Coalition Formation with Spatial and Temporal Constraints

Sarvapali D. Ramchurn[1] Maria Polukarov[1] Alessandro Farinelli[1,2] Cuong Truong[1]
Nicholas R. Jennings[1]
[1]IAM Group, School of Electronics and Computer Science
University of Southampton, UK
{sdr,mp3,af2,cnt105,nrj}@ecs.soton.ac.uk
[2]Computer Science Department
University of Verona,Italy
alessandro.farinelli@univr.it

ABSTRACT

The coordination of emergency responders and robots to undertake a number of tasks in disaster scenarios is a grand challenge for multi-agent systems. Central to this endeavour is the problem of forming the best teams (coalitions) of responders to perform the various tasks in the area where the disaster has struck. Moreover, these teams may have to form, disband, and reform in different areas of the disaster region. This is because in most cases there will be more tasks than agents. Hence, agents need to *schedule* themselves to attempt each task in turn. Second, the tasks themselves can be very complex: requiring the agents to work on them for different lengths of time and having *deadlines* by when they need to be completed. The problem is complicated still further when different coalitions perform tasks with different levels of efficiency. Given all these facets, we define this as The Coalition Formation with Spatial and Temporal constraints problem (CFSTP). We show that this problem is NP-hard—in particular, it contains the well-known complex combinatorial problem of Team Orienteering as a special case. Based on this, we design a Mixed Integer Program to optimally solve small-scale instances of the CFSTP and develop new *anytime* heuristics that can, on average, complete 97% of the tasks for large problems (20 agents and 300 tasks). In so doing, our solutions represent the first results for CFSTP.

Categories and Subject Descriptors

I.2.11 [Distributed Artificial Intelligence]: Multi-Agent Systems

General Terms

Algorithms

Keywords

Coalition Formation, Routing and Scheduling, RoboCupRescue

1. INTRODUCTION

Forming teams of agents which are able to effectively work together on tasks is a key issue for many practical applications. In particular, the coordination of emergency responders and robotic agents (e.g., Unmanned Aerial Vehicles and Unmanned Ground Vehicles) to operate in disaster scenarios is a very challenging and important problem for multi-agent systems [5]. Coalitions of responders must coordinate to form teams that are able to tackle fires, dig out victims, and unblock roads in the most effective way. Moreover, since typically the number of agents is limited, these teams are likely to continually form, disband, and reform, in numerous of different parts of the affected area. Hence, agents need to (re)schedule their activities to execute tasks over time. Furthermore, each task might require a certain amount of work to be completed (the *workload* of the task) so that agents need to contribute to different tasks different amounts of time, also depending on their abilities to execute a particular task. Second, tasks may be of different urgency or have *deadlines* by when they need to be completed (otherwise victims die or buildings burn down completely): if a building is on fire, the agents need to arrive there before it is burnt down and spend a specific amount of effort (spraying water and evacuating victims) in order to extinguish it. Finally, some of the tasks may require agents with different capabilities. For example, the task of rescuing a victim from rubble requires agents that have the ability to locate and dig, and those that can heal the victim. The problem may be further complicated if the agents have tools or knowhow that do not work well together, and hence, different teams may be effective to different degrees. For example, a team of two fire-brigade agents, one with a water tank and one with a hose, may not be useful at all if the hose equipment does not fit the water tank properly.

Motivated by the importance of emergency response scenarios, a number of approaches have recently emerged, particularly within the multi-robot routing domain, to address some of these issues. For example, Koenig et al. provide a number of auction-based multi-robot routing algorithms that allow the allocation to tasks that provide quality guarantees on the solutions computed [2, 7]. However, their work focuses on finding the best paths for robots to visit tasks and does not consider team capabilities, task workloads or deadlines. On the other hand, [13] provide algorithms to allocate tasks in large scale systems and dynamic environments, based on token passing. Their approach ensures that the right agents are routed to the right tasks based on capability thresholds (i.e., how much of certain capabilities is needed to complete the task). Now, while their approach does consider agent 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 to work on the same task. Recently however, [6] provide anytime algorithms based on Mixed Integer Programming (MIP) to allocate teams of robots to tasks requiring teams with specific capabilities. While their work bridges a major gap in the literature, there is still a lack of solutions for tasks with deadlines and coalitions with the same capabilities but different degrees of effectiveness. Now, such a dearth of solutions can be partially blamed on the fact that the problem of allocating coalitions to tasks in space and time is not as well defined as other important combinatorial optimisation prob-

Cite as: Coalition Formation with Spatial and Temporal Constraints, S. D. Ramchurn, M. Polukarov, A. Farinelli, C. Truong, N. R. Jennings, *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, van der Hoek, Kaminka, Lespérance, Luck and Sen (eds.), May, 10–14, 2010, Toronto, Canada, pp. 1181-1188

lems such as the Travelling Salesman Problem (TSP) (on which most work on multi-robot routing is based). Nevertheless, we believe it is important to define the problem, relate it to other existing combinatorial problems, so that approximate solution techniques and anytime heuristics (that provide increasingly better solutions if given more time) can be re-used, and to devise new solutions that tackle the specific features of this problem.

Against this background, in this paper we define the problem of allocating coalitions of agents to spatially distributed tasks with workloads and deadlines so as to maximise the total number of tasks completed over time. More precisely, we introduce the following setting. Consider a set of tasks and a set of agents, with their given locations and corresponding travel times between them. Each task is associated with a workload and deadline so that the task is considered completed if and only if the required amount of work on it is done by the given time threshold. Furthermore, if a number of agents are working on a task simultaneously, their contribution per time interval may not be necessarily additive. Thus, while each of agents i and j, on their own, can do x amount of work in a time interval, they may produce $y > 2x$ ($y < 2x$ in case of sub-additive domains), if they collaborate and work on the task at the same time. Given these spatial and temporal constraints and coalitional effects, the goal is to schedule agents to tasks so that the number of completed tasks is maximised. We term this model as *Coalition Formation with Spatial and Temporal constraints problem* (CFSTP).

Our contribution includes a formalisation of the problem and a definition of its relationship with other existing combinatorial optimisation problems. Moreover, we show how to solve it optimally and approximately using MIP and scheduling heuristics respectively. Specifically, this paper advances the state of the art in the following ways. First, we show that the CFSTP is NP-Hard and generalises the well known Team Orienteering problem (TOP), which itself generalises the TSP. In so doing, we build the case for developing new algorithms for the CFSTP since it is not possible to *directly* apply those used in other problems. Second, we provide an optimal solution to the CFSTP using MIP as a benchmark to solve small-sized problems where computation time is not an issue. Third, we devise new *anytime* heuristics to find approximate solutions fast and hence provide the first lower bounds on the solutions that can be found in any given CFSTP. Finally, we empirically evaluate our algorithms and show that they compute (in less than 5 seconds) 97% efficient solutions for non-trivial problems involving up to 20 agents and 300 tasks (with uniformly distributed workloads and deadlines). Thus, our work encompasses all aspects of the CFSTP, from the model, generalisations, optimal solution, and an anytime heuristic. Moreover, our algorithms can be regarded as the first to ever solve the CFSTP and are therefore the benchmarks against which future algorithms for the CFSTP can be evaluated.

The rest of the paper is organised as follows. In Section 2, we provide notation and basic definitions, and present the formal CFSTP model in Section 3, using a disaster management scenario as a running example. Then, Section 4 examines the complexity of the problem, shows how the CFSTP generalises the TOP and hence is NP-hard. In Section 5, the MIP formulation for the CFSTP is provided, as well as some results on the scalability of the approach. Section 6 describes our heuristics that provide approximate solutions to the CFSTP. Section 7 empirically evaluates our algorithms and Section 8 concludes.

2. BACKGROUND

We first provide the basic notation and then go on to expand on how coalitions can be represented and how allocations of agents to tasks can be used to generate allocations of coalitions to tasks.

2.1 Basic Definitions

Agents are noted as $a_1, \ldots, a_n \in A$ that have to complete a number of tasks $v_1, \ldots, v_m \in V$ located in different parts of a space (more than one task may be located in the same place); the set of all possible task locations is denoted by L_V. A concrete example of such a scenario exists in the RoboCupRescue simulator where a number of ambulances have to allocate themselves to victims trapped in buildings or fire brigades need to form coalitions to extinguish fires across a city [5]. The time taken for an agent to travel from one location to another is given by a function $\rho : (L_A \cup L_V) \times L_V \rightarrow [0, \infty]$ (assuming all agents can move at the same speed) where L_A is the set of agents' initial 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 the time until which the victim will survive without being rescued or the time until which the 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 as d_{max} 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.

2.2 Coalitions

Agents may form coalitions for several reasons. First, the number of tasks may be much larger than the number of agents. Hence, agents need to schedule themselves in, possibly different, groups to try and maximise the number of tasks completed. Second, the workload for a given task may be too high for one agent to complete it by the deadline of that task. Hence, agents need to work together on the same task at the same time to complete it in time.

We define what it means for an agent to "work" on a task later in this section. First, however, we denote the fact that an agent a works on a task v at a given time t by $\tau_t^{a \rightarrow v}$. We define $T = \{\tau_t^{a \rightarrow v}\}_{a \in A, v \in V, t \in \{0, \cdots, d_{max}\}}$ 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 $\tau_t^{C \rightarrow v}$ 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 $t' \in \{0, \cdots, d_{max}\}$ within which we want to explore the coalitions that could exist,[1] we define the corresponding (feasible) allocation of coalitions, $\Gamma(T', t')$, over the given time period, as follows:

$$\Gamma(T', t') = \left\{ \tau_t^{C \rightarrow v} \mid C = \{a \mid \tau_t^{a \rightarrow v} \in T'\}, v \in V, t \leq t' \right\} \quad (1)$$

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

Obviously, physical 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 $[0, t']$, $t' \leq d_{max}$. Therefore, coalitions that exist at different locations at the same time do not overlap.

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

The work that a coalition performs on a task in each time unit (or, step) decreases the workload of that task.[2] For example, consider the case of a victim buried under debris in the RoboCup Rescue domain [5], by removing part of the debris rescue agents reduce the amount of work necessary to dig out the victim. The extent to which the workload decreases is dependent on the *value* of the coalition, given by the function $u : 2^A \rightarrow \mathbb{N}^+$. 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, $u(C \cup \{a\}) \geq u(C)$.[3] For now we will also assume tasks are, in turn, homogeneous. Since tasks are considered to be atomic, only one coalition can perform one task at a time. For example, the above assumptions hold when we consider the problem of allocating ambulances to rescue victims in the RoboCup Rescue domain.

Given the above definitions, we next define the CFSTP that this setup generates and detail the associated constraints.

3. COALITIONS WITH CONSTRAINTS

The goal of CFSTP is to maximise the number of tasks completed given all possible allocations of agents to tasks. In particular, this allocation needs to take into account two main types of constraints: temporal and spatial. The former take care of restrictions with respect to the time taken by agents to finish a task, while the latter restrict the movement of agents around the tasks given the time available to them. In what follows, we detail these constraints and then formulate the objective function we aim to maximise. We will assume that the solution should contain some allocation of agents to tasks as the set $T' \subseteq T$.

3.1 Temporal Constraints

To specify constraints on the agents' completion of tasks, we define a binary-valued function $W : V \times \mathbf{\Gamma} \rightarrow \{0, 1\}$ as follows:

$$W(v, \Gamma) = \begin{cases} 1, & w_v - \sum_{\tau_t^{C \rightarrow v} \in \Gamma} u(C) \leq 0 \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

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

$$\Delta(v, \Gamma) = \begin{cases} 1, & \max_{\tau_t^{C \rightarrow v} \in \Gamma} t \leq d_v \wedge W(v, \Gamma) = 1 \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

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

3.2 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 agents in travelling from task to task (i.e., 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 others). These spatial constraints therefore apply over the existence of coalitions. If agent a is routing to task v from location $l \in L_A \cup L_V$ (which is either its initial location or another task) at a given time t, the starting time $s_a^v \in [0, d_v]$ at which agent a starts working on the task v must satisfy the following:

$$s_a^v \geq t + \rho(l, v) \quad (4)$$

Note that given the condition in (4), coupled with the deadline constraint (3), and assuming that travel times are proportional to distances between the locations (and hence satisfy the triangle inequality), we can restrict all possible assignments to the following:

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

where $l_a \in L$ is the location of agent a at time t.

Similarly, at a given time t', if we knew the partial solution (i.e., the allocations of each agent) 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 task v and a subset of agents $A' \subseteq A$, we can specialise the set of all possible allocations to:

$$T(A', v, t') = \left\{ \{\tau_t^{a \rightarrow v}\}_{t \in \{t' + \rho(l_{t'}^a, v), \cdots, d_v\}} \right\}_{a \in A'} \quad (5)$$

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 $l_a \in L_A$, then:

$$s_a^v \geq \rho(l_a, v) \quad (6)$$

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

$$s_a^v \geq s_a^{v'} + |\cup_{t \in \{\rho(l_a, v'), \cdots, t' \leq d_{v'}\}} \tau_t^{a \rightarrow v'}| + \rho(v', v) \quad (7)$$

where t' is the time at which a stops working at v'.

Similar to (6), (7) 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 $\rho(v', v)$ we get the earliest time by which task v is reached by agent a.

Note that the routing of agents to tasks in T may not actually be feasible (i.e., where the agents reach the tasks before their deadlines) and one of the challenges is to find those routes that are feasible and efficient (i.e., minimise the time travelling and maximise the tasks completed). To find such an assignment we note this problem is very similar to solving a Vehicle Routing Problem (VRP) with time windows [14]. Thus, to find an assignment that is consistent with the constraints defined in section 3.1 is, in turn, equivalent to finding a feasible schedule for the tasks [9]. 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 that can be attempted (as we show next).

3.3 The Objective Function

The main objective of the CFSTP is to maximise the number of tasks completed. This can be expressed as follows:

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

subject to constraints in equations (6) and (7).

It is important to note that 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*! In order to understand the practical implications of trying to find an optimal

[2]Note that we here assume that workloads and deadlines are independent (e.g., digging down rubble does not improve the health of a victim) of the task in any way. In future work we will consider removing this assumption.

[3]While these assumptions are reasonable in most robot routing and emergency response domains, and help speed up computation, we aim to remove them in future work.

solution in this case, we propose an optimal algorithm based on MIP in section 5 and evaluate it with increasing numbers of agents and tasks. We show that while the MIP algorithm is a good benchmark for small sized problems, as expected, it does not scale up for large problems requiring quick solutions. Hence, in section 6 we propose novel heuristics to try to combat the complexity of the CFSTP. Before doing so however, in the next section we elaborate on the complexity and other combinatorial problems related to the CFSTP. Our aim is to show that the CFSTP is a novel optimisation problem that generalises several existing combinatorial problems. This sets the stage for further analysis of all the possible special cases of the CFSTP (which is beyond the scope of this paper).

4. COMPLEXITY ANALYSIS

In this section we show that CFSTP falls within the class of NP-hard problems, and show how it is a new generalisation of the Team Orienteering Problem.

4.1 NP-Hardness of the CFSTP

The well-known Orienteering Problem (OP)[4] is defined as follows. Given n nodes in the Euclidean plane, each with a score $\sigma(i) \geq 0$ where $\sigma(1) = \sigma(n) = 0$, find a route of maximum score through these nodes that begins at 1 and ends at n, and whose duration is no greater than a given value $t_{\max}$. This problem (and, in particular, its special case with unit scores, i.e., $\sigma(i) = 1$ for any $i \neq 1, n$) has been shown to be NP-hard [3], as it contains the Travelling Salesman Problem (TSP) as a special case. Similarly to the proof in [3], the TSP can be reduced to our problem or, alternatively, one can observe that the unit-score version of the OP is a special case of CFSTP, hence the following result.

THEOREM 1. *CFSTP is NP-hard.*

PROOF. To see this, consider a single-agent version of CFSTP in which the set of tasks corresponds to the subset of nodes $\{i \neq 1, n\}$, the agent's initial location is given by node 1, and the travel times $\rho(\cdot)$ between the locations are set up accordingly with the distances $\delta(\cdot)$ between the corresponding nodes on the plane. Set the workloads $w_v = 0$ for all tasks v and the deadlines to $d_v = t_{\max} - \delta(i_v, n)$, where $\delta(i_v, n)$ is the distance between node i_v that corresponds to task v, and the terminal node, n. It is easy to see that any feasible solution for this problem is feasible for the corresponding OP, as the deadlines are set in a way that from any visited location (including the last one in the route) the terminal node can be reached by the time $t_{\max}$. For optimality, notice that since travel times between the locations are determined by distance on the Euclidean plane, the triangle inequality holds, and hence $\rho(i, n) \leq \rho(i, i_0) + \sum_{j=1}^{k} \rho(i_{j-1}, i_j) + \rho(i_k, n)$, for any sequence of nodes $(i, i_0, i_1, \ldots, i_k, n)$. Thereby, our deadline requirements are not "overrestricting". Therefore, if we had a polynomial algorithm for solving CFSTP, we could find a route starting at 1 and ending at n, that maximises the number of locations visited by time $t_{\max}$, thus solving the OP in polynomial time. Since the OP is NP-hard, the CFSTP is at least as difficult; in fact, even its simplified, single-agent version with zero task workloads, is so. □

4.2 Generalising the Team Orienteering Problem

Having shown that CFSTP is NP-hard based on its generalisation of the OP (which basically ignores deadlines, workloads, and coalitions), we now turn to defining how it generalises the extended version of the OP, the Team Orienteering Problem (TOP) [1], in which the performance of the entire set of agents is taken into account. In TOP, each member of the team (assuming that all start at the same point) tries to visit as many locations as possible within a prescribed time limit, and then ends at the terminal point (also joint to all). Once a team member visits a location and is awarded its associated score, no other agent can be awarded a score from the same point. Thus, each agent has to choose a sequence of locations to visit so that there is minimal overlap in the locations visited by the members of the team, the time limit is not violated, and the total team score is maximised. We now show that the TOP can be viewed as a special case of the CFSTP. Specifically, we reduce from the set of TOP with integer scores. We then proceed and argue that considering integer scores is sufficient as for any TOP with real scores there is a corresponding problem with integer scores whose set of optimal solutions is included in that of the original problem.

Consider an instance of the TOP with integer scores and reduce it to CFSTP as follows. With each node $i \neq 1, n$ with score $\sigma(i)$, associate a location, l^i, with tasks $\{v_1^i, v_2^i, \ldots, v_{\sigma(i)}^i\}$. Assign zero travel times between any two tasks at the same location; for tasks at different locations set travel times according to the distances between the corresponding nodes. As before, set zero workloads to all the tasks and for each task v^i at location l^i define its deadline as $d_{v^i} = t_{\max} - \delta(i, n)$, where $\delta(i, n)$ is the distance from i to the terminal node. Since the workloads are zeros, any time an agent visits a location, it can complete all the tasks at it, and thus collect the full score of the corresponding control point. Obviously, if another agent visits the same location, this will not bring any additional score to the team. By similar arguments regarding feasibility and optimality of routes we gave for the single-competitor case, we conclude that the sets of optimal solutions for the two problems coincide.

Now, it remains to show that every TOP can be represented by a TOP with integer scores. This is true because the number of nodes, and hence, the number of scores, is finite, and therefore so is the number of their possible subsums—representing possible objective values of the problem. Indeed, a feasible solution is given by the subset of nodes so that there is a route that has a total length no greater than $t_{\max}$ through these nodes, and the corresponding objective value is given by the sum of scores over the selected subset of nodes; hence, there are at most $\sum_{q=1}^{n-2} \binom{n-2}{q}$ feasible objective values. Arrange them in increasing order, and let x be the minimal difference between two values in the sequence. Multiply all the sequence by $M > \frac{n}{x}$ (accordingly, each score is multiplied by M). Note that the difference between any two values in the modified sequence is at least n. When all scores are rounded up, each subsum—value in the sequence—will grow by at most $n - 2$, and hence this will not affect the order of the values in the sequence (note that if there were several solutions with the same objective value, they may result in different values after rounding; however, the order of these "groups" of values will remain the same). Now, the solution that corresponds to the maximal value in the resulting sequence gives a maximal value in the original sequence of objective values. Thus, the set of optimal solutions for the TOP with these modified, integer, scores is contained in the set of optimal solutions of the original TOP.

From the literature,[5] most related studies have typically focused on routing optimisation under time and capacity constraints, and ignore the issue of coalition formation and (possibly non-linear) coalitional contributions to tasks they are assigned to. Thus, the

[4]Also referred to as the "generalised traveling salesman problem" [15].

[5]Perhaps, the most relevant models to our problem are the Orienteering Problem with Time Windows [4] in which a point can only be visited within a specified time interval, and the Vehicle Routing Problem with Time Windows and a Limited Number of Vehicles [8] where a feasible solution may contain unserved customers and/or relaxed time windows.

CFSTP is the first attempt to deal with the problem of optimal coalition formation under temporal and spatial constraints. Obviously, this results in high complexity and difficulty of solution.

5. MIP FORMULATION

We present here our MIP formulation of the CFSTP to establish a benchmarking optimal solution technique.

Specifically, the aim of CFSTP is to maximise the number of tasks completed by the agents such that the deadline of each completed task is met. Our decision variables include binary variables $\delta_v \in \{0,1\}$ for each $v \in V$, indicating whether the task v is successfully completed, and the objective is to maximise the sum of these variables over the set of tasks (see 3.3). The *schedule* decision variables include s_a^v (integer) representing the absolute time at which agent $a \in A$ starts its work on task $v \in V$, and with a slight abuse of notation we can represent assignments as binary variables $\tau_t^{a\to v} \in \{0,1\}$ indicating whether agent $a \in A$ is at location $v \in V$ in the time interval t, $t \in T = \{0,1,\ldots,d_{max}\}$ (d_{max} is defined in section 2.1). We assume that the allocation process starts from the time interval $t = 1$, and use the zero time interval for sake of formulation only. Thus, for all agents and locations we set zero service at time $t = 0$ and use a convention that zero starting time for a particular agent at a particular location essentially means that the agent *never* starts working at this location. The coalition values $u(C) \in \mathbb{R}$ are assumed to be given as an input. Given this we use $\tau_t^{C\to v} \in \{0,1\}$ as binary variables to indicate whether a coalition of agents C, $C \in 2^A$, is the one working on task $v \in V$ in the time interval $t \in T$ (see (10) below). We use *routing* variables r_{ij}^a (binary) to indicate whether agent $a \in A$ travels from location $i \in L_A \cup L_V$ to location $j \in L_V$. Now, we can finally formulate CFSTP in terms of the objective function and a set of constraints.

Objective function:

$$\max \sum_{v\in V} \delta_v. \tag{9}$$

subject to:

• *completion* constraints:

$$\sum_{t=0}^{d_{max}} \sum_{C\in 2^A} \tau_t^{C\to v} u(C) \geq \delta_v w_v, \forall v\in V, \tag{10}$$

$$\sum_{t=0}^{d_{max}} \sum_{C\in 2^A} \tau_t^{C\to v} \leq M\cdot \delta_v, \forall v\in V, \tag{11}$$

$$\sum_{C\in 2^A} \tau_t^{C\to v} \leq \delta_v, \forall t\in[0,d_{max}], v\in V, t\in[0,d_{max}], \tag{12}$$

• *deadline* constraints:

$$s_a^v + \sum_{t=0}^{d_{max}} \tau_t^{a\to v} \leq d_v, \forall v\in V, a\in A, \tag{13}$$

• *starting time, routing, and service consistency* constraints:

$$\rho_{(l_a,j)} \leq s_a^j + M(1-r_{l_aj}^a), \forall a\in A, j\in L_V, \tag{14}$$

$$s_a^i + \sum_{t=0}^{d_{max}} \tau_t^{a\to i} + \rho(i,j) \leq s_a^j + M(1-r_{ij}^a), \forall a\in A, i,j\in L_V, \tag{15}$$

$$\sum_{v\in V} \tau_t^{a\to v} \leq 1, \forall a\in A, t\in T, \tag{16}$$

$$1-2|t-(s_a^v)| \leq \tau_t^{a\to v} - \tau_{t-1}^{a\to v}, \forall a\in A, v\in V, t\in T\setminus\{0\}, \tag{17}$$

$$\sum_{t=0}^{d_{max}} |\tau_{t+1}^{a\to v} - \tau_t^{a\to v}| \leq 2, \forall a\in A, v\in V, \tag{18}$$

$$1-|t-(s_a^j-\rho(i,j)| - M(1-r_{ij}^a) \leq |\tau_t^{a\to i} - \tau_{t-1}^{a\to i}|, \quad \forall a\in A, i,j\in L_V, t\in T\setminus\{0\}, \tag{19}$$

$$r_{ii}^a = 0, \forall a\in A, i\in L_V, \tag{20}$$

$$r_{l_aj}^a + \sum_{i\in L_V} r_{ij}^a \leq 1, \forall a\in A, j\in L_V, \tag{21}$$

$$\sum_{j\in L_V} r_{ij}^a \leq 1, \forall a\in A, i\in L_V\cup\{l_a\}, \tag{22}$$

$$\sum_{i\in L_V} r_{ij}^a \leq M\sum \tau_t^{a\to j}, \forall j\in L_V, a\in A, \tag{23}$$

$$\sum_{i\in L_V} r_{ij}^a \leq M\sum \tau_t^{a\to i}, -\forall i\in L_V, a\in A, \tag{24}$$

• *linking* constraints:

$$\sum_{a\in A, i\in L_V\cup\{l_a\}} r_{ij}^a \geq \delta_i, \forall j\in L_V, \tag{25}$$

$$\sum_{a\in C} \tau_t^{a\to v} \geq |C|\cdot \tau_t^{C\to v}, \forall C\in 2^A, a\in C, v\in V, t\in[0,d_{max}], \tag{26}$$

$$s_a^v \leq M\sum_{t=0}^{d_{max}} \tau_t^{a\to v}, \forall a\in A, v\in V, \tag{27}$$

$$\sum_{t=0}^{d_{max}} \tau_t^{a\to v} \leq M s_a^v, \forall a\in A, v\in V, \tag{28}$$

$$s_a^v \geq \sum_{i\in L_V\cup\{l_a\}} r_{il_v}^a, \forall a\in A, v\in V, \tag{29}$$

$$s_a^v \leq M\sum_{i\in L_V\cup\{l_a\}} r_{il_v}^a, \forall a\in A, v\in V, \tag{30}$$

$$\sum_{i\in L_V} \tau_t^{a\to i} \leq 1, \forall t\in[0,d_{max}], \tag{31}$$

$$\tau_0^{a\to v} = 0, \forall a\in A, v\in V, \tag{32}$$

where $\rho(i,j)$ represents the travel time from $i \in L_A \cup L_V$ to $j \in L_V$, and M is a large positive number.

The completion constraints in (10) to (11) determine whether the total contribution of all the agents to each task is either greater than or equal to the task's workload (if the task is completed) or zero (otherwise). In particular, (12) enforces unique assignments of coalitions to completed tasks. Constraint (13) requires the deadline of each (completed) task to be met (note that by (10), (11) and by the convention about starting times, in the case of an uncompleted task, the left hand side of the inequality will be equal to zero).

The routing consistency constraints in (14) and (15) imply that the starting time of a particular agent at a particular location is greater than or equal to the time at which the agent is able to arrive at the location after it had finished its work at the previous location, or from its initial location (for its first task). The constraints in (16) require an agent to be working at most at one location in a particular time interval. Constraint (17) implies that the binary service variable $\tau_t^{a\to v}$ of agent a at location v has different values at times $t-1$ and t if the corresponding starting time equals t; constraint (18) then requires that an agent changes its service status (i.e., "working" v/s "not working") at a particular location at most twice. Coupled with (32), this implies that if an agent is working at a particular location, then its service indicator changes it value from zero to one at the starting time, and then from one to zero when the agent stops its service. Constraint (19) links between service binary variables and finishing service times, in a similar way. Routing constraints in (20)-(22) require that an agent does not travel from a location to itself, and, given a particular location, it can arrive there from only one previous location and can leave from there for at most one next location.

The linking constraints in (25)-(30) determine that the following conditions are satisfied: (i) if there is an agent that arrives to a particular location, the task at this location should be completed; (ii) a coalition is considered to be working at a particular location in a given time interval if and only if all its members are working at that location in that time interval; (iii) an agent starts working at a

particular location if and only if it spends some time at this location; (iv) an agent starts working at a particular location if and only if it actually arrives there from a previous, or from its initial, location. Moreover, coupling constraint (18) with (26) also ensures the coalition (of which the agents are members) service time variable only changes twice. Also constraints (31) and (32) ensure that the algorithm allows agents to service only one task per time step and does not assign agents to tasks at time equals zero respectively.

We implemented the solution presented in this section using IBM ILOG CPLEX 9.1 and found that for small scenarios with not more than 4 agents and 7 tasks, the algorithm takes more than 2 hours to find the optimal solution. Since the problem is NP-Hard, it was expected that the running times would grow exponentially. While this permits solving small problems off-line, such running times are not acceptable for the type of domains (e.g., emergency response and surveillance missions) that need solutions to the CFSTP online. In particular, there is a need for algorithms that can return solutions quickly and anytime (i.e., the algorithm starts by giving partial solutions and improves upon them if given more time). We present such a solution in the next section.

6. THE SCHEDULING HEURISTICS FORMULATION

Having established the benchmark for optimal algorithms for the CFSTP, we now turn our attention to designing approximate procedures that can return efficient solutions quickly. Thus, here, we first present a set of heuristics that allow us to generate assignments $\tau_t^{a\rightarrow v}$ for each agent in the system and then define a procedure which optimises over these possible assignments.[6] We incorporate these procedures into our Coalition Formation with Look-Ahead (CFLA) algorithm. Generally speaking, the CFLA is based upon the following main objectives:

1. Maximise the number of tasks completed—to achieve this we only allocate agents to tasks that they can complete.

2. Maximise the working time of the agents—to achieve this we make sure that the smallest coalitions possible are allocated to each task. In so doing, we ensure that the maximum number of tasks are attempted at any one time and we minimise the travel time of all the agents (since making more agents travel to the same tasks increases the total travel time and hence less time is actually being used to work). However, the tradeoff in maximising the working time is that the agents may take longer to finish some tasks and can therefore not finish future tasks. Thus, to minimise the risk of losing future tasks, in the CFLA we perform a one-step look-ahead (more than one step leads to significantly more computation) to find out the consequence of completing each task with each possible coalition. Hence, while our solution may not return an optimal solution, it clearly establishes an empirical lower bound on the optimal solution (see section 7).

3. Minimise the time taken by coalitions to complete the maximum possible number of tasks—to achieve this we need to make sure that the most efficient coalitions are chosen to complete the maximum possible tasks that can be reached. Specifically, an optimal algorithm would try to balance the allocation of the agents *across* all the tasks to minimise the overall time to complete the maximum possible number of tasks. However, doing so, would require searching an exponentially large (in the number of tasks) space. In contrast, we adopt a greedy approach to allocate the most efficient coalition to the most important tasks (where importance can be defined in terms of how many other tasks could be completed in the next time step when agents are allocated to the given task), which generates good solutions in reasonable time.

Based on above principles, the algorithm works as an iterative process as follows: Step 1: define which tasks can be reached by their deadline by which agents: Step 2—define which coalition of agents should be allocated to a given task; Step 3—define which tasks to attempt first; Step 4—repeat from step 1 assuming agents start from the point where they finish their previous task until all tasks are allocated. We now detail each step in turn.

6.1 Step 1: Defining Feasible Allocations

Recall from section 2.2 that the set T contains all possible assignments of agents at all times. Then let $T' \subseteq T$ and t' be the point in time such that $T_{t'} = \{\tau_t^{a\rightarrow v}\}_{a\in A, v\in V, t\in\{0,\cdots,t'\}}$. Here we show how to exactly define $T_{t'}$. Thus, we need to find which tasks are *accessible* by each agent in the system. By "accessible", we mean that the agent can reach the task at time t' given its earlier position l_a^t at time $t < t'$. Let us assume that there exists a set $A_{busy}^{t'}$ that stores all agents that are working (or travelling to a task) at time interval t'. We show how to construct $A_{busy}^{t'}$ when agents get allocated in section 6.3. Now, also assume that the set of tasks that have not been completed at time t is V_t. Then, using algorithm 1 we can construct $T_{t'}$. To do so, the algorithm loops through all agents that

Algorithm 1 Define the set of feasible assignments.

Require: $A_{busy}^{t'}$
1: $T_{t'} = \emptyset$ {*Initialise with empty set.*}
2: for all $v \in V_t$ do
3: for all $a \in A_{t'} = A \setminus A_{busy}^{t'}$ do
4: if $t + \rho(l_a^t, l_v) \leq t'$ where $t \leq t'$ {*a can reach v at t'*} then
5: $T_{t'} \leftarrow T_{t'} \cup \{\tau_{t'}^{a\rightarrow v}\}$ {*include the agent*}
6: end if
7: end for
8: end for

are not busy and checks in step 4 that the agent can arrive at the task at time t'. By defining the allocation of agents at every time t using T_t we are able to generate new positions for agents for later time points and hence define new starting positions for $t'' > t'$.

Now, having computed $T_{t'}$, we can construct assignments at time t of coalitions to tasks $\Gamma(T_{t'}, t)$ (defined in 2.2). However, as we are aiming to maximise the number of tasks attempted, we next try to find the smallest possible coalitions that can service the tasks.

6.2 Step 2: Choosing the Best Coalition

In this step, we compute the best set of agents that can be allocated to complete a given task. Our approach is based on a minmin objective whereby we try to minimise both the size of the coalitions used and the time at which a task can be completed. To this end, here, we detail an earliest-completion-first (ECF) allocation algorithm (adapted from [11]) that iterates over the set of possible agent allocations (defined in step 1) to generate the possible allocations of coalitions to tasks $\tau_t^{C\rightarrow v}$ at time t. Using our ECF algorithm we try to minimise the completion time for the given task so that agents can be allocated to other tasks as soon as possible. Thus, in step 2 of algorithm 2, we first define the size of the smallest coalition that can complete the task. This is a non-trivial process that requires searching through all coalitions that can reach the task and checking whether they can complete it. This procedure, in the worst case, involves searching $2^{|A_v|}$ coalitions where A_v is the set of agents that can reach v. Given this, our solution involves applying binary search to select the best coalition size, coupled with [10]'s linear time algorithm to loop through the coalitions of the selected size, resulting in computation of $O(\sum_{k=0}^{size_{min}} 2^k)$. Having done so, the

[6]It is important to note here that we do not use one of the many heuristics that have been used in TSP or OP because such heuristics generally do not incorporate a notion of coalition values and, as a result, would not constitute *admissible* heuristics for the CFSTP.

Algorithm 2 ECF algorithm.

Require: $T_{t'}, v \in V_t$

1: Define $\Gamma(T_{t'}, d_v)$ as in equation (1).
2: Find $size_{min} = \min_{\tau_t^{C \to v} \in \Gamma(T_{t'}, d_v)} |C|$ where $\sum_t^{d_v - t'} u(C) \cdot t - w_v \geq 0$ {*get the minimum size of the coalition that can complete the task — use binary search and algorithm by [10] for this.*}
3: set $t_v^{min} \leftarrow d_v$
4: for all $\tau_t^{C \to v} \in \Gamma(T_{t'}, d_v)$ where $|C| = size_{min}$ {*use [10] to cycle through coalitions of a given size.*} do
5: $work \leftarrow w_v - \sum_{\tau_{t'}^{C \to v}, t' \in \{t, \cdots, d_v\}} u(C_v)$
6: if $work \leq 0$ then
7: $t_{minmax} = \min_{t_{max}} (w_v - \sum_{\tau_{t'}^{C \to v}, t' \in \{t, \cdots, t_{max}\}} u(C_v)$ {*minimum time at which the task can be completed.*}
8: if $t_{minmax} < t_v^{min}$ {*check if this is a better coalition.*} then
9: $t_v^{min} = t_{minmax}$ {*record minimum time.*}
10: $C^* = C_v$ {*record the ECF coalition allocation.*}
11: end if
12: end if
13: end for
14: return C^*

algorithm loops through coalitions of the given size ($size_{min}$) to find the one that can complete the task the earliest (steps 4 to 13).

As can be seen, the ECF algorithm is completely myopic as it does not consider the effect of allocating a coalition on other tasks to be allocated at a later time $t' > t$. To be able to capture such effects, it is first important to represent the effect a given allocation can have on future allocations and then apply an algorithm that can take into account such effects. To this end, in the next section we detail just such a procedure which is the core of the CFLA.

6.3 Steps 3 & 4: Allocating Tasks

Given the procedure to choose the best coalition for a given task, the algorithm proceeds by going through each task to check how many tasks will be accessible in the future. Based on this, it prioritises tasks that can lead to the most tasks being completed in future. The algorithm (shown in algorithm 3) proceeds as follows. We define a set $V_{completed}$ for all tasks that have been completed. This set is populated as the algorithm runs through all tasks and allocates coalitions as it goes along. Then, for each task that has not been completed yet, it picks the best coalition that can be allocated to it (steps 5 to 7) using a combination of algorithms 1 and 2. Then, for each other task that has not been visited yet, the algorithm checks to see if, after assigning the coalition selected (step 9), the available set of agents can complete other tasks at later times (see steps 10 to 19). For each reachable task, the count of reachable tasks is increased by one. In fact, in so doing, we effectively traverse a tree (of depth 1) rooted at a given task and where leaf nodes represent other tasks that can be completed at a later time given that a particular coalition has been allocated at the root. The degree of such a tree rooted at each task is then used as the measure to decide whether to complete the root task or not (see steps 22 to 25).

As can be seen, we fix the look-ahead to one step since the computation as shown earlier, grows exponentially in the number of agents and hence, by considering further future steps the required computation would grow even more. We also show in section 7 that a one step look-ahead generates good-enough solutions.[7] Finally, it is important to note that as the CFLA incrementally builds allocations, the algorithm is inherently *anytime* and could be stopped before completion to give partial solutions (i.e., to more important tasks) that get better as the algorithm is given more time.

[7]Note that it is unlikely that values will be defined for all 2^n potential coalitions (which we assume here and in our experiments). Reducing the number of *feasible* coalitions could significantly reduce computation and therefore permit deeper look-aheads.

Algorithm 3 Allocating Tasks.

1: $degree_v = 0$
2: repeat
3: for all $v \in V \setminus V_{completed}$ do
4: set $T'_t = T_t$ {*where t is current time of operation*}
5: Generate T_{d_v} from algorithm 1
6: Compute $T^v_{d_v}$ based on T_{d_v} {*Get only those relevant to v.*}
7: Generate C^*_v using algorithm 2 for agent v.
8:
9: Create $\tau_{t''}^{C \to v}$ for all $t'' = t, \cdots t_v$ where t_v is equal to the time at which $w_v - \sum_{\tau_{t'}^{C \to v}, t' \in \{t, \cdots, t_v\}} u(C_v) \leq 0$ {*keep track of the time of completion*}
10: $A^{t,t_v}_{busy} = C_v$
11: $A^{t_v}_{busy} = \cup_{\tau' \leq t_v, \tau'' > t_v} A^{\tau', \tau''}_{busy}$
12: $A_{t_v} \leftarrow A \setminus A^{t_v}_{busy}$ {*Update the list that can reach the task at t_v.*}
13: Update $T'_{t=t_v}$ {*update the set of assignments according to the completion time.*}
14: $count = 0$ {*a possible number of tasks completed next after task v*}
15: for all $(v_i \in V \setminus V_{completed} \cup \{v\}$ {*for other tasks*}
16: Generate $A_{v_i}^{t=d_{v_i}}$ using updated $T'_{t=t_v}$.
17: {*Check to ensure all members of this group can reach task v_i*} do
18: if $\exists C_{v_i} \subseteq A_{v_i}^{t=d_{v_i}}$ where $(w_{v_i} - \sum_{\tau_{t'}^{C \to v}, t' \in \{t_v + travel(l_v, l_{v_i}), \cdots, d_{v_i}\}} u(C_v) \leq 0$ {*agents can still complete task.*} then
19: $count = count + 1$ {*increase the number of tasks that can be completed after v is completed by C_v*}
20: end if
21: end for
22: $degree_v \leftarrow count$
23: end for
24: Assign C_{v^*} to v^* where $v^* = \arg\max_v (degree_v)$ {*use algorithm 2*}
25: $V_{completed} \leftarrow V_{completed} \cup \{v^*\}$
26: Update T_t {*remove allocated agents from available agents, and update T_t on new agents' positions.*}
27: until $V = \emptyset$

7. EMPIRICAL EVALUATION

The worst case guarantee of the CFLA is only $\frac{1}{|V|}$ since it is always able to allocate the best coalitions to a feasible task if there exists one (a simple verification of algorithm 3 proves this). Comparing with our MIP solution reveals that while, at times the CFLA can return high quality solutions, the size of the problems that can be tested (i.e., less than 15 tasks and 5 agents) make the results hard to generalise. Moreover, the MIP solution takes hours if not days compared to the CFLA (which only take seconds) as expected. Therefore, here we evaluate the additional benefit of performing a one-step look-ahead in the CFLA by comparing a common scheduling algorithm, earliest-deadline-first (EDF) [11], that allocates ECF coalitions to the tasks that have the earliest deadlines. Our hypothesis is that, by exploiting the look-ahead, the CFLA should be more efficient (i.e., complete more tasks) and more effective (i.e., completing them faster) than the EDF approach. To this end, here we present an average case study by making no assumption about the structure of the problem and assigning random workloads and deadlines. This is important because the performance of the heuristics significantly depends on a number of parameters including, but not limited to: (i) the number and position of agents and tasks, (ii) the demand (workload and deadline) of each task and (iii) the coalition values.

7.1 Experimental Setup

Both the CFLA and EDF were implemented in Java. In our experiments we draw coalition values from a uniform distribution as $u(C) = k \times |C|$ where $k \sim U(1, 2)$ (to simulate super-additive coalitions). Moreover, for task deadlines and workloads, we generated these values from a uniform distribution over the ranges of the deadlines as $d_v \sim U(5, 600)$ and $w_v \sim U(10, 50)$, respectively (to

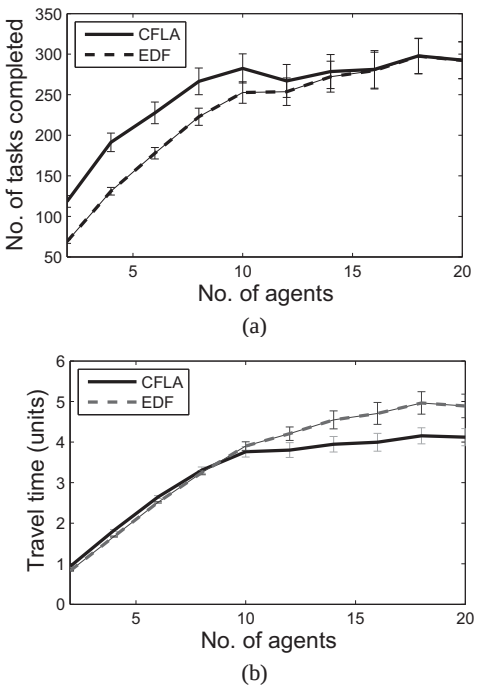

Figure 1: Time taken to visit all tasks.

simulate wide ranging scenarios). The number of tasks was fixed at 300 and the number of agents was varied from 2 to 20 (since the number of coalitions is $2^{|A|} - 1$ we are limited in the size of the problem we can generate). The positions of the agents were allocated randomly on a 50 by 50 grid where the travel time between two points was computed as the Manhattan distance between the points (meaning maximum travel time of 100 time units). With these settings in place, the problem cannot be solved by each agent performing tasks on their own and hence the selection of good coalitions can guarantee a large number of completed tasks. We measured the percentage of completed tasks, the travel time (i.e., number of time steps), and computation time required to complete them. We ran 100 randomly generated instances of the above setup for each number of agents and we report in Figure 1 the mean performance and plot the 95% confidence interval (as error bars) for each instance for the percentage of completed tasks and computation time, respectively. We next describe the results.

7.2 Results

The CFLA is found to complete more tasks than the EDF (up to 31% in the best case) particularly when the number of agents in the system is low (see figure 1(a)). Also, as the number of agents increases, it becomes more apparent that CFLA becomes more efficient (by up to 16.5% for 20 agents) than EDF in visiting tasks (see figure 1(b)) even though they complete nearly the same number of tasks (see figure 1(a)). These results are statistically significant given that the error bars do not overlap and hence the results confirm our hypothesis. However, the computation time for CFLA (4.5s for 20 agents) grows exponentially faster than that of EDF (1.5s for 20 agents) due to the look-ahead procedure. Finally, the results show that in non-trivial scenarios CFLA is 85% efficient with only 10 agents and 97% efficient with 20 agents.

8. CONCLUSIONS

In this paper we have introduced a novel coordination problem: the CFSTP, which can be used to capture many real world applications including the coordination of emergency responders in major disasters, and is relevant to other application domains, such as surveillance and patrolling of wide geographical area or multi-robot exploration of hostile environments. We have shown that CFSTP is a combinatorial optimisation problem that generalises many other hard combinatorial problems and is NP-Hard. Given this, we provided the first benchmark optimal algorithm for CFSTP. Given that the latter is only appropriate for small and non-urgent problems, we devise the CFLA algorithm which fast, anytime, and implements a look-ahead heuristic that can return good solutions in reasonable time (less than 5s for 20 agents and 300 tasks). Hence, we show that, on average, the CFLA can find allocations (in less that 5 seconds) that complete up to 97% of the tasks given a ratio of 1:15 of agents to tasks.

Future work will look at developing anytime optimal algorithms to solve the CFSTP. In particular, we aim to build decentralised solutions to dynamic versions (where the number of tasks and agents may not be known in advance) of the CSFTP based on our preliminary work in [12]. We also aim to gather better datasets from realistic applications such as emergency response, in order to formulate benchmarks for the CFSTP.

Acknowledgments

This research was undertaken as part of the ALADDIN (Autonomous Learning Agents for Decentralised Data and Information Systems) project and is jointly funded by a BAE systems and EPSRC (Engineering and Physical Sciences Research Council) strategic partnership (EP/C548051/1). We also kindly thank the anonymous reviewers for their very useful comments.

9. REFERENCES

[1] I.-M. Chao, B. L. Golden, and E. A. Wasil. The team orienteering problem. *European Journal of Operational Research*, 88:464–474, 1996.

[2] K. Daniel and S. Koenig. Fast winner determination for agent coordination with SBB auctions. In *Proc. 8th Int. Joint Conf. on Autonomous Agents and Multiagent Systems-Volume 2*, pages 1197–1198, 2009.

[3] B. Golden, L. Levy, and R. Vohra. The orienteering problem. *Naval Research Logistics*, 34(3):307–318, 1987.

[4] M. Kantor and M. Rosenwein. The orienteering problem with time windows. *Journal of Operational Research Society*, 43(6):629–635, 1992.

[5] H. Kitano. Robocup rescue: A grand challenge for multi-agent systems. In *Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS)*, pages 5–12, 2000.

[6] M. Koes, I. Nourbakhsh, and K. Sycara. Heterogeneous multirobot coordination with spatial and temporal constraints. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, pages 1292–1297. AAAI Press, June 2005.

[7] M. Lagoudakis, E. Markakis, D. Kempe, P. Keskinocak, A. Kleywegt, S. Koenig, C. Tovey, A. Meyerson, and S. Jain. Auction-based multi-robot routing. In *Proceedings of the International Conference on Robotics: Science and Systems*, pages 343–350, 2005.

[8] H. C. Lau, M. Sim, and K. M. Teo. Vehicle routing problem with time windows and a limited number of vehicles. *European Journal of Operations Research*, 148:559–569, 2003.

[9] M. Pinedo. *Scheduling: theory, algorithms, and systems*. Springer Verlag, 2008.

[10] T. Rahwan and N. R. Jennings. Coalition structure generation: dynamic programming meets anytime optimisation. In *Proceedings of the Twenty Third Conference on Artificial Intelligence*, pages 156–161, 2008.

[11] K. Ramamritham, J. Stankovic, and W. Zhao. Distributed scheduling of tasks with deadlines and resource requirements. *IEEE Transactions on Computers*, 38(8):1110–1123, 1989.

[12] S. D. Ramchurn, A. Farinelli, K. S. Macarthur, M. Polukarov, and N. R. Jennings. Decentralised coordination in RoboCup Rescue. *The Computer Journal*, (to appear), 2010.

[13] P. Scerri, A. Farinelli, S. Okamoto, and M. Tambe. Allocating tasks in extreme teams. In *AAMAS '05: Proc. 4th Intl. Joint Conf. on Autonomous Agents and Multi-Agent Systems*, pages 727–734. ACM, 2005.

[14] P. Toth and D. Vigo, editors. *The vehicle routing problem*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.

[15] T. Tsiligirides. Heuristic methods applied to orienteering. *Journal of the Operations Research Society*, 35(9):797–809, 1984.