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

Christina Theocharopoulou Ioannis Partsakoulakis George A. Vouros Kostas Stergiou

Dept. of Information & Communication Systems Engineering
University of the Aegean, Samos, Greece
{chtheo, jpar, georgev, konsterg} @aegean.gr

ABSTRACT

This work proposes a method for allocating temporally interdependent tasks to homogeneous or heterogeneous cooperative agents in dynamic large-scale networks. This method views searching, task allocation and scheduling as an integrated problem that has to be efficiently solved in such networks. Solving the general problem optimally in a decentralized way is very hard and can only be solved by a centralized method, be approximated by means of heuristics, or by relaxations of the original problem. Our method facilitates effective searching through the dynamic assignment of gateway roles to agents and the exploitation of routing indices. In combination to searching, it exploits distributed constraint satisfaction techniques and dynamic re-organization of agent teams to efficiently handle the allocation of complex tasks with interdependent subtasks.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence – *multiagent systems*

Keywords

Task and resource allocation, coordination, cooperation, and distributed constraint processing.

1. INTRODUCTION

This paper views searching, task allocation and scheduling in dynamic large-scale networks of randomly deployed agents as an integrated problem and proposes an approach that combines methods for the dynamic construction of overlay networks of gateway agents, the construction and maintenance of routing indices, and for distributed constraint satisfaction. 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 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) [5]. 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 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).

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. The proposed method facilitates effective and efficient searching through the dynamic assignment of gateway roles to 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 emergent situations, they propagate requests to subsidiaries, which act so as to form teams depending on the requirements of each situation: They check the required capabilities and jointly schedule their activities taking into account interdependencies, as well as their other scheduled activities. Agents may propagate a request in case they do not have the required capabilities and/or resources. It is important to notice that in such a setting agents can not be considered to form any network with specific properties: They are considered to be randomly deployed in a terrain, where events requiring action appear randomly. In such a dynamic setting, where agents may join or leave the network, and new requests arrive, efficient task allocation and coordination of activities is crucial, considering that agents form a random network of acquaintances.

Being interested in the development of decentralized policies that can support agents to manage their resources efficiently, the proposed method builds on self-organization approaches for ad-hoc networks, token-based approaches for coordination in large-scale systems, and distributed constraint satisfaction. Aiming to increase the efficiency and the benefits of a multiagent system, this paper:

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

(a) Integrates searching, task allocation and scheduling in large-scale dynamic systems of cooperative agents. The proposed method is demonstrated by allocating temporally inderdependent tasks with specific capability requirements to time-bounded agents.
(b) Demonstrates 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 this complex problem efficiently.
(c) It provides experimental results from a simulated setting that demonstrate the efficiency of the proposed method.
The rest of the paper is structured as follows: In Section 2 we review related work, while in Section 3 we state the problem. In Section 4 we describe the individual techniques employed, while in Section 5 we present the overall method for integrating searching, task allocation and scheduling. Experimental results are shown in Section 6. Finally, Section 7 concludes the paper and points to future work.

2. RELATED WORK

As already pointed, decentralized control of large-scale dynamic systems of cooperative agents is a hard problem in the general case. In cases where the system has to perform joint activities that involve temporally interdependent subsidiary tasks, controlling at the planning stage 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 [5]. 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.

However, in a large-scale dynamic 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, 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 interdependent tasks appropriately. This last example shows that transitions are interdependent too, which further complicates the computations.

Decentralized control of such a system in cases where agents have to act jointly in the presence of temporal constraints, is a challenge. This challenge has been recognized in [8], 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 dynamic 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 [13], 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 constraints, extending the problem along the following dimensions: (a) Handling temporal constraints among tasks, (b) dealing with dynamic networks of acquaintances, (c) dealing with agents that do not have the capabilities to perform every task, and (d) integrating searching with task allocation and scheduling.

Token based approaches are promising for scaling coordination to large-scale systems effectively. The algorithm proposed in [15] focuses on routing models and builds on individual token-based algorithms. Additionally, in [15] authors provide a mathematical framework for routing tokens, providing a three-level approximation to solving the original problem. These works do not 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.

It has to be noticed that in this paper we do not deal with communication decisions for optimizing information sharing/exchange as it done in [6], or for proactively exchanging information [19]: This is orthogonal to our research which can 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 [16, 17, 12]).

3. PROBLEM DEFINITION

In this paper we consider time-bounded agents with specific discrete capabilities, and tasks that require the use of a certain resource for a fixed duration. We assume that each agent has a single type of resource and at each time point at most one task 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.

The acquaintance network of agents is modelled as a graph $AN=(N,E)$, 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 $N(A_i)$. 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_i) (resp., *latest*(R_i)) denotes the time point r_{ik} in R_i, for which there is not any other point r in R_i with $r < r_{ik}$ (resp. $r_{ik} < r$). The priority is a unique identifier (remaining battery life, number of neighbors, etc.) that is assigned to each agent that enters the multi agent system [1]: This is necessary for the computation of an overlay network (explained in 4.1) and for the non-cyclic update of routing indices (explained in 4.2)

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 it may require the joint achievement of a set of 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 $<a, start(t_i), end(t_i), Cap_i>$, where a, $start(t_i)$ and $end(t_i)$, 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 task t_i consisting of a set of sub-tasks $\{g_{i1}, .., g_{ik}\}$ there is 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}.

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

(a) for each atomic task $t_i = <a, start(t_i), end(t_i), Cap_i>$ 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:

$\{(Cap_i \subseteq Cap_j) \wedge (\exists R \subseteq R_j \text{ s.t. } |R| \geq a \wedge start(t_i) \leq earliest(R) \wedge latest(R) \leq end(t_i))\}$

On the contrary, $perform(A_j, t_i)=0$

(b) for each joint 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 maximally satisfied. For that purpose, following [10], 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 4.3 we explain how the cost functions are evaluated and how the aggregation is computed. The aim is to increase the benefit of the system, i.e. the ratio of tasks allocated to the total number of requests, and to increase the message gain, i.e. the ratio of the benefit to the number of messages exchanged.

4. SELF-ORGANIZATION, SEARCHING AND SCHEDULING

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

4.1 Dynamic overlay network 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 [3]. 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 [1]:

- 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 [1,3], nodes may be considered to cover an information space, or the space of capabilities/resources required for the execution of tasks. The algorithm of Dai and Wu [3] 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 [1]. The algorithm is as shown in Figure 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 either in this sub-graph

or

the neighbour of at least one node in the subgraph),

then *opt* for a non-gateway node.

Else *opt for* a gateway node.

Figure 1. Computing overlay network of gateways

Dominating nodes (gateway agents) are dynamically computed as 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.

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 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 [2].

To compactly capture information about the availability of time units, each gateway 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 $Vi=(<0,3>,<1,2>,<2,1>,<3,0>,<4,0>,<5,1>,<6,0>,<7,2>,<8,1>,<9,0>)$ represents the time line shown in Figure 2. 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.

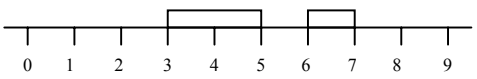

Figure 2. Vector of time-units availability

Assuming that m (the number of time units) 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 apply to any type of resources that can be committed to a single request.

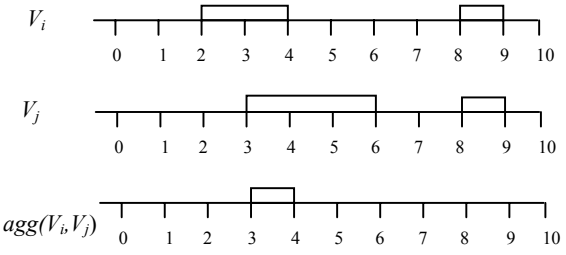

Figure 3. Aggregation of vectors

Figure 3 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. 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 availability of all agents in $N(A) \cup \{A\}$. Specifically, given $RI(A)$, A updates the indices of its neighbours as follows: A sends the aggregation of the vectors of the agents in $N(A) \cup \{A\} - \{A_i\}$ to each agent A_i in $N(A)$.

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 $\{RI_{capl},..., RI_{capm}\}$. 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 indices' updates propagate in cycles [2]: 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 aggregation 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. This is done dynamically and in parallel to the propagation of requests.

4.3 Distributed constraint solving

A *Distributed Constraint Satisfaction Problem* (dis CSP) consists of a set of agents, each of which controls one or more variables, a (finite) domain for each variable, and a set of constraints [18]. 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* (dis COP) is a dis CSP with an optimization function which the agents must cooperatively optimize. The dis COP framework has recently emerged as a promising framework for modelling and solving certain task and resource allocation problems in multiagent systems.

To efficiently capture interdependencies between subtasks of a complex task, we model complex tasks as dis COPs as follows:

- 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

$A=\{A_1,...,A_m\}$ is located using the gateway and routing indices searching infrastructure described in Sections 4.1 and 4.2. This is further explained in Section 5. Each $A_j \in A$ has the necessary capabilities and resource availability to undertake at least one subtask.

- For each subtask $g_i=<a, start(g_i), end(g_i), Cap_i>$ there is a variable X_i controlled by an agent $A_j \in A$, corresponding to the start time of g_i.
- The domain $D(X_i)$ of each variable X_i is a set $R \subseteq R_j$ of time points that agent A_j can allocate to the start time of g_i. $D(X_i)$ includes each time point $r \in R_j$ such that all time units between r and $r+\alpha$ are available on A_j's time vector.
- There is a hard intra-agent constraint between any two subtasks that are allocated to an agent A_j specifying that the execution of the two subtasks cannot overlap[2]. To be precise, for any two subtasks g_i and g_l allocated to A_j, with durations α_i and α_l respectively, there is a hard intra-agent constraint c_{il} $(\in C_t) = (X_i + \alpha_i \le X_l) \vee (X_l + \alpha_l \le X_i)$. 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 complex tasks. This occurs when an agent participates in the allocation of more than one complex task.
- 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 [10]). Recall that a precedence constraint specifies a temporal distance between the executions of the corresponding subtasks. For example, constraint $X_i + \alpha_i + 5 \le X_l$ 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. Cost function f_{il} is defined as follows:

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

where $d_i \in D(X_i)$, $d_l \in D(X_l)$ and M is a non negative integer. M is a measure of the constraint's violation defined as the minimum distance that the starting time of one of the subtasks has to be moved 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, according to the algorithm described further below. For example let us consider the constraint c_{il}: $X_i + \alpha_i + 5 \le X_l$ 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: $Q \le K$, M is defined as follows:

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

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

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

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 :

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

We consider summation as the aggregation operator, but this is not a requirement.

To achieve efficient task allocation in dynamic settings where several requests may enter the system simultaneously while agents may enter or leave the system at will, it is essential that a fast, scalable, and dynamically adaptable task allocation method is used. Therefore, computationally expensive complete dis CSP and dis COP algorithms, such as Adopt [10] and DPOP [11] may not be suitable. Even sophisticated distributed incomplete local search methods, such as distributed breakout [7], can be too expensive in large-scale environments. Instead, we use a simple local search method and utilize dynamic self-organization of agent teams to cope with local optima.

This method can be seen as an instantiation of the Distributed Stochastic Algorithm (DSA) [4] in which each agent may control more than one variable, the optimization criterion takes into account the cost functions of the soft constraints and only a min conflicts hill-climbing procedure [9] (without random moves) is run. Figure 4 depicts this algorithm.

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

Figure 4. Hill-Climbing algorithm executed by all agents

When an agent A_j 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 constraint graph (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). These assignments can be made using some amount of randomness. At the moment we do not employ random moves, and only make a new assignment if it can reduce the sum of the cost functions. Note that no single agent has complete knowledge of $F(C_t)$. Therefore, each agent tries to minimize the aggregation of the cost functions for the constraints that it is

[2] 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.

"aware of". That is, the constraints that involve variables that it controls. In the next section we explain how we handle failures (local optima) and dynamic changes in teams (i.e. sudden exit of agents) using self-reorganization.

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 be able to find optimal (or simply satisfying) allocations even if they exist. This means that some complex tasks may not be served although there may be agents in the systems that can cooperatively serve them.

5. INTEGRATION OF SEARCHING, TASK ALLOCATION, AND SCHEDULING

This section describes the interplay of the methods described in the previous section to efficiently handle task allocation and scheduling in large dynamic networks of agents.

Given a network of acquaintances *AN*, this is first self-organized into a network where a set of agents form a connected overlay sub-network of "gateways" *GN*. Each time a change happens in *AN* (due to uncontrollable events), agents may need to self-reorganize into 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 *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 [5]. Gateways' views (i.e. routing indices) are maintained by means of capability-informing and resource-informing tokens.

This dynamic, self-organizing searching infrastructure supports the formation of teams for the performance of joint activities: Given a joint task that is decomposed into atomic temporally interdependent subsidiary tasks, agents have to search for a set of agents where each one has the resources and the capabilities to perform at least one subtask. All the requests for task allocation, as well as tokens, travel in *GN*.

Specifically, considering a request about the atomic task g_i=<*a, start*(g_i), *end*(g_i),cap_i> submitted to an agent A_j, searching proceeds as follows: If the request can not be served by A_j, then in case A_j is a non-gateway node, it forwards the task request to a gateway neighbour. If A_j is a gateway node, it examines whether the task can be served by any of its non-gateway neighbours (i.e. by any of the agents it covers). If not, then it forwards the task request to all its lower-priority gateway neighbours which, according to the stored indices, may propagate the request to an agent that can serve it. In case there is no gateway neighbour with lower priority, then A_j propagates the request to all its gateway neighbours. Since requests propagate through many different gateways, it is possible that more than one agent that can serve a request is found. 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).

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. The search for each of these atomic tasks proceeds as it has been described in the previous paragraph. The agents to whom the sub-stasks are allocated (i.e. have the required capabilities and resources) form a logical network of potential team-mates *PTN,* who jointly try to schedule their activities with respect to the temporal constraints and their other activities, using the distributed local search algorithm of Section 4.3. In case they are not successful in forming a common schedule, and depending on the violated constraints, they reform *PTN* until a team is formed or the living time (*TTL)* of the request for the joint activity expires.

Reformation of a *PTN* happens as follows: Request originators set a time limit that concerns the time-to-progress as far as the reconciliation of temporal constraint is concerned. This means that if this time-to-progress is exceeded and no solution to the dis COP has been found (i.e. agents in *PTN* did not form a team), then the request originator asks one of the agents to release its subtask and propagate the request for this subtask. Reformation of a *PTN* may proceed as long as the corresponding joint task exceeds a specific time-to-live (*TTL*): In this case the task is considered as unsatisfied.

We have to point that agents do not commit to the performance of specific tasks (they are dealing with planning without interleaving execution). This allows them to reconsider their schedules when they face requests for participation in new joint activities.

6. EXPERIMENTS AND RESULTS

To model networks of randomly deployed agents, we assume that the geographical distance among agents determines the topology of the system. Each node *A* establishes connections with nodes that "live" in a specific distance from him, according to a given radius r. In our experiments we assume that all nodes of the network are randomly placed in a limited square region and each one determines its neighbors by using parameter r: All nodes located in a cycle with center *A* and radius r are neighbors of *A*.

The following experiments concern networks *AN=(N,E)*, with |N|=500 nodes. We assume that each agent A_i possesses S_i amount of time resource. 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*(t_i), *end*(t_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.

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 have compared our strategy to a method that uses simple *blind flooding* propagation of task-requests as a baseline case. In this method, each agent that cannot satisfy a given task forwards a task-request to all his neighbours. Searching using blind flooding can provide a reasonable estimation of the maximum number of tasks that can be satisfied (using a fixed method for task allocation), and thus the maximum benefit that can be achieved.

We have ran two types of experiments: (a) allocation of atomic tasks (figures 5, 6) where $|Cap_{AN}|$ takes values 1, 5 and 10 and (b)

allocation of joint tasks (figures 7-9), where $|Cap_{AN}|$ takes values 1, 5 and 10 and the maximum number of subtasks that constitutes a complex task varies (taking values 3 and 7). Both the capabilities of the agents and the ones required by the tasks are randomly assigned. In the first type of experiments blind flooding achieves high benefit but performs searching with a huge number of task messages. On the other hand our method performs searching with considerably smaller communication cost, while at the same time achieving high benefit.

In the second type of experiments flooding succeeds high benefit as $|Cap_{AN}|$ increases (figure 7), but also has very high communication costs, compared to our method (figure 8). We also ran the same experiments with TTL increased by 600%. As expected, flooding achieves higher benefit than before, albeit by producing a much higher number of messages. The message gain of our method remains considerably higher than that of flooding, but it has to be noticed that the benefit in problems with complex tasks is less than that of flooding.

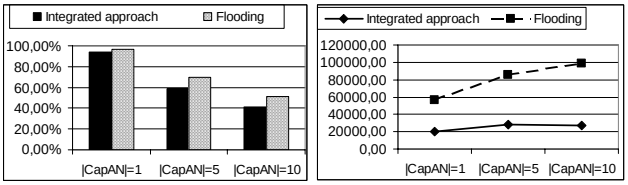

Figure 5. (left) The system's benefit and (right) exchanged messages

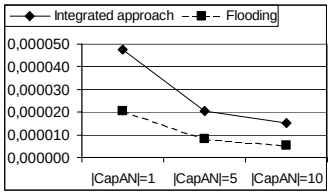

Figure 6. Message gain

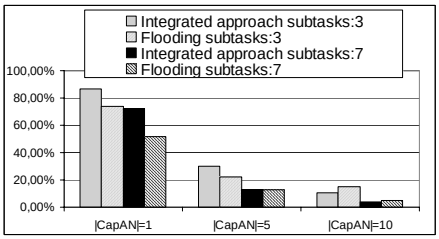

Figure 7. The system's benefit for joint tasks with at most 3 or 7 subtasks for varying number of capabilities.

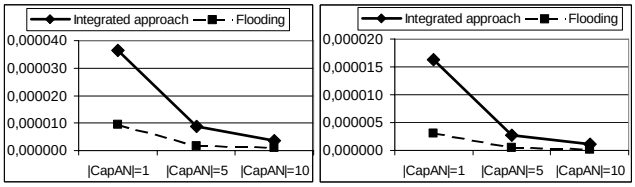

Figure 8. Message gain for the experiments of figure 7. (left) 3 subtasks, (right) 7 subtasks

6.1 Reorganization in open networks

In real networks the influx of new events is continuously changing the state of the system. Agents may appear or disappear which implies that additional resources may become available or that available resources may disappear. In such topologies, gateways have to be reorganized and RIs must be constantly updated. In the following we present results from two types of experiments with networks consisting of 500 agents, sharing 5 different types of capabilities and aiming to allocate a) atomic tasks and b) joint tasks with at most 3 subtasks.

We have examined dynamic networks where 20% of the network's agents exit and then later re-enter the system and networks where this percentage is 50%. Each agent leaves for at most 30% of the system's lifetime. Figures 10 and 11 present results from the first type of experiments (atomic tasks). For comparison, we also give results from static networks. Even though the number of nodes decreases dynamically by 20% or by 50%, our method succeeds high benefit (very close to that achieved in a static network). As far as the exchanged messages are concerned, our method increases their total number only by 6%, when the percentage of agents that exit and re-enter the system is at 20%, and only by 11%, when the percentage of agents that exit and re-enter is at 50%.

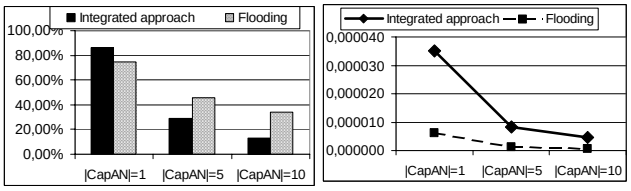

Figure 9. Results for joint tasks with at most 3 subtasks and increased TTL: (left) benefit and (right) message gain

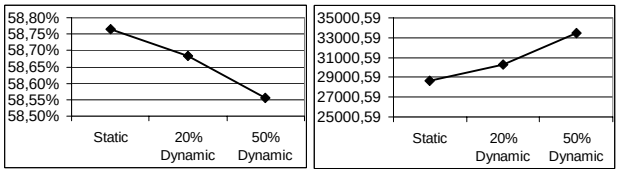

Figure 10. (left) System's benefit, (right) number of messages

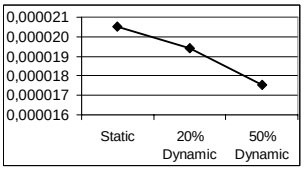

Figure 11. Message gain

Figures 12 to 14 present results from the second type of experiments (joint tasks). As shown, our method is scalable: even if the number of agents that exit and re-enter the system increases substantially, the benefit is kept high. Also, the total number of exchanged messages increases only by 4% when the agents that exit and re-enter the system are at 20% and only by 13% when this percentage is at 50%. Figure 14 gives the numbers of agents that are committed to perform certain numbers of complex tasks. As it is shown, our method distributes the workload relatively evenly among the agents, since most of them participate in at least 2 complex tasks.

7. CONCLUSIONS AND FUTURE WORK

We introduced a new method for searching and task allocation in dynamic large-scale networks of agents. This method facilitates effective searching through the dynamic assignment of gateway

roles to agents and the exploitation of routing indices. In addition, it uses dis CSP techniques together with dynamic self-organization of agent teams to perform task allocation. Preliminary experimental results demonstrate the efficiency and scalability of the proposed method in dynamic random networks.

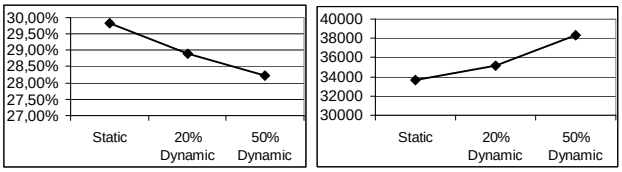

Figure 12. a) the system's benefit b) number of messages exchanged

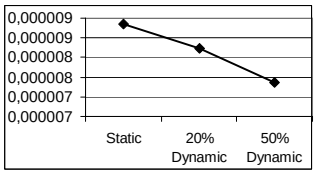

Figure 13. Message gain

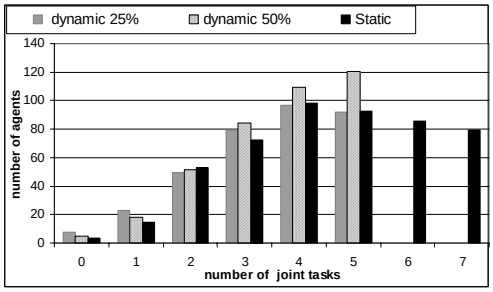

Figure 14. Numbers of agents committed to joint tasks.

There is still much work to be done. Among other issues we intend to investigate the following:

a) Dynamic assignment of priorities to agents based on their attributes, in order to better manage the distribution of the workload. Preliminary experiments in this direction showed that priorities and workloads have a high connection. Specifically a) higher priority nodes are more likely to become gateways and b) gateways are the nodes with high workload, as far as the number of received messages is concerned. This implies that if nodes with high workload manage to reduce their priorities, then it is more likely that their workload will be also reduced.

b) Exploitation of more advanced dis CSP and dis COP techniques (such as the algorithm of [13]) for task allocation, and in-depth investigation of the trade-offs between constraint solving and dynamic organization for achieving feasible task allocations. At the moment we quickly abandon search in the formulated dis COPs (as soon as a local optimum is met), meaning that our emphasis is on dynamic organization.

c) Consideration of other types of utilities, apart from the degree of the constraints' violation, as well as varying degrees of capabilities for the agents.

8. REFERENCES

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

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

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

[4] 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.

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

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

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

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

[9] 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.

[10] 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.

[11] Petcu A. and Faltings B., A Scalable Method for Multiagent Constraint Optimization, IJCAI 2005, 266-271.

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

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

[14] 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.

[15] 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

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

[17] 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.

[18] 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.

[19] 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,*2004, 64-71.