The Computer Journal Advance Access published March 24, 2010

doi:10.1093/comjnl/bxq022

Decentralized Coordination in RoboCup Rescue

SARVAPALI D. RAMCHURN[1,*], ALESSANDRO FARINELLI[1,2], KATHRYN S. MACARTHUR[1] AND NICHOLAS R. JENNINGS[1]

[1]*School of Electronics and Computer Science, University of Southampton, Southampton SO17 1BJ, UK*
[2]*Computer Science Department, University of Verona, Verona, Italy*
**Corresponding author: sdr@ecs.soton.ac.uk*

Emergency responders are faced with a number of significant challenges when managing major disasters. First, the number of rescue tasks posed is usually larger than the number of responders (or agents) and the resources available to them. Second, each task is likely to require a different level of effort in order to be completed by its deadline. Third, new tasks may continually appear or disappear from the environment, thus requiring the responders to quickly recompute their allocation of resources. Fourth, forming teams or coalitions of multiple agents from different agencies is vital since no single agency will have all the resources needed to save victims, unblock roads and extinguish the fires which might erupt in the disaster space. Given this, coalitions have to be efficiently selected and scheduled to work across the disaster space so as to maximize the number of lives and the portion of the infrastructure saved. In particular, it is important that the selection of such coalitions should be performed in a decentralized fashion in order to avoid a single point of failure in the system. Moreover, it is critical that responders communicate only locally given they are likely to have limited battery power or minimal access to long-range communication devices. Against this background, we provide a novel decentralized solution to the coalition formation process that pervades disaster management. More specifically, we model the emergency management scenario defined in the RoboCup Rescue disaster simulation platform as a coalition formation with spatial and temporal constraints (CFST) problem where agents form coalitions to complete tasks, each with different demands. To design a decentralized algorithm for CFST, we formulate it as a distributed constraint optimization problem and show how to solve it using the state-of-the-art Max-Sum algorithm that provides a completely decentralized message-passing solution. We then provide a novel algorithm (F-Max-Sum) that avoids sending redundant messages and efficiently adapts to changes in the environment. In empirical evaluations, our algorithm is shown to generate better solutions than other decentralized algorithms used for this problem.

Keywords: ALADDIN special issue; emergency responders; decentralized solution to the coalition formation process that pervades disaster management

Received 11 August 2009; revised 26 January 2010
Handling editor: Nick Jennings

1. INTRODUCTION

Major disasters are characterized by highly dynamic and uncertain parameters that can pose significant challenges to emergency responders. For example, during earthquakes or terrorist attacks, civilians may become trapped under rubble, fires may start and spread across a city and roads may become blocked as the disaster unfolds. Moreover, communication facilities might be significantly reduced by natural phenomena (e.g. fires taking out communication antennae or smoke interfering with radio signals) or become overloaded [1]. Against this backdrop, the actors (or agents) in the system are required to perform a number of tasks (e.g. rescuing civilians or extinguishing fires) in different parts of the affected area. Now, each task may require a given level of effort (e.g. digging civilians with specific drills, fires requiring large amounts of water) and may have to be performed by a certain deadline (otherwise, civilians die or the city is devastated). Moreover, the problem is compounded by the fact that new tasks may continuously appear or old tasks may disappear, requiring the agents to change their plans frequently.

In order to complete as many tasks as possible by their deadline, it is important that agents attempt the tasks in the right order at the right time. Since the tasks are spatially distributed, each agent must be provided with a route plan and a work schedule in order to visit the maximum number of tasks in the minimum time possible. They also need to do this while taking into account the temporal constraints applying over each task to ensure each task they decide to route to can be completed in time (i.e. by considering the amount of time it takes to complete the task and the deadline of the task). More importantly, agents need to form teams or *coalitions* of multiple responders from different agencies. This is because no single agency will have all the resources needed to save all the victims, unblock the roads and extinguish the fires. However, by working together in a coalition, the agents can achieve such tasks more efficiently as a result of their synergistic abilities. Hence, it is critical that the processes of coalition formation and management are effectively enacted.

In terms of underpinning technologies, it is important that such coalitions are chosen in a decentralized fashion (meaning each agent decides for itself what coalitions to form and join and that it interacts only with those agents that are nearby). By so doing, the system has no single point of failure and is more robust to long-range communication failures (or limited battery power) caused by the disaster. Moreover, the agents must be able to disband the existing coalitions and reform new ones as tasks are completed and new tasks appear in the system. Thus, the algorithms need to return solutions fast (i.e. within seconds or minutes) and be able to quickly adapt to changes in the environment (i.e. when tasks appear or disappear) as any time wasted can result in significant losses.

To date, the problem of forming task-achieving teams of agents in a geographical area has mainly been studied in the multi-robot routing domain (see Section 2 for more details). However, this work typically assumes that the agents have no synergies in coalitions (i.e. their abilities are simply additive) or that they each have distinct capabilities (i.e. they are perfectly substitutable). On the other hand, research in the coalition formation area has grown significantly in recent years [2–5]. However, to date, most approaches are typically centralized and usually focus on either the coalition value calculation problem (i.e. finding how effective each coalition is) or the coalition structure generation problem (i.e. selecting the best coalitions to be implemented) (see Section 2 for more details). In particular, most of this work ignores the fact that coalitions may need to be dynamically formed over time and that agents need to perform very complex tasks (i.e. spatially distributed and having a deadline).

Against this background, we model the coalition formation problem posed in disasters and provide a novel decentralized algorithm for it. Moreover, our algorithm is also able to efficiently adapt to new tasks arriving or existing tasks being removed from the environment. More specifically, we consider a sub-part of the disaster management problem defined by the RoboCup Rescue (RCR) disaster simulation platform [6]. Thus, we formulate the problem as a coalition formation with spatial and temporal constraints (CFST) problem and provide a distributed constraint optimization (DCOP) formulation for it in order to show how the optimization problem can be decomposed. Given this, we solve it using a new decentralized algorithm that is able recompute solutions efficiently when the allocation needs to be changed to accommodate different sets of tasks.

In more detail, this paper advances the state of the art in the following ways:

(i) We introduce CFST as a general model for the task allocation problem faced by ambulances and fire brigades in RCR and in disaster management at large. Thus, our model captures most task allocation problems that involve some form of temporal (i.e. deadline and time to complete a task) and spatial constraints (i.e. positions of agents and tasks) such as those existing in logistics planning or crew scheduling [7]. Given this, we define both optimal and approximate solutions for the problem.
(ii) We develop a new DCOP formulation for the approximate solution to the CFST problem and solve it using a novel decentralized algorithm based on the state-of-the-art Max-Sum algorithm [8].
(iii) We show that our algorithm can complete 10% more tasks than the current best decentralized algorithm for this problem (on average) and requires up to 91% fewer messages and 99% less computation than the standard Max-Sum algorithm in order to converge to a solution.

The rest of the paper is organized as follows. In Section 2, we describe related work in the area of multi-robot routing and coalition formation in general. Then, in Section 3, we describe the task allocation problem posed by RCR. Given this, Section 4 describes our representation for the CFST problem and provides both optimal and approximate solutions for it. Building on this, we provide a DCOP formulation of the problem and show how to apply the Max-Sum algorithm to solve CFST in a decentralized fashion in Section 5. In Section 6, we provide a novel algorithm that builds upon Max-Sum to adapt to disruptions more effectively and in Section 7 we empirically evaluate it. Finally, Section 8 concludes.

2. RELATED WORK

To date, the disaster management problems as simulated by the RCR platform (which we discuss in the next section) have only been attempted by the RoboCup competition entrants (since it started in 2001). Unfortunately, the problems posed by the simulator and solutions to these have rarely, if at all, been defined, formalized and solved. This is because the entries to the competition were mainly designed to win the competition rather than solve the general task allocation problems posed by RCR. Thus, these entries mainly involved ad-hoc solutions to (approximations of) the task allocation problems[1] in RCR that exploited bugs or were tailored to the scenarios generated by the platform[2] and hence not generalizable across most problems. However, a number of grounded approaches to the task allocation problems similar to those in RCR do exist.[3] We divide these into two groups: centralized and decentralized. In the next subsections, we discuss each of these approaches and also survey the general area of coalition formation.

2.1. Centralized algorithms

Here, we note the work of Zheng and Koenig [10] who consider a problem where only a specific number of agents can perform a certain task. This means that the problem is agnostic to the actual coalition serving a given task and that their algorithm considers a much smaller search space than ours since in our case more than one set of agents (of any number) can complete a given task. They also assume a central planner that allocates tasks to agents. Similarly, Gelenbe and Timotheou [9] use a centralized mechanism based on random neural networks to allocate responders to perform a number of rescue tasks.

2.2. Decentralized algorithms

Works by Scerri *et al.* [11] and Ferreira *et al.* [12] have applied DCOP and other decentralized heuristics to the general assignment problem. Now, while these approaches do consider heterogeneous agents (i.e. agents with different capabilities) and execution constraints for tasks (e.g. two tasks that must be executed at the same time), they ignore the benefit of forming coalitions of agents (i.e. with synergistic capabilities) to work on the same task.

In a different vein, Maheswaran *et al*. consider completely decentralized solutions to an allocation problem which considers teams of agents but ignores the spatial constraints of the CFST and show how different DCOP formulations of the problem result in different degrees of computational and communication efficiency when used with typical DCOP algorithms such as ADOPT [13] or DPOP [14]. In our work, we adopt a similar approach to theirs in building our DCOP formulation for the CFST, but solve it using the more scalable and general Max-Sum algorithm [8].

Approximate algorithms such as Max-Sum or the distributed stochastic algorithm (DSA) require very little local (re)computation and communication, and are, as such, well suited for large scale distributed applications in which the optimality of the solution can be sacrificed in favour of computational and communication efficiency [15, 16]. However, the quality of provided solutions heavily depends on the specific application domain and thus these approaches can often result in solutions of varying quality. This limits their applicability in many application domains (particularly safety critical ones) and hence it is important to evaluate such algorithms empirically (as we do in Section 7).

In this paper, we opt for the Max-Sum algorithm for two main reasons:

(i) Max-Sum does not incur the exponential coordination overhead typical of complete techniques and, moreover, it does not need the agents to organize themselves into a DFS tree as most other complete DCOP algorithms do (e.g. ADOPT, OptAPO [17] and DPOP). This means that agents do not need to hold the connectivity graph in memory and update it as connections are made or broken. Hence, the Max-Sum algorithm is more robust and appropriate for the context we study where there is high uncertainty in the environment in which the agents are operating.

(ii) Max-Sum, contrary to most other DCOP algorithms, naturally works with n-ary constraints. That is, the algorithm is able to find a solution when variables are constrained with more than one other variable. In the domain we study, the agents (acting as variables) are naturally constrained by more than one other agent (e.g. an agent can possibly reach multiple tasks but can only complete them if it forms coalitions with some other agents).

In general, the closest work to ours is that of Chapman *et al.* [18] (a longer version of which is included in this special issue) who develop a game theoretic framework and apply an approximate algorithm to solve the task allocation problem in the RCR scenario. Similar to our approach, they consider an approximation of the optimal solution to a simplified version of the CFST (where agents' abilities are simply additive). However, since their algorithm is an extension of DSA, they find only a local maximum (as opposed to the global optimum) of the function (which also approximates the optimal solution) they try to optimize. Moreover, Chapman *et al.* assume that the assignment is static; that is, tasks do not change within the current allocation. If they did change, the agents would have to recompute the whole allocation. Finally, their allocation mechanism ignores how different coalitions form based on when the agents arrive at a task (i.e. synergistic effect). In our model, we find the optimum value of the objective and can adapt,

[1]For example, the allocation of ambulances to victims in the simulation can be approximated by arbitrarily choosing one set of routes through all victims.

[2]For example, some buildings had pre-determined properties that teams could exploit to extinguish fires faster or victims' health decay could be accurately modelled. In the new version of the simulator, developed by the ALADDIN project, these bugs have been corrected and less information is now available for teams to design scenario specific algorithms.

[3]It is not our intention here to elaborate on all possible algorithms or techniques that have been developed for disaster management (e.g. see [1, 9]). Instead, we focus on those that are clearly relevant to the problem of allocating coalitions of agents to multiple tasks.

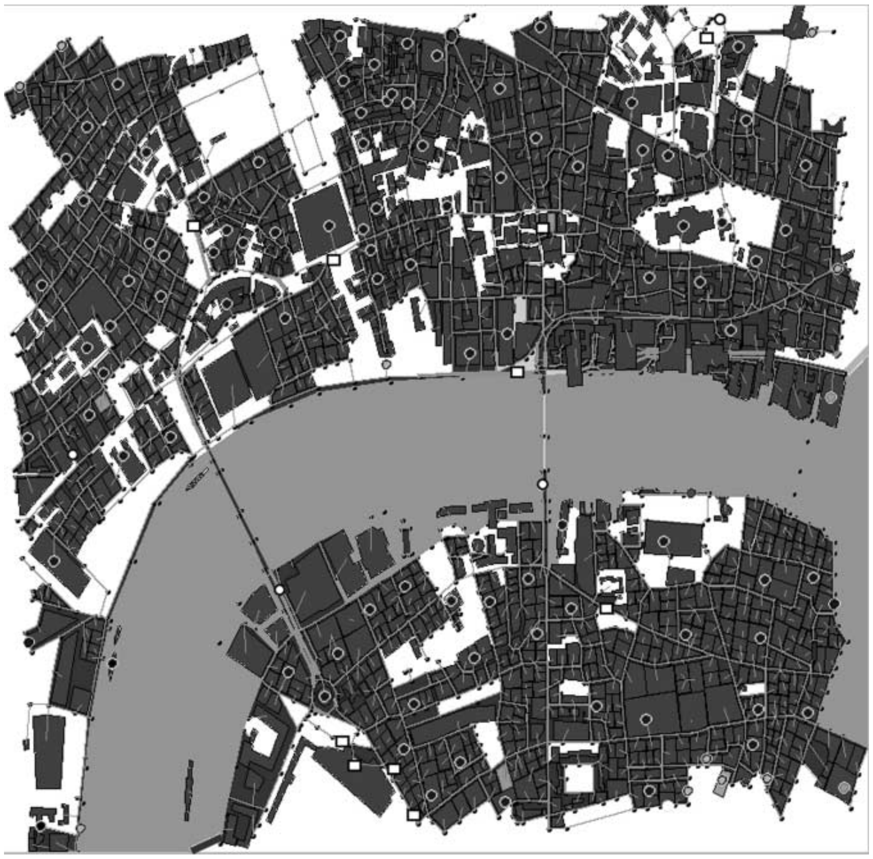


FIGURE 1. Part of London map in RCR.

with minimal recomputation, to re-assign coalitions to new tasks coming into the environment or old tasks being cancelled in such a way that more tasks are completed.

2.3. Coalition formation

Our work also targets the area of coalition formation in general. In particular, we note that most of the coalition formation techniques are typically centralized [2, 5]. An exception to this is [3, 4], for example, who provide algorithms to distribute the coalition value calculation problem. In general, however, these algorithms assume that *all* coalitions are actually feasible and ignore the fact that the task allocation problem dictates the set of coalitions that can be formed (e.g. some tasks might require coalitions of at least three agents to be completed by their deadline or some tasks might require one ambulance and two fire brigade agents to be completed). In our algorithm, we do take into account the domain and therefore distribute the computation accordingly. Finally, our contribution can also be seen as the first attempt to distribute, in a practical domain, the coalition structure generation problem which involves selecting the best coalitions to be enacted [2, 5]. To date, there exists no other algorithm which provides a decentralized solution to this problem.

3. TASK ALLOCATION IN RCR

The RCR simulation project was set up after the great Hanshin-Awaji earthquake in Japan, as a competition to stimulate research into multi-agent systems to aid in disaster rescue [6]. The simulation platform is currently maintained by researchers working within the ALADDIN project[4] and is continuously being extended and improved to create high fidelity simulations of fire spreading in a city, traffic congestion and perception and communication abilities of agents.[5]

In more detail, the project is based on the development of a disaster simulation platform that attempts to reproduce the conditions prevailing in real disasters so as to test coordination strategies that emergency responders might use in such situations. The platform is completely distributed (i.e. the simulators and agents can be run on multiple machines) and simulates the aftermath of an earthquake in a city. Fires

[4]http://www.aladdinproject.org.

[5]The ALADDIN project has also developed a building evacuation simulator (BES) [19, 20] that is complementary to the RCR. As opposed to the RCR which considers events in an open space requiring large-scale rescue operations, the BES considers the simulation of disasters in enclosed spaces such as buildings or ships. Hence, the basic elements of the task allocation problem we solve in this paper could also be simulated in the BES on a smaller scale.

erupt in different parts of the city, buildings have collapsed and civilians are trapped in them, and some roads are blocked. The simulation starts with a number of emergency response agents spread across the city and these agents must be designed to complete a number of tasks, including: (i) extinguishing the fires (by fire fighting agents), (ii) digging out the civilians (by ambulance agents) and (iii) unblocking the roads (by police agents). Figure 1 shows a 2D map of London used by the simulation platform with 70 ciivilians (black dot), 11 fire brigade agents (grey dot), 9 ambulance agents (white dot with black border) and 11 police agents (white square with black border). At the start of the simulation, the agents only have knowledge of the map and have to search the city to find fires, civilians and blocked roads.

In this work, we model, generalize and solve the task allocation problems faced by the ambulance and fire brigade agents which is to find and rescue the civilians buried in buildings and extinguish fires, respectively. In such allocation problems, agents have to first search for the victims and fires (which are unknown at the start of the simulation—only the map of the disaster space is known). As the positions of the tasks (i.e. victims or fires) become known, the agents need to decide and continuously update the sequence of the tasks they will attempt. This sequence needs to take into account both the spatial and temporal features that constrain the set of possible solutions as follows. First, given each task is located in a different part of the disaster space, the agents need to compute the most efficient tour of feasible tasks while minimizing the time they spend travelling (in case new tasks appear). Second, given the tasks' deadline and the level of effort required to complete them, the agents need to choose the best time to arrive and complete each task (i.e. by its deadline) in order to be able to reach other tasks before their deadline.

Now, it is also critical for agents *to coordinate to form coalitions* at each task since the demands of each rescue or extinguishing task cannot be met by one single agent. Hence, the agents need to coordinate their arrival time at each task in order to form the coalition that can complete the task (i.e. a coalition only exists if all agents in the coalition are present at the same time at the task). Moreover, since the number of victims or fires is likely to be much larger than the number of ambulances and fire fighters, it is important that the best agents (e.g. fire brigades with more water, ambulances with more capabilities) are allocated to the most demanding tasks to guarantee that as many tasks as possible are completed. Finally, since the environment is very dynamic and new tasks can appear in the system, it is important that the solution chosen by the agents can quickly be adapted to incorporate new tasks or changes to the existing tasks. However, performing this operation optimally is not trivial as it involves providing a route plan for each agent to ensure that the best coalitions are formed at the right place at the right time. Now, due to the nature of disasters, computational units in the system might be damaged and communication with them may severely be impaired. Power systems may be down, and responders may have limited battery power to transmit and receive data over long ranges. Hence, shorter range peer-to-peer approaches to coordination are more useful in such domains.

To this end, in the next section, we first provide the basic definitions required to model the problem as a coalition formation problem with spatial and temporal constraints. Then, in Section 5, we provide a fully decentralized solution for it that can quickly adapt to changes in the problem structure.

3.1. Basic definitions

Agents are noted as $a_1, \dots, a_n \in A$ that have to complete a number of tasks $v_1, \dots, v_m \in V$ that are located in different parts of a city (though more than one task may be located in the same place). The time taken for an agent to travel from one location to another is given by a function $\rho : (L \cup V) \times V \to [0, \infty]$ (assuming all agents can move at the same speed) where L is the set of all possible initial agent locations in the environment. Each task $v \in V$ has a *demand* consisting of two parameters as follows: *deadline*, $d_v \in [0, \infty]$ (e.g. representing time until which the victim will survive without being rescued or the time until which fire can be controlled), and *workload*, $w_v \in [0, \infty]$ (e.g. denoting the amount of work (in time units), that has to be done to extract the victim or extinguish the fire). We will denote $d_{\max}$ as the latest deadline, that is, $d_{\max} = \max_{v \in V} d_v$. Moreover, we assume that time is discrete such that agents travel or perform tasks in measurable time units (e.g. seconds, minutes or hours) starting at time equals zero.

3.2. Coalitions

Agents may form coalitions for several reasons. First, the workload for a given task may be too high for a single agent to perform by the deadline of that task. For example, a fire can be extinguished before it burns down the whole building if a number of fire brigades extinguish it on multiple sides rather than one fire brigade on one side. Secondly, even if all tasks have workloads that are manageable by a single agent (i.e. an agent can complete tasks by their deadline), the completion time of each task may be too late for the agents to have enough time to attempt other tasks (whose deadline may have passed). Following from the same fire example, the fire brigade might be able to extinguish one fire but by that time another major building might have been burnt down completely. Third, the distance to be travelled by each agent to any task may be too long to reach the task in time to complete it by its deadline.

In the remainder of this section, we elaborate on how agents form coalitions and the effect these coalitions have on task completion. We define what it means for an agent to 'work' on a task in later parts of this section. First, however, we denote the fact that an agent a works on a task v at a given time t by $\tau_t^{a \to v}$. We define $T = \{\tau_t^{a \to v}\}_{a \in A, v \in V, t \in \{0, \cdots, d_{\max}\}}$ as the set of all possible allocations of agents to tasks. When one or more agents work together on the same task, they work as a coalition, $C \in 2^A$; in a similar way, we denote by $\tau_t^{C\to v}$ the fact that a coalition C works on task v at time t. In effect, the coalition captures the synergistic effect of the agents' capabilities which helps them complete tasks faster than they would if they worked separately (at different points in time) on the same task. Now, given an agent allocation $T' \subseteq T$ and a time horizon $t' \in \{0, \ldots, d_{\max}\}$ within which we want to explore the coalitions that could exist,[6] we define the corresponding (feasible) allocation of coalitions, $\Gamma(T', t')$, over a given time period, as follows:

$$\Gamma(T', t') = \left\{\tau_t^{C\to v} \mid C = \{a \mid \tau_t^{a\to v} \in T'\}, v \in V, t \le t'\right\}. \tag{1}$$

The above definition basically means that a coalition C exists at task v at time t if all agents $a \in C$ work on task v at time t. This also means that only one coalition exists at a given task at any one time. Given this, we denote by $\Gamma = \{\Gamma(T, d_{\max})\}$ the set of all (maximal) coalition assignments generated by T.

Obviously, physically embodied agents cannot be allocated to all tasks at all times and, therefore, the solution to the allocation problem will involve agents working only on some tasks at some points in time. More precisely, we will say that an allocation of agents is feasible if it assigns an agent to two different tasks only in time points whose difference is greater than the travel time between the corresponding tasks. Given this, note that if $T' \subseteq T$ is a feasible agent allocation, then it generates a feasible coalition allocation, $\Gamma(T', t')$, over any time period $[0, t']$, $t' \le d_{\max}$. This, in particular, means that coalitions that exist at different locations at the same time do not overlap.

The work that a coalition performs at a task in each time unit (or, step) decreases the workload of that task.[7] The extent to which the workload decreases is dependent on the *value* of the coalition, given by the function $u : 2^A \to \mathbb{N}^+$. The function $u(\cdot)$ basically expresses how well the agents involved in the coalition work together and how their capabilities match. For example, if agents a_1 and a_2 have a coalition value of $u(\{a_1\}) = 1$ and $u(\{a_2\}) = 1$, then, if they work together they may generate a value $u(\{a_1, a_2\}) = 3$, if their capabilities are synergistic. We will assume for now that coalition values are independent of the task the agents work on and that coalitions of more agents are usually better or equal to coalitions of smaller numbers of agents, that is $u(C \cup \{a\}) \ge u(C)$.[8] Moreover, we will assume tasks are, in turn, homogeneous. Since tasks are considered to be atomic, only one coalition can perform one task at a time.

Given the above definitions, in the next section, we define the problem that this setup generates and the associated constraints.

[6]This will become useful when we discuss the algorithms to generate a solution.

[7]Note that we here assume that the work done does not affect the deadline of the task in any way. For example, the depth at which a victim is trapped does not determine how long he/she will live. In future work, we will consider removing such an assumption.

[8]This is typically true in the settings where larger teams of responders can extinguish fires and rescue civilians faster.

4. COALITION FORMATION WITH SPATIAL AND TEMPORAL CONSTRAINTS

The goal of the CFST is to maximize the number of tasks completed given all possible allocations of agents to tasks. The allocation of agents needs to take into account a number of constraints. These can be grouped broadly into two classes: spatial and temporal. The former restrict the movement of agents around the tasks given the time available to them, while the latter take care of the restrictions with respect to the time taken by agents to finish a task. In what follows, we first detail the constraints and then move onto the objective function(s) we try to maximize. We will assume that the solution should contain some allocation of agents to victims as the set $T' \subseteq T$.

4.1. Spatial constraints

The fact that tasks are spatially distributed implies that there is a cost to switching from one task to another. This cost is captured by the time spent by coalitions in travelling from task to task (captured by the function ρ), or the delay for a coalition to be formed when several agents need to meet to work on a task (i.e. some agents have to wait for other agents). These spatial constraints therefore apply over the existence of coalitions. If agent a is routing to location v from location $l \in L$ (which is either its initial location or another task) at a given time t, the starting time $s_a^v \in [0, \infty]$ at which agent a starts working on the task v must satisfy the following:

$$s_a^v \ge t + \rho(l, v). \tag{2}$$

Note that given the condition in (2), coupled with the fact that tasks cannot be attempted after their deadline (we elaborate on this in the next subsection), and assuming that travel times are proportional to distances among the locations (and hence satisfy the triangle inequality), we can restrict all possible assignments to the following:

$$T = \left\{\{\tau_t^{a\to v}\}_{t\in\{\rho(l_a,v),\cdots,d_v\}}\right\}_{a\in A, v\in V}\},$$

where $l_a \in L$ is the initial location of agent a.

Similarly, at a given time t', if we knew the solution up to this point, we could replace the initial location of agent a with its current location, $l_{t'}^a$. Thus, at each time t', given a specific victim v and a subset of agents $A' \subseteq A$, we can specialize the set of allocations to:

$$T(A', v, t') = \left\{\{\tau_t^{a\to v}\}_{t\in\{\rho(l_{t'}^a,v),\cdots,d_v\}}\right\}_{a\in A'}\}. \tag{3}$$

Now, depending on where the agent is routing from, its starting time at a particular location is restricted in two ways. First, if agent a arrives at v from its initial location $l_a \in L$, then:

$$s_a^v \ge \rho(l_a, v). \tag{4}$$

Second, if within the assignment T', agent a moves to v from another task v', then:

$$s_a^v \ge s_a^{v'} + \mid \cup_{t\in\{\rho(l_a,v'),\cdots,d_{v'}\}} \tau_t^{a\to v'} \mid + \rho(v', v). \tag{5}$$

Similar to (4), the above condition requires that an agent will not start working on a task before reaching it. Here, the second term in the right-hand side represents the amount of time that agent a spends in total on task v'—the sum of this and the starting time of a on v' gives the earliest time agent a can leave task v'; by adding to this the travel time between v' and v we get the earliest time by which task v can be reached by agent a.

4.2. Temporal constraints

Having defined the constraints that determine where an agent can route to at what time, we now define constraints that determine what the assignments that will result in tasks being completed. Thus, we define a binary-valued function $W : V \times \Gamma \rightarrow \{0, 1\}$ as follows:

$$W(v, \Gamma) = \begin{cases} 1, & w_v - \sum_{\tau_t^{C \rightarrow v} \in \Gamma} u(C) \leq 0 \\ 0, & \text{otherwise.} \end{cases} \quad (6)$$

Thus, $W(\cdot)$ expresses the fact that a task can only be completed if all the work done on the task by all coalitions equals or is greater than the workload of that task. However, the coalitions can only be effective up to the deadline of the task, after which the task is deemed to have failed. To express the success or failure of a task, we define the function $\Delta : V \times \Gamma \rightarrow \{0, 1\}$ as follows:

$$\Delta(v, \Gamma) = \begin{cases} 1, & \max_{\tau_t^{C \rightarrow v} \in \Gamma} t \leq d_v \wedge W(v, \Gamma) = 1 \\ 0, & \text{otherwise.} \end{cases} \quad (7)$$

Thus, $\Delta(\cdot)$ returns 1 only if the given task can be completed as per the schedule of agent assignments specified.

Note that, given the above temporal and spatial constraints, the routing of agents to tasks in T may not actually be feasible (e.g. an agent being assigned to tasks it cannot reach before their deadline or joining a coalition to complete a task too early or too late) and one of the challenges is to find those routes that are consistent. We note that finding such an assignment is very similar to solving a vehicle routing problem [21]. In particular, to find an assignment that is consistent with the constraints defined in Section 4.2 is, in turn, equivalent to finding a feasible schedule for the tasks [22]. Thus, the problem is a complex combination of both routing and scheduling that generates a search space that grows exponentially in the number of tasks and agents. We next define the objective of the CFST problem and explain how we may approximate this objective in order to avoid searching through the whole search space.

4.3. Objective functions

Given the size of the problem that the CFST generates, there could be a number of ways in which the objective is specified to try to approximate the exact objective function. In what follows, we first express the exact objective function and then go on to define a myopic approximation of it. Given this, we show how our myopic approximation is sound (i.e. does not return invalid solutions).

4.3.1. Optimal solution

The main objective of the CFST problem remains to maximize the number of tasks completed. This can be expressed as follows:

$$\arg\max_{\Gamma \in \Gamma} \sum_{v \in V} \Delta(v, \Gamma), \quad (8)$$

subject to constraints in Equations (4) and (5).

As can be seen, the above objective is simply to maximize the number of tasks completed with respect to the given constraints. However, the space over which the function iterates is very large (in the worst case, we might need to consider nearly $|V|!d_{\max}^{|V|}$ possible plans for each agent and the number of coalitions that need to be considered at each task is at worst $2^{|A|}$). In the worst case, the optimal solution to the problem is simply not computable in reasonable time.[9] Given that the time taken to generate solutions in the disaster management domain is critical in saving the maximum number of lives, it is important to devise algorithms that solve the problem quickly even if they are not optimal. Moreover, since the set of tasks is likely to change over time, it may not be worth considering solutions that consider allocations far ahead in time (i.e. long sequences of tasks to be completed). Thus, in the next section, we define a myopic objective function that tries to approximate the above objective in the next subsection and later evaluate its effectiveness in maximizing the number of tasks completed in Section 7.

4.3.2. Myopic solution

The general approach in this section is to allocate agents in a myopic way in that the agents do not consider sequences of tasks and only consider one task at a time. This is particularly appropriate to dynamic situations (such as in RCR) where solutions that consider allocations far ahead in time can be quickly invalidated if new (more critical) tasks arrive in the system. Moreover, the agents should try to maximize the number of tasks completed in the minimum time available (i.e. to avoid wasting time) in order to maximize the probability of completing future tasks. Then, after any task is completed, the agents that were working on that task are considered free and are re-allocated on any remaining task.

To this end, at time t, we restrict the set of tasks to those that are not yet allocated to any coalition of agents to $V^t \subseteq V$, and for each $v \in V^t$ we define the set of agents $A_v^t \subseteq A$ that are able to arrive at v before its deadline, that is $A_v^t = \{a | \rho(l_t^a, v) \leq d_v\}$, where $l_t^a \in L$ represents a's current location. Given A_v^t, for each task, we use equation (3) to compute the set $T(A_v^t, v)$ of possible allocations of the agents to the task, and hence the set

[9]We have implemented and solved the problem using mixed-integer programming and found that problems involving only 4 agents and 10 tasks can take hours to solve.

of possible coalition allocations $\Gamma(T(A_v^t, v), d_v)$. Let us also denote $A^t = \cup_{v \in V^t} A_v^t$.

In order to find the allocations of coalitions that will complete the tasks as fast as possible, let $t_{\min}^v(\cdot)$ be a general function that returns the earliest time at which a task can be completed given the set $T' \subseteq T$ of coalitions allocated to it and is computed as follows:

$$t_{\min}^v(t, \Gamma(T', t)) = \begin{cases} \min_{t' \in S^v(t, \Gamma(T', t))}(t), & S^v(t, \Gamma(T', t)) \neq \emptyset \\ X, & \text{otherwise} \end{cases}$$

where

$$S^v(t, \Gamma(T', t)) = \{t' \in \{t, \cdots, d_v\} \mid \Delta(v, \Gamma(T(A_v^t, v), t')) = 1\}$$

and $X \gg d_{\max}$. That is, X is used to express the fact that the task cannot be completed by its deadline given the set of allocated coalitions.

Having defined the necessary constructs to find a timely allocation of coalitions given the possible coalitions, at time t we restrict the general set of allocations Γ to consider assignments given by V^t and A_v^t, $v \in V^t$, as $\Gamma^t = \bigcup_{v \in V^t} \Gamma(T(A_v^t, v), d_v)$. Then every subset of Γ^t represents possible allocations of coalitions from time t onwards. Given this, the goal is to find a set of such allocations that maximizes the following objective function:

$$\max_{\Gamma^t \subseteq \Gamma^t} \sum_{v \in V} \left(X - t_{\min}^v(t, \Gamma^t)\right) \qquad (9)$$

and satisfies the following: $\nexists \tau_{t'}^{C \to v}, \tau_{t''}^{C' \to v'}$ such that $C \cap C' \neq \emptyset$ and $v \neq v'$, and the condition in Equation (4) holds. This basically means that the set of selected coalitions should not overlap (i.e. an agent is only assigned to one task in this allocation). Having allocated as above, at next time step, the set of tasks is reduced to $V^{t+1} = \{v \mid v \in V^t, \nexists \tau^{C \to v} \in \Gamma^t\}$, that is to the set of tasks that have not been allocated up to then, and the set of agents A_v^{t+1}, $v \in V^{t+1}$ is computed appropriately, choosing from the currently unassigned agents (i.e. it is reset to the empty set if all agents are allocated or is refilled with the agents that have been freed).[10]

As can be seen from the above objective function, we implicitly consider the routing of agents to form coalitions at different tasks by only considering one task at a time and determining the agents that can reach it. In so doing we reduce the computation of solutions significantly (to $O(k)$ plans to be considered for each agent where k is a constant) as we do not have to search for the optimal tour of all tasks in the system for all agents. However, we can still show that the myopic solution maintains soundness (but not completeness) as follows.

First, we can easily deduce that in time step t, an agent is allocated to only one task since coalitions do not overlap and the starting times of the agents are consistent as per Equation (4). Second, since the set of agents A^t excludes those that have been allocated at $t' < t$, it is not possible that an agent is previously allocated to a given task. This means that there will be no solutions that have agents being allocated to two tasks at the same time. Now, we can also deduce from the objective function in Equation (9) that it will return 0 in case no tasks can be completed (since $t_{\min}^v(\cdot)$ returns X when no coalitions can complete any task) and only be greater than 0 in case at least one task can be completed. However, this does not prevent the coalitions from being assigned to the tasks even though they cannot be completed. We do not exclude such a possibility since more agents may become free (after completing some tasks previously allocated) and work with these coalitions to complete the tasks.

Thus, so far, we have only considered centralized solutions to the CFST. However, in the disaster management problem we believe it is important that computation is distributed as argued in Section 3. Given this, we next proceed to define our decentralized solution.

[10]Given this, it only makes sense to recalculate when we have freed agents.

5. A DECENTRALIZED SOLUTION

Here, we describe a DCOP formulation for the CFST problem that allows agents to divide up the objective function defined in (9), in such a way that different sub-functions can be computed separately by individual agents and a coordinated solution can be found by message passing between them.

5.1. The DCOP formulation

Formally a DCOP can be defined as a tuple $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{F} \rangle$, where $\mathcal{A} = \{a_1, \ldots, a_k\}$ is a set of agents, $\mathcal{X} = \{x_1, \ldots, x_n\}$ is a set of variables, each variable x_i is owned by exactly one agent a_i, but an agent can potentially own more than one variable. The agent a_i is responsible for assigning values to the variables it owns. $\mathcal{D} = \{D_1, \cdots, D_n\}$ is a set of discrete and finite variable domains, each variable x_i can take the value in the domain D_i. Then, $\mathcal{F} = \{f_1, \ldots, f_m\}$ is a set of functions that describe the constraints among variables. Each function $f_i : D_{i_1} \times \cdots \times D_{i_{r_i}} \to \Re$ depends on a set of variables $\mathbf{x_i} \subseteq \mathcal{X}$, where $r_i = |\mathbf{x_i}|$ is the arity of the function. Each function assigns a real value to each possible assignment of the variables it depends on.

There are many ways in which we can formalize the CFST problem as a DCOP, depending on what we choose to represent with variables and how we define the constraints among them. These choices have impact on the communication overhead, the computational load and the computation distribution among agents. Here we decide to assign to each agent a variable that represents the agent's current target, that is, the task that the agent will attempt. This formalization has several benefits. First, it removes the complexity of modelling coalitions, as they are dealt with inside the utility function. Second, it minimizes the loops in the constraint network, simplifying computation and avoiding redundant information propagation. A formulation similar to this one can be seen in the work of [23]; in which a number of stationary sensors are required to coordinate to track moving targets.[11]

In our DCOP formalization, variable domains consist of the task locations that are reachable fast enough for the agent to arrive and make a useful contribution (e.g. the set of victim locations that the agent can reach before the victims' deadline). If x_i represents the variable for which agent a_i is responsible, given t is the current time, the domain of this variable is then $D_i = \{v_k \in V^t | t + \rho(x_i, v_k) \le d_{v_k}\}$, where x_i is the current task the agent is assigned to or the agent itself (i.e., its initial location) and d_{v_k} represents the deadline for task v_k (as specified above).

The constraint functions in this formulation represent the 'utility' of each task, taking into account all variables whose domains contain the location of the task in question. More formally, assuming $x_i \in \mathcal{X}$ and $v_j \in V^t$, x_i is a variable to f_j (the utility function of v_j) if and only if $v_j \in D_i$. As such, in this DCOP formalization, there are N variables, one for each agent, and M constraint functions, one for each task.

Now, to embed the myopic scheduling strategy of the agents (presented in Section 4.3) into the objective function, we compute the utility function for each task as follows:

$$f_i(\mathbf{x_i}) = X - t^{v_i}_{\min}(\Gamma(T', d_{\max})), \qquad (10)$$

where $T' = T(A_{v_i}(\mathbf{x_i}), v_i)$ and $A_{v_i}(\mathbf{x_i}) = \{x_k | x_k \in \mathbf{x_i} \wedge x_k = v_i\}$ such that the constraint in Equation (4) is met. Note that $\sum_{v_i \in V^t} f_i(\mathbf{x_i})$ is equivalent to our myopic solution specified in Equation (9). The fact that agents have a variable which assigns them to a task ensures that they can only be part of one coalition in the allocation.

Finally, notice that solutions provided by the DCOP formalization presented here can be tied directly back to our initial representation presented in Section 3. For example, consider a situation where an agent a_1 assigns itself to task v_1 and works on the task at time steps 3–5 and agents a_2 and a_3 assign themselves to task v_2, working on this task for time steps 3 and 4. This outcome can be represented by $T_{v_1} = \{\tau_3^{a_1 \to v_1}, \tau_4^{a_1 \to v_1}, \tau_5^{a_1 \to v_1}\}$ and $T_{v_2} = \{\tau_3^{a_2 \to v_2}, \tau_4^{a_2 \to v_2}, \tau_3^{a_3 \to v_2}, \tau_4^{a_3 \to v_2}\}$. Then, the allocations of coalitions can be deduced from $\Gamma(T_{v_1} \cup T_{v_2}, d_{\max})$.

Having defined the formulation of the decentralized solution to the CFST, we next describe the Max-Sum algorithm that solves it.

[11]Notice that following this formulation, it is not obvious which agent is responsible for the computation of the utility functions, as utilities are defined for tasks that can be performed by multiple agents. While the assignment has an impact on the agent's computational load, it does not impact the performance of the algorithm in terms of solution quality and communication overhead. Here we focus on providing an efficient decentralized solution to the coalition formation problem and we do not address the problem of balancing the computational load among the agents. As such we can use any assignment of utility functions to agents that avoids redundant computation. A simple policy is to assign the computation for shared utility functions to the agent that has the lowest ID.

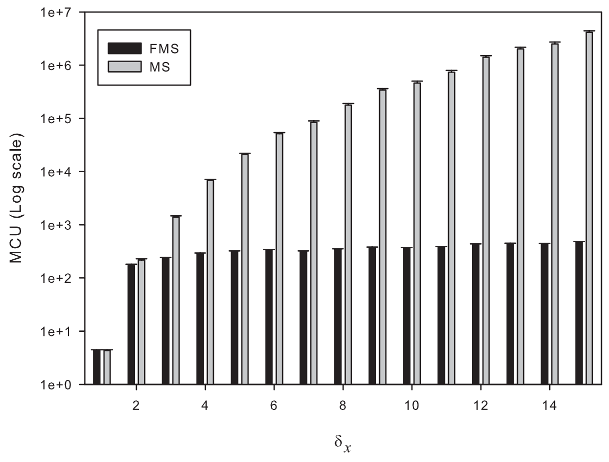


FIGURE 2. The example from Fig. 3 formulated as a factor graph, with agents as variables (circle), and victim locations as factors (squares). The lines now connect the factors to the variables over which they are defined.

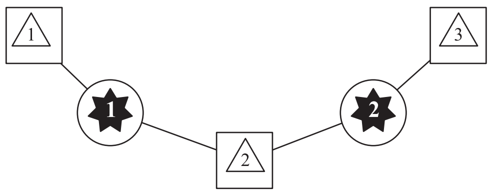


FIGURE 3. An example scenario, containing two rescue agents (black stars) and three victims (white triangles). The lines connect agents to victims they can reach before their deadline.

5.2. Applying the Max-Sum algorithm

To apply Max-Sum to the DCOP formalization presented above, we adopt a Factor graph representation of the problem [24]. A factor graph is a bipartite graph where vertices are variables, which represent the variable of the DCOP formalization, or functions, which represent the DCOP constraints. The edges of the factor graph connect functions to variables over which they apply. The situation in Figure 2 is represented as a factor graph in Figure 3. The global utility function in this example would be

$$F(x_1, x_2) = f_1(x_1) + f_2(x_1, x_2) + f_3(x_2). \qquad (11)$$

It is important to note, that when the factor graph is cycle free, the algorithm is guaranteed to converge to the global optimal solution such that it finds the combination of states that maximizes the sum of the functions. When applied to cyclic graphs (e.g. in the case where more than one agent can attempt exactly the same tasks), there is no guarantee of convergence but extensive empirical evidence demonstrates that this family of algorithms generate good approximate solutions [24, 25]. In particular, the Max-Sum algorithm has been shown to outperform previous approximate algorithms on standard DCOP benchmarks [8].[12]

In order to apply Max-Sum, there are two types of messages that need to be specified. Given our factor graph formulation of the problem, we need to specify messages that need to be sent from variable to function and vice-versa as follows:

- From variable to function:

$$q_{i\to j}(x_i) \text{ for all values of } x_i$$

where

$$q_{i\to j}(x_i) = \alpha_{ij} + \sum_{k\in\mathcal{M}_i\setminus j} r_{k\to i}(x_i), \tag{12}$$

where $\mathcal{M}_i$ is a vector of function indices, indicating which function nodes are connected to variable node i, and α_{ij} is a scalar chosen such that $\sum_{x_i} q_{i\to j}(x_i) = 0$, in order to normalize the message and hence prevent them increasing endlessly in the cyclic graphs that we face here.

- From function to variable:

$$r_{j\to i}(x_i) \text{ for all values of } x_i$$

where

$$r_{j\to i}(x_i) = \max_{\mathbf{x}_j\setminus i}\left[f_j(\mathbf{x}_j) + \sum_{k\in\mathcal{N}_j\setminus i} q_{k\to j}(x_k)\right], \tag{13}$$

where $\mathcal{N}_j$ is a vector of variable indexes, indicating which variable nodes are connected to function node j and $\mathbf{x}_j\setminus i \equiv \{x_k : k \in \mathcal{N}_j \setminus i\}$.

The messages flowing into and out of the variable nodes within the factor graph are sets of values that represent the total utility of the network for each of the possible states of the variable. At any time during the propagation of these messages, an agent is able to determine which task it should undertake such that the sum over all the task utilities is maximized. This is done by locally calculating the function, $z_i(a_i)$, from the messages flowing into agent i's variable node:

$$z_i(x_i) = \sum_{j\in\mathcal{N}_i} r_{j\to i}(x_i) \tag{14}$$

and hence finding $\arg\max_{x_i} z_i(x_i)$.

Notice that, although the Max-Sum algorithm is approximating the solution to a global optimization problem, it involves only local communication and computation. Also note that Max-Sum typically runs continuously. This means that if any change is made to the factor graph (e.g. some tasks are found to have a different workload or deadline or new tasks appear) such that the utility computed by the factors changes, Max-Sum will have each factor recompute its messages and generate a new solution.

[12]The results in Section 7.1 further confirm this.

Now, Max-Sum was not specifically designed to deal with the dynamic changes in the factor graph that typically arise in the RCR scenario. As a result, there are inefficiencies in the re-computation of the solution (specifically the maximizations in Equation (13) and over Equation (14)). First, when new messages are generated from changes in the system, but the contents of these messages do not change the solution, the factors still re-compute solutions and forward these messages to variables when they need not. It is important to avoid such re-computations because the space to be searched by each factor can be quite large. In the general case, this is *exponential* in the number of states of the variables to a given factor. Second, Max-Sum does not make any assumptions about the states of the variables and hence iterates over a large state space whereas, in our case, there are several properties of the tasks and the agents that could be exploited to reduce the state space.

Given this, in the next section we detail how we extend Max-Sum to avoid factors computing over all possible variable states. Moreover, we significantly modify the algorithm to allow variables to detect disruptions in the factor graph and, wherever possible, prevent the connected factors from recomputing their messages. In so doing, we devise an algorithm that is more tailored to the domain we consider and is robust to changes in the environment (i.e. can re-compute solutions quickly when new tasks appear or existing tasks are completed).

6. THE FAST MAX-SUM ALGORITHM

The F-Max-Sum algorithm extends the standard Max-Sum in two main ways. First, in order to reduce the number of states over which each factor has to compute its solution, we introduce new functions on variable and factor nodes that single out the states that matter to them. Second, we introduce new functions that allow each variable to decide when to send messages to its other connected factors when changes happen in the factor graph (i.e. a factor is removed or added). We detail these two extensions in the following sub-sections and show how they allow us to make significant computational and communication savings in Section 7.

6.1. Reducing communication and computation

In order to reduce the number of states each factor needs to compute its solution over, we restrict the domain of each variable to only two states per each connected factor representing the fact that an agent is assigned to a specific task (the factor) or not. With this change, we can now specialize the message computation performed by the original Max-Sum which applies over all states of the variables involved (see Equations (12) and (13)). Hence, in what follows, we introduce new functions to manage these new messages that inform the factors of these states and show why F-Max-Sum retains the same properties as Max-Sum with this reduction in state space:

- From variable to function:

$$q_{i\to j}(x_i = v_j) = q \quad \text{and} \quad q_{i\to j}(x_i = v_{-j}) = q'$$

where

$$q = \alpha_{ij} + \sum_{k\in\mathcal{M}_i\setminus j} r_{k\to i}(v_{-k}) \quad \text{and}$$

$$q' = \alpha_{ij} + r_{b\to i}(v_b) + \sum_{k\in\mathcal{M}_i\setminus b,j} r_{k\to i}(v_{-k}). \qquad (15)$$

where $b = \arg\max_{x_i\neq v_j} z_i(x_i)$ represents the best task to which x_i can be allocated apart from v_j. The above message differs from the usual Max-Sum message (see Equation (12)) in that the variable sends only the utility values for two states; where the agent a_i assigns itself to v_j and where it does not. This is possible here because the utility gained by function f_j by assigning a_i to v_k for $k \neq j$ is the same for all k since the agent does not help in saving v_j in this case. Hence, what matters to f_j is only the utility that the rest of the system gets for *not* assigning the agent to v_j.

- From function to variable:

$$r_{j\to i}(x_i) \text{ for } x_i = v_j \text{ and any } x_i = v_{-j}$$

where

$$r_{j\to i}(x_i) = \max_{\mathbf{x}_j\setminus i}\left[f_j(\mathbf{x}_j) + \sum_{k\in\mathcal{N}_j\setminus i} q_{k\to j}(x_k)\right] \quad \text{and}$$

$$-j \in \mathcal{M}_i \setminus j. \qquad (16)$$

The above differs from the usual Max-Sum message (see Equation (13)) in that the factor does not need to compute the utility the system gets for all values of x_i. Instead, it only computes for a_i being assigned to v_j or not. When a_i is not assigned to v_j, $f_j(\mathbf{x}_j)$ is independent of the specific task that the agent is assigned to and hence we can generalize this allocation to be any task other than v_j. Note that this operation is different from the one in Max-Sum which would have searched assignments of the variable which do not improve the utility of the factor in any way. Thus, we effectively prune the space that would have originally been searched by Max-Sum without losing any information

Since now a variable node receives only two values per factor it is connected to, the variable node needs to sum these messages in a different way from Max-Sum. Basically, if a variable node x_i has received $r_{j\to i}(x_i)$ for $x_i = v_j$ and $x_i = v_{-j}$, the variable simply adds $r_{j\to i}(v_j)$ to all other messages $r_{k\to i}(v_{-k})$ where v_{-k} is defined as above. Hence, the computation of the function $z(\cdot)$ is now (compared with Equation (14)):

$$z_i(x_i) = \left(r_{j\to i}(v_j) + \sum_{k\in\mathcal{N}_i\setminus j} r_{k\to i}(v_{-k})\right)$$

given $x_i = v_j$.

In so doing, we get the total utility for all states of the variable. Then, the variable can choose which value it takes as $\arg\max_{x_i} z_i(x_i)$ as before.

Note that our approach to extend Max-Sum can be adapted to other types of problems where domain-specific properties can be exploited to reduce the state space. In more detail, the extensions we present in F-Max-Sum can be generally applied to any domain where functions show a similar dependency structure from the variable. That is, the function has a significant change only for particular values of the domain (e.g. when the variable is allocated to the task that the function represents in our case). This clearly has a significant impact on the computation that factors go through when performing the maximization step, as we reduce the domain size of the variable. In particular, for our domain, a factor that depends on n variables that have a domain composed of d values each, will need to perform d^n computations, while with our extension this reduces to 2^n.

Also note that since F-Max-Sum does not specifically try to reason about cycles in the factor graph, F-Max-Sum cannot be guaranteed to converge on graphs with cycles similar to Max-Sum. However, both F-Max-Sum and Max-Sum are guaranteed to converge on acyclic graphs.[13] We next detail how F-Max-Sum adapts to disruptions.

6.2. Managing disruptions

We define a *disruption* in the graph as the addition or removal of a task from V which results in the addition or removal of a factor from the factor graph. This may happen for a number of reasons. First, a task v_i is removed from V, for example, when a victim has been rescued or has perished. Second, a task v_i is added to V when new information is received (e.g. a new building on fire or a new victim is located). Third, if the wrong information is received about a task v_i having a certain demand when it does not (e.g. a false alert about a trapped victim), then it will be removed from V.

Now, in order to define how the algorithm chooses to send a message, we first assume that at time t, the allocation computed by each variable is $\arg\max_{x_i} z^t(x_i)$. Then, at time t' two types of disruption may happen:

(i) A new task v_k appears $t' > t$—the set of tasks that needs to be assigned is augmented to $V^{t'} = V^t \cup \{v_k\}$. This also means that the set of indices of factor nodes is increased to $\mathcal{M}^{t'} = \mathcal{M}^t \cup \{j\}$.

(ii) An existing task v_k disappears at time $t' > t$—the set of tasks that needs to be assigned is decreased to $V^{t'} = V^t \setminus v_k$. This also means that the set of factor nodes is decreased to $\mathcal{M}^{t'} = \mathcal{M}^t \setminus \{j\}$.

Given the above, the domains of the variables will also change accordingly. The result is that the variables x_i with $v_k \in D_i$ will

[13]A simple verification of the messages sent for F-Max-Sum will confirm this.

need to make different decisions based on whether the factor is removed or has just been added. Essentially this means that:

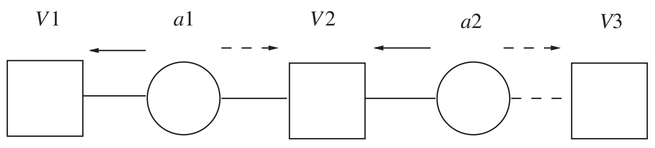
(i) if f_j is added, then $z^{t'}(x_i) = z^t(x_i) + r_{j\to i}(x_i)$.
(ii) if f_j is removed, then $z^{t'}(x_i) = z^t(x_i) - r_{j\to i}(x_i)$.

Then, the decision that the agents make is as follows. Assuming for each f_k for $v_k \in D_i$ we define the utility for $x_i = v_k$ and $x_i = v_{-k}$ as $\{z^t(x_i = v_k), z^t(x_i = v_{-k})\}$, then:

(i) If both $z^t(x_i = v_k) = z^{t'}(x_i = v_k)$ and $z^t(x_i = v_{-k}) = z^{t'}(x_i = v_{-k})$, then x_i does not need to transmit a newly computed $q_{i\to k}(x_i)$ based on the new domain of x_i.
(ii) Otherwise, x_i has to send the message. This is because, if the utility that a_i achieves by being allocated (or not) to v_j changes, then the system's utility might change as well and hence messages need to be sent around in any case.

For example, consider Fig. 4 that shows a situation with two agents a_1, a_2 and two victims v_1, v_2 at time t. Suppose at time $t' > t$ a third victim v_3 is discovered and thus the factor graph changes as shown in the figure. The tables represent the utility functions for each victim. The assignment of an agent to the victim is noted in binary format to represent whether it is allocated (1) or not (0). The value in the left-most column represents the utility (earliest time) obtained for the assignment selected (e.g. $\{a_1 = 0, a_2 = 1\}$ results in a utility of 4 at $V2$).

The best allocation at time t is computed as $\{a_1 = v_1, a_2 = v_2\}$ which gives a value of $4+2 = 6$, while when the new victim is discovered then the best allocation is $\{a_1 = v_2, a_2 = v_3\}$ which gives a value of $4+3 = 7$. The change in the factor graph will result in a different $z_2^{t'}(a_2)$, which will trigger information propagation leading F-Max-Sum to change the allocation and obtain the best value. In particular, notice that when v_3 is discovered we have $z_2^t(a_2 = v_2) = 4$ and $z_2^t(a_2 = v_{-2}) = 0$



$V2$	$a1$	$a2$
0	0	0
4	0	1
4	1	0
5	1	1

$V1$	$a1$
0	0
2	1

V	$a2$
0	0
3	1

FIGURE 4. Example of a new task v_3 being discovered in the system. The resulting factor (V3) is added (shown by the dotted line) to an existing factor graph . Then a_2 needs to decide whether it sticks to v_2 or moves to v_3. The tables show the utility (in the left-most column) for the assignment of each agent to each victim. At time t only the tables for v_1 and v_2 exist and at time t' the table for v_3 is introduced.

while $z_2^{t'}(a_2 = v_2) = 4$ and $z_2^{t'}(a_2 = v_{-2}) = 3$, which results in $z_2^t(a_2 = v_2) = z_2^{t'}(a_2 = v_2)$ but $z_2^t(a_2 = v_{-2}) \neq z_2^{t'}(a_2 = v_{-2})$ and message propagation is necessary to deal with the change in the factor graph and reach a better allocation.

In the case where the utility does not change (i.e. $z_2^t(a_2 = v_{-2}) = z_2^{t'}(a_2 = v_{-2})$) with respect to v_k, then F-Max-Sum prevents any message from being sent (see the rules above) as the rest of the system (starting from f_k) will not be affected (since a_2 cannot do any better by changing its assignment). Instead, in Max-Sum, any change in utility in any state of the variable initiates a new message (see Equation (12)) and hence results in it recomputing Equation (13) (which maximizes over all states of all its connected variables) only to find out that the allocation does not change. Comparatively, F-Max-Sum only needs to check over v_k's states to decide whether to send a message or not, thus minimizing the computation needed to implement any changes in utility. Thus, by distributing some of the computation on the variable nodes (in addition to factor nodes) and filtering out messages, F-Max-Sum can avoid both the redundant computation and messages of Max-Sum.

Also note that F-Max-Sum and Max-Sum have essentially the same behaviour when it comes to computing the solution since they both take into account the same information (i.e. changes in utility that will affect the assignment of agents to tasks). F-Max-Sum is simply more efficient at doing so.

Having described Max-Sum and F-Max-Sum, in the next section we empirically evaluate F-Max-Sum in the RCR domain and compare it with Max-Sum with respect to disruptions.

7. EMPIRICAL EVALUATION

In this section, we evaluate and benchmark F-Max-Sum to show how effective it is at finding good solutions and how it scales with the number of tasks. Moreover, we examine how effective F-Max-Sum is in adapting to changes in the task set. Our experiments are therefore divided into two parts. First, we compare the effectiveness of F-Max-Sum in maximizing the number of tasks completed compared with a number of other strategies. This aims to validate our use of the myopic solution as a good approximation to the optimal solution. Second, since Max-Sum and F-Max-Sum compute the same solutions, we apply both to the same data set and evaluate their performance in responding to changes in the task set.[14] In so doing, we show how F-Max-Sum is more efficient in both computation and communication.

7.1. Experiment 1: solution quality

In this experiment, we choose the set of agents A such that $|A| = 10$ and vary the set of tasks V in increments of 5

[14]We focus on the application of Max-Sumand F-Max-Sumto tree-structured problems in order to tease out their key differences and leave their evaluation over cyclic graphs for future work.

such that $|V| \in \{10, 15, \ldots, 60\}$. Fifty instances of agent and task positions are randomly generated for each set of tasks. Moreover, the deadline of each task is drawn from a uniform distribution that is dependent on the number of tasks as $d_v \in U(0, 10 \times |V|)$ and the workload is drawn from a uniform distribution that is also dependent on the number of tasks as $w_v \in U(0, \frac{10\times|V|}{2})$. In so doing, we set up an average-case problem where the deadlines and workloads are balanced with respect to the number of tasks. That is, there is a fair distribution of easy (long deadline small workload) and hard (short deadline and large workload) tasks.

We compare F-Max-Sum against two other strategies; namely OPGA (see Section 2), which is the only other decentralized algorithm designed for a similar problem to ours, and a centralized approach (which we devised based on the objective function defined in Section 4.3.2). Note that the centralized algorithm also includes a one-step look-ahead process which optimizes the sequence of tasks attempted by the coalitions. Basically, the algorithm not only chooses the best coalition to allocate to each task at time t, but also determines how the allocation would allow agents to reach and complete other tasks at time $t + 1$. It then returns the allocation that maximizes the total number of completed tasks at t and $t + 1$. In so doing, the centralized algorithm acts as an upper bound on the solutions generated by F-Max-Sum and OPGA.

All of the strategies were evaluated on the 50 instances and run over $10 \times |V|$ time steps (i.e. at each time step, each strategy is used to allocate available agents to tasks). The mean total number of tasks completed by each strategy is shown in Fig. 5, along with 95% confidence intervals.

As can be seen from Fig. 5, F-Max-Sum outperforms OPGA by up to 9% (for 60 tasks), and shows this trend persists as the number of tasks increases. This is because the OPGA only seeks to find local maxima, whereas F-Max-Sum finds the optimal of all considered solutions. In addition to this, OPGA is based upon the DSA (see Section 2), and thus has the associated problem of agents sometimes thrashing between assignments. In more detail, we found that F-Max-Sum can complete up to 76% of the tasks given the settings of our experiments, compared with up to 67% with OPGA and up to 90% with a centralized one-step lookahead search. In the next section, we evaluate the robustness of F-Max-Sum against disruptions and see how its performance compares with Max-Sum.

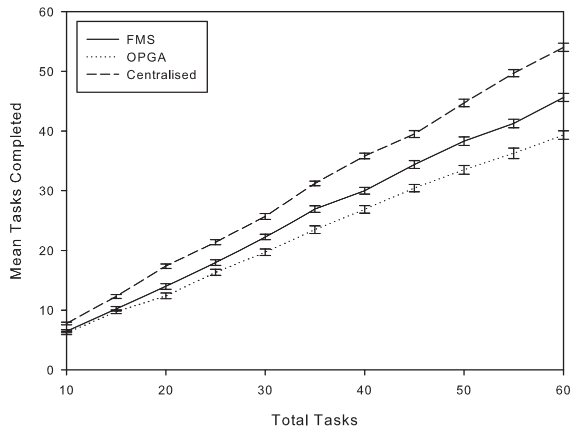


FIGURE 5. Number of tasks completed by agents—comparing the F-Max-Sum (FMS) algorithm to OPGA and a centralized algorithm.

7.2. Experiment 2: robustness

When disruptions occur in the environment, the current set of tasks changes and the agents may need to be re-assigned. In the worst case, this means that the whole allocation of agents to tasks needs to be recomputed. In this experiment, we evaluate how efficient F-Max-Sum is compared with Max-Sum in dealing with changes in the set of tasks. To do this, we generated two distinct sets of random tree-structured factor graphs: one varying the average degree of variable nodes (noted as δ_x), and the other, varying the average degree of factor nodes (noted as δ_f). Given these graphs, in our experiments, we remove one factor (equivalently a task in the RCR scenario) and let the algorithm recompute the solution. It should be noted that we focus our experiments on tree structured graphs on which Max-Sum is guaranteed to converge for the same reasons as noted in footnote 14.

Now, on the one hand, the degree of each factor node is equal to the number of variables it is connected to (i.e. how many agents can save the victim). Hence, it determines the space of possible combinations (or coalitions) the factor needs to search through, which is exactly 2^{δ_f} combinations. On the other hand, the degree of each variable node is equal to the number of factors it is connected to, and hence the number of possible allocations (i.e. $\delta_x - 1$) it can affect if one of its factors is removed from the graph (i.e. a task is removed). Given these features of the problem, our goal is to evaluate how F-Max-Sum can minimize the number of messages that needs to be propagated in the graph and the computation performed by all remaining factor nodes when a given factor is removed from the graph.

Against this background, we separately varied the values of δ_x and δ_f between 1 and 15, in increments of 1. In each simulation, the value that was not being controlled (i.e. δ_f when we controlled δ_v, and vice-versa) was randomly selected from a uniform distribution in [3, 6]. For each value of the controlled variable, 50 random tree instances were generated. We ran both F-Max-Sum and Max-Sum on each instance generated and recorded the following values:[15]

Mean computation units used (MCU)—the average number of combinations of states evaluated at a factor node.

Mean total number of messages sent (TNS)—the average total number of messages sent, by both variable and factor nodes.

[15]These are typical measures used in the DCOP community [13].

Mean total size of messages sent (TSS)—the average total size of all messages sent by all nodes, measured in bytes. This reflects the number of total number of variable states communicated throughout the graph.

Given that we expect F-Max-Sum to outperform Max-Sum on several fronts, we postulate the following hypothesis: Hypothesis: *F-Max-Sum has lower MCU, TNS and TSS for all values of δ_x and δ_f than Max-Sum.*

The intuition behind this hypothesis is that, on the one hand, by allowing variables to only send two messages to the connected factors (representing whether they assign themselves to the task or not), F-Max-Sum prevents those factors from evaluating a large number of redundant states (i.e. those states where an agent is allocated to tasks other than the one associated to the given factors). Moreover, for increasing values of δ_x, the number of factors connected to each variable increases. Then, since F-Max-Sum filters messages sent by variables when disruptions occur (see Section 6.2), as δ_x grows, F-Max-Sum becomes more effective than Max-Sum in preventing unnecessary recomputation and messages.

The results shown in Figs 6–8 confirm our hypothesis. Thus, for increasing values of δ_x, F-Max-Sum takes up to 38% less TNS, up to 91% less TSS and up to 99% less MCU than Max-Sum. For increasing values of δ_f, we notice a similar trend where F-Max-Sum improves upon Max-Sum by up to 70% in MCU, up to 33% in TNS and up to 57% in TSS. These results show that, indeed, F-Max-Sum has the most significant improvements over Max-Sum in graphs where large numbers of factors are connected to each variable. However, we can see that for the lower values of δ_x, the difference in MCU and TSS of F-Max-Sum and Max-Sum is much smaller. This is because the variables have a much smaller domain. In more detail, we showed in Section 6.1 that F-Max-Sum reduces the MCU of Max-Sum from d^n to 2^n, and the size of each message sent to 2 in all cases, as opposed to it depending on d. Thus, it follows that F-Max-Sum will only make a noticable difference where $d > 2$. Given this, we can see that the improvement given by F-Max-Sum increases very quickly as the value of δ_x grows.

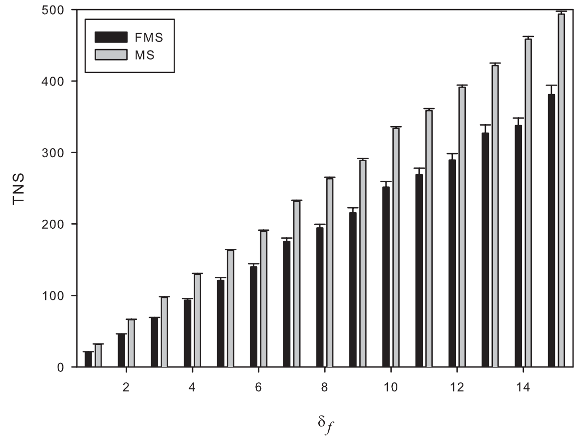


FIGURE 6. Total messages sent over varying edges per each type of node—comparing the FMS algorithm to Max-Sum (MS).

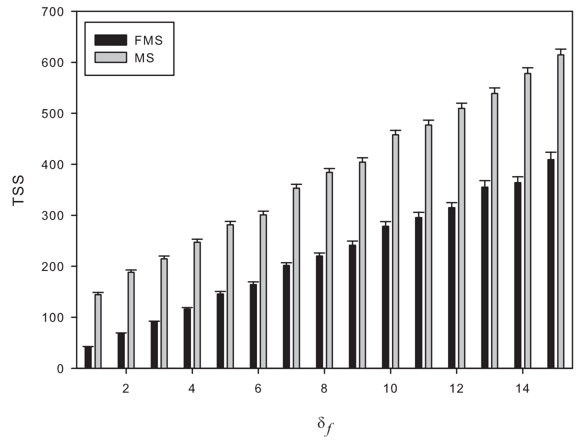


FIGURE 7. Total message size sent over varying edges per each type of node—comparing the FMS algorithm to MS.

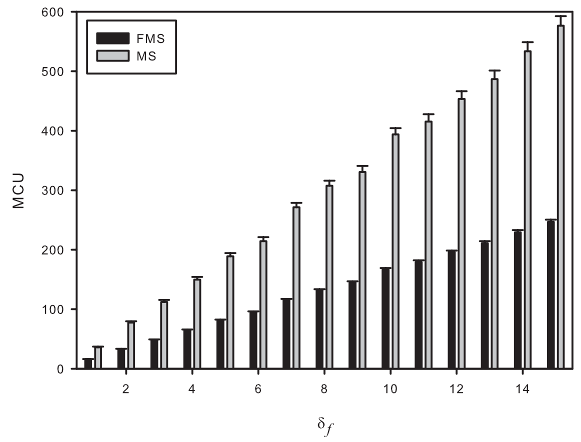


FIGURE 8. Computation units used over varying edges per each type of node—comparing the FMS algorithm to MS.

8. CONCLUSIONS AND FUTURE WORK

In this paper, we have modelled the RCR domain in terms of a CFST constraints. We then provided a DCOP formulation of the problem and showed how to solve it using the Max-Sum algorithm. On the basis of this, we then developed the novel F-Max-Sum algorithm that improves upon Max-Sum to deal with disruptions in its underlying factor graph more effectively. In so doing, we have provided the first full solution to the problem of decentralized coalition formation that pervades disaster management. Our solution is also one that is able to efficiently adapt to a dynamic environment.

While our experiments show how effective F-Max-Sum is in finding good allocations, it will be important to show in future how the algorithm scales with increasing numbers of agents and different profiles of deadlines and workloads. Moreover, we also aim to evaluate the performance of F-Max-Sum under more general settings of variable and factor degrees. Finally, we aim to extend the algorithm to consider the addition and removal of agents from the environment and show how F-Max-Sum can be extended to achieve convergence on cyclic graphs.

FUNDING

This research was undertaken as part of the ALADDIN(Autonomous Learning Agents for Decentralized Data and Information Networks) project and is jointly funded by a BAE Systems and EPSRC (Engineering and Physical Research Council) strategic partnership (EP/C548051/1).

REFERENCES

[1] Timotheou, S. and Loukas, G. (2009) Autonomous Networked Robots for the Establishment of Wireless Communication in Uncertain Emergency Response Scenarios. *Proc. 24th ACM Symp. Applied Computing, Track on Intelligent Robotic Systems*, Honolulu, HI, USA, pp. 1171–1175.

[2] Sandholm, T.W., Larson, K., Andersson, M., Shehory, O. and Tohme, F. (1999) Coalition structure generation with worst case guarantees. *Artif. Intell.*, **111**, 209–238.

[3] Shehory, O. and Kraus, S. (1998) Methods for task allocation via agent coalition formation. *Artif. Intell.*, **101**, 165–200.

[4] Rahwan, T. and Jennings, N.R. (2007) An algorithm for distributing coalitional value calculations among cooperative agents. *Artif. Intell.*, **171**, 535–567.

[5] Rahwan, T., Ramchurn, S.D., Giovannucci, A. and Jennings, N.R. (2009) An anytime algorithm for optimal coalition structure generation. *J. Artif. Intell. Res.*, **34**, 521–567.

[6] Kitano, H. (2000) Robocup Rescue: A Grand Challenge for Multi-agent Systems. *Proc. 4th Int. Conf. Multi-agent Systems (ICMAS)*, Boston, MA, USA, pp. 5–12.

[7] Hillier, F. and Lieberman, G. (2004) *Introduction to Operations Research*. McGraw-Hill Science, Engineering & Mathematics.

[8] Farinelli, A., Rogers, A., Petcu, A. and Jennings, N.R. (2008) Decentralised Coordination of Low-Power Embedded Devices using the Max-sum Algorithm. *Proc. 7th Int. Conf. Autonomous Agents and Multi-Agent Systems (AAMAS'08)*, Estoril, Portugal, pp. 639–646.

[9] Gelenbe, E. and Timotheou, S. (2008) Random neural networks with synchronized interactions. *Neural Comput.*, **20**, 2308–2324.

[10] Zheng, X. and Koenig, S. (2008) Reaction Functions for Task Allocation to Cooperative Agents. *Proc. Int. Joint Conf. Autonomous Agents and Multi-agent Systems (AAMAS'08)*, Estoril, Portugal, pp. 559–566.

[11] Scerri, P., Farinelli, A., Okamoto, S. and Tambe, M. (2005) Allocating Tasks in Extreme Teams. *Proc. 4th Int. Joint Conf. Autonomous Agents and Multi-Agent Systems*, Utrecht, Netherlands, pp. 727–734. ACM New York, NY, USA.

[12] Ferreira, P.R., Santos, F., Bazzan, A.L.C., Epstein, D. and Waskow, S. (2009) Robocup rescue as multiagent task allocation among teams: experiments with task interdependencies. *Auton. Agents Multiagent Syst.*, to appear.

[13] Modi, P.J., Shen, W., Tambe, M. and Yokoo, M. (2005) ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artif. Intell. J.*, **161**, 149–180.

[14] Petcu, A. and Faltings, B. (2005) DPOP: A Scalable Method for Multiagent Constraint Optimization. *Proc. 19th Int. Joint Conf. Artif. Intell., (IJCAI'05)*, Acapulco, Mexico, pp. 266–271.

[15] Fitzpatrick, S. and Meetrens, L. (2003) *Distributed Sensor Networks: A Multi-agent Perspective*. Kluwer Academic.

[16] Maheswaran, R.J., Pearce, J. and Tambe, M. (2005) A Family of Graphical-Game-based Algorithms for Distributed Constraint Optimization Problems. *Coordination of Large-Scale Multiagent Systems*, pp. 127–146. Springer, Heidelberg Germany.

[17] Mailler, R. and Lesser, V. (2004) Solving Distributed Constraint Optimization Problems using Cooperative Mediation. *Proc. 3rd Int. Joint Conf. Autonomous Agents and Multiagent Systems (AAMAS'04)*, pp. 438–445.

[18] Chapman, A., Micillo, R.A., Kota, R. and Jennings, N.R. (2009) Decentralised Dynamic Task Allocation: A Practical Game-Theoretic Approach. *Proc. 8th Int. Conf. Autonomous Agents and Multiagent Systems (AAMAS'09)*, Budapest, Hungary, pp. 915–922.

[19] Filippoupolitis, A., Hey, L., Loukas, G., Gelenbe, E. and Timotheou, S. (2008) Emergency Response Simulation using Wireless Sensor Networks. *Proc. 1st Int. Conf. Ambient Media and Systems*, Quebec City, Canada, pp. 1–8.

[20] Filippoupolitis, A. and Gelenbe, E. (2009) A Distributed Decision Support System for Building Evacuation. *Proc. 2nd IEEE Int. Conf. Human System Interaction*, Catania, Italy, pp. 323–330.

[21] Toth, P. and Vigo, D. (eds) (2002) *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications. Philadelphia.

[22] Pinedo, M. (2008) *Scheduling: Theory, Algorithms and Systems*. Springer.

[23] Waldock, A., Nicholson, D. and Rogers, A. (2008) Cooperative Control using the Max-sum Algorithm. *2nd Int. Workshop on Agent Technology for Sensor Networks*, Estoril, Portugal, pp. 65–70.

[24] Kschischang, F.R., Frey, B.J. and Loeliger, H.A. (2001) Factor graphs and the sum-product algorithm. *IEEE Trans. Inf. Theory*, **47**, 498–519.

[25] MacKay, D.J.C. (2003) *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press.