capabilities and resources, (b) *task allocation*: allocating tasks to appropriate agents, and (c) *scheduling*: constructing a commonly agreed schedule for these agents. Each of these issues has received considerable interest in the literature.

The control process can be modelled as a decentralized partially-observable Markov decision process [9]. The computation of an optimal control policy in this case is simple given that global states can be factored, the probability of transitions and observations are independent, the observations combined determine the global state of the system and the reward function can be easily defined as the sum of local reward functions.

Decentralized control in systems where agents have to act jointly in the presence of temporal constraints, is a challenge. This challenge has been recognized in [13], where authors propose an anytime algorithm for centralized coordination of heterogeneous robots with spatial and temporal constraints, integrating task assignment, scheduling and path planning. In addition to providing a centralized solution, they do not deal with ordering constraints between tasks. Although the authors demonstrate that their method can find schedules for up to 20 robots within seconds, it is unclear how it would perform in networks with hundreds of agents. On the other hand, it has to be pointed that their algorithm, being efficient for small groups and providing error bounds, may be combined with our approach, given small groups of potential team-mates. However this is an issue for further research since the inclusion of ordering constraints will impact the solution complexity.

Extreme teams is a term coined in [23], emphasizing on four key constraints of task allocation: (a) domain dynamics may cause tasks to appear and disappear, (b) agents may perform multiple tasks within resource limits, (c) many agents have overlapping functionality to perform each task, but with differing levels of capability, and (d) inter-task constraints may be present. In this paper we deal with these four key issues, extending the problem along the following dimensions: (a) handling temporal constraints among tasks, (b) dealing with agents that do not have the capabilities to perform every task, and (c) integrating searching with task allocation and scheduling.

Token-based approaches are promising for scaling coordination to large-scale systems effectively. The algorithm proposed in [29] focuses on routing models and builds on individual token-based algorithms. Additionally, in [29] authors provide a mathematical framework for routing tokens, providing a three-level approximation to solving the original problem. Token-

based approaches do not inherently deal with scheduling constraints and dynamic settings. In our approach, tokens concerning the availability of resources and capabilities are being used for constructing agents' partial views of the network status using routing indices. Routing is further restricted to the connected dynamic sub-network of gateway agents which manage searching.

In [15,16] a protocol was proposed for solving a distributed resource allocation problem while conforming to soft real-time constraints in a dynamic environment. Although this is not the same problem as task allocation, it is worth mentioning that the approach towards solving the resource allocation problem taken in [15] has certain similarities with our approach. To be precise, resource allocation was solved modelled as an optimization problem, similar to a Partial Constraint Satisfaction Problem. The protocol of [15] uses constraint satisfaction based pruning techniques to cut down the search space, coupled with a hill climbing procedure.

It has to be noticed that in this paper we do not deal with communication decisions for optimizing information sharing/exchange as done in [10], or for proactively exchanging information [34]: This is orthogonal to our research which may further increase the efficiency of the proposed method. However, we point that this can not be done in any way such that agents share a global view of the environment state (as in [22, 30, 31]).

The (C_TÆMS) representation [2] is a general language (based on the original TÆMS language [6]) widely used for distributed planning and scheduling in multiagent systems. In order to deal with uncertainties, C_TÆMS tasks have probabilistic utilities and durations. Agent coordination and scheduling in dynamic and uncertain environments using C_TÆMS has been in addressed by various approaches within the DARPA Coordinators program [14, 19, 25]. These approaches are more general than ours in the type of tasks they consider, since for the purposes of this paper we limit ourselves to tasks with fixed duration. Musliner et al. use distributed Markov Decision Processes (MDPs) as the underlying formalism to capture uncertainty [19]. Smith et al. use Simple Temporal Networks (STNs) to infer feasible start times for the tasks allocated to agents in the system [25]. Once dynamic changes occur, agents heuristically determine task insertions in their schedule and change the STN constraints in order to make such insertions feasible. Finally, [14] introduced the *Predictability and Criticality Metrics (PCM)* approach in which dynamic changes are handled by making schedule modifications, chosen heuristically through simple metrics from within a set of possible modifications that can increase the team utility.

In a more relevant note to our work, in [26] the authors solve a class of multi-agent task scheduling problems by mapping specifications expressed in a subset of C_TÆMS to distributed constraint optimization problems (DCOPs) and hence allowing the use of algorithms such as Adopt [18]. Apart from the mapping, this work focuses on the use of constraint propagation to prune the domains of the variables in the resulting DCOPs and hence achieve efficient solving. We should note that this work and all the C_TÆMS ones mentioned above, focus on how to handle distributed planning and scheduling. In contrast, the approach we present here views the whole process of task allocation and scheduling as tightly interconnected problem and to this extend we propose a combination of methods, concerning all of its aspects, to tackle it.

Finally, we need to point out that the approach presented in this paper extends the work proposed in [27]. Indeed, this paper is an extended version of [27] that explains the overall approach in more detail and, most significantly, it presents and evaluates configurations of the overall method using new methods (compared to that of [27]) for task allocation and scheduling. Experimental results given in Section 7 demonstrate that the new methods outperform the ones proposed in [27].

3 Problem Formulation

In this section we formally define the problem we deal with, discussing the various assumptions made, and we give an overview of our proposed approach. We first describe the setting of the agent network and the type of task requests we consider.

3.1 Agent Network

In this paper we assume large-scale networks of collaborating agents that are distributed geographically. Therefore, the network's connectivity factor varies between different geographical regions. The acquaintance network of agents is modelled as a connected graph $G=(N,E)$, where N is the set of agents and E is a set of bidirectional edges denoted as non-ordered pairs (A_i,A_j). The (immediate) neighbourhood of an agent A_i includes all the one-hop away agents (i.e. each agent A_j such that $(A_i, A_j) \in E$. The set of neighbours (or acquaintances) of A_i is denoted by $N(A_i)$.

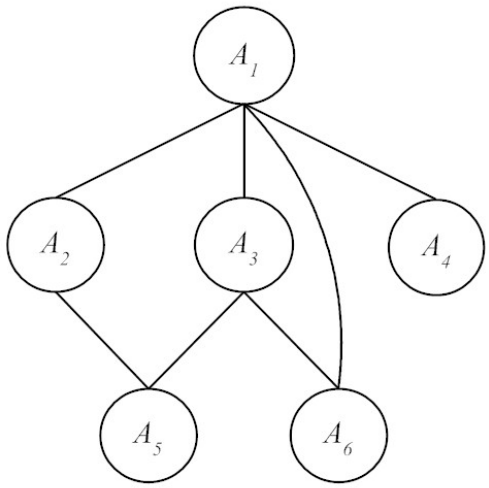
Agents are considered to be time-bounded with specific discrete capabilities. We assume that each agent has a single type of resource and at each time point at most one of the requested tasks (presented in subsection 3.2) can use any such resource. Under these assumptions the problem of allocating tasks to the resources becomes equivalent to the problem of finding available time intervals in the agents' schedule to allocate to the various tasks. Therefore we can consider that the only resource that an agent manages is a set of time units. These assumptions are not restrictive as the method can be extended to other resources, which can be treated mostly as we treat the availability of time units or agents' capabilities.

Therefore, each agent A_i is being attributed with a set of capability types $Cap_i=\{cap_{i1}, cap_{i2}... cap_{im}\}$, a set of time-units $R_i=\{r_{i1}, r_{i2},...,r_{in}\}$, totally ordered with a relation "*consecutive*" denoted by "<", and a priority P_i which is a positive integer. The function *earliest*(R) (resp., *latest*(R)) denotes the time point r_{ik} in $R, R \subseteq R_i$, for which there is not any other point r in R with $r < r_{ik}$ (resp. $r_{ik} < r$). The priority is a unique identifier that is assigned to each agent when it enters the multi agent system [3]: This is necessary for the computation of an overlay network (explained in Section 4.1) and for the non-cyclic update of routing indices (explained in 4.2), and it may depend on several factors such as the battery life of the agent, its availability of resources, its capabilities etc. The initial state of an example network is shown in Figure 1. In this network each agent has 10 consecutive time-units, which are available for allocation, and a specific set of capabilities.

3.2 Task Requests

$N = \{A_1, A_2, A_3, A_4, A_5, A_6\}$

$E = \{(A_1,A_2), (A_1,A_3), (A_1,A_4), (A_1,A_6), (A_2,A_5), (A_3, A_5), (A_3,A_6)\}$



For all A_i $i = 1,2,...,6$ $earliest(R_i) = 0$, $latest(R_i) = 10$

A_1: $cap_1 = \{1,3\}$, $P_1 = 1$, A_4: $cap_4 = \{2,3\}$, $P_4 = 4$

A_2: $cap_2 = \{1,2,3\}$, $P_2 = 2$ A_5: $cap_5 = \{3\}$, $P_5 = 5$

A_3: $cap_3 = \{1\}$, $P_3 = 3$ A_6: $cap_6 = \{1,2\}$, $P_6 = 6$

Figure 1. A network of acquaintances

We assume that there is a set of requests concerning k independent tasks $T = \{t_1,..., t_k\}$. Each task t_i can be either *atomic* or *complex*, in which case it may require the joint achievement of a set of atomic subtasks $\{g_{i1}, g_{i2},.., g_{ik}\}$. Atomic tasks require the commitment of a single agent. Each atomic task t_i (or subtask g_{il}) is a tuple $<a_i, start_i, end_i, Cap_i>$, where α_i, $start_i$ and end_i, are non-negative integers representing the maximum number of time units that the task needs to be completed, the earlier time point when t_i should start to be executed and the latest time point that t_i should finish executing, respectively. Cap_i is the set of agent capabilities that are required for the successful execution of the task.

For each complex task t_i consisting of a set of sub-tasks $\{g_{i1},.., g_{ik}\}$ there is also a set of constraints C_i corresponding to the interdependencies between t_i's subtasks. In this paper we consider binary precedence constraints specifying temporal distances between the executions of

subtasks. For example, constraint $start(g_{ij})+3 \leq start(g_{il})$ means that the execution of subtask g_{il} must start at least 3 time units after the execution of subtask g_{ij}.

3.3 Problem Specification and Overview of our Approach

Given the above, the problem that this article deals with is as follows: Given a network of agents $G=(N,E)$ and a set of requests for performing a set of tasks T, we require

(a) for each atomic task $t_i = <a_i, start_i, end_i, Cap_i>$ in T, to find an agent A_j in N, such that $perform(A_j, t_i)=1$.

This holds if A_j has the required capabilities and time resources:

$\{(Cap_i \subseteq Cap_j) \wedge (\exists\ R \subseteq R_j$ s.t. $|R| \geq a_i \wedge start_i \geq earliest(R) \wedge latest(R) \geq end_i)$, where R is an interval of consecutive time points$\}$. On the contrary, $perform(A_j, t_i)=0$

(b) for each joint (complex) task t_i in T with subsidiary tasks $\{g_{i1},..., g_{ik}\}$ to find a set of agents G that form a network of potential team-mates (PTN) such that

(i) for each sub-task there is an agent in G that can perform it: $\sum_{gik} perform(A_l, g_{ik}) = |\{g_{i1},..., g_{ik}\}|$

(ii) the precedence constraints between the subtasks of t_i are satisfied.

For that purpose, following [17], we assume that for each constraint $c_{kl} \in C_i$ between two subsidiary tasks g_{ik} and g_{il} there is a *cost function* f_{kl} providing a measure of the constraint's violation. The total satisfaction of the complex task's constraints is defined as an aggregation of these cost functions. In Section 5.1 we explain in detail how complex tasks are modeled, how the cost functions are evaluated and how their aggregation is computed.

The aim is twofold. First, to increase the benefit of the system, i.e. the ratio of tasks successfully allocated to the total number of requests. As explained above, we consider an atomic task to have been successfully allocated if the system has located an agent that has the available resources and capabilities to serve the requested task. A complex task is considered successfully allocated if a team of agents is formed such that all sub-tasks of the complex task are successfully allocated and all inter-task constraints are satisfied. Our second goal is to increase the message gain, i.e. the ratio of the benefit to the number of messages exchanged.

To achieve these goals we propose an approach that builds on self-organization approaches for ad-hoc networks, token-based approaches for coordination in large-scale systems, and distributed constraint optimization. Namely, the agents in the network are organized in a dynamic overlay structure consisting of dominating nodes, which act as "heads of service", and non-dominating ones which only provide resources (see Section 4.1). The dominating nodes are responsible for forwarding task requests through the network as they are the ones that have some, incomplete, view of the available resources. This is achieved by equipping each dominating node with a routing index which is a compact summary of the resources available through this node and its neighbours (see Section 4.2). Once a complex task request enters the system it is forwarded through the network until a set of agents that have the available resources and capabilities to serve it are located. The way the task "travels" through the network, either broken down to its subtasks or as a whole, is determined by certain heuristics that are explained in Section 6. Once an appropriate set of agents is located, they receive the constraints between the subtasks of the complex task; they form a potential team, and try to find a joint schedule for the task so that the interdependencies between its subtasks are satisfied. This is done using distributed constraint optimization techniques (see Section 5). If no satisfying schedule is found then the potential team is reorganized by releasing one or more agents from the team and having other appropriate agents enter it. This process, which is explained in Section 6, is repeated until the task has been successfully scheduled or its deadline expires.

4 Self-Organization and Searching

This section presents the individual techniques employed in the proposed method. Specifically, it describes the construction of dynamic overlay networks of gateway agents, and the construction and maintenance of routing indices.

4.1 Dynamic Overlay Networks of Gateways

A dominating set of nodes in a network is a connected subset of nodes that preserves and maintains the "coverage" of all the other nodes in the network [4]. In a connected network where

each node owns a distinct priority, a node *A* is fully covered by a subset *S* of its neighbours in case the following hold [3]:

- *S* is connected
- Each neighbour (excluding the nodes in *S*) of *A* is a neighbour of at least one node in *S*
- All nodes in *S* have higher priority than *A*.

A node belongs to the dominating set if no subset of its neighbours fully covers it.

Although originally proposed for area coverage and monitoring [3, 5], nodes may be considered to cover an information space, or the space of capabilities and/or resources required for the execution of tasks. The algorithm of Dai and Wu [5] for the computation of a dominating set of nodes allows each node to decide about its dominating node status without requiring excessive message exchange and based only in local knowledge: the knowledge of a node's neighbours is sufficient [3]. The algorithm is as shown in Table 1:

1. Collect information about one-hop neighbours and their priorities 2. Compute the sub-graph of neighbouring nodes that have higher priority 3.**If** (this sub-graph is connected **and** (every one-hop neighbour is in this sub-graph **or** it is the neighbour of at least one node in the sub-graph)), **then** *opt* for a non-gateway node. **Else** *opt for* a gateway node.

Table 1. Computing the gateway status of an agent

Dominating nodes (gateway agents) are dynamically computed in case the acquaintance network changes. Having computed an overlay network of gateways that constitute the backbone of the system and "cover" all the other (non-dominating or non-gateway) nodes in the network, the propagation of requests and indices' updates can be restricted to this set of nodes. Specifically, according to our approach, gateway agents have the responsibility to forward requests to their neighbours and keep a record of their neighbours' availability and capabilities. Non-gateways forward all requests to gateways and possess only information about their own availability. Therefore, as the percentage of gateways in a network decreases, the searching task is expected to become more efficient, although the maximum workload of the agents is expected to increase.

A simple network of 10 agents is depicted in the following figure for illustration purposes. The form of the presented example network is similar to the ones produced in any of the presented of experiments sets. Here, we consider nodes with low index numbers having the highest priorities (e.g. agent 0 has higher priority than agent 1).

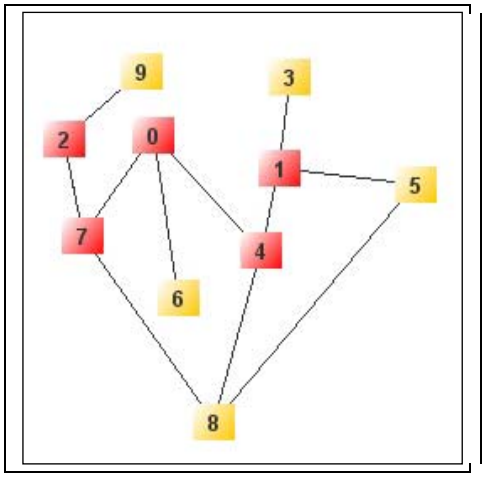


Figure 2. Example of a simple network of 10 agents with 3 edges at maximum for each one. Agents 0,1,2,4 and 7 are gateway agents.

4.2 Routing Indices

Given a network of agents *G=(N,E)*, and the set of neighbours *N(A)* of an agent *A* in *N*, the routing index (RI) of *A* (denoted by *RI(A)*) is a collection of $|N(A) \cup \{A\}|$ vectors of resources' and capabilities' availability, each corresponding to a neighbour of *A* or to *A* itself. Given an agent A_i in *N(A),* then the vector in *RI(A)* that corresponds to A_i is an aggregation of the resources that are

available to A *via* A_i. The key idea is that given a request, A will forward this request to A_i if the resources/capabilities available to the network *via* A_i can serve this request.

As it can be understood from the above, the efficiency and effectiveness of RIs depend on the way availability is being modelled and on the way this information for a set of agents is aggregated in a single vector [4].

To compactly capture information about the availability of time units, each agent A_i has a time vector V_i of m tuples $<j,s>$ representing the time-units available to the agent. Each non-negative integer s represents the number of consecutive non-allocated time-units that follow the time point j, $0 \leq j \leq m$. This vector can be depicted as a time line of m points. For example, the time vector $Vi=(<0,3>,<1,2>,<2,1>,<3,0>,<4,0>,<5,1>,<6,0>,<7,3>,<8,2>,<9,1>)$ represents the time line shown in Figure 3 with m=10. The vector specifies the existence of 3 available time-units starting from the point 0, 1 available time unit after the point 5, and 2 time units after the time point 7.

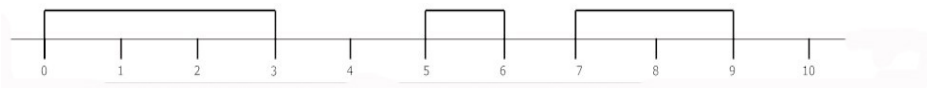


Figure 3. Vector of time-units availability

Assuming that m (the total number of time units per agent) is constant for all agents, given two time vectors V_i and V_j, their aggregation, denoted by $agg(V_i,V_j)$, is a vector that comprises elements $<j_{ak},s_{ak}>$, with $0 \leq k \leq m$, such that, given the elements $<j_{ik},s_{ik}>$ and $<j_{lk},s_{lk}>$ of V_i and V_j respectively, with $j_{ik}=j_{lk}$, then $j_{ak}=j_{ik}= j_{lk}$, and $s_{ak}=max(s_{ik},s_{lk})$. For instance, given the tuples $<j_{i4}, s_{i4}>$= <4,4> and $<j_{j4}, s_{j4}>$= <4,0> the corresponding tuple in the aggregation is $<j_{a4}, s_{a4}>$=<4, max(4,0)>=<4,4>. This type of aggregation can generally be applied to any type of resources that can be committed to a single request.

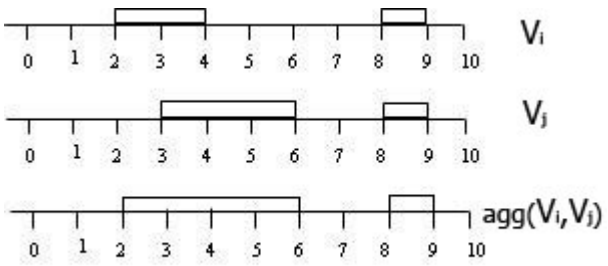


Figure 4. Aggregation of vectors

Figure 4 shows an example for aggregating the vectors V_i and V_j of two agents A_i and A_j respectively with m=10. The vectors are depicted as time lines. More precisely, agent A_i has committed to allocate the intervals (2,4) and (8,9) to two tasks. Similarly, the agent A_j has committed to allocate the interval (3,6) to one task and (8,9) to another. The resulting vector ($Agg(V_i,V_j)$) shows the availability of the two nodes as a whole, without distinguishing the availability of each node.

The aggregation of all vectors in the routing index of an agent A gives information about the maximum availability of any agent in $N(A) \cup \{A\}$. Specifically, given $RI(A)$, A updates the indices of its neighbour A_i by sending the aggregation of the vectors of the agents in $N(A) \cup \{A\} \setminus \{A_i\}$ to A_i.

Every time the vector that models the availability of resources in a node changes, the node has to compute and send the new vector of aggregations to the appropriate neighbours. Then, its neighbours have to propagate these updates to their neighbours and so on, until they reach nodes whose routing indices are not affected.

To capture the availability of time-units in conjunction to the availability of capabilities, we extend RIs to multiple routing indices (MRIs). An *MRI* for the agent A_i, $MRI(A_i)$, has the generic form $\{RI_{cap1},..., RI_{capm}\}$. Each RI_{capj} represents the available resources that A_i can reach through neighbours that own the capability cap_j. *MRIs* not only provide an agent with the information about the temporal availability of its neighbours, but they also capture information about the available agents that own specific capabilities.

Routing indices are rather problematic when the updates of indices propagate in cycles [4]: In the worst case, information about resources availability is misleading, leading to inefficient search mechanisms. Although cycles can be detected, known techniques are not appropriate for open networks where network configurations change frequently. We have managed to deal with cyclic updates of agents' indices by forcing each agent to propagate indices' updates only to neighbouring agents with higher priority. Considering that routing indices are being maintained only by gateway agents, these record the corresponding indices for the non-gateway neighbours and the aggregations of indices for gateway neighbours with lower priority. Updating the indices of gateways by aggregating the indices of their gateway neighbours with lower priority has the following effects: (a) Since agents have distinct priorities, indices can not be updated in a cyclic way, avoiding the distracting affects of cycles to searching. (b) Priorities denote the "search and bookkeeping abilities" of agents: Agents with high priorities index their neighbours and guide search. (c) Gateways with lower priority do not know about the indices of their gateway neighbours with higher priority. Since requests propagate from low to high priority gateway agents, some of the high priority gateways may function like "traps" since they may not know how to fulfil or propagate requests. Experiments showed that the benefits gained by avoiding cycles outweigh this disadvantage.

As agents schedule tasks, the availability of agents with certain resources and capabilities changes, causing the update of indices. The update of MRIs due to this phenomenon is done dynamically, albeit not instantly, and in parallel to the propagation of requests.

5 Distributed Constraint Solving

A *Distributed Constraint Satisfaction Problem* (dis CSP) consists of a set of agents, each of which controls one or more variables, each variable has a (finite) domain, and there is a set of constraints [33]. Each constraint is defined on a subset of the variables and restricts the possible combinations of values that these variables can simultaneously take. A constraint that involves variables controlled by a single agent is called an *inter-agent* constraint. One that involves constraints of different agents is called an *intra-agent* constraint. A *Distributed Constraint Optimization Problem* (DCOP) is a dis CSP with an optimization function which the agents must cooperatively optimize. The DCOP framework has recently emerged as a promising framework for modelling a wide variety of multi-agent coordination problems. Such problems are, for example, distributed planning, distributed scheduling and distributed resource allocation. We now present the way complex tasks are modelled as DCOPs in our framework and then we describe the constraint solving techniques we have used for efficient task allocation and scheduling.

5.1 Modeling Complex Tasks as DCOPs

To efficiently capture interdependencies between the subtasks of a complex task, we model complex tasks as DCOPs. We assume that for each complex task t consisting of a set of sub-tasks $\{g_1,...,g_k\}$ and a set of constraints C_t, a set of agents $A=\{A_1,...,A_m\}$, with $m \leq k$, is located using the gateway and routing indices searching infrastructure described in Sections 4.1 and 4.2. This is also explained in Section 6. Each $A_j \in A$ has the necessary capabilities and resource availability to undertake at least one subtask. The DCOP model is as follows:

- For each subtask $g_i=<a_i, start_i, end_i, Cap_i>$ there is a variable X_i controlled by an agent $A_i \in A$, corresponding to the start time of g_i.
- The domain $D(X_i)$ of each variable X_i is a set $R \subseteq R_i$ of time points that agent A_i can allocate to the start time of g_i (R_i is the total set of time units for the agent A_i). $D(X_i)$ includes each time point $r \in R_i$ such that all time units between r and $r+\alpha$ are available on A_i's time line.
- There is a hard intra-agent constraint between any two subtasks that are allocated to an agent A_i, specifying that the execution of the two subtasks cannot overlap[1]. To be precise, for any two subtasks g_i and g_l allocated to A_i, with durations α_i and α_l respectively, there is a hard disjunctive intra-agent constraint c_{il} $(\in C_t) = (X_i + \alpha_i \leq X_l) \vee (X_l + \alpha_l \leq X_i)$. Note that such a constraint exists between any two subtasks (or atomic tasks in general) that are allocated to the same agent even if they belong to different tasks. This occurs when an agent participates in the allocation of more than one complex task.

[1] This is under our assumption that each resource can be used by at most one task at any point in time. In general, these constraints depend on the type of resource.

- A binary precedence constraint $c_{il} \in C_t$ between two variables X_i and X_l is modelled as a *soft* constraint with a cost function $f_{il} : D(X_i) \times D(X_l) \rightarrow N$ which associates a cost to each pair of value assignments to X_i and X_l (similarly to [18]). Recall that a precedence constraint specifies a temporal distance between the executions of the corresponding subtasks. For example, the constraint $X_i + \alpha_i + 5 \leq X_l$ specifies that the execution of g_l must start at least 5 time units after g_i has finished. Such a constraint may be inter-agent if the two subtasks have been allocated to different agents or intra-agent if the two subtasks have been allocated to the same agent. The cost function f_{il} is defined as follows:

$$f_{il}(d_i, d_l) = \begin{cases} 0, \; iff \; d_i \; \& \; d_l \; satisfy \; c_{il} \\ M > 0, \; otherwise \end{cases}$$

where $d_i \in D(X_i)$, $d_l \in D(X_l)$ and M is a non negative number. M is a measure of the constraint's violation "degree", defined as the minimum distance that the starting time of one of the subtasks has to be shifted on the time line of the corresponding agent in order for the constraint to be satisfied. Agents try to minimize the cost function associated with such a constraint by repeatedly changing the values of the variables they control. For example let us consider the constraint c_{il}: $X_i + \alpha_i + 5 \leq X_l$ and assume that X_i and X_l take values d_i and d_l respectively such that: $d_i + \alpha_i + 5 > d_l$. Since the constraint is violated, one of the agents that control variables X_i and X_l must change the value of its variable at least by M= $d_i + \alpha_i + 5 - d_l$ in order to satisfy the constraint. For example a new value for X_l that satisfies the constraint could be $d_l' = d_l$+M. Generally, given a constraint $c: Q \leq K$, M is defined as follows:

$$M = \begin{cases} 0, \; Q \leq K \\ Q - K, Q > K \end{cases}$$

Similarly we can define M for other types of interdependencies. For example, given a constraint $c: Q < K$, M is defined as follows:

$$M = \begin{cases} 0, \; Q < K \\ Q - K + 1, Q \geq K \end{cases}$$

Collectively, the agents in A try to minimize the aggregated function $F(C_t)$ which is a measure of violation for all the constraints of a complex task t :

$$F(C_t) = \sum_{\forall X_i, X_l} f_{il}(d_i, d_l)$$

Note that in the general case no agent in A has complete knowledge of the function $F(C_t)$. Hence, the use of a distributed constraint optimization method by the agents in A is a necessity. In this paper we consider summation as the aggregation operator for the optimization functions, but this is not a requirement.

5.2 Techniques for the Scheduling of Complex Tasks

To achieve efficient task allocation and scheduling in dynamic settings, where several requests may enter the system simultaneously, it is essential that a fast, scalable, and dynamically adaptable method is used. Several distributed constraint satisfaction and optimization techniques have been proposed in the literature. For example, there exist simple distributed local search methods, such as the distributed stochastic algorithm (DSA) [8], and the distributed breakout algorithm [11]. These are fast and scalable methods but as a downside they are incomplete. On the other hand, a number of complete distributed CSP and DCOP algorithms have been proposed, such as Adopt [18], DPOP [21] and their extensions (e.g. [1, 6, 32]). These methods guarantee that the computed results are optimal and have been shown to perform satisfactorily in terms of run time for agent networks of up to medium size. However, they cannot yet handle large networks. As will be explained in detail below, in the context of this work we do not consider very large DCOPs. Although agent networks may consist of many hundreds, or even thousands, of nodes, it is safe to assume that complex tasks that enter the network will consist of relatively few subtasks in most practical cases. Hence, by modelling each complex task as a separate DCOP, the sizes of the DCOPs that are generated can be efficiently handled by methods such as Adopt.

The task allocation and scheduling approach we follow in this paper combines distributed constraint optimization with dynamic agent team reorganization. Concerning the scheduling of tasks via constraint optimization, we have implemented and compared two different approaches; the first one is an incomplete local search method based on DSA, and the second one is a complete optimization algorithm based on Adopt. Before going into further details in the subsections that follow, let us briefly explain our approach.

Given a complex task t, and assuming that there is a set of agents to which the atomic subsidiary tasks of t have been allocated (i.e. a Potential Teammates' Network – PTN), the agents in the PTN apply a DCOP algorithm. If a solution is found then t is considered as successfully scheduled and the agents in PTN form a network of teammates. If no solution is found, or a time-

out occurs, then PTN self-organizes. That is, one or more of the agents in the PTN exit the PTN, tasks are being allocated to new agents that have the appropriate capabilities, and thus, these agents join the PTN. Then, the agents in the new PTN try again to schedule the subtasks of *t* using a DCOP algorithm. This process continues until *t* is successfully scheduled or a time-out occurs, in which case *t* is considered as a failed request.

Before describing in detail in the two DCOP methods that have been implemented (subsections 5.2.1 and 5.2.2), we give a motivating example.

Example 5.1

Consider a task with 3 individual subtasks $t_i = \{g_{i1}, g_{i2}, g_{i3}\}$ that enters the agent network. For the individual sub-tasks we can assume the following:

$g_{i1}=<a_{i1}=1, start_{i1} \geq 0, end_{i1}<5, Cap_{i1}=1>$

$g_{i2}=<a_{i2}=2, start_{i2} \geq end_{i2}, end_{i2}<8, Cap_{i2}=1>$

$g_{i3}=<a_{i3}=2, start_{i3} \geq end_{i3}, end_{i3}<10, Cap_{i3}=1>$

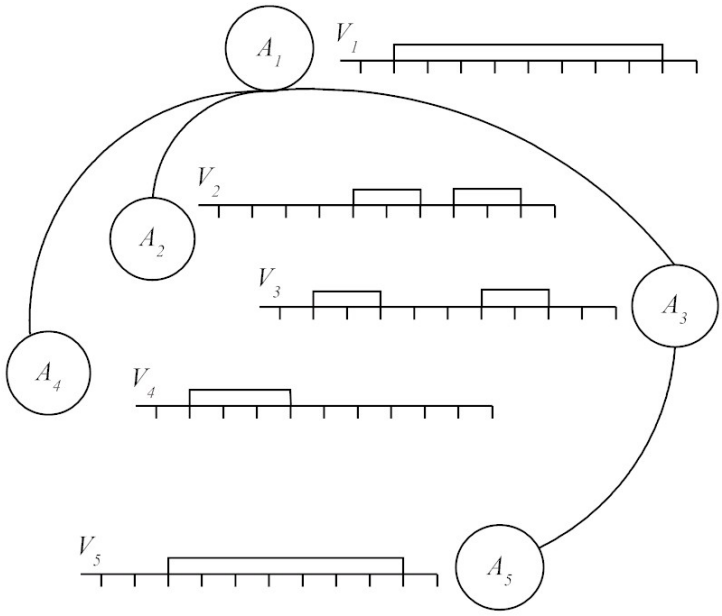
Let us also assume that we have an agent network $G=(N,E)$ of 5 agents defined as follows:

$N = \{A_1, A_2, A_3, A_4, A_5\}$.

$E = \{(A_1, A_2), (A_1, A_3), (A_1, A_4), (A_3, A_5)\}$.

The agent network and the availability of resources for each agent are shown graphically in Figure 5. For simplification reasons and without loss of generality we can assume that all capabilities (for the tasks and agents) have the same value ($Cap = 1$).

In Figure 5 there are 2 gateway agents (A_1, A_3) and the priorities between them are set according to their index (i.e. A_1 has lower priority than A_3 since its index is lower). Consequently only A_3 has a complete view of all the network's resources. A_1 can only record information about the availability of itself and of agents A_2, A_4. Routing indices are not shown as they are simple to compute. Also, the availability of each agent in shown in vector form.



$V_1=(<0,1>,<1,0>,<2,0>,<3,0>,<4,0>,<5,0>,<6,0>,<7,0>,<8,0>,<9,1>)$

$V_2=(<0,4>,<1,3>,<2,2>,<3,1>,<4,0>,<5,0>,<6,1>,<7,0>,<8,0>,<9,1>)$

$V_3=(<0,1>,<1,0>,<2,0>,<3,3>,<4,2>,<5,1>,<6,0>,<7,0>,<8,2>,<9,1>)$

$V_4=(<0,1>,<1,1>,<2,0>,<3,0>,<4,6>,<5,5>,<6,4>,<7,3>,<8,2>,<9,1>)$

$V_5=(<0,2>,<1,1>,<2,0>,<3,0>,<4,0>,<5,0>,<6,0>,<7,0>,<8,0>,<9,1>)$

Figure 5. A network of acquaintances

Assuming that the gateway agent A_1 has received the task request, it performs a search in its immediate neighbourhood in order to find agents that can provide the requested resources for the satisfaction of the task. As already described, resources belonging to A_3 are completely invisible to A_1, since the priority of the latter is lower than that of the former. Consequently, A_1 will try to

allocate the task to itself and/or to agents A_2, A_4. One possible assignment is {(A_1 , g_{i1}), (A_2 , g_{i2}), (A_4 , g_{i3})} where each tuple denotes a possible sub-task to agent allocation. Doing so, the three agents A_1, A_2, A_4 form a PTN and will jointly try to schedule the three subtasks.

The next step after the formation of the PTN consists of the construction of a new overlay network according to the inter-agent constraints among the agents in the PTN (let is call this type of overlay networks "c-overlay"). As far as the DCOP algorithms are concerned, two agents <A_i, A_j> are considered neighbours in the c-overlay network if there is a constraint between their variables. As with the overlay of gateways (with which c-overlay networks should not be confused), the c-overlay network is constructed on-demand and dynamically depending on the existing inter-agent constraints. This is done by any PTN that has been formed in order to schedule a complex task. Such a network is either reformed when the PTN dynamically re-organizes, or it is dissolved when PTN is dissolved. The later happens when the complex task has been either allocated successfully or its allocation has failed.

Since an agent can be a member of more than one PTN that executes a DCOP algorithm, it can simultaneously belong to many c-overlay networks. But since each DCOP is for scheduling a different task, messages created for a single c-overlay network cannot travel on edges belonging to another such network.

Returning to Example 5.1, once the three agents in the PTN have received the subtasks allocated to them, they will try to jointly schedule the atomic subsidiary tasks of the complex task by modelling it as DCOP. In this case, the c-overlay network shown in Figure 6 will be formed.

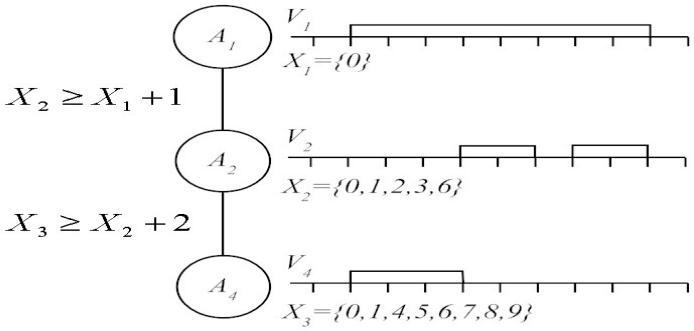


Figure 6. A c-overlay network for the network of agents in Figure 5

Since there are three subtasks, there are three corresponding variables X_1, X_2, X_3 that are controlled by agents A_1, A_2, A_4, respectively. Constraints are depicted as bi-directional edges beetwen agents. The constraints that involve one variable (unary constraints) are enforced on the domains of the corresponding variables. For example, value *9* for agent A_1 is not included in the set of potential values for X_1 , since this variable must be smaller than 5.

This is the initial state of the DCOP solving process. Henceforth, each agent will assign values to its variables, send messages to other agents about its state, and react to incoming messages according the DCOP algorithm used. Message exchange for the DCOP is carried out exclusively on the c-overlay network constructed due to the constraints in the DCOP for this specific complex task (e.g. due to the network depicted in Figure 5).

5.2.1 Solving DCOPs using Local Search

The first method for solving DCOPs that we implemented and evaluated is a local search procedure. This is an instantiation of the Distributed Stochastic Algorithm (DSA) [8]. In this method each agent may control more than one variable and the optimization criterion takes into account the cost functions of the constraints involved. As demonstrated in [35], DSA displays good performance in certain constraint optimization problems compared to distributed breakout. In its simplest form the local search method is based on a min conflicts hill-climbing procedure [17] without stochastic moves. Figure 7 depicts this basic algorithm.

1: assign values to variables so that the aggregation of cost functions is minimized
2: **while** (termination condition has not been met) **do**
3: **for** each new value assignment of a variable *X*
4: send the new value to agents that control a variable involved in a constraint with *X*
5: **end for**
6: collect the neighbours' new values, if any, and compute constraint violations and cost functions

7: choose values for the variables so that the aggregation of cost functions is minimized
8: assign the selected values to the variables
9: **end while**

Figure 7. Basic Local Search algorithm executed by all agents

Let us now describe the operation of the basic algorithm. Each agent A_j initially assigns values to its variables so that the sum of the cost functions is minimized. This can be a dynamic process since subtasks may dynamically allocated to the agent A_j during the operation of the system. In other words, as soon as an agent receives a subtask g_i, it tries to assign a value to X_i (i.e. find a start time for g_i) that minimizes the sum of the cost functions. This may involve changing the assignment of other variables that A_j controls (line 6). If a new assignment of some variable X is made, this is communicated to A_j's neighbours in the corresponding c-overlay network (i.e. the agents that share constraints with A_j involving variable X). All similar assignments made by A_j's neighbours are collected and the constraint violations are recomputed. This may lead to the assignment of new values to A_j's variables (line 7). The choice of value assignments is an important issue of the algorithm. The basic algorithm makes a new assignment only if it can reduce the sum of the cost functions. Note that no single agent has complete knowledge of the cost function $F(C_t)$ of a complex task t (unless all subtasks of t are assigned to a single agent). Therefore, each agent tries to minimize the aggregation of the cost functions for the constraints that it is "aware of". That is, the constraints that involve variables that it controls. In case no assignment that reduces the aggregation of the cost functions can be made, then the agent will not make any change to its variable assignments. In this case, the agent has reached a local optimum and will have to wait for messages from its neighbours in the corresponding c-overlay network.

This basic algorithm may quickly reach a local optimum as it is not equipped with any technique for escaping such situations (e.g. stochastic moves). In case all agents in the team reach a local optimum, or a termination condition related to the time allowed for constraint solving is met, then, as already explained, the PTN re-reorganizes itself. Self-organization is further explained in Section 6.

Note that local search methods comply with the requirements for speed and dynamicity, but on the other hand, because of their inherent incompleteness, may not find optimal (or simply feasible) allocations, even if they exist. This means that some complex tasks may not be served, although there may be agents in the system that can cooperatively serve them.

5.2.2 Solving DCOPs using Adopt

Alternatively to the Local Search method we evaluated an alternative configuration of the proposed method using the complete DCOP algorithm Adopt [18]. The majority of the existing methods for DCOP (e.g. local search methods) are not able to provide theoretical guarantees on global solution quality given that agents have to operate asynchronously. Nevertheless, we can overcome this disadvantage by allowing agents to make local decisions based on cost estimates. This approach, introduced in [18], results in a polynomial-space algorithm for DCOP named *Adopt*. Adopt guarantees a globally optimal solution. Furthermore it allows agents to execute asynchronously and in parallel. As noted in [18] "The Adopt algorithm consists of three key ideas: a) a novel asynchronous search strategy where solutions may be abandoned before they are proven suboptimal, b) efficient reconstruction of those abandoned solutions, and c) built-in termination detection". A sketch of Adopt's operation is as follows:

First, agents form a prioritized tree structure. The priorities in this structure are decided after considering the constraints between variables inherent in the CSP problem which we have to solve. In [18] there is a precise explanation on how this is done. The priority ordering is then used to perform a distributed backtrack search using a best-first search strategy. To be more precise, based on the current available information, each agent keeps on choosing the best value for its variables. That is, each agent always chooses the variable value which minimizes its lower bound as defined in [18].

To efficiently reconstruct a previously explored solution, Adopt uses a stored lower bound as a backtrack threshold. When an agent knows from previous search experience that lb is a lower bound for its subtree, it should inform the agents in the subtree not to bother searching for a solution whose cost is less than lb. Bound intervals track the progress towards the optimal solution. This is the core of the built-in termination detection mechanism. A bound interval consists of both a lower bound and an upper bound on the optimal solution cost. When the size of the bound interval shrinks to zero (the lower bound equals the upper bound) the cost of the optimal solution has been determined and agents can safely terminate when a solution of this cost is

obtained. This technique increases the efficiency of the algorithm. Furthermore it requires only polynomial space in the worst case.

The prioritized Depth-First Search (DFS) tree defines parent and child relationships and of course priorities between the agents. Variable value assignments (VALUE messages) are sent down the DFS tree while cost feedback (COST messages) propagate back up the DFS tree. It may be useful to view COST messages as a generalization of NOGOOD messages from DisCSP algorithms. THRESHOLD messages are used to reduce redundant search and sent only from parents to children. A THRESHOLD message contains a single number representing a backtrack threshold, initially zero.

For the agent group formed in Example 5.1 a possible sequence of events in order to solve the formed DCOP is as follows. Agent A_1 chooses $X_1 = 0$ first. Agent A_2 receives this value through a VALUE message and re-evaluates its own value. If, for instance, it was $X_2 = 0$ this will change to $X_2 = 2$. Then A_2 sends a COST message back. Since the constraint between A_1 and A_2 is satisfied the cost is zero. A_2 also sends a VALUE message to A_4. Now A_1 upon receiving a zero cost re-evaluates its upper bound and sets it to zero. This is equal to the THRESHOLD value as well. Consequently A_1 sends to A_2 a TERMINATION message and these agents halt their DCOP solution procedure. Under the same procedure agent A_4 eventually receives the final value A_2 has chosen and a terminate message. Therefore A_4 re-evaluates its own value, setting $X_3 = 4$ and terminates.

6 Searching, Task Allocation and Scheduling

This section describes the interplay of the methods for the scheduling of tasks (described in the previous sections), with the methods for the (re-) formation of teams in large networks of agents: Special emphasis is given to the searching of agents and to the allocation of tasks to agents, forming a network of potential teammates (PTN).

The primary task in a network of acquaintances *AN*, is to organize itself into a network where a set of agents form a connected overlay sub-network of "gateways" *GN*. Each time a change occurs in *AN* (due to uncontrollable events), agents may need to reorganize themselves forming a new *GN*. Self-organization happens by means of agents' local criteria using the algorithm explained in Section 4.1 for the computation of dominating nodes.

Given an arbitrary *GN,* each of the non-gateway agents connects to at least one gateway agent. To facilitate searching and maintenance of routing indices, gateway agents maintain routing indices for the resources and capabilities available to non gateway neighbours. Also, every gateway agent in *GN*, stores aggregated indices of its gateway neighbours with lower-priority. This forms an aggregated and approximate view of the network state, resulting in a jointly fully observable setting [9]. Gateways' views (i.e. routing indices) are maintained by means of capability-informing and resource-informing tokens.

Requests concerning atomic or complex tasks enter the agent network in an arbitrary fashion. Any agent can be considered as an entry point for a demand over the network's resources.

This dynamic self-organizing searching infrastructure supports the formation of teams for the performance of joint activities: Given a request for a joint task *t* originated by an agent in the network, then all its sub-tasks $\{g_1,..,g_i,...,g_k\}$ must be allocated to the appropriate agents, i.e. the agents that have the resources and the capabilities to perform each subtask. The search for the appropriate agent for each of these atomic tasks proceeds as it will be described in section 6.1. The appropriate agents (i.e. have the required capabilities and resources) form a logical network of potential teammates (PTN), who jointly try to schedule their activities with respect to the constraints associated to t, conjunctively with constraints that must hold for their other activities. This is done by means of one of the DCOP algorithms of Section 5. In case they are not successful in forming a common schedule, and depending on the violated constraints, they reform the PTN until a team is formed successfully (i.e. a team that has successfully scheduled all subsidiary tasks) or the time-to-live (TTL) of the request for the joint activity expires. Depending on the DCOP algorithm used, this occurs either when the algorithm determines inconsistency (in the case of Adopt) or is trapped in a local minimum (in the case of DSA).

We have tried two different approaches to the process of PTN reformation that are detailed below in Sections 6.1.1 and 6.1.2. Briefly, in the first approach reformation consists of forwarding the whole complex task to another gateway agent. In the second approach, the request originator asks one (randomly selected) agent involved in a constraint violation to release its subtask and then propagates the request for this subtask. In the first approach, the new PTN formed may involve a completely different set of agents, while in the second approach the new PTN involves

only one different agent than before. Reformation of a PTN may proceed as long as the corresponding joint task exceeds its TTL: In this case the task is considered as unsatisfied.

We have to point out that agents are allowed to reconsider their existing schedules when they face requests for participation in new joint activities as long as they have not committed their resources to already successfully allocated tasks. That is, if an agent is involved in a PTN that has not yet been resolved and at some point the agent joins another PTN (i.e. it now participates in two PTNs at the same time) then it can try to accommodate all subtasks that have been assigned to it by making the necessary shifts in its schedule. However, decisions made about already successfully allocated tasks are not backtracked on in order to find a better overall allocation. That is our approach, in its current design and implementation, is tuned to greedily try and accommodate incoming requests in the best possible way. Once agents commit to certain tasks, these commitments cannot be undone in order to accommodate requests arriving later. Hence, an agent that has committed part of its resources to allocated tasks stays idle until a new request arrives.

We now turn our attention to the searching and allocation tasks, and describe two alternative methods for achieving them.

6.1 Searching and Task Allocation

Upon the arrival of a complex task g_i, one of the following two cases holds: A_j is either a gateway agent or a non-gateway agent. If it is a gateway agent, then depending on whether g_i is a complex or an atomic task, it tries to locate the appropriate agent(s). This is further explained in Sections 6.1.1 and 6.1.2. In case A_j is a non-gateway agent, we have considered two possible modes of operation:

- *Inactive non-gateway mode*: the non-gateway agent immediately forwards the tasks to a gateway agent. In this case, the complex task is forwarded to the one-hop away gateway agent that has the higher priority among the gateways covering A_j.
- *Active non-gateway mode*: The non-gateway agent performs a quick local placement effort, checking whether it can satisfy the request itself. According to this mode, the non-gateway A_j checks whether its own resources and capabilities can satisfy all requirements concerning g_i.

In the case of a single atomic task, the task start time, end time and actual demand are the sole parameters taken into consideration. Similarly, in the case of a complex task, A_j will check if it has the time resources to accommodate all the subtasks, while respecting the constraints between them. Since the agent has a clear view of the whole task, this is straightforward.

At this time point, A_j being aware of the task requirements can decisively conclude whether it is capable to satisfy these requirements. Note that in this case there is no need to formulate a DCOP for the given task. If the agent decides that it can successfully accommodate the requested task, it updates its timeline appropriately to reflect the current situation. The task is marked as satisfied and A_j continues receiving requests from other agents.

In case A_j decides that it cannot accommodate the requested task using only its own resources, it will send this task to one of its gateway agents. As in the inactive non-gateway mode, it will send the task to the one-hop away gateway agent with the highest priority.

Comparing the two alternatives, we expect the active mode to speed up the system as some requests will be immediately handled by the agents used as entry points. On the other hand, we expect that the inactive mode will result in more efficient allocation as this process will be handled only by the gateway agents who, through their routing indices, have a better view of the agents' availability.

Henceforth we assume that the complex task is in the hands of a gateway agent, either because this agent has been used as the entry point to the system, or because the non-gateway agent that first received the request has sent it to a gateway agent. This may have happened either immediately, or after the non-gateway agent has decided that it cannot satisfy the task requirements on its own. In the sub-sections that follow we specify two different methods for searching and allocating tasks.

6.1.1 Method A

According to this method, the gateway node A_j that has received the task will first examine whether the task can be served by any of its non-gateway neighbours (i.e. by any of the agents it covers). If the task is atomic the course of action is straightforward, meaning that A_j only searches

for the first agent with adequate capabilities and resources. In the case of a complex task A_j breaks it down into individual subtasks and derives the constraints between them. Then it searches for neighbouring agents, including itself, that have the required resources and capabilities to accommodate subtasks. In both the cases of an atomic task and a subtask of a complex task, if more than one neighbouring agents are capable of serving, according to capabilities and resources, it then the first found is selected. That is, the agents' numbering scheme is followed. When all subtasks have been assigned, the agents form a PTN and subsequently, they form a new c-overlay network, as required by the DCOP algorithm. As they are now aware of the constraints between the subtasks allocated to them, they start executing the DCOP algorithm to determine if there is a solution.

If A_j cannot locate any non-gateway agent with the necessary resources and capabilities, or if the PTN formed cannot solve the DCOP, then A_j forwards the request to its gateway neighbour with the highest priority. In case there is no gateway neighbour with higher priority than A_j, then A_j propagates the request to all of its gateway neighbours. Since requests propagate through many different gateways, it is possible that there will be more than one agent that can serve a request. In such a case, all these agents inform the request originator about their availability, and the originator decides to whom the task shall be allocated (for example, based on their workload).

The execution of method A is summarized as a flowchart in the following figure.

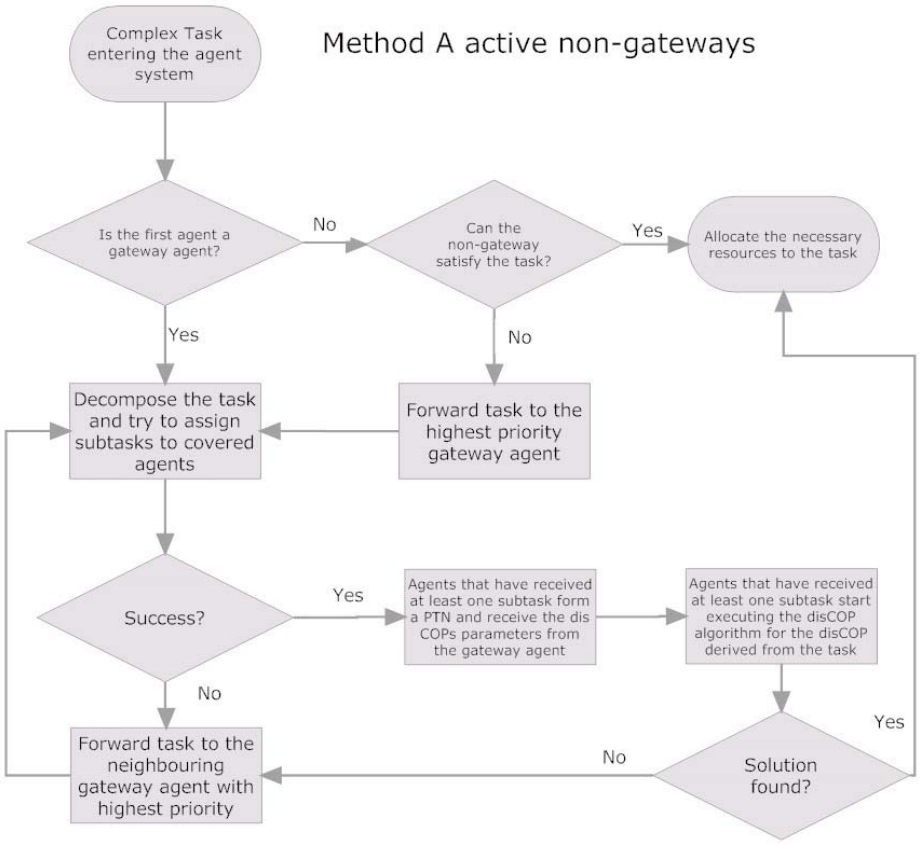


Figure 8. Flowchart for Method A

6.1.2 Method B

As an alternative to the approach described above, we introduce a method that is based on a more elaborate initialization process between neighbouring gateway agents. While the main course of action is the one described in the previous paragraph when the gateway agent A_j cannot accommodate a task in its immediate neighbourhood (i.e. in the one-hop away non-gateway agents), the procedure thereafter is different. Recall that through the use of routing indices, a gateway agent is fully aware not only of its own resources but of the resources and capabilities that are available via its neighbours: Therefore a gateway agent is aware of (a) its own resources and capabilities at any given time, (b) the resources and capabilities of all non-gateway agents that

exist in its neighbourhood and (c) the resources and capabilities of the agents that can be reached by the gateway agents and have lower priority while existing at a one hop distance.

Consequently, if A_j can not satisfy the requirements for resources or capabilities upon receiving a request, it tries to form a group with the "eligible" subtask recipients in order to satisfy the task. Eligible candidates are all neighbouring non-gateway agents and lower priority gateway agents. However, the gateway agent's view of its neighbours is not always accurate. This is because gateway agents have no way of knowing what task requests the other agents process at any moment, since routing indices are not updated immediately after each change in the availability of the agents. For instance, if an agent is in the process of using constraint solving techniques to decide if a previously submitted task or subtask can be accommodated by it, we may have a situation in which its own accurate view pertaining to its own resources is not in accordance with the (out of date) view that neighbouring gateway agents have.

Consequently, in method *B* the gateway agent A_j consults its neighbouring agents in order to decide whether a certain part of the task is going to be forwarded to one of them. Before doing this the gateway agent checks its routing indices to decide which agent seems most capable for receiving one or more subtasks. These remote agents are contacted and the final decision is made by each one of them after they have checked their own accurate view of their resources' availability. The first agent that answers positive to a request concerning a specific subtask is the one that receives it. The procedure for resolving the constraints involved cannot begin before the allocation procedure for the particular task has ended.

The major difference between Methods A and B is the gateway's 'view' during the allocation process. In Method A the gateway allocates each subtask based merely on its routing indices. On the contrary, in Method B the gateway consults its neighbours to acquire a clearer up-to-date view of their resources. The routing indices act as a first lead but the procedure continues and the gateway asks for an accurate snapshot of the potential recipients' timeline. Another point of difference is that in Method A the gateway searches for potential candidates in its immediate neighbourhood only. In method B the request may propagate through gateway agents with lower priority. Therefore, in Method B a subtask can be allocated to an agent that resides several hops away.

At this point the status of the recipient agent (being gateway or not) can lead to alternative ways in order to accomplish the assignment process:

- If the recipient is a non-gateway agent it can respond to a request by simply checking its own view. Therefore, the answer is plainly positive or negative.
- If the remote agent is a gateway agent (note that its priority is always lower than the one of the agent that initiated the assignment procedure) and this agent cannot handle the subtask or subtasks in question, an additional step is involved before the final answer. The lower priority gateway agent starts a similar procedure to the one already started by the requesting gateway node, trying to forward the requests in discussion to its own neighbourhood. If unsuccessful, the higher priority gateway is notified that the lower priority gateway agent cannot satisfy the request. Otherwise, the lower priority agent sends the agents' ids that can possibly satisfy each subtask. Due to gateway priorities, it is impossible for a given subtask to circle in the agent network during the allocation process. In case an agent cannot be assigned the subtask requested by the gateway, the next neighbouring agent that seems capable to accommodate the specific fraction of the initial request is consulted.

Consecutive negative answers from all neighbouring agents result in sending the task to a higher priority one-hop-away gateway agent. Since lower priority gateway agents do not have information about resource availability of their higher priority gateway agents, the decision is based on a simple request over the amount of total resources and capabilities that the higher priority gateways have in their view. This means that if a gateway agent cannot accommodate a task, and consequently must send it to a higher priority gateway agent, it only asks for the resources (time units) available via its one-hop-away gateways. The actual recipient is the one among them that possesses the highest number of available time units in its view.

The execution of method B is summarized as a flowchart in the following figure.

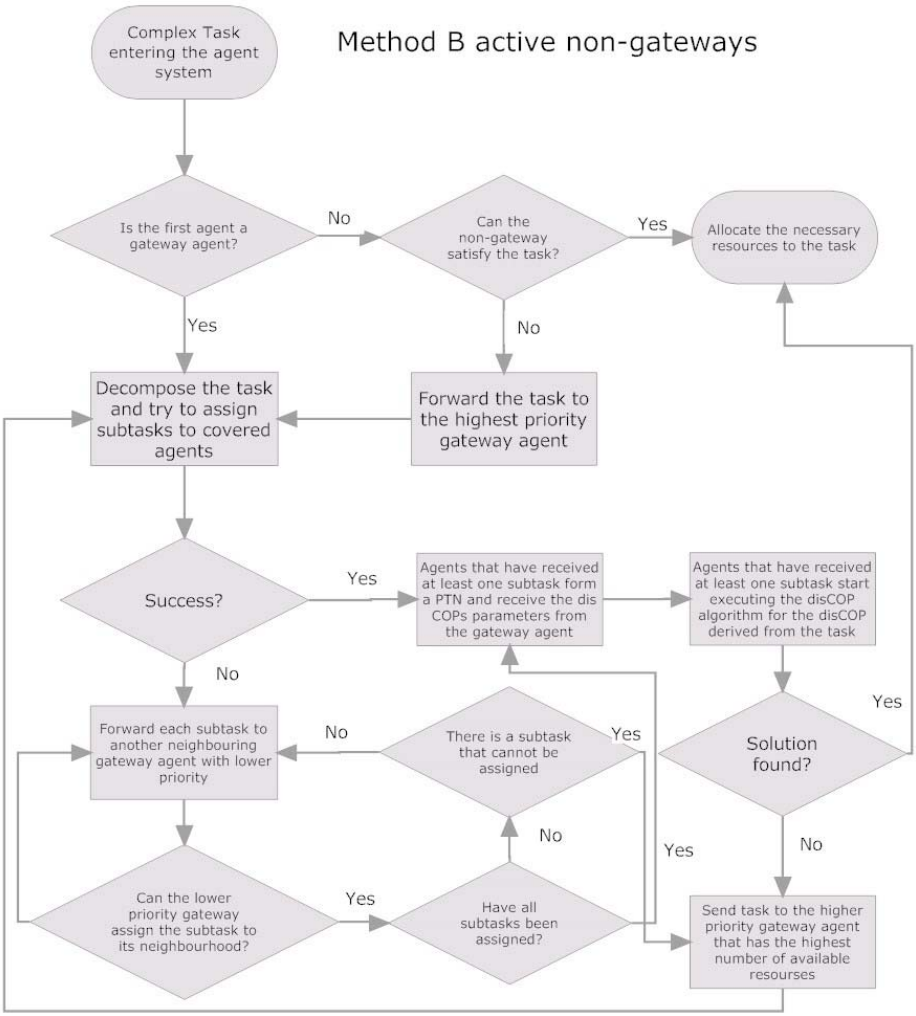


Figure 9. Flowchart for Method B

Comparing the two methods for searching and task allocation we can say that Method A uses simple (and fast) means for propagating the requests to those directions where it seems that there is a high possibility to locate the appropriate agents. In contrast, Method B uses a slightly more sophisticated (and hence slower) approach by first contacting neighbouring agents and acquiring a more accurate view of their availability. Therefore, Method B helps gateways in having more options during the task allocation process and increases the possibility of directing requests towards parts of the network where it is more likely that the requested task will be successfully allocated and scheduled.

6.1.3 Discussion

Analyzing the complexity of designing organizations Horling [12] has shown that the complexity of constructing an organization template and allocating agents to organizational roles is NEXP-Complete. This agrees with complexity results by Nair, Tambe and Marsella [20]. Knowledgeable and heuristic methods [24] may prune the search space for constructing suboptimal organizations. Our approach does not deal with designing organizational templates: Given a set of complex tasks and the network of acquaintances, agents need to be assigned to specific atomic tasks with respect to the required and own resources and capabilities. Therefore, the organizational structures are quite simple with respect to the roles agents need to play (i.e. the tasks to perform) and the network has to be clustered in specific teams of agents that can jointly perform the requested tasks. In this setting, each agent may be assigned multiple atomic tasks (i.e. participate in multiple teams), which need to be scheduled consistently to the atomic and team constraints. According to this, given a specific overlay network of gateways, the complexity of our approach lies mostly to the searching and constraint problem solving tasks. The searching task

aims at reducing the complexity of allocating agents to specific tasks by exploiting indices of agents' resources and capabilities. Searching is bounded by the number of gateways and the maximum number of immediate acquaintances of each gateway agent. The decision of which agents to participate in a team is distributed among the agents and it is subject to their *joint* ability to resolve the constraints: In such a case a simple method (e.g. a contract net protocol) would not be suitable for decision making given that each agent needs to satisfy jointly with its potential teammates the constraints of a task, in conjunction to all the constraints imposed by the other teams in which it aims to participate. According to the above, our approach is mostly suitable in cases where the gateway agents are proportionally less than the number of agents in the acquaintance network, and in cases where there are complex tasks that need to be jointly performed by teams of agents.

7 Experiments and Results

The experiments we carried out aimed at evaluating two important aspects of our methods. More specifically, we evaluated the two methods for DCOP solving described in Section 5.2 and the various combinations of methods for searching and task allocation put forward in Section 6.1. We first discuss the way test problems were generated and then we present the experimental results.

7.1 Problem Generation

We experimented with two models for the generation of the agent network. The first one generates networks of randomly deployed agents, assuming that the geographical distance among agents determines the topology of the system. Each node A establishes connections with all nodes that exist in a specific distance from it, according to a given radius r. That is, all nodes located in a cycle with center A and radius r are neighbors of A. Networks are constructed by distributing randomly |N| agents in an $n \times n$ area, each with a "visibility" ratio equal to r. The acquaintances of an agent are those that are "visible" to the agent and those from which the agent is visible (since edges in the acquaintance network are bidirectional). Although this method of generation, which we will call *geographic* henceforth, distributes agents randomly in an area, the "visibility" ratio ensures that only agents that are "close by" can communicate directly. Hence, some kind of structure is introduced. The experiments in Sections 7.2 and 7.3 concern networks $AN=(N,E)$, with |N|=500 nodes, randomly placed in a 250x250 grid. The radius parameter r was set to 25 and only connected networks were considered in the experiments. Note that since the area of placement is large and r is small, these settings tend to create relatively sparse networks where a message originating at some agent may need to travel through many edges in order to reach distant agents. In Section 7.4 we discuss an alternative random generation method that constructs the agent network in a way that allows for messages to reach any other agent by traveling through fewer edges on average.

We assume that each agent A_i possesses an amount of S_i time units. Agents are simultaneously requested to jointly fulfil a set of tasks $T=\{t_1,t_2, \ldots ,t_n\}$, where $t_i=<a_i, start_i, end_i, Cap_i>$ such that $\sum_{i=1}^{n} a_i = \sum_{i=1}^{|N|} S_i$: This is a worst-case scenario where the system has to simultaneously fulfil the maximum number of tasks that fit its resource capacity. In the experiments below, S_i was uniformly set to 10 for all agents. Therefore, the total available and required capacity was 500x10=5000 time units.

For simplicity we assume that the cardinality of each Cap_i is 1, which means that a unique type of capability is sufficient for t_i's satisfaction. We assume that each $A_i \in N$ is also attributed with a unique type of capability, such that $\bigcup_{\forall t_i \in T} Cap_{t_i} = \bigcup_{\forall A_i \in N} Cap_{A_i} = Cap_{AN}$. We divided our experiments into three sets in terms of the capabilities that agents have and that tasks require. The first set includes experiments where all agents have the same capability type (i.e. Cap_i =1) and all tasks require an agent with Cap_i =1. In the second set there are two possible values for Cap_i, simply denoted by 1 and 2. Hence, every agent has Cap_i =1 or Cap_i =2 and accordingly, each atomic task either requires one agent with Cap_i =1 or one with Cap_i =2. Finally, in the last set of experiments there are three possible values for Cap_i (1, 2, and 3).

The complex tasks in the set of task requests T are generated sequentially in a way such that the total duration of all subtasks does not exceed the network's total capacity. In the experiments

presented below each complex task consists of, at maximum, 3 or 7 subtasks. The TTL of all tasks was set to 10. The actual number of subtasks for each complex task is chosen randomly with a uniform distribution. The next step in the generation of a complex task involves deciding the duration of its subtasks. This is done by randomly setting the duration of each subtask so that their total duration is at most S_i, i.e. at most equal to the uniform capacity of the agents. To be more specific, assuming a complex task with 3 subtasks, this is generated as follows. We first pick a subtask t_j and randomly set its duration d_j to a number between 1 and S_i. We then select another subtask t_k and randomly set its duration d_k to a number between 1 and S_i - d_j. Finally, the duration of the remaining subtask is randomly set to a number between 1 and S_i - d_j - d_k. If at some point during this process there is no available choice for the duration of a subtask, because of previously set durations of other subtasks, then the process is restarted.

We then generate the set of constraints between the subtasks. Given X subtasks in a complex task, this is done by first selecting randomly y% of the possible constraints between the subtasks (i.e. $\lfloor (y/100) \times X \times (X\text{-}1)/2 \rfloor$ constraints). For example, when y=50 in a complex task with 7 subtasks we generate randomly $\lfloor 0.5 \times (7 \times 6)/2 \rfloor$ =10 constraints between the subtasks. For all the experiments presented below, y was set to 50. This value resulted in creating complex tasks that are relatively hard while at the same time rarely being over-constrained. Then for each constraint we choose a random label among the following set: {>, <, =, ≥, ≤)}. For instance, if the chosen label is '>' it means that the start time of the second subtask participating in the constraint must be greater than the end time of the first participator. For example, if the first subtask's duration is 2 and its start time is 1, the second subtask's start time must be greater than 3.

As a final step, we use ADOPT to check if the generated complex task is actually satisfiable. If it happens to be over-constrained then it is dropped and a new complex task is generated. In all experiments all complex tasks enter the agent system through a randomly chosen agent at the start of the system run.

In the reported experiments we report averages over 10 experiments for each individual case of parameter settings. Throughout the following sections we compute and compare three basic measures:

1. *Benefit*: The percentage of complex tasks scheduled over the complex tasks requested to be scheduled.
2. *Messages*: This is the total number of messages exchanged between any two agents throughout the task allocation and scheduling process
3. *Message Gain*: The ratio of the benefit over the total number of exchanged messages.

In some cases we also report additional useful information such the number of gateway agents created and the numbers of PTNs formed during the task allocation process.

7.2 Adopt vs Local Search for solving DCOPs

Experiments here compare the two methods described in Section 5.2 for solving the DCOPs derived from the temporal interdependencies of tasks. That is, we evaluate the performance of the system's scheduling component when either Local Search or Adopt is used to solve the DCOPs. For a fair comparison, experiments for both algorithms ran in a system that uses the same method for searching and task allocation (Method A with active non-gateways). Therefore, the experiments presented here illustrate the contrast between a complete (Adopt) and an incomplete (Local Search) algorithm for solving DCOPs generated from complex tasks. As we detail below, results show that Adopt, compared to Local Search, improves the efficiency without considerably increasing the cost in terms of exchanged messages.

The parameters of the generated geographic networks are as follows:

- 500 nodes
- n=250, r=25
- 1 or 2 or 3 distinct capability types
- 3 or 7 subtasks per task at maximum

Note that with the above settings for n, r, and 500 nodes, the average density of the generated networks is around 1.95%. The average number of nodes that obtained gateway status was 227.

Figures 10 and 11 give the benefit and message gain of the compared methods when complex tasks include at maximum 3 subtasks. Accordingly, Figures 12 and 13 give the same information when complex tasks consist of 7 subtasks at maximum. As displayed in Figures 10 and 11, when Cap_i is 1, both Adopt and Local Search achieve a very high benefit while Local Search displays

higher message gain because of the increased message exchange that Adopt incurs. However, as the number of available capabilities increases and the problems become harder, Adopt achieves a much higher benefit and also outperforms Local Search in terms of message gain. This is because Adopt is able to solve more DCOPs and hence successfully schedule more complex tasks than Local Search. This success outweighs Adopt's extra message cost and results in higher message gain.

As displayed in Figures 12 and 13, when the complexity of the task requests increases, Adopt is constantly better than Local Search both in terms of benefit and message gain for all values of Cap_i. It is notable that when there are 3 types of capabilities (which is the hardest case) a system that uses Adopt can achieve nearly three times the benefit achieved by Local Search.

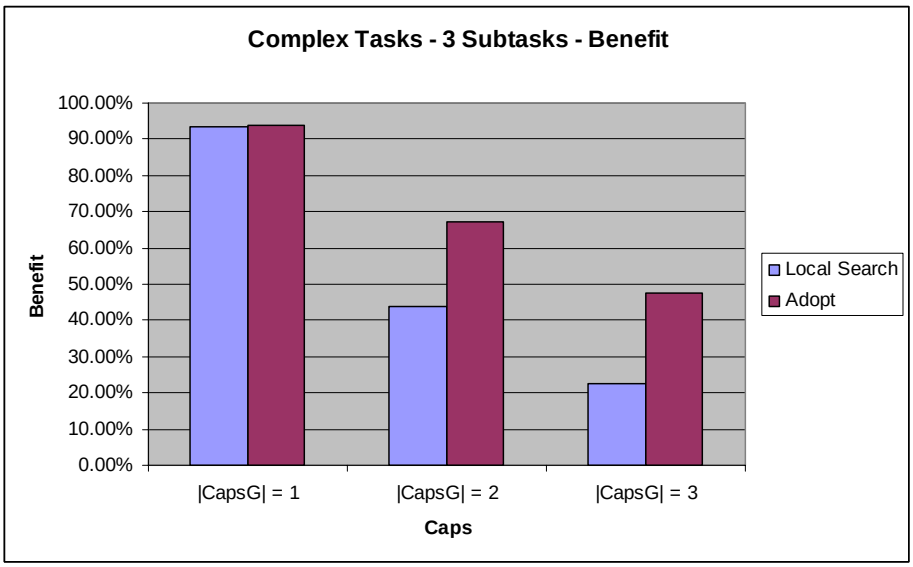


Figure 10. Benefit achieved by Adopt and Local Search when there are at most 3 subtasks per complex task.

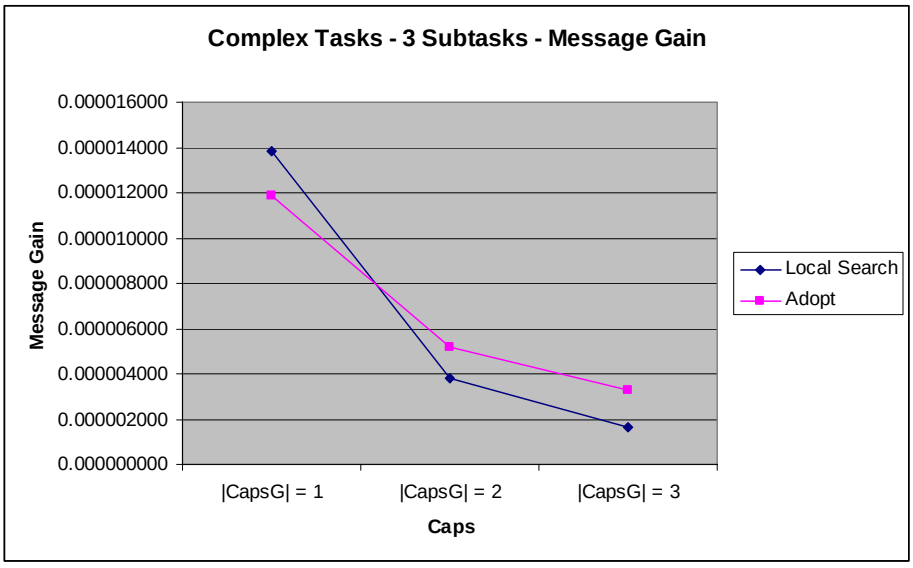


Figure 11. Message gain achieved by Adopt and Local Search when there are at most 3 subtasks per complex task.

To summarize, the use of Adopt displayed increased efficacy as far as the obtained benefit is concerned. Furthermore, the number or exchanged messages did not increase considerably compared to Local Search. As a result, Adopt demonstrated a better message gain than Local Search. Therefore, we can safely conclude that, on hard problems, the method that applies Adopt for scheduling is more efficient than the one that applies Local Search.

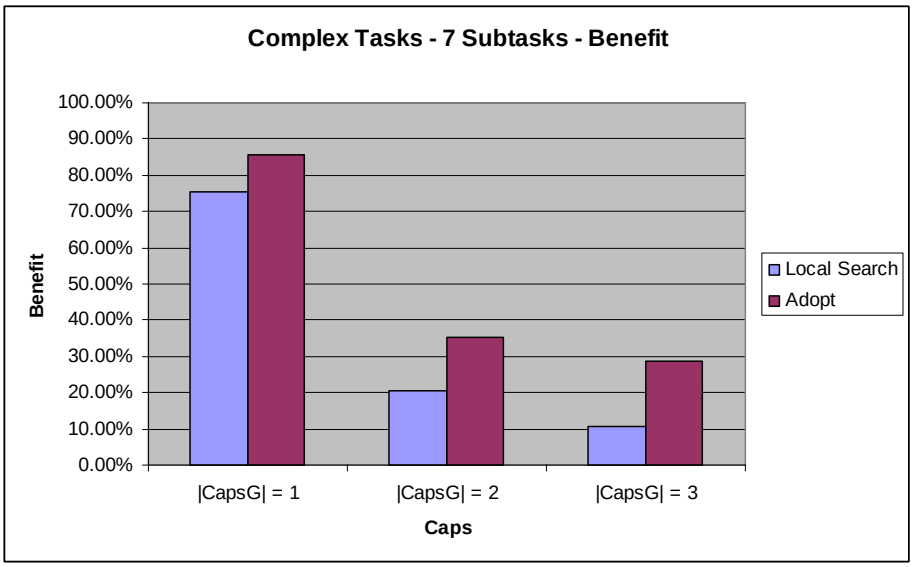


Figure 12. Benefit achieved by Adopt and Local Search when there are at most 7 subtasks per complex task.

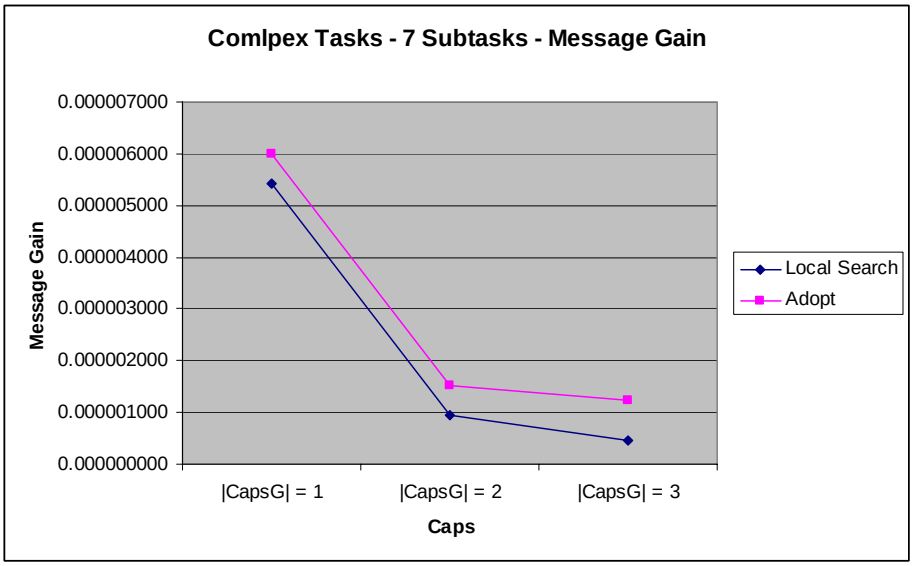


Figure 13. Message gain achieved by Adopt and Local Search when there are at most 7 subtasks per complex task.

7.3 Evaluating different approaches to searching and task allocation

Having established that Adopt offers considerable advantages in terms of the system's benefit, we now evaluate the different approaches to searching and task allocation using the same networks as in Section 7.2 (i.e. |N|=500, r=25, n=250). For all the experiments presented hereafter in the paper we have used Adopt for scheduling despite the relative increase in the number of exchanged messages that it sometimes occurs compared to Local Search. We evaluated the following methods outlined in Section 6.1:

1. *Method A with active non-gateways*
2. *Method A with inactive non-gateways*
3. *Method B with active non-gateways*
4. *Method B with inactive non-gateways*

Considering the two methods (A and B) for performing task allocation and scheduling, recall that as discussed in Paragraph 6.1.2, Method B seems to be more flexible in finding agents with

free resources and consequently in forming PTNs. Indeed, experiments verified this as Method B consistently demonstrated better results than Method A in each single experiment (10 experiments for each parameter setting). Both variations of Method B performed better than any variation of Method A in terms of the total number of satisfied complex tasks. A small downside is that Method B produced more messages. However, Method A produced an average message gain of 0.5984×10^{-5} when each complex task has at most 3 subtasks, while for Method B the message gain was 0.6271×10^{-5}. The average difference of 0.0287×10^{-5} in favor of Method B amounts to 4.79% of Method A's message gain. In the case of 7 subtasks the average message gain was 0.2759×10^{-5} for Method A and 0.2853×10^{-5} for Method B. Therefore, the average difference in favor of Method B was 0.0094×10^{-5} (3.41% of Method A's message gain). These are the average variations in message gain between Methods A and B with either active or inactive non-gateway agents and capability types 1,2 or 3. Method A achieved a slightly better average message gain only in the (simple) case of complex tasks with at most 3 subtasks and 1 type of capability. Therefore, in the rest of Section 7.3 we opt to present results from the two variations of Method B (active and inactive non-gateways) only. Section 7.4.1 presents a detailed statistical analysis comparing Method A to Method B. We now first present some information regarding the generated sets of tasks, and then we give results from the two variants of Method B.

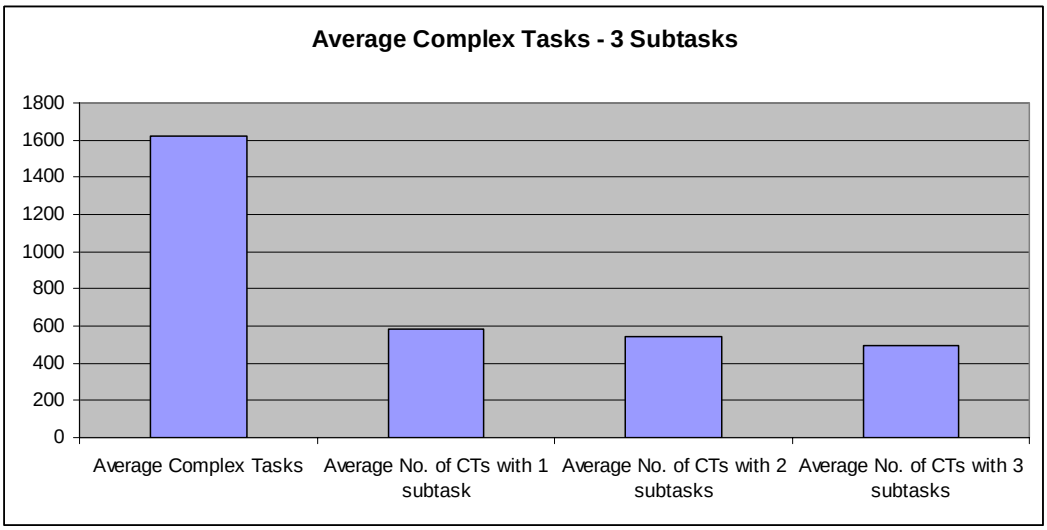


Figure 14. Average number of complex tasks for 3 subtasks at maximum and average number of complex tasks with exactly 1,2 and 3 subtasks respectively.

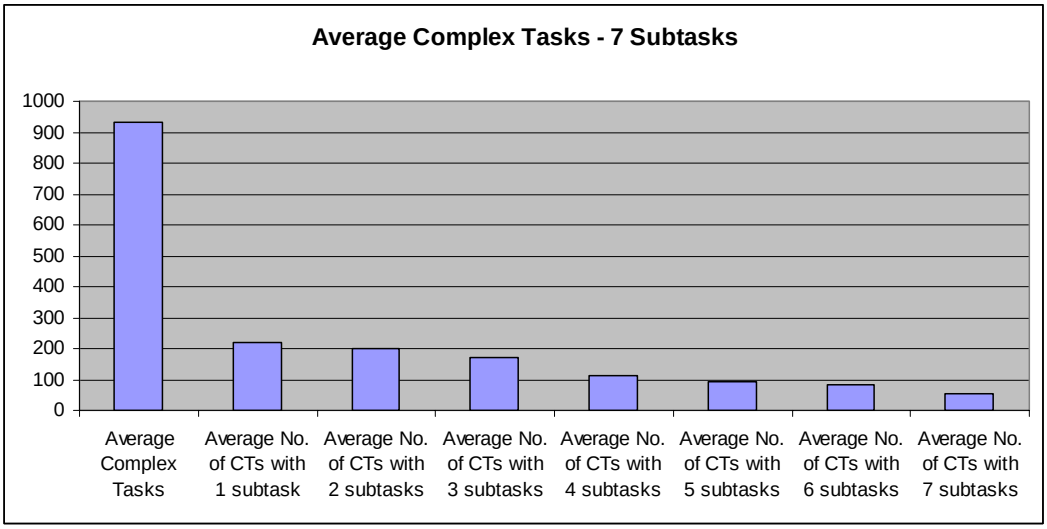


Figure 15. Average number of complex tasks for 7 subtasks at maximum and average number of complex tasks with exactly 1,2,…,7 subtasks respectively.

In the first column of Figures 14 and 15 we show the average number of complex tasks created in a set of task requests *T* when the maximum number of subtasks per complex task is 3 and 7 respectively. We also give the average number of complex tasks with exactly 1,2,3 subtasks for the first case and 1,2,…,7 subtasks for the second case. We can see that although the number of subtasks per complex task was uniformly selected, complex tasks with few subtasks appeared more often than ones with many subtasks. This is due to the fact that preprocessing with Adopt detected unsatisfiability for many of the latter complex tasks, and therefore they were not included in the generated set *T*.

Figures 16 and 17 present the average duration of the complex tasks in a set of tasks *T* for the two cases (3 or 7 subtasks at maximum). In these figures, we also give the average duration of the complex tasks broken down to the number of subtasks. As expected, in the first case shorter tasks are generated, meaning that their allocation and scheduling is more likely to succeed as they can often be assigned to a single agent. In contrast, complex tasks with many subtasks have an average total duration closer to the capacity of the agents which means that most likely a PTN with several agents needs to be formed in order to serve them, especially in the cases where 3 distinct capability types exist.

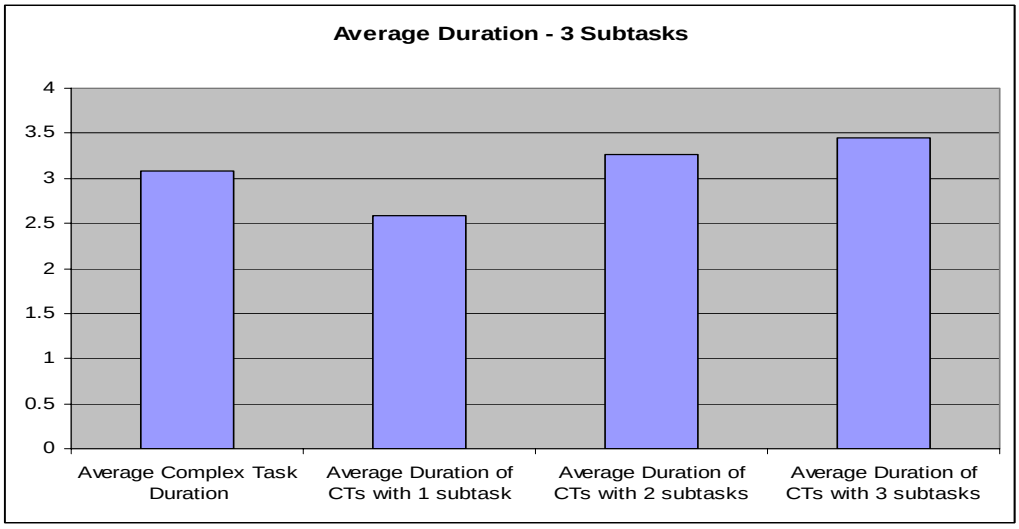


Figure 16. Average duration of complex tasks of 3 subtasks at maximum and average duration of complex tasks with exactly 1,2 and 3 subtasks respectively.

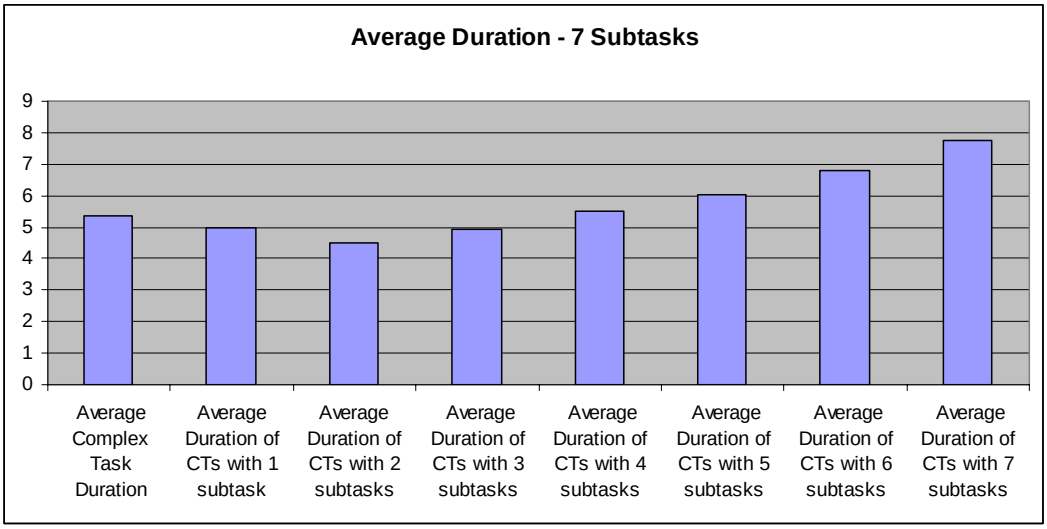


Figure 17. Average duration of complex tasks of 7 subtasks at maximum and average duration of complex tasks with exactly 1,2 … 6 and 7 subtasks respectively.

Figures 18 and 19 report the number of PTNs formed while trying to resolve a complex task, the number of attempts that succeeded in finding resources for a complex task, and the number of attempts failed(and thus required reconsideration/reformation of the unsuccessful PTN). In Figure

18 we give these measurements for the case of 3 subtasks at maximum, while Figure 19 contains this data for 7 subtasks at maximum. Note that in both cases there are two methods (the two versions of Method B) and three different types of existing capabilities.

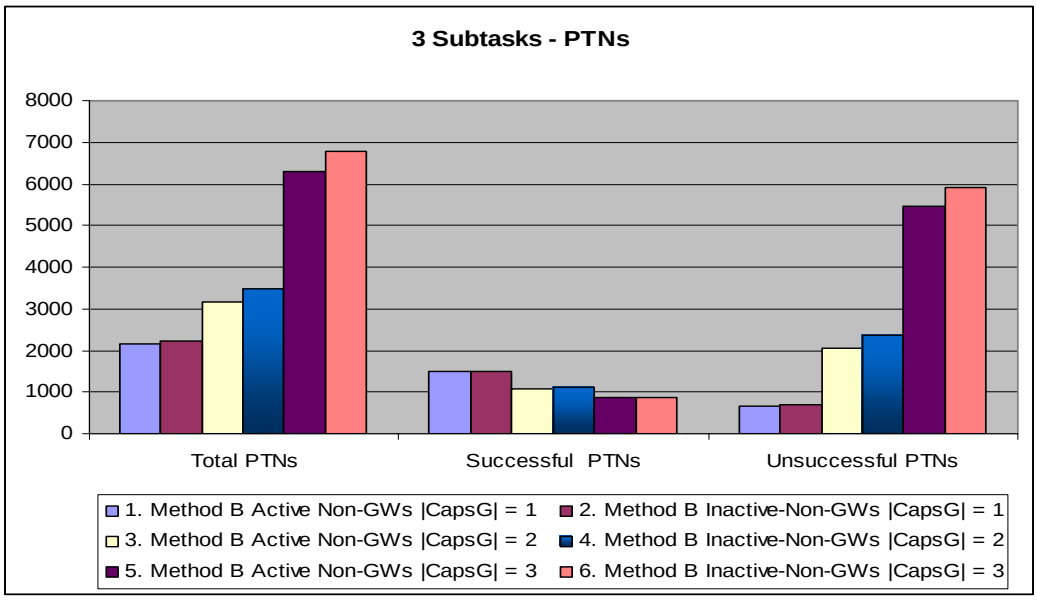


Figure 18. Number of total PTNs formed for each case of problems concerning complex tasks with maximum 3 subtasks, followed by the number of those that were successful and unsuccessful.

Both figures demonstrate that the inactive non-gateways method forms a larger number of PTNs on average. Although the number of successful PTNs does not vary much between the two methods, the number of unsuccessful PTNs is visibly higher for the inactive non-gateways variant of Method B. The number (over all values of capability types) of unsuccessful PTNs for each successful PTN (or for each successfully allocated task) ranges from 0.44 when there are at most 3 subtasks within any complex task up to 17.01 unsuccessful PTNs for each successful one when there are at most 7 subtasks. These numbers are approximately 2.98% higher on average for the inactive non-gateways method compared to the active non-gateways method. For the active non-gateways method we have an average of 6.56 unsuccessful PTNs for each successful one (this average number takes into account all this paragraph's experiments of Method B following the active non-gateways model). This number becomes 6.75 for the inactive non-gateways method. The active gateways method forms fewer PTNs because in some cases a task that enters the system through a non-gateway node can be directly accommodated by this node. In contrast, in the non-active gateways method the task will be immediately forwarded to a gateway agent and therefore the process of searching for a PTN to accommodate the task will begin. Despite this, and as results below demonstrate, the non-active gateways method provides greater flexibility to the system since non-gateways that receive incoming tasks do not engage their resources immediately, and in this way fill up their timeline, but allow for gateway agents that a have a more accurate view of the system's available resources to forward the task to agents that may have better availability to serve it.

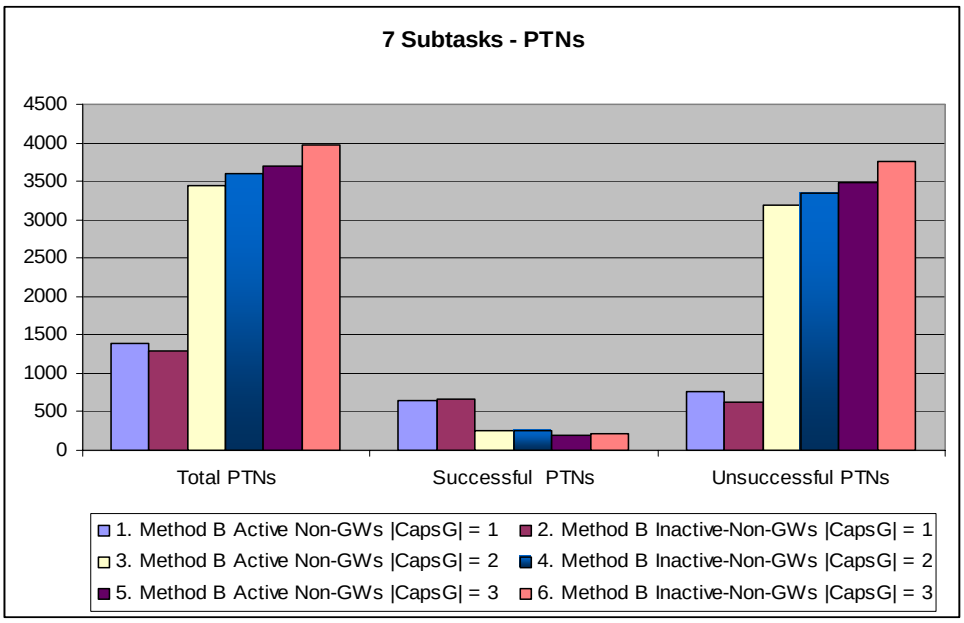


Figure 19. Number of total PTNs formed for each case of problems concerning complex tasks with maximum 7 subtasks, followed by the number of those that were successful and unsuccessful.

Concluding this section we give the average benefit and message gain for the two alternatives of Method B. Figures 20 and 21 depict the benefit for the cases of 3 and 7 maximum subtasks per complex task respectively, while Figures 22 and 23 depict the message gain for the two cases. In each figure we give the average numbers for problems with 1, 2, and 3 possible capabilities.

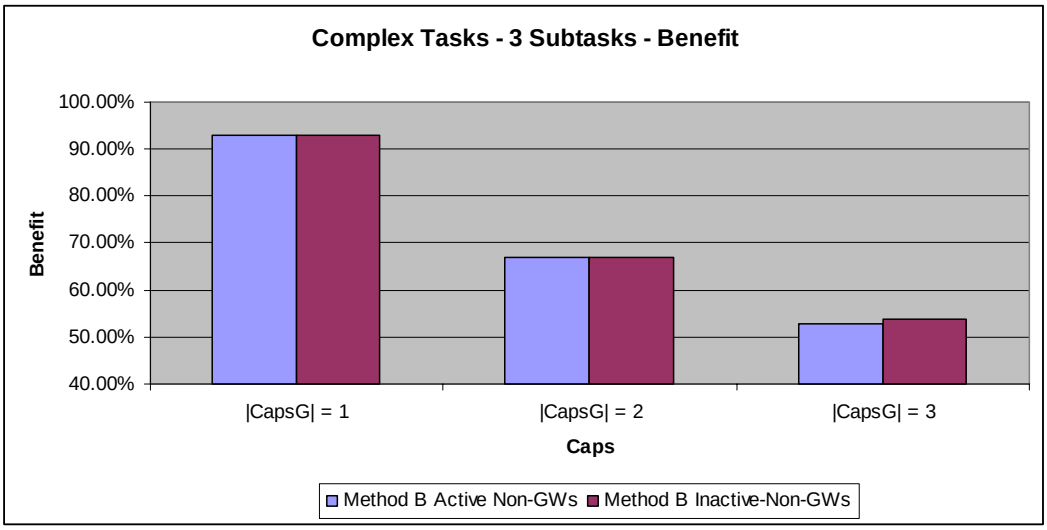


Figure 20. Benefit achieved when there are 3 subtasks per complex task.

As displayed, the inactive non-gateways method achieves slightly better performance compared to the active non-gateways method, especially in the case of 7 subtasks. Despite the fact that the inactive non-gateway method forms more PTNs on average, the increase in the number of messages exchanged incurred is not important enough to affect the message gain factor considerably. Hence, the inactive non-gateways method also obtains a higher message gain. However, the difference in the message gain obtained is reduced as the number of subtasks in each complex task increases. This is to be expected as a higher number of subtasks within each complex task decreases the potential of each active non-gateway agent to serve a request, resulting in more messages being exchanged while searching for PTNs to serve the tasks.

Relating these results to the discussion on Figures 14-17, it is interesting to note that the decline in benefit achieved when moving from fewer to more subtasks per complex task is not significant (see the first two bars in Figures 20 and 21) in the case of a single capability for all agents.

However, the decline is rapid as the number of capabilities increases as it becomes increasingly difficult to locate the appropriate agents and successfully form PTNs.

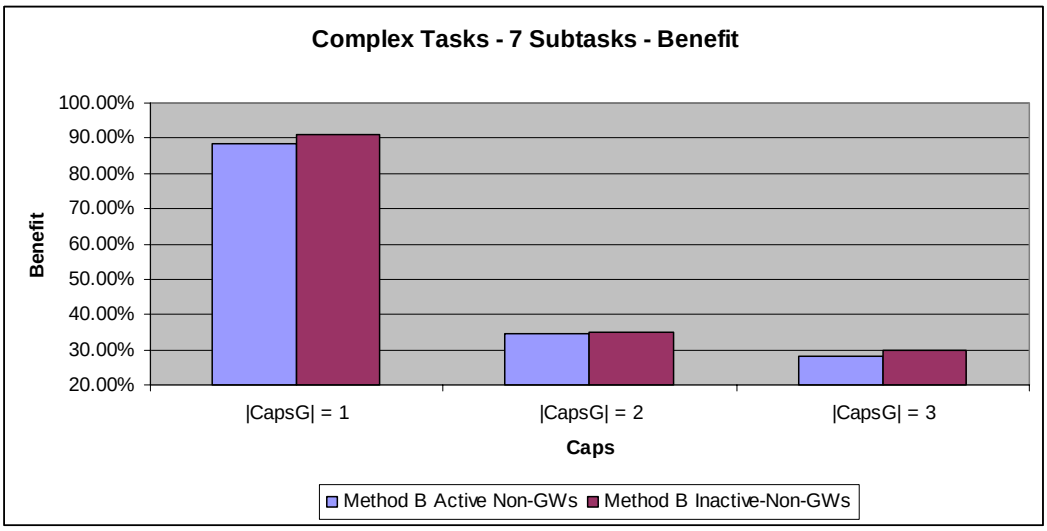


Figure 21. Benefit achieved when there are 7 subtasks per complex task.

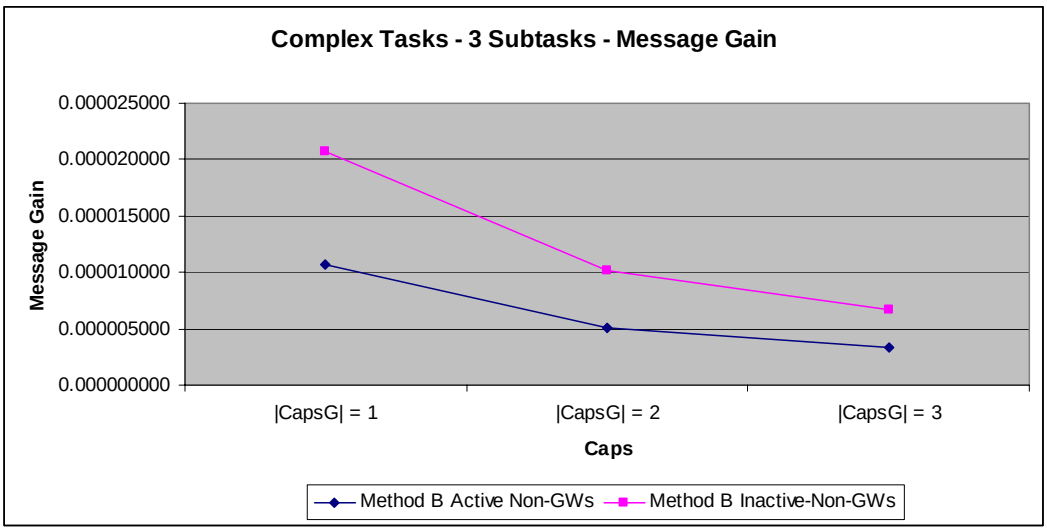


Figure 22. Message Gain when there are 3 subtasks per complex task.

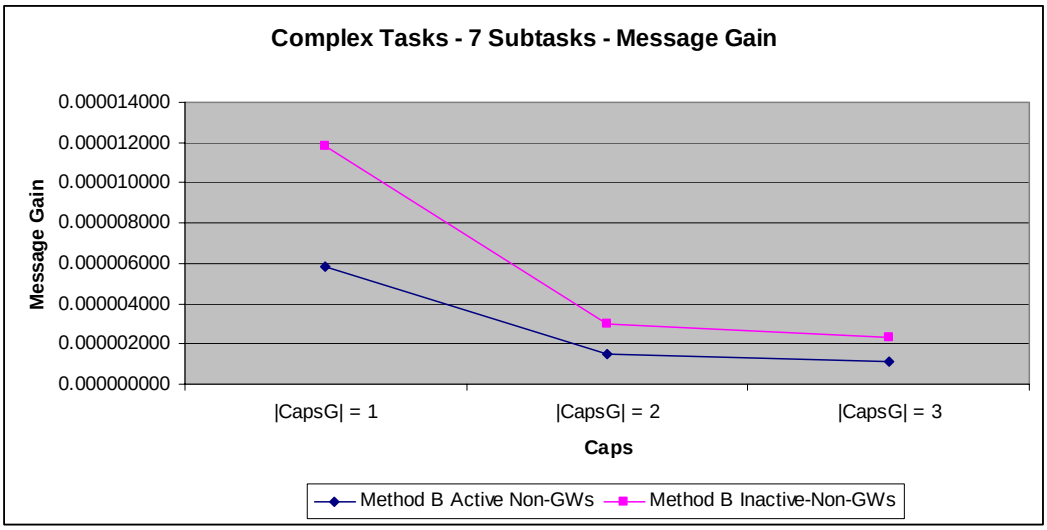


Figure 23. Message Gain when there are 7 subtasks per complex task.

7.4 Evaluating different approaches to searching and task allocation networks with random connections

In this section we study the performance of the proposed methods in different agent network settings aiming to investigate whether and to what extent the topology of the network influences the performance of our proposed techniques.

The networks in this set of experiments were generated in a way such that connections can be created between any two nodes in the network, discarding the radius parameter. Hence, we call this the *random_connections* generation method. The generator takes a parameter n denoting the maximum number of edges that can be attached to any node in the agent network. That is, the maximum number of agents each agent can have in its immediate neighborhood. For each agent, the actual number m of its neighbors is selected randomly between 1 and n. The generation process starts by randomly selecting an agent A_i and a number m, $m>0$. Then, we randomly select m agents to connect to A_i and the corresponding edges are added to the network. This process continues until the connections of all agents have been determined. When determining the connections of an agent we take into account the connections it may have already acquired through previous steps in the process. We ensure that the resulting network is connected by randomly adding edges between any disconnected components. For the experiments presented here n varied from 3 to 15 with an increasing step of 2.

The average number of complex tasks generated for the experiments of this section was 1637 for the case of 3 subtasks and 933 for the case of 7 subtasks. Note that the generation of complex tasks is not influenced by the underlying network's topology and therefore their characteristics are very close to the ones shown in Figures 14 and 15. The average graph density of the graphs generated for n= 3, 5, 7,…, 15 is shown below (Figure 24).

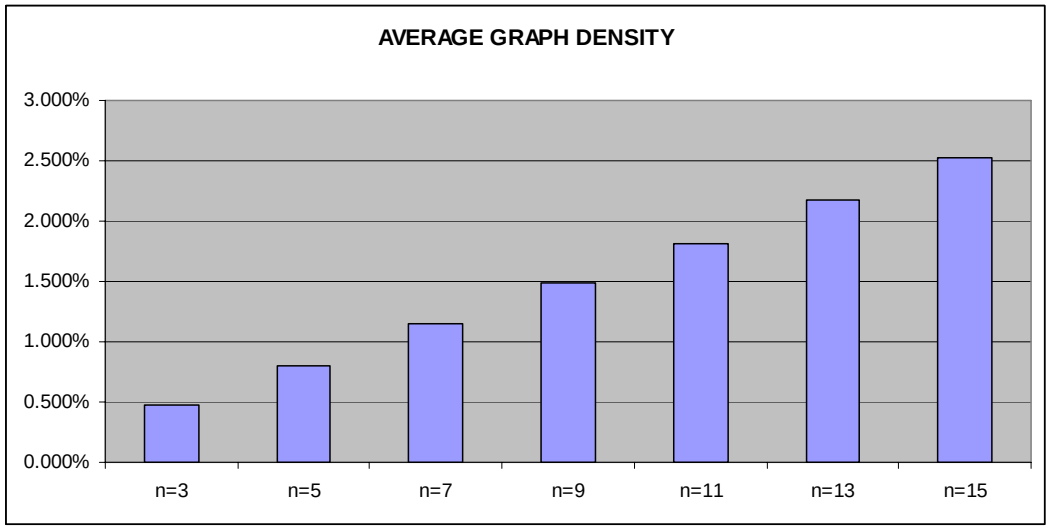


Figure 24. Average graph density for n= 3, 5, …, 15

Note that the generated graphs are sparse as the average graph density ranges from 0.48% for n=3 to 2.52% for n=15. After each agent network is generated and each agent acquires knowledge of its neighbors we decide which ones will be assigned the gateway status using the algorithm given in Section 4.1 [2]. Information on this is presented in Figure 25 where we give the average number of agents that opted for gateway status over the maximum number of connections for each agent. As expected, as the density of the network increases, the number of gateways decreases.

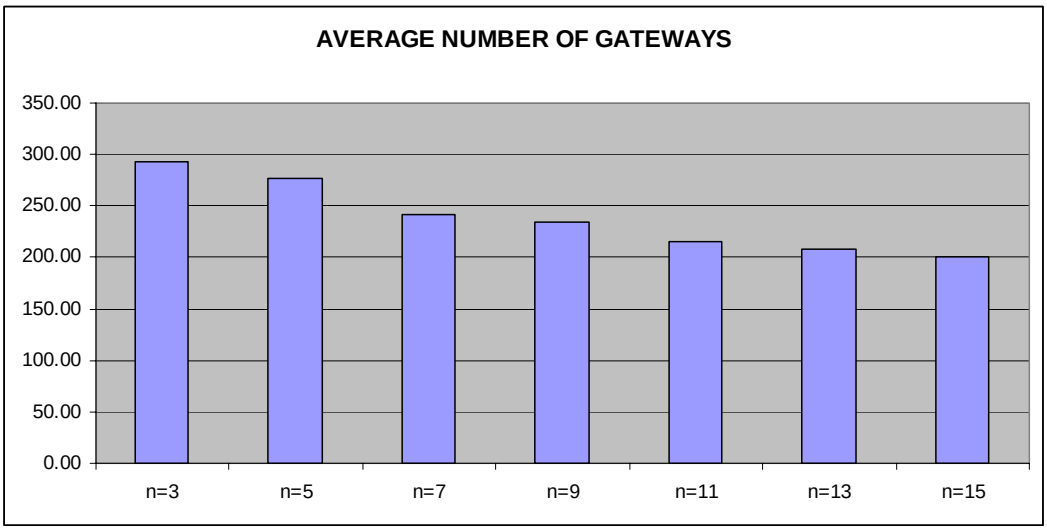


Figure 25. Average number of gateway agents for n maximum possible immediate neighbors.

We have performed a detailed evaluation the proposed searching and allocation methods (the variants of Method A and Method B) on the problems generated with the random connections generation method whose characteristics are explained in Figures 24 and 25. Before presenting the results of our most competitive methods (the two variants of Method B) in Section 7.4.2, we first give a statistical analysis confirming the (slight) advantage of Method B over Method A.

7.4.1 Method A vs. Method B

As it is also reported in Section 7.3, Method B is generally slightly better than Method A. We now give results from a statistical analysis, performed using non-parametric Wilcoxon signed-rank tests that verify this. The Wilcoxon signed-rank test is a non-parametric statistical hypothesis test for the case of two related samples or repeated measurements on a single sample. The test is based on the magnitude of the difference between the pairs of observations. The Wilcoxon signed-ranks tests assume a continuous value distribution. It can be used as an alternative to the paired Student's t-test when the population cannot be assumed to be normally distributed. Table 2 compares Method B with active non-gateways to Method A with active non-gateways, while Table 3 gives similar results for the case of inactive non-gateways. We give the mean and standard deviation of the difference in benefit and the difference in the number of messages sent by the compared methods. We also give the Z-value and the p-value for each combination of Capability types/Max. subtasks. If the computed p-value is equal or less to 0.05, then the two methods can be said to differ significantly. Otherwise, no statistically significant difference can be assumed between the two methods.

Capa-bility types / Max. Sub-tasks	Benefit_B_a - Benefit_A_a				Messages_B_a - Messages_A_a			
	Mean	**SD**	**Z-value**	**p-value**	**Mean**	**SD**	**Z-value**	**p-value**
1/3	-0.11	1.40	-0.296(b)	0.767	3398	4080	-2.090(a)	0.037
2/3	1.66	1.90	-2.191(a)	0.028	3751	4339	-2.090(a)	0.037
3/3	1.74	1.97	-2.701(a)	0.007	6394	4720	-2.599(a)	0.009
1/7	1.97	2.22	-2.191(b)	0.028	6663	5341	-2.803(b)	0.005
2/7	2.48	3.00	-2.191(a)	0.028	1115	5701	-0.255(a)	0.799
3/7	1.48	3.07	-1.362(a)	0.173	14584	6865	-2.803(a)	0.005

Table 2. Statistical results comparing Method B with active non-gateways (denoted by B_a) to Method A with active non-gateways (denoted by A_a). Column 1 gives the number of capability types and the maximum number of subtasks per complex task. Columns 2-5 give statistics regarding the difference in achieved benefit. Columns 6-9 give statistics regarding the difference in sent messages. P-values and Z-values that indicate statistical significance are underlined. A sample of 10 instances was used for each combination of values for the capabilities and the maximum subtasks per complex task.

Results from Tables 2 and 3 show that in all cases, except in the case of 1 capability and 3 subtasks per task in Table 2, Method B achieves a higher benefit on average than Method A. Moreover, in most cases the difference is statistically significant as the p-values indicate. In contrast, in all cases Method B has a higher average cost than Method A, measured by the messages sent. Also, in most cases the difference in the number of messages is statistically significant. Having established that Method B achieves a higher benefit than Method A, in the rest of Section 7.4 we only present results from the two variants of Method B.

Caps / Max. Sub-tasks	Benefit_B_i - Benefit_A_i				Messages_B_i - Messages_A_i			
	Mean	**SD**	**Z-value**	**p-value**	**Mean**	**SD**	**Z-value**	**p-value**
1/3	0.36	1.65	-0.663(a)	0.508	8494	4492	-2.803(a)	0.005
2/3	1.95	1.83	-2.395(a)	0.017	1504	3338	-1.274(a)	0.203
3/3	3.01	2.94	-2.395(a)	0.017	6574	6685	-2.701(a)	0.007
1/7	6.39	2.12	-2.803(b)	0.005	5387	6903	-2.090(b)	0.037
2/7	2.51	1.89	-2.599(a)	0.009	2127	6358	-0.968(a)	0.333
3/7	1.98	1.91	-2.293(a)	0.022	11861	8850	-2.599(a)	0.009

Table 3. Statistical results comparing Method B with inactive non-gateways (denoted by B_i) to Method A with inactive non-gateways (denoted by A_i). Column 1 gives the number of capabilities and the maximum number of subtasks per complex task. Columns 2-5 give statistics regarding the difference in achieved benefit. Columns 6-9 give statistics regarding the difference in sent messages. P-values and Z-values that indicate statistical significance are underlined. A sample of 10 instances was used for each combination of values for the capabilities and the maximum subtasks per complex task.

7.4.2 Active vs. Inactive Non-Gateways

In Figures 26 and 27 we compare the benefit achieved by the two variations of Method B in problems with complex tasks consisting of at most 3 and 7 subtasks respectively, while in Figures 28 and 29 we show the message gain for the same classes of problems. The numbers given are averages for all values of the maximum number of agents' neighbors *n* (3 up to 15). We do not present separate results for the different values of *n* because changes in the graph's density in the range of 0.48 to 2.52 did not have a significant impact on the system's benefit. Of course, for much denser networks this may not be true and we intend to investigate this in more detail in the future. While the average density for networks with *n*=3 was substantially lower than the one for networks with *n*=15 (around 5 times lower), the system's benefit throughout all sets of experiments had a variation of only 1.5% between the highest and the lowest benefit obtained for networks of any density, noting that slightly higher benefit was achieved as the density was increased. Hence, it seems that the proposed method is quite robust regarding the network's density, at least for the random_connections generation method.

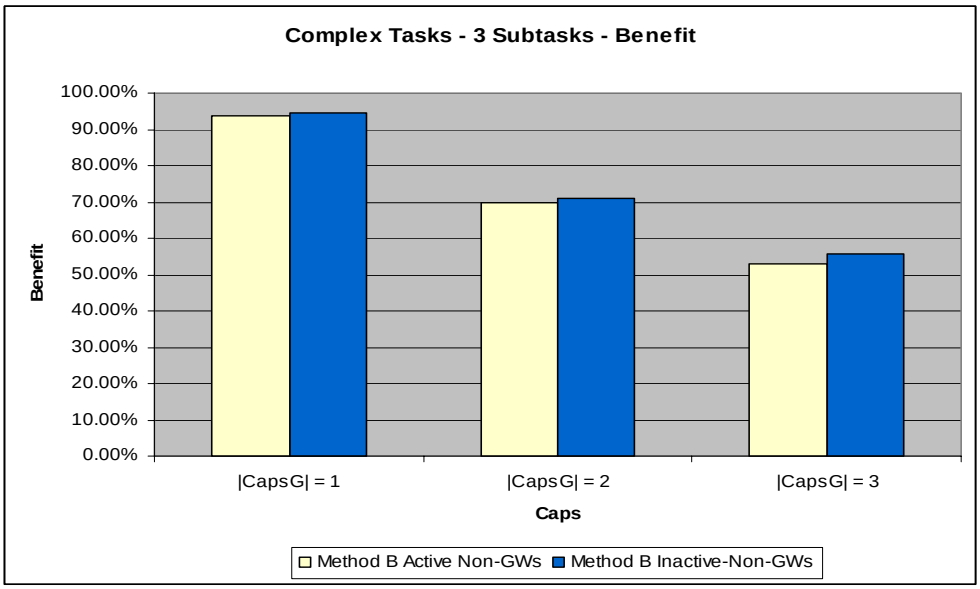


Figure 26. Benefit achieved when there are at most 3 subtasks per complex task.

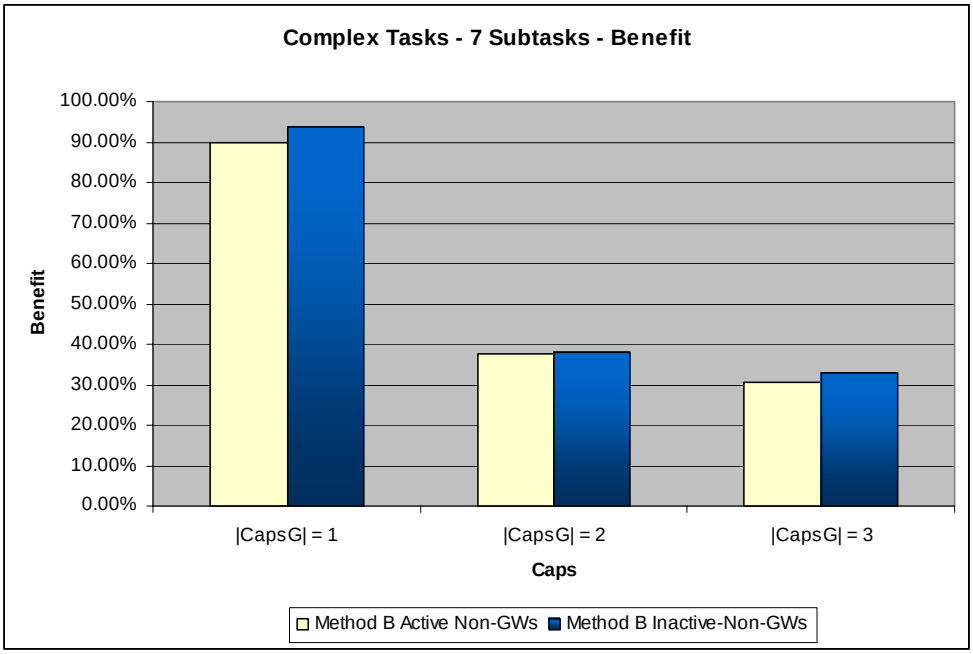


Figure 27. Benefit achieved when there are at most 7 subtasks per complex task.

Results show that the system's benefit remains high independent of the allocation method selected. The second method (method B with inactive non-gateways "non-GWs") performs slightly better compared to the other one in terms benefit but it incurs more exchanged messages, as in Section 7.3. Hence, the active non-gateways method achieves slightly better message gain in the case of 3 subtasks per complex task. However, in the case of 7 subtasks this is reversed due to the higher difference in benefit in favor of the inactive non-gateways method.

To obtain a better understanding of the difference in performance between the two variants of Method B, we performed a statistical analysis similar to the one presented in Section 7.4.1. Table 4 presents the results of this analysis that confirm that the inactive non-gateways variation always achieves a higher benefit on average compared to the active gateways variation. However there is only a small difference which is not always statistically significant. On the other hand, the inactive non-gateways method incurs a slight increase in the number of exchanges messages, but this increase is very small and rarely has a statistical significance.

Caps / Max. Sub-tasks	Benefit_B_i - Benefit_B_a				Messages_B_i - Messages_B_a			
	Mean	**SD**	**Z-value**	**p-value**	**Mean**	**SD**	**Z-value**	**p-value**
1/3	0.63	1.47	-1.073(a)	0.283	5473	5792	-2.395(a)	0.017
2/3	0.88	2.21	-0.866(a)	0.386	442	3440	-0.153(b)	0.878
3/3	2.66	2.00	-2.803(a)	0.005	1981	4899	-1.274(a)	0.203
1/7	3.98	2.20	-2.803(b)	0.005	1913	5169	-0.866(b)	0.386
2/7	0.32	2.30	-0.255(b)	0.799	2244	6860	-1.070(a)	0.285
3/7	2.22	2.23	-2.090(a)	0.037	1504	6635	-0.866(a)	0.386

Table 4. Statistical results comparing Method B with active non-gateways (denoted by B_a) to Method B with inactive non-gateways (denoted by B_i). Column 1 gives the number of capability types and the maximum number of subtasks per complex task. Columns 2-5 give statistics regarding the difference in achieved benefit. Columns 6-9 give statistics regarding the difference in sent messages. P-values and Z-values that indicate statistical significance are underlined. A sample of 10 instances was used for each combination of values for the capabilities and the maximum subtasks per complex task.

Comparing the results of Section 7.4 with the ones presented in Section 7.3 we can conclude that the proposed method for allocation and scheduling is quite robust with respect to the problem generation method (at least for geographical and random_connections networks). That is, the benefit achieved does not vary considerably between the two generation methods. Specifically, the variation between the results of paragraphs 7.3 and 7.4 with respect to the system's benefit was 1.9% on average, with maximum value of 3.85%, for the case of 3 subtasks per complex task. In the case of 7 subtasks the variation was 2.6% on average and the maximum difference was 3.14%. In all cases the benefit achieved on the networks generated with the random_connections method is higher.

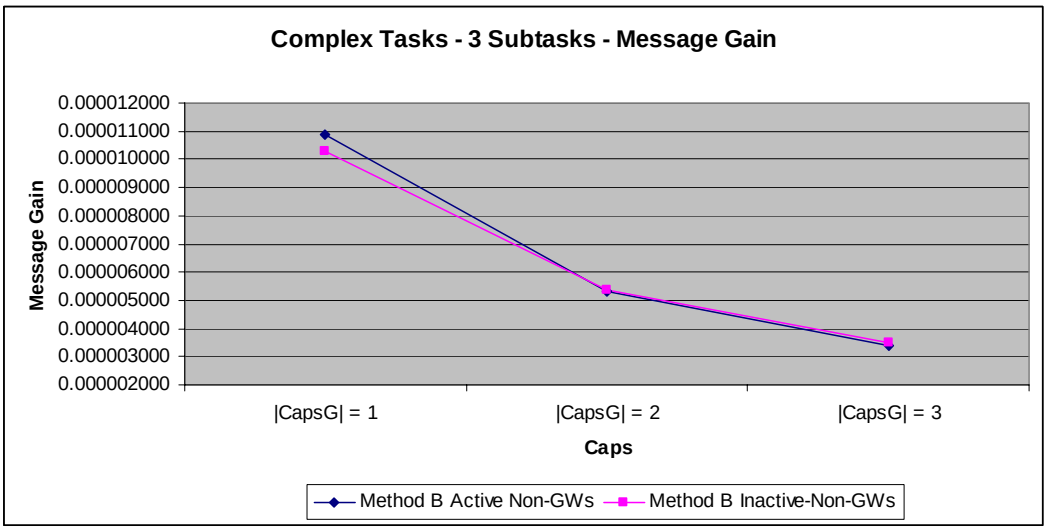


Figure 28. Message Gain when there are 3 subtasks per complex task.

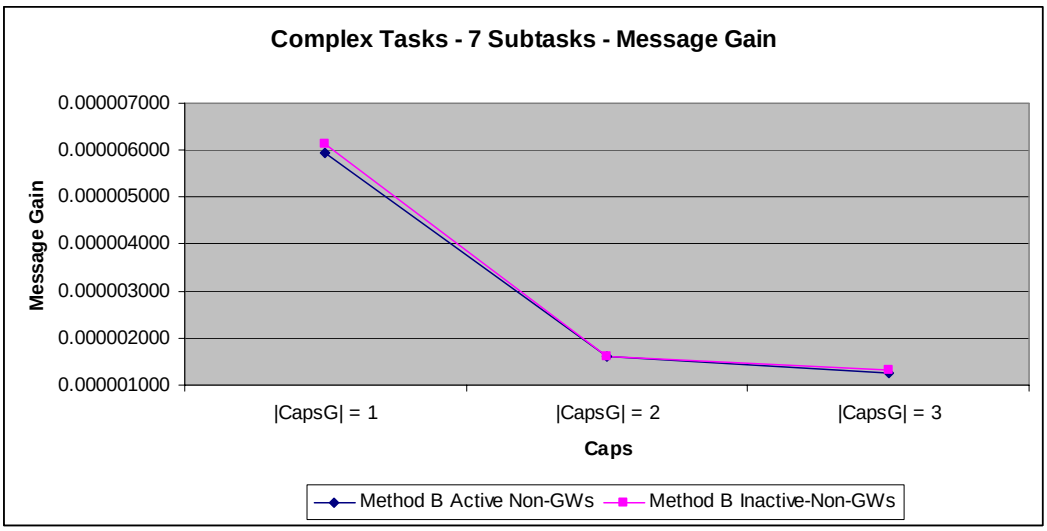


Figure 29. Message Gain when there are 7 subtasks per complex task.

Comparing the results from the geographical and random_connections types of networks that have the same average density (i.e. around 1.95%) the average benefit variation is roughly 1%, again in favor of the random_connections generation method. Although this is not significant statistically, we conjecture that on the random_connections networks the higher benefit may be due to th that a message can reach any node in the network in fewer hops (on average) than in geographical networks. This means that a task request that enters through a specific agent A_i in a geographical network, and requires resource availability and capabilities offered by distant agents only, may not be served because its TTL may expire before the appropriate agents are located and the corresponding PTN is formed. In contrast, in random_connections networks the appropriate agents could be reached faster (because of the random shortcuts), within the TTL, and the request might therefore be served. Of course, this conjecture requires further and more detailed experimental evidence.

7.4.3 Changing the distribution of capabilities

Up to this point capabilities have been assigned to agents using a uniform distribution. In practice though, many systems tend to behave in a different way as certain agent capability types are frequent and others are scarce. Hence, we also experimented with problem sets where capabilities are distributed in a different way. To be precise, the agent networks here were randomly generated using the random_connections method and capabilities of 3 distinct types were assigned to the agents following the Zipfean distribution. In this case, the number of agents

having capability type 1 was much higher than the one of agents having capability type 2. In turn, agents with capability type 2 are more than ones with capability type 3. The performance of the system is shown in Figures 30 (benefit) and 31 (message gain). As can be seen, there is a sharp drop in the benefit achieved compared to Figures 26 and 27, especially in the case of 3 subtasks. For example, in this case, the benefit of the non-active gateways method drops from approx. 60% down to just over 30%. Similar results hold with relevance to the message gain obtained.

These results are not unexpected since the system has increasing difficulty in finding agents capable of serving atomic tasks that require capabilities 2 or 3. Hence most of the complex tasks that include subtasks requiring these capabilities will not be served, resulting in reduced benefit. In addition, the difficulty to locate appropriate agents results in a larger volume of messages exchanged. On the other hand, if all the subtasks in a complex task require agents with capability type 1, it will be very easy to locate them, form a PTN and subsequently successfully schedule the task. However, this is not a very common case. Note that we did not use the Zipfean distribution to assign required capabilities to subtasks, but rather the uniform one. If we had used the Zipfean distribution for the subtasks then the distribution of capabilities to agents would follow that of capabilities to subtasks, resulting in easier problems.

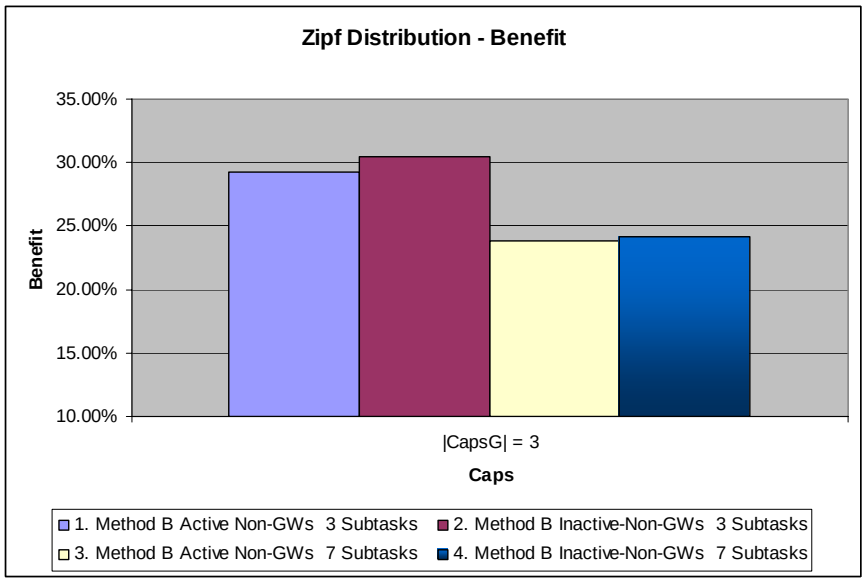


Figure 30. Benefit for networks with possible capability values 1, 2 or 3 following the Zipfean distribution.

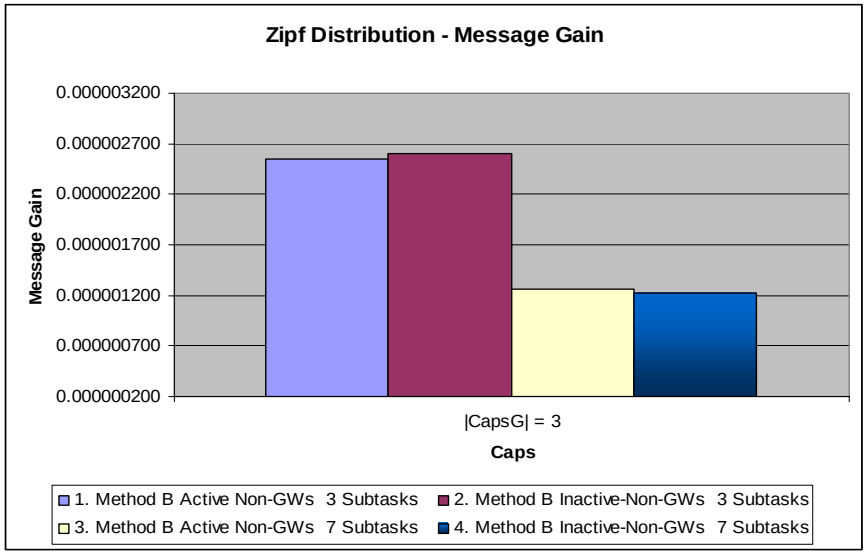


Figure 31. Message Gain for networks with possible capability values 1, 2 or 3 following the Zipfean distribution.

To conclude the experimental evaluation, viewing the task allocation and scheduling as a two step process we can note that different approaches to each step have different repercussions on the

system's efficiency. As demonstrated in Subsection 7.2, different approaches to the scheduling step can lead to significant variance in the amount of successfully scheduled tasks (benefit). Hence moving from local search to a sophisticated complete DCOP algorithm like Adopt is indeed very beneficial. On the contrary, in this and the previous subsections we have shown that trying to improve the task allocation process through more elaborate mechanisms does not have a similarly significant impact on the system's benefit since the use of the gateway/routing indices infrastructure is by itself very effective. However, it remains to be seen if even more intricate task allocation processes can have a significant impact on the system's benefit for complex problems.

7.5 Discussion

The experimental results given in Sections 7.3 and 7.4 demonstrate that the benefit achieved by our method sharply declines as the problems become harder by increasing the number of available capabilities, especially when the number of subtasks per complex task is large. A natural question that follows is whether this is due to some inherent deficiency of our approach or to the extreme hardness of the generated instances. One way to answer this question is to measure the distance between the optimal benefit that can be achieved in the generated instances and the benefit achieved by our method. However, this is far from easy to do in practice since the problems we deal with are highly combinatorial and hence very hard. In practice, finding the optimal benefit in large enough random instances would require collecting all the generated tasks in a single agent and solving the resulting problem in a centralized way through some elaborate branch & bound search algorithm. Developing such an algorithm requires extensive research and is outside the scope of this paper. Therefore, measuring the optimal benefit in our randomly generated instances is beyond our reach at the moment.

To obtain an indication of our method's ability to approach the optimal benefit, we have run experiments with an alternative task generation method which guarantees that all generated tasks are satisfiable, not only on their own, but also considered collectively. Recall that the generation methods detailed above ensure that each task is satisfiable when considered individually but do not guarantee that all tasks are satisfiable when considered together. This alternative generation method proceeds as follows.

At first the agents' timelines are processed one by one and broken into smaller fragments. Each fragment's duration is determined randomly and is between 1 and 5 time units (half of the agent's full capacity). Then, the complex tasks are constructed. Assuming a task with x subtasks is required, we randomly choose x time fragments from the previously broken down timelines. Each individual fragment of every timeline is chosen only once. For instance, when generating a complex task with 3 subtasks, we randomly select 3 timeline fragments from 3 agents (which are not necessarily different). Each fragment becomes an individual subtask within the complex task. Constraints are then posted between the subtasks making sure that they are satisfied. For example, assuming two subtasks (timeline fragments) t_1=[1, 3] and t_2=[5, 8], a constraint $S_1+3<S_2$ may be posted. After a complex has been created, the corresponding timeline fragments are removed from the respective agents' timelines and the process is repeated until no more satisfiable complex tasks can be created. Any remaining timeline fragments are considered as atomic tasks.

This generation method guarantees that all tasks are satisfiable. This is because, ideally, any subtask of a given task can be allocated to the agent where the corresponding timeline fragment, from which it was created, belongs. Then this agent can schedule this subtask in exactly the "correct" timeline fragment. Therefore, the optimal benefit under this generation method is 100%.

We ran experiments with Method B in combination with inactive non-gateways on randomly generated problems with 200 agents where the total capacity required by the generated tasks is equal to the total capacity of the agents. The results are given in Table 5. The first column gives the number of capabilities and the maximum number of subtask per complex task. Columns 2 to 4 give the average performance of our method measured in the metrics used across all our experiments. Column 5 gives the average % coverage of the total capacity of the agents (i.e. the aggregation of their timelines) once the scheduling of the tasks has been completed. For instance, 96.78% coverage in the case of one capability and 3 subtasks per task at maximum means that once the allocation and scheduling of the tasks was completed, the 3.22% of the total capacity was free (i.e. not allocated to any task). Column 6 gives the uniform TTL of the generated tasks.

Capabilities – Max. Subtasks	**Benefit**	**Messages**	**Ben/Mess**	**%Timeline Covered**	**TTL**
1 - 3	97.38%	102368	0.000009513	96.78%	10
1 - 7	95.17%	157422	0.000006046	94.83%	10

3 - 3	63.27%	167347	0.000003781	62.76%	30
3 - 7	39.83%	198563	0.000002006	39.21%	30
3 - 3	67.49%	173168	0.000003897	68.15%	50
3 - 7	43.84%	203722	0.000002152	42.87%	50

Table 5. Average performance of Method B with inactive non-gateways on problems where all tasks are guaranteed to be satisfiable.

Table 5 demonstrates that as the TTL of the tasks increases the benefit achieved also increases. Recall that the TTL in all experiments presented above was set to 10, which is relatively low. This explains, to some extend, the drop in the achieved benefit as the tasks get harder. However, even with TTL=50, the benefit of the system is still under 50% in the case of 3 capabilities and 7 subtasks per complex task. This is of course very far from the optimal 100% benefit but we must note that it is considerably higher than the benefit achieved under the previously used task generation methods. This is evident if we compare the results of the last line in Table 5 with those given in Figures 21 and 27.

Concluding the above, we believe that these results do not demonstrate a deficiency in our approach but they rather show that the problems we tackle are very hard. As explained, the generation method we try here guarantees that all tasks can be satisfied, but in practice it will be nearly impossible for any heuristic method to locate the appropriate agents and schedule the subtasks to the "correct" timeline fragments for all the generated tasks.

8 Conclusions

We proposed a novel method for allocating atomic and complex tasks in large-scale networks of homogeneous or heterogeneous cooperative agents. In contrast to prior work, we treat searching, task allocation and scheduling as a single problem and propose a decentralized method for all these tasks where no accumulated or centralized knowledge or coordination is necessary. Efficient searching for agent groups that can facilitate task requests is accomplished through the use of a dynamic overlay structure of gateway agents and the exploitation of routing indices. The task allocation and scheduling of complex tasks is accomplished by combining dynamic reorganization of agent groups and distributed constraint optimization methods. Experimental results displayed the efficiency of the proposed method.

In the immediate future we plan to perform a more in-depth experimental investigation of the effect that the various parameters have on the system's performance. Specifically, we are referring to the way networks are generated, their density (as dictated by n and r for geographical networks for example), the size of the complex tasks in terms of subtasks they include, the density of the constraint graph for each complex task, and the number of available capabilities.

Further work targets investigating in more detail the trade-off between DCOP solving and dynamic reorganization of PTNs during the scheduling process. First we intend to clarify through extensive experimentation the contribution that the DCOP algorithm has to the resolution of complex tasks compared to the dynamic reorganization of PTNs. That is, we will measure the percentage of complex tasks that are successfully allocated without reorganization and the extent of reorganization that occurs on average. This investigation will hopefully lead to the development of efficient heuristic methods for choosing whether to continue employing sophisticated DCOP algorithms or simply reorganize the PTN once conflicts are encountered during the scheduling process.

Other, more general, directions for future work include extending our framework to consider other types of resources and tasks with uncertain durations as, as well as to situations where the agents have preferences on the tasks they can serve. Also, it would be interesting to study the use of alternative DCOP algorithms within our method, such as DPOP and its extensions.

Finally, a direction we aim to pursue is the extension of our approach so that changes to agent commitments can be made as tasks arrive dynamically so that a better overall allocation can be achieved. This requires modifying the scheduling and allocation methods involved in our approach, and will most likely increase their run times, but may well result in higher system benefit being obtained.

Acknowledgements

We would like to thank the anonymous reviewers of an earlier version of this paper for their insightful comments that helped improve this paper.

References

[1] Atlas J., Decker K., "A complete distributed constraint optimization method for non-traditional pseudotree arrangements". In *Proc. of* AAMAS 2007: 111, 2007.

[2] M. Boddy, B. Horling, J. Phelps, R. Goldman, R. Vincent, A. C. Long, R. Kohout, and R. Maheswaran. *C-TAEMS language spec. v. 2.02*, 2006.

[3] Carle J., Simplot-Ryl D., "Energy-Efficient Area Monitoring for Sensor Networks", *Ad-Hoc Networks, IEEE Computer Society*, pp. 40-46, 2004.

[4] Crespo, A., Garcia-Molina, H. Routing indices for peer-to-peer systems, in *Proc. of the 28th Conference on Distributed Computing Systems*, July 2002.

[5] Dai F., and Wu J.,. "Distributed Dominant Pruning in Ad Hoc Networks", In Proc. *IEEE 2003 Int'l Conf. Communications (ICC 2003),* pp.353-357, 2003.

[6] Davin J., Modi P.J., "Hierarchical variable ordering for distributed constraint optimization". In Proc. of AAMAS 2006: 1433-1435, 2006.

[7] K. Decker. "TAEMS: A Framework for Environment Centered Analysis & Design of Coordination Mechanisms". In *Foundations of Distributed Artificial Intelligence, Chapter 16*, pages 429–448. G. O'Hare and N. Jennings (eds.), Wiley Inter-Science, January 1996.

[8] Fitzpatrick S. and Meertens L., "An experimental assessment of a stochastic, anytime, decentralized, soft colourer for sparse graphs", in: Proc. of the 1st *Symp. on Stochastic Algorithms: Foundations and Applications*, pp. 49–64, 2001.

[9] Goldman, C., and Zilberstein, S. "Decentralized Control of Cooperative Systems: Categoriation and Complexity Analysis". *JAIR* 22:143-174, 2004.

[10] Goldman, C., and Zilberstein, S. "Optimizing Information Exchange in Cooperative Multi-agent Systems". In *Proc. Proc. of AAMAS 2003*, ACM Press, July 2003.

[11] Hirayama K. and Yokoo M., "The distributed breakout algorithms". *Artificial Intelligence*, 161:89-115, 2005.

[12] B.Horling, "Quantitative Organizational Modelling and Design for Multi-Agent Systems", PhD Dissertation, Univ. of Massachusetts Amherst, 2006.

[13] Koes, M., Nourbakhsh, I., and Sycara K. "Heterogeneous Multirobot Coordination with Spatial and Temporal Constraints". In *Proc. of AAAI 2005,* 1292—1297, 2005.

[14] R. T. Maheswaran, P. Szekely, M. Becker, S. Fitzpatrick, G. Gati, J. Jin, R. Neches, N. Noori, C. Rogers, R. Sanchez, K. Smyth, and C. VanBuskirk. "Predictability & criticality metrics for coordination in complex environments". In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2008)*:647-654, 2008.

[15] Mailler, R. and Lesser, V. "A Cooperative Mediation-Based Protocol for Dynamic, Distributed Resource Allocation". *IEEE Transaction on Systems, Man, and Cybernetics, Part C*, Special Issue on Game-theoretic Analysis and Stochastic Simulation of Negotiation Agents, 36(1):80-91, 2006.

[16] Mailler, R. "Using Prior Knowledge to Improve Distributed Hill Climbing". In *Proc. of the 2006 International Conference on Intelligent Agent Technology* (IAT), 2006.

[17] Minton S., Johnston M.D., A.B. Philips, and P. Laird, "Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems", *Artificial Intelligence* 58 (1–3) 161–205, 1992.

[18] Modi P.J., Shen W.M., Tambe M., and Yokoo M., "Adopt: Asynchronous Distributed Constraint Optimization with Quality Guarantees", *Artificial Intelligence*, 161:149-180, 2005.

[19] D. J. Musliner, E. H. Durfee, J. Wu, D. A. Dolgov, R. P. Goldman, and M. S. Boddy. "Coordinated plan management using multiagent MDPs". In *Proceedings of the 2006 AAAI Spring Symposium on Distributed Plan and Schedule Management*, March 2006.

[20] R. Nair, M. Tambe, and S. Marsella. "Role allocation and reallocation in multiagent teams: Towards a practical analysis". In Proceedings of Second International Joint Conference on Autonomous Agents and Multi-agent Systems (AAMAS-03), pages 552–559, 2003.

[21] Petcu A. and Faltings B., "A Scalable Method for Multiagent Constraint Optimization". In *Proc. of IJCAI 2005*, 266-271, 2005.

[22] Pynadath, D.V., Tambe, M. "Multiagent Teamwork: Analyzing the Optimality and Complexity of Key Theories and Models". In *Proc. of AAMAS 02,* 873-880, 2002.

[23] Scerri, P., Farinelli, A., Okamoto, S., Tambe, M. "Allocating Tasks in Extreme Teams". In *Proc. Of AAMAS 05,* 2005.

[24] Sims, M., Corkill, D., and Lesser V. “Knowledgeable Automated Organization Design for Multi-Agent Systems”, Autonomous Agents and Multi-Agent Systems, Volume 16, Issue 2 (April 2008), pp 151 – 185, 2008.

[25] S. Smith, A. T. Gallagher, T. L. Zimmerman, L. Barbulescu, and Z. Rubinstein. “Distributed management of flexible times schedules”. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multiagent System (AAMAS-2007)*:74, 2007.

[26] Sultanik E., Modi P.J., Regli W., “On Modeling Multiagent Task Scheduling as a Distributed Constraint Optimization Problem”. In *Proc. of* IJCAI 2007, 1531-1536, 2007.

[27] Theoharopoulou C., Partsakoulakis I., Vouros G., Stergiou K. “Overlay Networls for Task Allocation and Coordination in Large-Scale Networks of Cooperative Agents”. In *Proc. of AAMAS-2007*, 55, 2007.

[28] Xu, Y., Scerri, P., Yu, B., Lewis, M., and Sycara, K. “A POMDP Approach to Token-Based Team Coordination”. In *Proc of AAMAS’05*, (July 25-29, Utrecht) ACM Press, 2005.

[29] Xu, Y., Scerri, P., Yu, B., Okamoto, S., Lewis, M., and Sycara, K. “An Integrated Token Based Algorithm for Scalable Coordination”. In *Proc. of AAMAS’05,* 407-414, 2005.

[30] Xuan, P., Lesser, V., Zilberstein, S. “Communication Decisions in Multi-agent Cooperation: Model and Experiments”. In *Proc of AGENTS’01,* 2001, 616-623, 2001.

[31] Yen, J., Yin, J., Ioeger, T.R., Miller, M.S., Xu, D. and Volz, R. CAST: Collaborative Agents for Simulating Teamwork. In *Proc. of IJCAI 2001,* 1135-1144, 2001.

[32] Yeoh W., Felner A., Koenig S., “BnB-ADOPT: an asynchronous branch-and-bound DCOP algorithm”. In *Proc. of* AAMAS (2) 2008: 591-598, 2008.

[33] Yokoo M., Durfee E.H., Ishida T., and Kuwabara K., “Distributed Constraint Satisfaction Problem: Formalization and Algorithms”, *IEEE Trans. on Knowledge and Data Engineering*, 10:673-685, 1998.

[34] Zhang, Y., Volz, R., Ioeger, T.R., Yen, J. “A Decision Theoretic Approach for Designing Proactive Communication in Multi-Agent Teamwork”. In *Proc of SAC’04,* 64-71, 2004.

[35] Zhang, W., Wang, G., Xing, Z., and Wittenburg, L. “Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks”. *Artificial Intelligence*, 161:55-87, 2005.