SN Computer Science (2021) 2:165
https://doi.org/10.1007/s42979-021-00523-w

ORIGINAL RESEARCH

Anytime and Efficient Multi-agent Coordination for Disaster Response

Luca Capezzuto[1] · **Danesh Tarapore**[1] · **Sarvapali D. Ramchurn**[1]

Received: 22 September 2020 / Accepted: 13 February 2021 / Published online: 23 March 2021

Abstract
The *Coalition Formation with Spatial and Temporal constraints Problem* (CFSTP) is a multi-agent task allocation problem where the tasks are spatially distributed, with deadlines and workloads, and the number of agents is typically much smaller than the number of tasks. To maximise the number of completed tasks, the agents may have to schedule coalitions. The state-of-the-art CFSTP solver, the *Coalition Formation with Look-Ahead* (CFLA) algorithm, has two main limitations. First, its time complexity is exponential with the number of agents. Second, as we show, its look-ahead technique is not effective in real-world scenarios, such as open multi-agent systems, where new tasks can appear at any time. In this work, we study its design and define a variant, called *Coalition Formation with Improved Look-Ahead* (CFLA2), which achieves better performance. Since we cannot eliminate the limitations of CFLA in CFLA2, we also develop a novel algorithm to solve the CFSTP, the first to be simultaneously anytime, efficient and with convergence guarantee, called *Cluster-based Task Scheduling* (CTS). In tests where the look-ahead technique is highly effective, CTS completes up to 30% (resp. 10%) more tasks than CFLA (resp. CFLA2) while being up to 4 orders of magnitude faster. We also propose S-CTS, a simplified but parallel variant of CTS with even lower time complexity. Using scenarios generated by the RoboCup Rescue Simulation, we show that S-CTS is at most 10% less performing than high-performance algorithms such as Binary Max-Sum and DSA, but up to 2 orders of magnitude faster. Our results affirm CTS as the new state-of-the-art algorithm to solve the CFSTP.

Keywords Coalition formation · Spatial and temporal constraints · Anytime · Convergence guarantee · Disaster response · RoboCup rescue simulation

Introduction

Disasters, man-made and natural, can cause severe loss of life, damage to infrastructure and cascading failures in energy systems [1]. In the aftermath of a disaster, first responders have to be deployed to meet the needs of the community. They are responsible for complex tasks such as first aid and infrastructure restoration, which they must perform during periods of high stress and in environments with strict time constraints [5]. During these operations, it is fundamental to act as fast as possible since any delay can lead to further tragedy and destruction.

We focus on a class of disaster response problems that has been characterised by Ramchurn et al. [29] as *Coalition Formation with Spatial and Temporal constraints Problem* (CFSTP). We use the definitions of *coalition* and *coalition formation* given in [14, 29, 32]. Hence, a coalition is a flat and task-oriented organisation of agents, short-lived and disbanded when no longer needed, while coalition formation is a consequence of the emergent behaviour of the system [23]. In the CFSTP, the agents (e.g., ambulances or fire brigades) have to decide which tasks they are going to execute (e.g., save victims or extinguish fires). The decision is influenced by how tasks are located in the disaster area, how much time is needed to reach them, how much work they require (e.g., how large a fire is) and their deadlines (e.g., estimated time left before victims perish). Given these

This article is part of the topical collection "Advances in Multi-Agent Systems Research: EUMAS 2020 Extended Selected Papers" guest edited by Nick Bassiliades and Georgios Chalkiadakis.

✉ Luca Capezzuto
luca.capezzuto@soton.ac.uk

Danesh Tarapore
d.s.tarapore@soton.ac.uk

Sarvapali D. Ramchurn
sdr1@soton.ac.uk

[1] School of Electronics and Computer Science, University of Southampton, Southampton, UK

conditions, and considering that there could be many more tasks than agents, it is necessary that agents cooperate with each other by forming, disbanding and reforming coalitions over time [32]. Coalitions enable agents to complete tasks more efficiently than working individually. Moreover, some tasks may have constraints that could not be satisfied by single agents. For instance, a fire is extinguished faster when multiple fire brigades work on it together. Hence, the objective of the CFSTP is to schedule the right coalitions (e.g., the fastest ambulances and fire trucks with the largest water tanks) to the right tasks (e.g., sites with the most victims and the strongest fires) to ensure that as many tasks as possible are completed.

Our interest is in algorithms that are *anytime* (i.e., which can return partial solutions if they are interrupted before completion), have *theoretical properties* and can solve the CFSTP *efficiently* (i.e, approximation algorithms [25]). The reason is that anytime and approximate solutions are fundamental in real-world domains, where it is necessary to have theoretical guarantees, but it may be computationally not feasible or economically undesirable to produce an optimal solution [42]. In particular, as we said above, the faster the disaster response, the lower the losses incurred. We also assume that the agents are situated in a *open* [13] system, that is, at any time, agents can join in or leave and new tasks can appear.

To date, the most effective way of solving the CFSTP is to reduce it to a *Distributed Constraint Optimisation Problem* (DCOP) [10] and solve it with the Max-Sum algorithm [9]. The variants relevant to our scope are *Fast Max-Sum* (FMS) [29] and *Binary Max-Sum* (BinaryMS) [27]. FMS has a time complexity which is exponential in the number of agents [29, Section 6.1] but it can find optimal solutions when the problem is represented by an acyclic factor graph [10]. On the other hand, BinaryMS can only find approximate solutions, but its time complexity is polynomial in the number of agents. Nonetheless, since both use binary decision variables, they require a pre-processing phase with exponential time to solve CFSTP instances with n-ary decision variables. Multi-agent approaches that solve problems similar to the CFSTP make use of social insects [8], automated negotiation [11, 12, 39] and evolutionary computation [41], but without considering the anytime property. In the iTax taxonomy of Korsah et al. [20], the CFSTP is defined as a *Cross-schedule Dependent Single-Task Multi-Robot Time-extended Assignment* (XD [ST-MR-TA]) problem [20]. To date, the approaches proposed to solve XD [ST-MR-TA] problems utilise linear programming [2, 18, 19], automated negotiation [21] and memetic algorithms [22]. However, either they do not produce anytime solutions [21, 22], or they do not have theoretical properties [2], or they are based on a simpler model [18, 19].

Against this background, we focus on the state-of-the-art algorithm to solve the CFSTP, namely the *Coalition Formation with Look-Ahead* (CFLA) algorithm [30]. Our rationale is that CFLA is anytime and, although its computational time is exponential in the worst case, due to its design [30, Section 6] and to the performance of current computers, a well-engineered implementation can find a solution to problems with dozens of agents and hundreds of tasks in minutes, when it is not necessary to terminate early. Specifically, we advance the state of the art in the following ways:

- We define CFLA2, a novel variant of CFLA that minimises limitations and improves performance.
- Since we cannot eliminate the limitations of CFLA in CFLA2, we design CTS, the first CFSTP solver to be simultaneously anytime, efficient and with convergence guarantee. In tests where the look-ahead technique is highly effective, CTS is up to 4 orders of magnitude faster than CFLA and CFLA2.
- Finally, we propose a simplified, parallel and more efficient variant of CTS, called S-CTS. In problems generated with the RoboCup Rescue Simulation [16], we show that S-CTS can compete with high-performance DCOP algorithms, while being up to 2 orders of magnitude faster.

The rest of the paper is organised as follows. In "Problem Formulation", we give our CFSTP model. "Coalition Formation with Improved Look-Ahead" details CFLA2 and "Cluster-Based Task Scheduling" presents the CTS algorithm. Next, we give comparison tests between CFLA, CFLA2 and CTS, then we show the performance of S-CTS in the RoboCup Rescue Simulation and finally conclude.

Problem Formulation

We present below a refined constraint optimisation model of the CFSTP [30]. More precisely, we extend the definition of coalition value, define the constraints with fewer and simpler equations, and introduce the concept of solution degree.

Basic Definitions

Let $V = \{v_1, \ldots, v_m\}$ be a set of m tasks and $A = \{a_1, \ldots, a_n\}$ be a set of n agents.[1] Let L be the finite set of all possible task and agent locations. Hence, more than one agent or task can be at the same location. Time is denoted by $t \in \mathbb{N}$, starting at $t = 0$, and agents travel or execute tasks with a

[1] Although not necessary, it is typically assumed that $m \gg n$.

base time unit of 1. The time units needed by an agent to travel from its current location to a new task location are given by the function $\rho : A \times L \times L \to \mathbb{N}$. Unlike [30], we put A in the domain of ρ to characterise agents with different speeds.[2] Task locations do not change over time, while agent locations can. Each task v has a *demand* $D_v = (w_v, d_v)$, where $w_v \in \mathbb{R}^+$ is the *workload* of v, or the amount of work required to complete v, and $d_v \in \mathbb{N}$ is the *deadline* of v, or the time until which agents can work on v. Our notion of work will be clear in "Coalition Values". Hence, workloads are positive and some tasks may have a deadline of zero. In other words, a problem may have tasks that cannot be completed in time, independently of the algorithm chosen to solve it. We denote the location of agent a at time t by $l_a^t \in L$, the times at which a starts and finishes working on task v by $s_a^v \in [0, d_v]$ and $f_a^v \in [s_a^v, d_v]$, respectively, and the *latest deadline* by $d_{\max} = \max_{v \in V} d_v$.

Coalition Allocations

Agents are cooperative [38] and can work together to complete a task. A subset of agents $C \subseteq A$ is called a *coalition*. At time t, the rationale for allocating coalition C to task v is that C completes v in the fewest time units. An *agent allocation* is denoted by $\tau_t^{a \to v}$ and represents the fact that agent a works on task v at time t. The *set of all agent allocations* is denoted by

$$T = \left\{ \tau_t^{a \to v} \mid a \in A,\ v \in V,\ t \in [0,\ d_{\max}] \right\} \tag{1}$$

and contains all possible agent allocations. A *coalition allocation* is denoted by $\tau_t^{C \to v}$ and represents the fact that coalition C works on task v at time t. Given a set of agent allocations $T' \subseteq T$ and a time $t' \leq d_{\max}$, the *set of coalition allocations corresponding to* T' over the time period $[0, t']$ is denoted by

$$\Gamma(T', t') = \left\{ \tau_t^{C_v \to v} \mid v \in V,\ C_v = \left\{ a \mid \tau_t^{a \to v} \in T' \right\}, t \leq t' \right\}. \tag{2}$$

Furthermore, the *set of all coalition allocations* is denoted by

$$\Gamma = \Gamma(T, d_{\max}). \tag{3}$$

Similar to T, Γ contains all possible coalition allocations. An agent allocation $\tau_t^{a \to v}$ is also denoted as a *singleton coalition allocation* $\tau_t^{\{a\} \to v}$.

Coalition Values

Each coalition allocation has a *coalition value*, given by the function[3] $u : P(A) \times V \to \mathbb{R}_{\geq 0}$, where $P(A)$ is the power set of A and $\mathbb{R}_{\geq 0}$ is the set of non-negative real numbers. Unlike [30], we put V in the domain of u to characterise the fact that the same coalition may execute different tasks with different performances. Hence, given a coalition allocation $\tau_t^{C \to v}$, the value $u(C, v)$ expresses the amount of work that coalition C does on task v at each time t. The workload w_v decreases linearly over time, depending only on $u(C, v)$. Moreover, $u(C, v)$ is independent of time and the effect of C working on v.

Constraints

There are three constraint types: structural, temporal and spatial. Structural constraints require that each task can be allocated to only one coalition at a time. For each task v, this is characterised by defining the set $\Gamma_v \subseteq \Gamma$, which contains only coalition allocations to v, is maximal with respect to inclusion and such that

$$\tau_t^{C_1 \to v}, \tau_t^{C_2 \to v} \in \Gamma_v \implies C_1 = C_2. \tag{4}$$

Temporal constraints require that each task v can be completed only within its deadline d_v. This is characterised by the function $\Delta : V \times \Gamma \to \{0, 1\}$, defined as follows:

$$\Delta(v, \Gamma) = \begin{cases} 1, & \sum_{t \leq d_v,\ \tau_t^{C \to v} \in \Gamma_v} u(C, v) \geq w_v \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

Equation 5 utilises Γ_v to consider only coalition allocations that satisfy the structural constraints.

Spatial constraints require that an agent will not start working on a task before reaching it:

$$\forall a \in A,\ \forall v \in V, \forall t \leq d_v, \quad s_a^v \geq t + \rho(a, l_a^t, l_v) \tag{6}$$

$$\forall a \in A, \forall v_1, v_2 \in V, \quad f_a^{v_1} + \rho(a, l_{v_1}, l_{v_2}) \leq s_a^{v_2}. \tag{7}$$

Equation 7 also implies that two tasks cannot be allocated to the same agent at the same time. In other words, coalitions that exist at a different location at the same time are disjoint [30, Section 2].

A set of agent allocations $T' \subseteq T$ is called *legal* if there exists $t \leq d_{\max}$ such that, $\forall v \in V$, $\Delta(v, \Gamma(T', t)) = 1$. A set of coalition allocations $\Gamma' \subseteq \Gamma$ is called *feasible* if, $\forall v \in V$, $\Delta(v, \Gamma') = 1$, and Γ' satisfies Eqs. 6 and 7. Consequently, at

[2] In real-world scenarios, this avoids approximating different speeds to the same one.

[3] In cooperative game theory, this is a *characteristic function* [3, Section 2.1].

time t, if $\tau_t^{C_1 \to v_1}$ and $\tau_t^{C_2 \to v_2}$ are feasible coalition allocations and $l_{v_1} \neq l_{v_2}$, then $C_1 \cap C_2 = \emptyset$.

When a task v is allocated to a coalition C, each agent $a \in C$ starts working on v as soon as it reaches the location of v, without waiting for the remaining agents. In other words, there are no synchronisation constraints [24].

Objective Function

The objective function of the CFSTP is to find a feasible set of coalition allocations that maximises the number of completed tasks:

$$\arg \max_{\substack{\Gamma' \subseteq \Gamma, \\ \Gamma' \text{feasible}}} \sum_{v \in V} \Delta(v, \Gamma'). \tag{8}$$

To solve Eq. 8, an exhaustive search may require to verify all the possible agent allocations until $d_{\max}$. Consequently, the time complexity of finding an optimal solution to the CFSTP is $O(|A| \cdot |V|! \cdot (d_{\max})^{|V|})$ [30].

A feasible set of coalition allocations $\Gamma' \subseteq \Gamma$ is called a *solution with degree k* if $\sum_{v \in V} \Delta(v, \Gamma') = k$, with $0 < k \leq |V|$. Moreover, Γ' is called a *partial solution* if $k \leq |V|$ and an *optimal solution*[4] if $k = |V|$. Hence, the argument of the maxima in Eq. 8 is a solution with the highest degree.

Ramchurn et al. [30] proved that the CFSTP is NP-hard [25] and a generalisation of the Team Orienteering Problem [4], which is a generalisation of the Travelling Salesman Problem [37]. As we said in "Introduction", CFLA is the state-of-the-art CFSTP solver. In the next section, we show how to improve it.

Coalition Formation with Improved Look-Ahead

We now present the *Coalition Formation with improved Look-Ahead* (CFLA2), an extension of the CFLA algorithm [30]. More precisely, its look-ahead technique ("Phase 3: Defining the Degree of Each Task") has two modifications that, as we shall see in "Comparison Tests", enhance the overall performance.

The concept of CFLA2 is the same as CFLA, but for completeness, we briefly report it in "The Concept of CFLA2 ". After that, we detail the procedures that compose CFLA2, explaining how they differ from the ones of CFLA. Finally, we list the limitations that CFLA2 continues to keep from CFLA, which are the rationale for our new algorithm in "Cluster-Based Task Scheduling".

Both CFLA and CFLA2 use the same four phases, but Ramchurn et al. [30, Section 6] describe them in three algorithms. To improve the readability, we describe them in four algorithms ("Phase 1: Defining the Legal Agent Allocations" to "Phase 4: Overall Procedure of CFLA2 ").

The Concept of CFLA2

CFLA2 is a centralised, anytime and greedy algorithm that solves Eq. 8 by maximising the working time of the agents and minimising the time required by coalitions to complete tasks. It is divided into four phases:

1. Defining the legal agent allocations ("Phase 1: Defining the Legal Agent Allocations").
2. For each task v, choosing the best coalition C ("Phase 2: Selecting the Best Coalition for Each Task").
3. For each task v, doing a 1-step look-ahead ("Phase 3: Defining the Degree of Each Task") to define its *degree* δ_v, or the number of tasks that can be completed after the completion of v.
4. At each time $t \in [0, d_{\max}]$, allocating a task not yet completed and with the highest degree ("Phase 4: Overall Procedure of CFLA2").

Phase 1: Defining the Legal Agent Allocations

Algorithm 1: getLegalAgentAllocations (Phase 1 of CFLA2)

Input: time t
Output: the set of legal agent allocations at time t

$L_t \leftarrow \emptyset$
for $a \in A^t_{free}$ **do** // for each free agent a
  for $v \in V_{unc}$ **do** // for each uncompleted task v
    if $t + \rho(a, l^t_a, l_v) \leq d_v$ **then** // if a can reach v at t within d_v
      $L_t \leftarrow L_t \cup \{\tau^{a \to v}_{t'}\}_{t+\rho(a,l^t_a,l_v) \leq t' \leq d_v}$

At time t, Algorithm 1 determines which free agents[5] (A^t_{free}) can reach which uncompleted tasks (V_{unc}) before their deadlines. The resulting set of legal agent allocations is denoted by L_t. This phase is identical in CFLA.

Phase 2: Selecting the Best Coalition for Each Task

Algorithm 2: ECF (Phase 2 of CFLA2)

Input: task v, a set of legal agent allocations L_t
Output: ECF coalition C

$A^t_v \leftarrow$ define from L_t the agents that can reach v at t within d_v
$C^*_v \leftarrow \emptyset$ // the ECF coalition
$t^*_v \leftarrow d_v + 1$ // time at which C^*_v completes v
$i \leftarrow 1$
while $i \leq |A^t_v|$ *and* $C^*_v = \emptyset$ **do**
  for $C \in$ *all combinations of* i *agents in* A^t_v **do**
    if $\sum_{\tau^{C' \to v}_{t'} \in \Gamma_v,\ C' \subseteq C,\ t' \in [t, d_v]} u(C', v) \geq w_v$ **then**
      $t_{minmax} \leftarrow \arg\min_{t_{max}} \left(w_v - \sum_{\tau^{C' \to v}_{t'} \in \Gamma_v,\ C' \subseteq C,\ t' \in [t, t_{max}]} u(C', v) \right)$
      if $t_{minmax} < t^*_v$ **then**
        $t^*_v \leftarrow t_{minmax}$
        $C^*_v \leftarrow C$
  $i \leftarrow i + 1$

[4] Optimal solutions may not exist ("Basic Definitions").

[5] That is, agents who neither are travelling to nor working on a task.

Given a task v and a set of legal agent allocations L_t (computed by Algorithm 1), Algorithm 2 returns the *Earliest-Completion-First* (ECF)[6] coalition C_v^* that can be allocated to v [30, Section 6.2]. More precisely, the algorithm minimises both the size of C_v^* and the time at which it completes v. This is achieved by iterating from the smallest to the largest possible coalition size (line 5) and iterating through all possible coalitions of each size (line 6). When the procedure finds a coalition C that can complete v within its deadline d_v (line 7), then $|C|$ is the minimum size of the coalitions that can complete v. Hence, C_v^* is identified among the coalitions that have size $|C|$ (lines 8–11). The summation at lines 7–8 is the workload done by the coalition allocations defined from the asynchronous arrivals ("Constraints") of the agents of C (line 6) to the location of v.

Unlike the original formulation [30, Algorithm 2], Algorithm 2 clarifies that the minimum coalition size has to be determined by iterating through the subsets of the combinations[7] of A_v^t, which is the set of free agents that at time t can reach v within d_v.

Phase 3: Defining the Degree of Each Task

Algorithm 3: lookAhead (Phase 3 of CFLA2)

Input: task v, its ECF coalition C_v^*, the set of all agent allocations T
Output: the degree δ_v of task v

```
1  δ_v ← 0
2  f_v ← time at which C*_v completes v
3  for v_2 ∈ V_unc \ {v} do // for each other uncompleted task
4      if d_{v_2} ≥ d_v then // if v_2 expires after v
5          A^{f_v}_free ← agents that are free at f_v // derived from C*_v and T
6          A^{d_{v_2}} ← select from A^{f_v}_free the agents that can reach v_2 within d_{v_2}
7          i ← 1
8          while i ≤ |A^{d_{v_2}}| do // calculate δ_v
9              for C ∈ all combinations of i agents in A^{d_{v_2}} do
                   // if C can complete v_2
10                 if Σ_{τ_t^{C'→v} ∈ Γ_v, C'⊆C, t∈[f_v,d_{v_2}]} u(C,v) ≥ w_v then
11                     δ_v ← δ_v + 1 + (1 − η_{v_2})
12                     i ← |A^{d_{v_2}}| // break external loop too
13                     break
14             i ← i + 1
```

Given a task v, Algorithm 3 performs a brute-force search to define its degree δ_v ("The Concept of CFLA2 "). At line 8, with a procedure similar to line 5 in Algorithm 2, it checks how many tasks can be completed after the completion of v. Hence, Algorithm 3 assigns a score to each coalition allocation selected by Phase 2 ("Phase 2: Selecting the Best Coalition for Each Task") for each currently uncompleted task. These scores are then used by Phase 4 ("Phase 4: Overall Procedure of CFLA2 ") to choose the next task to execute.

Algorithm 3 differs from the original [30, Algorithm 3] in two points. First, it only considers uncompleted tasks that have a deadline greater or equal to d_v (line 4). This prevents from counting tasks that can be completed before the completion of v. As defined in "The Concept of CFLA2 ", δ_v represents the number of tasks that can be completed only after the completion of v. Second, at line 11, δ_v is not just incremented by 1, but also by $1 - \eta_{v_2}$, where η_{v_2} is the rescaling[8] of w_{v_2} to the range $[w_{\min}, w_{\max}]$, with $w_{\min}$ and $w_{\max}$ being, respectively, the minimum and maximum task workloads:

$$\eta_{v_2} = \frac{w_{v_2} - w_{\min}}{w_{\max} - w_{\min}}.$$

Hence, δ_v is also a measure of how much total workload remains after the completion of v. Maximising δ_v (line 12 of Algorithm 4) leads to the remaining tasks with the smallest workloads, which increases the probability of completing more.

Phase 4: Overall Procedure of CFLA2

Algorithm 4: Overall procedure (Phase 4 of CFLA2)

```
1  t ← 0
2  T ← {τ_t^{a→v}}_{a∈A, v∈V, t∈[0,d_max]} // the set of all agent allocations
3  V_unc ← V // uncompleted tasks
4  repeat
5      δ_max ← 0 // maximum task degree
6      v* ← NIL // next task to allocate
7      C* ← ∅ // coalition to which v* is allocated
8      L_t ← getLegalAgentAllocations(t) // Algorithm 1
9      for v ∈ V_unc do
10         C*_v ← ECF(v, L_t) // Algorithm 2
11         δ_v ← lookAhead(v, C*_v, T) // Algorithm 3
12         if δ_v > δ_max then
13             δ_max ← δ_v
14             C* ← C*_v
15     if v* ≠ NIL and C* ≠ ∅ then
16         Allocate C* to v*
17         V_unc ← V_unc \ {v*}
18         Reduce T according to new agent locations and availability
19     if A^t_free = A then // all agents are free
20         break
21     t ← t + 1
22 until V_unc = ∅ or t > d_max
```

Algorithm 4 shows the overall procedure. The repeat-until loop runs until all tasks are completed, or until the latest deadline is expired (line 22). At each time t, the set of legal agent allocations is updated (line 8) and a task allocation is defined (lines 9–18). If it is not possible to allocate other tasks, the algorithm stops early (line 19).

Analysis and Discussion

Algorithm 1 iterates through all free agents and uncompleted tasks. Assuming that line 4 requires constant time, the time complexity is $\alpha = O(|A| \cdot |V|)$.

Algorithm 2 iterates (line 5) from coalition size 1 to $|A_v^t|$, where A_v^t is the set of agents that can reach task v at time t. This requires $O(|A|)$ time in case $A_v^t = A$. For each

[6] This logic is adapted from the *Earliest-Deadline-First* (EDF) scheduling [33].

[7] To date, the most efficient technique to enumerate all such combinations is the Gray binary code [7, Section 7.2.1.1].

[8] Also known as min–max scaling or min–max normalisation.

$s \leq |A_v^t|$, all possible coalitions of size s could be examined (line 6), which requires $O(2^{|A|})$ time. Assuming that line 8 requires $O(d_{\max})$ time, the total time complexity is $\beta = O(|A| \cdot 2^{|A|} \cdot d_{\max})$.

Algorithm 3 iterates through all uncompleted tasks, which requires $O(|V|)$ time, while line 8 is computationally identical to line 5 in Algorithm 2. Hence, the time complexity is $\gamma = O(|V| \cdot 2^{|A|})$. Since it uses all previous algorithms, Algorithm 4 has a time complexity of:

$$O\big(d_{\max} \cdot (\alpha + |V| \cdot (\beta + \gamma))\big) = O\Big(\big(d_{\max} \cdot |V|\big)^2 \cdot 2^{|A|}\Big). \quad (9)$$

Therefore, despite having a lower complexity than an optimal CFSTP solver ("Objective Function"), CFLA2 has a run-time that increases quadratically with the number of tasks and exponentially with the number of agents, which makes it not suitable for systems with limited computational power or real-time applications. Other limitations are as follows.

1. It can allocate at most one task per time unit [30, Section 7]. More formally, at each time unit, the best-case guarantee of CFLA2 is to find a partial solution with degree $k = 1$.
2. In general, greedily allocating a task with the highest degree now does not ensure to allocate all uncompleted tasks in future. This is particularly relevant in an open system, where there is no certainty of having further uncompleted tasks ("Introduction").
3. The more the tasks can be grouped by degree, the more the look-ahead technique becomes a costly random choice. In other words, at time t, if some tasks $V' \subseteq V$ have all maximum degree, then Algorithm 4 selects v^* randomly from V'. Hence, the larger V' is, the less relevant Algorithm 3 becomes.
4. In Algorithm 4, all tasks have the same weight. That is, tasks with earlier deadlines may not be allocated before tasks with later deadlines. This is independent of the order in which the uncompleted tasks are elaborated (line 9) since the computation of $\delta_{\max}$ (line 12) would not be affected.

To overcome the limitations of CFLA2, in the next section we present CTS, a CFSTP solver that is anytime, efficient and with convergence guarantee, both in closed and open systems.

Cluster-Based Task Scheduling

The *Cluster-based Task Scheduling* (CTS) is an anytime and greedy algorithm that operates at the agent level, rather than at the coalition level. It is divided into the following two phases.

1. For each agent a, defining a task v such that v is the closest to a and d_v is minimal.
2. For each task v, defining the coalition of agents to which v has to be allocated.

Algorithm 5 is used in Phase 1, while Algorithm 6 enacts the two phases. We describe them, respectively, in "Selecting the Best Task for Each Agent" and "Overall Procedure of CTS".

Selecting the Best Task for Each Agent

Algorithm 5: getTaskAllocableToAgent (used in Phase 1 of CTS)

```
Input: time t, agent a
1  v_a^t ← (NIL, NIL) // array in which v_a^t[0] (resp. v_a^t[1]) is the unallocated (resp.
     allocated but still uncompleted) task allocable to agent a at time t
2  t_min ← (d_max + 1, d_max + 1) // array in which t_min[0] (resp. t_min[1]) defines the
     time units required by agent a to reach v_a^t[0] (resp. v_a^t[1])
3  d_min ← (d_max + 1, d_max + 1) // array in which d_min[0] (resp. d_min[1]) is the
     deadline of v_a^t[0] (resp. v_a^t[1])
4  for v ∈ V do // for each uncompleted task
5      i ← 0 // v is unallocated
6      if other agents are travelling to or working on v then
7          i ← 1 // v is allocated but still uncompleted
8      t_arr ← t + ρ(a, l_a^t, l_v)
9      if t_arr ≤ d_v and t_arr < t_min[i] and d_v < d_min[i] then // a can reach v
         within d_v and v is the closest to a
10         v_a^t[i] ← v
11         t_min[i] ← t_arr
12         d_min[i] ← d_v
13 if v_a^t[0] ≠ NIL then // prioritise unallocated tasks
14     return v_a^t[0]
15 return v_a^t[1]
```

Given a time t and an agent a, Algorithm 5 returns the uncompleted task v that is allocable, the most urgent and closest to a. By *allocable* we mean that a can reach v before deadline d_v, while *most urgent* means that v has the earliest deadline. The algorithm prioritises unallocated tasks, that is, it first tries to find a task to which no agents are travelling, and on which no agents are working ($v_a^t[0]$). Otherwise, it returns an already allocated but still uncompleted task such that a can reach it and contribute to its completion ($v_a^t[1]$). This ensures that an agent becomes free only when no other tasks are allocable and uncompleted.

Overall Procedure of CTS

Algorithm 6: Overall procedure of CTS (Phases 1 and 2)

Input: tasks V, agents A, locations L, task demands $\{D_v\}_{v \in V}$
Output: A set of coalition allocations Γ'

$t \leftarrow 0$
$\Gamma' \leftarrow \emptyset$ // the partial solution to return
$V_{allocable} \leftarrow \emptyset$ // allocable tasks
repeat
  for $a \in A$ **do** // Phase 1: satisfy spatial constraints
    if $a \in A^t_{free}$ **then**
      $v \leftarrow$ getTaskAllocableToAgent(t, a) // Algorithm 5
      if $v \neq$ *NIL* **then**
        if $v \notin V_{allocable}$ **then**
          $V_{allocable} \leftarrow V_{allocable} \cup \{v\}$
        $A^t_v \leftarrow A^t_v \cup \{a\}$
    else
      Update a's location
      if *a reached the task v it was assigned to* **then**
        Set a's status to *working on* v
  for $v \in V$ **do** // Phase 2: satisfy temporal constraints
    $C^t_v \leftarrow$ all agents working on v at time t
    if $v \in V_{allocable}$ **then**
      $\Pi^t_v \leftarrow$ list of all agents in A^t_v sorted by arrival time to v
      $C^* \leftarrow \emptyset$
      for $i \leftarrow 1$ **to** $|\Pi^t_v|$ **do**
        $C^* \leftarrow$ first i agents in Π^t_v
        $\lambda_i \leftarrow$ arrival time to v of the i-th agent in Π^t_v
        if $i+1 \leq |\Pi^t_v|$ **then**
          $\lambda_{i+1} \leftarrow$ arrival time to v of the $(i+1)$-th agent in Π^t_v
        else
          $\lambda_{i+1} \leftarrow d_v$
        $\varphi_v \leftarrow \varphi_v + (\lambda_i + \lambda_{i+1}) \cdot u(C^* \cup C^t_v, v)$ // w_v done at λ_{i+1}
        if $(d_v - \lambda_i) \cdot u(C^*, v) \geq w_v - \varphi_v$ **then**
          break // C^* is the minimum coalition to complete v
      $T_v = \bigcup_{a \in C^*} \left\{ \tau^{a \to v}_{\lambda_a} \right\}$ // λ_a is a's arrival time to v
      $\Gamma' \leftarrow \Gamma' \cup \Gamma(T_v, t)$ // add $\Gamma(T_v, t)$ (Section 2.2) to Γ'
      $V_{allocable} \leftarrow V_{allocable} \setminus \{v\}$
    if $C^t_v \neq \emptyset$ **then**
      $w_v \leftarrow w_v - u(C^t_v, v)$
      if $w_v \leq 0$ **then**
        Set free all agents in C^t_v
        $V \leftarrow V \setminus \{v\}$
  $t \leftarrow t + 1$
until $V = \emptyset$ *or* $t > d_{max}$ *or all agents are free*

The overall procedure is described in Algorithm 6. The repeat-until loop is the same as CFLA2, to preserve the anytime property. Phases 1 and 2 are represented, respectively, by the loops at lines 5 and 16.

Phase 1 loops through all agents. Here, an agent a may either be free or reaching a task location. In the first case (line 6), if an uncompleted task v can be allocated to a (lines 7 and 8), then v is flagged as allocable (line 9) and a is added to the set of agents A_v^t to which v could be allocated at time t (line 11). In the second case (line 12), a is travelling to a task v, hence its location is updated (line 13) and, if it reached v, it is set to *working on v* (line 14).

Phase 2 visits each uncompleted task v. If v is allocable (line 18) then it is allocated to the smallest coalition of agents in A_v^t (defined in Phase 1) that can complete it (lines 19–32). In particular, at lines 24–27, φ_v is the amount of workload w_v done by all the coalitions formed after the arrival to v of the first $i-1$ agents in Π_v^t (defined at line 19). After that, if agents are working on v (line 33), its workload w_v is decreased accordingly (line 34). If w_v drops to zero or below, then v is completed (lines 35–37). The algorithm stops (line 39) when all the tasks have been completed, or the latest deadline is expired, or no other tasks are allocable and uncompleted ("Selecting the Best Task for Each Agent").

The spatial constraints (Eqs. 6 and 7) are satisfied by executing Algorithm 5 only on free agents (line 6), while the temporal constraints (Eq. 5) are satisfied by allocating a task v to a coalition C only when C has the minimum size and can complete v within the deadline d_v.

Analysis and Discussion

The approach of CTS transforms the CFSTP from a $1-k$ task allocation to a series of 1–1 task allocations. In other words, instead of allocating each task to a coalition of k agents, we have that coalitions are formed by clustering (i.e., grouping) agents based on the closest and most urgent tasks. This is an eligibility criterion: unlike CFLA2, CTS exploits the distances between agents and tasks and the speeds of agents to reduce the time needed to define coalition allocations. Algorithm 5 runs in $\psi = O(|V|)$ time, assuming that the operation at line 8 has constant time. In Algorithm 6, the time complexity of Phase 1 is $O(|A| \cdot \psi) = O(|A| \cdot |V|)$, while Phase 2 runs in $O(|V| \cdot |A| \log |A|)$ because: in the worst case, $A_v^t = A$ and line 19 sorts A in $\Omega(|A| \cdot \log |A|)$ time using any comparison sort algorithm [6]; the loop at line 21 runs in $O(|A|)$ time. Since the repeat-until loop is executed at most $d_{\max}$ times, the time complexity of Algorithm 6 is

$$O\big(d_{\max} \cdot |V| \cdot |A| \log |A|\big). \tag{10}$$

If both phases are executed in parallel, the time complexity is reduced to:

$$\Omega\big(d_{\max} \cdot (|V| + |A| \log |A|)\big). \tag{11}$$

CTS does not have the limitations of CFLA2 ("Analysis and Discussion") because:

1. It can allocate at least one task per time unit. More formally, at each time unit, if one or more tasks are allocable, CTS finds a partial solution with degree $1 \le k \le |A|$.
2. It runs in polynomial time and does not use a look-ahead technique. Thus, it is efficient and can be used in open systems.

The following theorem is based on the definitions given in "Constraints".

Theorem 1 *CTS is guaranteed to find feasible coalition allocations.*

Proof We prove by induction on time t.

At $t = 0$, Phase 1 of Algorithm 6 selects a task v for each agent a such that v is allocable, the most urgent and closest to a ("Selecting the Best Task for Each Agent"). This implies that the agent allocation $\tau_0^{a \to v}$ is legal ("Constraints"). Then, Phase 2 ("Overall Procedure of CTS") allocates v to a only if it exists a coalition C such that $|C|$ is minimum, $\tau_0^{C \to v}$ is feasible ("Constraints") and $a \in C$.

At $t > 0$, for each agent a, there are two possible cases: a task v has been allocated to a at time $t' < t$, or a is free (i.e., idle). In the first case, a is either reaching or working on v (lines 12–15 in Algorithm 6), hence $\tau_t^{a \to v}$ is legal and $\tau_t^{C \to v}$ is feasible, where $a \in C$. In the second case, a is either at its initial location or at the location of a task on which it finished working at time $t' < t$. Thus, as in the base case, if it exists a coalition C and a task v such that $|C|$ is minimum, $\tau_t^{C \to v}$ is feasible and $a \in C$, then v is allocated to a. □

As shown in the two previous sections, Algorithm 5 iterates exactly once over a finite set of uncompleted tasks, while the repeat-until loop of Algorithm 6 is executed at most $d_{\max}$ times. Hence, a corollary to Theorem 1 is that CTS converges to a partial solution if it exists.

The counterexample given by Limitation 2 in "Analysis and Discussion" does not allow to prove the convergence of CFLA and CFLA2 in general settings. Since no current algorithm that solves the CFSTP is simultaneously anytime, efficient and with convergence guarantee ("Introduction"), CTS is the first of its kind.

Comparison Tests

We implemented CFLA, CFLA2 and CTS in Java,[9] and replicated the experimental setup of [30] because we wanted to evaluate how well CFLA2 and CTS perform in settings where the look-ahead technique is highly effective. For each test configuration, we solved 100 random CFSTP instances and plotted the average and standard deviation of: percentage of completed tasks; agent travel time ("Basic Definitions"); *task completion time*, or the time at which a task has no workload left; *problem completion time*, or the time at which no other tasks can be allocated.

Setup

Let $U(l, u)$ and $U^I(l, u)$ be, respectively, a uniform real distribution and a uniform integer distribution with lower bound l and upper bond u. Our parameters are defined as follows.

- All agents have the same speed.
- The initial agent locations are randomly chosen on a 50 by 50 grid, where the travel time of agent a between two points is given by the Manhattan distance (i.e., the taxicab metric or ℓ_1 norm) divided by the speed of a.
- Tasks are fixed to 300, while agents range from 2 to 40, in intervals of 2 between 2 and 20 agents, and in intervals of 5 between 20 and 40 agents.
- The coalition values are defined as $u(C, v) = |C| \cdot k$, where $k \sim U(1, 2)$. Hence, coalition values depend only on the number of agents involved, and all tasks have the same difficulty.
- Deadlines $d_v \sim U^I(5, 600)$ and workloads $w_v \sim U^I(10, 50)$.

Unlike [30], we set the number of maximum agents to 40 instead of 20, because it allows in this setup to complete all tasks in some instances. We did not perform a comparison on larger instances because of the run-time of CFLA and CFLA2: on commodity hardware, CTS takes seconds to solve instances with thousands of agents and tasks, while CFLA and CFLA2 take days. Consequently, the purpose of this section is to highlight the performance of CTS using CFLA and CFLA2 as a baseline. We aim to verify the scalability of CTS in a future investigation.

Results

In terms of completed tasks (Fig. 1a), the best performing algorithm for instances with up to 18 agents is CFLA2, while the best performing algorithm for instances with at least 20 agents is CTS. CFLA is outperformed by CFLA2 in all instances except those with 2 agents, and by CTS in instances with at least 10 agents. The reason why the performance of CFLA and CFLA2 does not improve significantly starting from instances with 20 agents is that the more agents (with random initial locations) there are, the more the tasks are likely to be grouped by degree.[10] CFLA2 has a trend similar to that of CFLA because it has the same limitations, but it performs better due to its improved look-ahead technique. CTS is not the best in all instances because its average task completion time is the highest (see the discussion on Fig. 1c below). This implies that the fewer the agents, the more the tasks may expire before they can be allocated. In our setup, 10 (resp. 20) is the number of agents starting from which this behaviour is contained enough to allow CTS to outperform CFLA (resp. CFLA2).

Regarding agent travel times (Fig. 1b), it can be seen that CTS is up to three times more efficient than CFLA and CFLA2. This is due to Algorithm 5, which allocates tasks to agents also based on their proximity. CFLA2 has lower agent travel times than CFLA for the following reason. The degree computation in CFLA2 also considers how much total workload would be left ("Phase 3: Defining the Degree of Each Task"). Higher degrees correspond to lower workloads, and tasks with lower workloads are completed first. Thus, fewer tasks are grouped by degree and more are likely to be completed. This means that the average distance between task locations in a CFLA2 solution may be lower than that of a CFLA solution. The agent travel times increase with all algorithms. This behaviour is also reported, but not explained, by Ramchurn et al. [30]. To explain it, let us consider a toy problem with one agent a_1 and one task v. If we introduce a new agent a_2 such that $\rho(a_2, l^0_{a_2}, l_v) > \rho(a_1, l^0_{a_1}, l_v)$, then the average travel time increases. In our setup, this happens because the initial agent locations are random.

In general, task completion times (Fig. 1c) decrease because the more agents there are, the faster the tasks are completed. The completion of task v is related to the size of the coalition C to which v is allocated: the highest the completion time, the smallest the size of C, hence the highest the working time of the agents in C. Task completion times are inversely related to agent travel times. Since CTS has the smallest agent travel times and allocates tasks to the smallest coalitions, it consequently has the highest task completion times. Therefore, in CTS, agents work the highest amount of times, and the number of tasks attempted at any one time is the largest.

The problem completion times (Fig. 1d) are in line with the task completion times (Fig. 1c) since the faster the tasks are completed, the less time is needed to solve the problem. The reason why the times of CFLA and CFLA2 do not

[9] https://doi.org/10.5281/zenodo.4320671.

[10] See Limitation 3 described in "Analysis and Discussion".

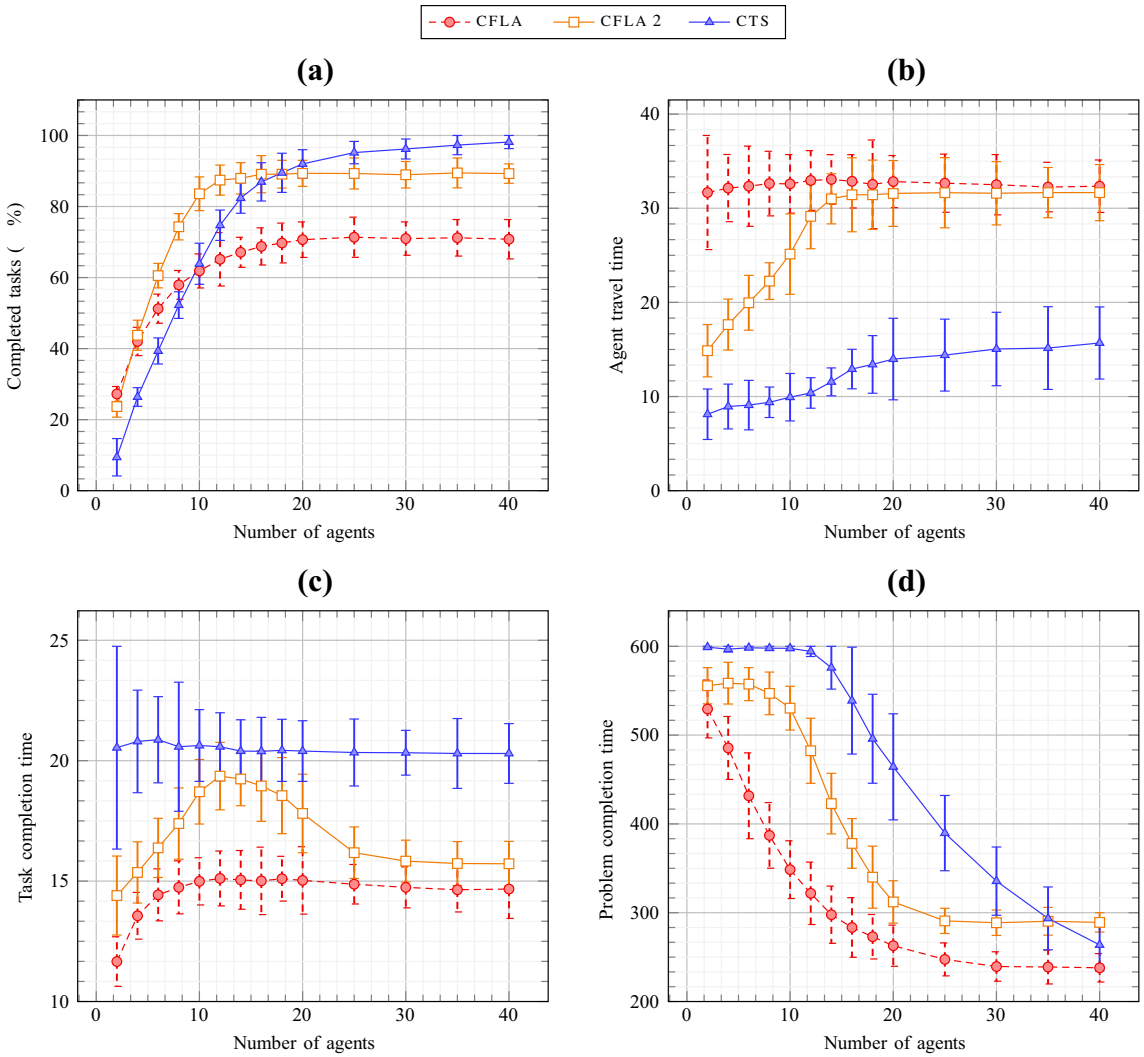

Fig. 1 Comparison of CFLA, CFLA2 and CTS on CFSTP instances with linear coalition values. In each figure, each point is the avg $\pm$ std/2, where avg is the average over 100 problems of the value indicated on the *Y*-axis and std is the standard deviation of avg. The tasks are fixed to 300, while the number of agents is denoted by the *X*-axis

decrease significantly from 20 agents up is linked to their performance (see the discussion on Fig. 1a above). On the other hand, the fact that the times of CTS decrease more consistently than those of CFLA and CFLA2 indicates that CTS is the most efficient asymptotically. In other words, CTS is likely to solve large problems in fewer time units than CFLA and CFLA2.

In terms of computational times, CTS is significantly faster than CFLA and CFLA2. For example, in instances with 40 agents and 300 tasks, on average[11] CTS is $45106\% \pm [2625, 32019]$ (resp. $27160\% \pm [1615, 20980]$) faster than CFLA (resp. CFLA2). The run-time improvement of CFLA2 is due to line 4 of Algorithm 3, due to which the look-ahead technique elaborates fewer tasks.

Tests with the RoboCup Rescue Simulation

In this section, we benchmark a variant of CTS ("Cluster-Based Task Scheduling") against high-performance DCOP solvers with the *RoboCup Rescue Simulation* (RCRS), one of the most important projects promoting multi-agent research on disaster response [16]. By reproducing the aftermath of an earthquake in a city, the RCRS allows verifying coordination approaches that could be enacted by first responders in such situations [15, 31].

[11] On a machine with an Intel Core i5-4690 processor (quad-core 3.5 GHz, no Hyper-Threading) and 8 GB DDR3-1600 RAM.

We conducted the tests with our fork of RMASBench[12] [17], a benchmark platform based on the RCRS. We chose it because it allows comparisons against ready-to-use implementations of BinaryMS ("Introduction") and the *Distributed Stochastic Algorithm* (DSA) [40]. We use them as a baseline because:

- Max-Sum and its variants are widely used and can obtain partial solutions with very high degrees ("Introduction"). In particular, BinaryMS can produce a solution within the time limit enforced by the RCRS[13] and with the same quality as FMS [27].
- Since numerous empirical evaluations have proven its efficacy in many different domains, DSA is a touchstone for testing DCOP and RCRS algorithms [10].

"Simplified CTS" describes how CTS can be adapted for use in the latest RCRS version.[14] The following two sections report our setup and results, respectively.

Simplified CTS

In the current RCRS version, deadlines and workloads are not accessible to agents. Thus, we cannot implement CTS since we can neither verify the spatial constraints in Phase 1 nor can we implement Phase 2 ("Overall Procedure of CTS"). However, the RMASBench allows to obtain the *utility* of a task, which is a quantitative measure that indicates the current importance of a task. Consequently, we implemented a modified Phase 1 in which each agent can independently choose to work on the closest task with the highest utility. We call this variant *Simplified CTS* (S-CTS).

The time complexity of S-CTS is $O(d_{\max} \cdot |V|)$ since the agents do not coordinate with each other and their choice is carried out in parallel. Although S-CTS may seem like a major handicap, we show below that it offers a reasonable trade-off between performance and complexity.

Setup

All tests are based on the Paris map, one of the most used in the RoboCup competition. We kept the default setup [27, Section 6.1] because, according to the authors, it maximises the performance of both BinaryMS and DSA.

In RMASBench, there are police patrols and fire brigades. A police patrol can unblock roads, while a fire brigade can extinguish fires. Having 2 agent types allows studying inter-team coordination aspects. Since this is not in our scope, we did not consider road blockades. As a result, our problems are easier and our baseline is more competitive. Figure 2 gives an example.

Fig. 2 Detail of an example problem on the Paris map. The red dots are fire brigades and the blue lines are their water jets. The colour of the buildings reflects their status: grey means no damage; yellow to red means on fire; blue to purple means that the fire has been extinguished, and black means that the building is burnt. The darker the colour, the greater the damage. On the centre-right is a fire station, to which the fire brigades return to refill

The RCRS is based on *scenarios* [31]. A scenario is a class of problems, whose main parameter is the number of agents. In RMASBench, there are 5 scenarios, respectively, with 15, 21, 27, 33 and 40 fire brigades. Other settings are as follows.

- The agents are homogeneous, that is, they all have the same speed and water tank size.
- There are 3 ignition points and each scenario is replicated 30 times. At each execution, a pseudo-random number generator influences the way the fires spread from ignition points to nearby buildings.
- To get a non-trivial number of fires, the agents are added 25 seconds after the start.
- Each simulation runs for a maximum of 5 minutes, ending earlier if all fires have been extinguished.
- The coalitions are super-additive [3, Section 2.1.2.2]. That is, $u(C, v) = |C|$.
- Deadlines and workloads are randomly generated by the RCRS.

[12] https://doi.org/10.5281/zenodo.4320658.

[13] That is, 1 s per problem time unit.

[14] https://github.com/roborescue/rcrs-server/releases/tag/2020-online-competition.

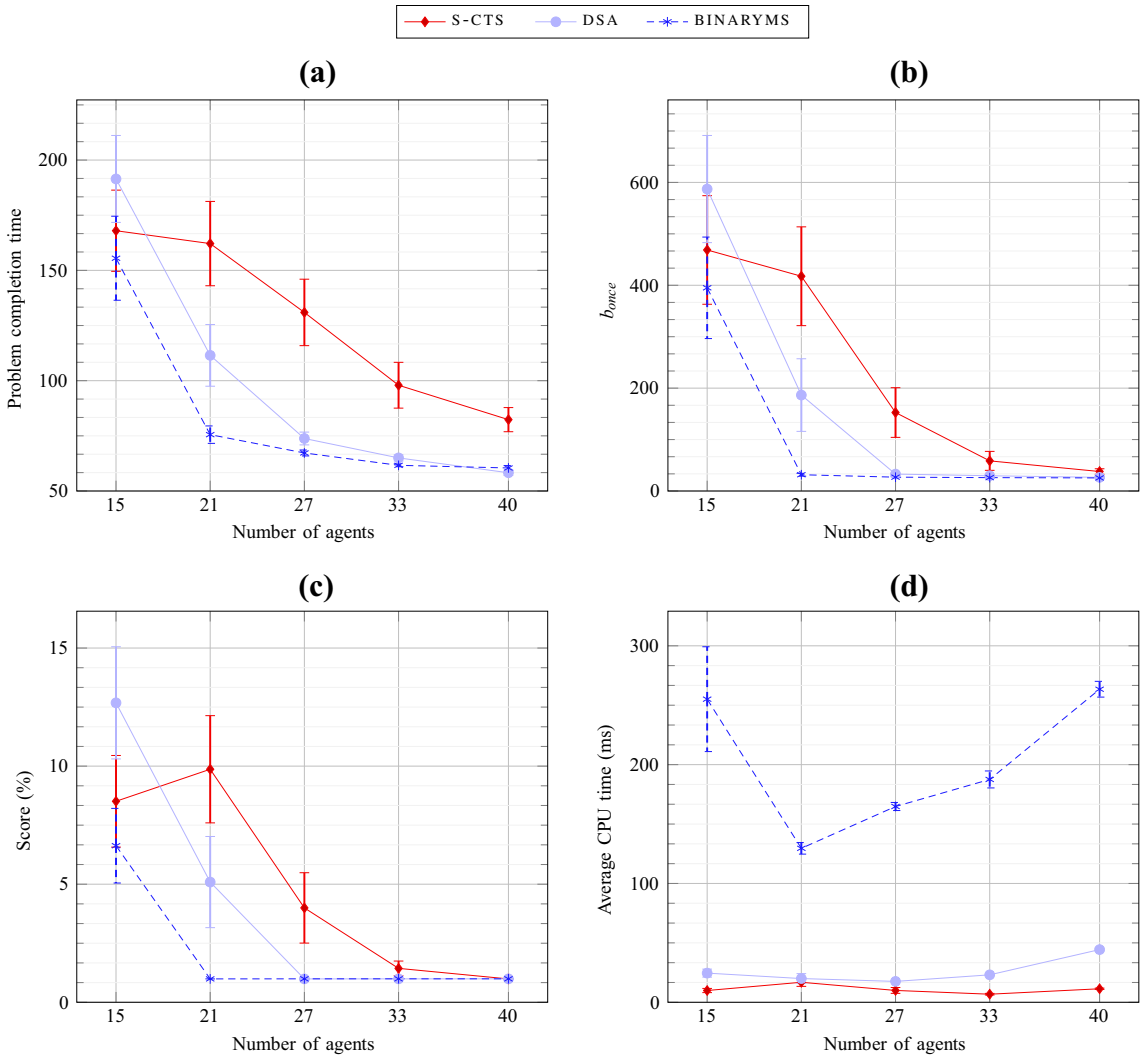

Fig. 3 Performance of S-CTS in RMASBench using DSA and Binary Max-Sum as baselines. In each figure, the *X*-axis defines the number of agents in the scenario, while each point is the avg $\pm$ std/2, where avg is the average over 30 simulations of the value indicated by the *Y*-axis, and std is the standard deviation of avg

For each scenario and algorithm, we plot the average and standard deviation of:

1. Problem completion time ("Comparison Tests").
2. The number of buildings that burned at least once, denoted by b_{once}.
3. *Score*, or the percentage of damage suffered by the city, where 100% means completely burnt. This is the main RCRS metric, defined on the total area of the city buildings and scenario-based parameters.
4. Average CPU time[15] per problem time unit.

We do not consider message-related metrics because S-CTS agents do not communicate ("Simplified CTS").

Results

The more the agents communicate with each other, the better they coordinate. In turn, this leads to lower completion times and numbers of burned buildings. Because there is no exchange of messages in S-CTS and BinaryMS has the highest communication overhead, they are, respectively, the least and the most performing in Fig. 2a, b.

Nevertheless, this does not result in a drastic drop in performance. In Fig. 2c, in the worst-case scenario (i.e., 21 agents), on average S-CTS scores about 10% (resp. 5%) less than BinaryMS (resp. DSA). This is not trivial, given that S-CTS is a simplification and that the scenarios used are

[15] Based on an Intel Xeon E5-2670 processor (octa-core 2.6 GHz with Hyper-Threading).

fine-tuned to maximise the performance of BinaryMS and DSA.

Regarding the average CPU time (Fig. 2d), S-CTS is up to 2 orders of magnitude (resp. 1) faster than BinaryMS (resp. DSA). This is because BinaryMS has a pre-processing phase that requires exponential time ("Introduction") while DSA, despite having a time complexity similar to that of S-CTS [10, Table 4], has a message-passing phase as well.

In Fig. 2a–c, the trends converge to zero because the more agents there are, the less relevant the solver becomes. In other words, the greater the number of agents, the higher the quality of solutions. We can deduce that the degree of agent communication is directly proportional to the score and inversely proportional to the CPU time. However, as we have seen, the performance difference between communication and no communication is not necessarily significant.

Conclusions

In this paper, we proposed two novel algorithms to solve the CFSTP. The first is CFLA2, an improved version of CFLA, and the second is CTS, which is the first to be simultaneously anytime, efficient and with convergence guarantee. CFLA2 can replace CFLA in offline settings or for small problems, while CTS provides a baseline for benchmarks with dynamic and large problems. Moreover, we showed how a simplified but parallel variant of CTS is enough to compete with high-performance solvers (i.e., BinaryMS and DSA) in the RCRS. Because it significantly outperforms CFLA and is more applicable than CFLA2, we can consider CTS to be the new state-of-the-art CFSTP solver. Due to its features ("Analysis and Discussion"), CTS can also be used in contexts that are not necessarily real-time, but can be still captured by the CFSTP model, such as multi-robot area coverage or exploration of environments that are dangerous for humans [30, Section 8].

The limitation of CTS is that it cannot define the quality of its approximation ("Analysis and Discussion"). Moreover, the fact that it maximises the agent working times ("Comparison Tests") implies that some agents may take longer to complete some tasks and therefore may not work on others. Thus, if an optimal solution exists, in general CTS cannot guarantee to obtain it. The CFSTP model also has some limitations, including

- The tasks have all the same weight and have no order. This does not capture scenarios such as search and rescue missions, where some tasks may have higher priority or must be completed before others [20, 24, 28].
- The task workloads are assumed static, when in reality they might be dynamic (e.g., fires that grow in intensity).
- Agent communication is perfect and without costs (i.e., *free comm* environment [26]). Instead, real-world communication channels may fail or have operational constraints, such as low bandwidth or limited network topology (e.g., sparse robot swarms [34]).
- Each agent knows its subproblem a priori (i.e., *total knowledge* or *deterministic environment behaviour* [10, Section 3]). In real-world domains, task states are partially or not known a priori, thus the agents must balance the *exploration* of the environment and the *exploitation* of the acquired information [35, 36].

Consequently, future work aims at

1. Extending CTS to give quality guarantees on the solutions found, and testing its scalability in dynamic benchmarks.
2. Extending the CFSTP model to eliminate the aforementioned limitations and capture more disaster response scenarios.
3. Designing an anytime, optimal and distributed algorithm to solve both the CFSTP and our extension.
4. Investigating efficient inter-team coordination in the RCRS [27]. Specifically, focusing on problems with fires, road blockades and victims trapped under the rubble.

Acknowledgements We thank Mohammad Divband Soorati, Ryan Beal and the anonymous reviewers for their corrections, comments and suggestions. We also thank the EUMAS 2020 Conference Chairs, Nick Bassiliades, Georgios Chalkiadakis and Dave de Jonge, for inviting us to publish in this special issue. Luca Capezzuto acknowledges the use of the IRIDIS High Performance Computing Facility, and associated support services at the University of Southampton, in the completion of this work.

Funding This research is sponsored by the https://www.axa-research.org/en/project/sarvapali-gopal-ramchurn AXA Research Fund. Danesh Tarapore is supported by an EPSRC New Investigator Award grant (EP/R030073/1).

Declarations

Code and Data Availability The source code of the tests reported in "Comparison Tests" to "Tests with the RoboCup Rescue Simulation" and the numerical data used to generate Figs. 1 and 3 are available at the following URLs: https://doi.org/10.5281/zenodo.4320671, https://doi.org/10.5281/zenodo.4320658. https://doi.org/10.5281/zenodo.4320673.

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Alexander ED. Principles of emergency planning and management. Oxford University Press; 2002.
2. Bogner K, Pferschy U, Unterberger R, Zeiner H. Optimised scheduling in human–robot collaboration—a use case in the assembly of printed circuit boards. Int J Prod Res. 2018;56(16):5522–40.
3. Chalkiadakis G, Elkind E, Wooldridge M. Computational aspects of cooperative game theory. Synth Lect Artif Intell Mach Learn. 2011;5(6):1–168.
4. Chao IM, Golden BL, Wasil EA. The team orienteering problem. Eur J Oper Res. 1996;88(3):464–74.
5. Coppola DP. Introduction to international disaster management. Elsevier; 2006.
6. Cormen TH, Leiserson CE, Rivest RL, Stein C. Introduction to algorithms. 3rd ed. MIT Press; 2009.
7. Donald, K.E.: The art of computer programming, volume 4, fascicle 2: generating all tuples and permutations. Pearson Education; 2005.
8. Dos Santos F, Bazzan ALC. Towards efficient multiagent task allocation in the robocup rescue: a biologically-inspired approach. AAMAS. 2011;22(3):465–86.
9. Farinelli A., Rogers A., Petcu A., Jennings N.R.: Decentralised coordination of low-power embedded devices using the max-sum algorithm. In: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - (AAMAS '08). International Foundation for Autonomous Agents and Multiagent Systems, Richland; vol. 2. 2008. p. 639–646.
10. Fioretto F, Pontelli E, Yeoh W. Distributed constraint optimization problems and applications: a survey. JAIR. 2018;61:623–98.
11. Gallud X, Selva D. Agent-based simulation framework and consensus algorithm for observing systems with adaptive modularity. Syst Eng. 2018;21(5):432–54.
12. Godoy J, Gini M. Task allocation for spatially and temporally distributed tasks. In: Proceedings of the 12th international conference on intelligent autonomous systems. Springer, Berlin; 2013. p. 603–12.
13. Hewitt C. The challenge of open systems. Cambridge University Press; 1990. p. 383–95.
14. Horling B, Lesser V. A survey of multi-organizational paradigms. Knowl Eng Rev. 2005;19(4):281–316.
15. Kitano H, Tadokoro S. Robocup rescue: a grand challenge for multiagent and intelligent systems. AI Mag. 2001;22(1):39. https://rescuesim.robocup.org.
16. Kitano H et al., RoboCup Rescue: search and rescue in large-scale disasters as a domain for autonomous agents research, IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.99CH37028), Tokyo, vol.6, 1999. p. 739–43. https://doi.org/10.1109/ICSMC.1999.816643.
17. Kleiner A, Farinelli A, Ramchurn S, Shi B, Maffioletti F, Reffato R. Rmasbench: benchmarking dynamic multi-agent coordination in urban search and rescue. In: Proc. of the 12th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2013), The international foundation for autonomous agents and multiagent systems (IFAAMAS); 2013. p. 1195–6.
18. Koes M, Nourbakhsh I, Sycara K. Heterogeneous multirobot coordination with spatial and temporal constraints. AAAI. 2005;5:1292–7.
19. Korsah GA. Exploring bounded optimal coordination for heterogeneous teams with cross-schedule dependencies. Ph.D. thesis, Carnegie Mellon University; 2011.
20. Korsah GA, Stentz A, Dias MB. A comprehensive taxonomy for multi-robot task allocation. Int J Robot Res. 2013;32(12):1495–512.
21. Krizmancic M, Arbanas B, Petrovic T, Petric F, Bogdan S. Cooperative aerial-ground multi-robot system for automated construction tasks. IEEE Robot Autom Lett. 2020;5(2):798–805. https://doi.org/10.1109/LRA.2020.2965855.
22. Liu C, Kroll A. Memetic algorithms for optimal task allocation in multi-robot systems for inspection problems with cooperative tasks. Soft Comput. 2015;19(3):567–84.
23. Mataric MJ. Designing emergent behaviors: from local interactions to collective intelligence. In: Proceedings of the second international conference on from animals to animats 2: simulation of adaptive behavior: simulation of adaptive behavior. MIT Press, Cambridge; 1993. p. 432–41.
24. Nunes E, Manner M, Mitiche H, Gini M. A taxonomy for task allocation problems with temporal and ordering constraints. Robot Auton Syst. 2017;90:55–70.
25. Computational complexity. Pearson; 1993.
26. Ponda SS, Johnson LB, Geramifard A, How JP. Cooperative mission planning for multi-UAV teams, chap. 60. Springer; 2015. p. 1447–90. https://doi.org/10.1007/978-90-481-9707-1.
27. Pujol-Gonzalez M, Cerquides J, Farinelli A, Meseguer P, Rodriguez-Aguilar JA. Efficient Inter-team task allocation in robocup rescue. In: Proceedings of the 2015 international conference on autonomous agents and multiagent systems (AAMAS '15). International foundation for autonomous agents and multiagent systems; 2015. p. 413–21.
28. Ramamritham K, Stankovic JA, Zhao W. Distributed scheduling of tasks with deadlines and resource requirements. IEEE Trans Comput. 1989;38(8):1110–23.
29. Ramchurn SD, Farinelli A, Macarthur KS, Jennings NR. Decentralized coordination in robocup rescue. Comput J. 2010;53(9):1447–61.
30. Ramchurn SD, Polukarov M, Farinelli A, Truong C, Jennings NR. Coalition formation with spatial and temporal constraints. In: Proceedings of the 9th international conference on autonomous agents and multiagent systems: (AAMAS '10). International Foundation for Autonomous Agents and Multiagent Systems, Richland, vol. 3; 2010. p. 1181–8.
31. RoboCup Rescue Simulator Manual. RoboCup rescue simulation team. Version 1.3. https://rescuesim.robocup.org/resources/documentation. 2020
32. Shehory O, Kraus S. Methods for task allocation via agent coalition formation. Artif Intell. 1998;101(1–2):165–200.
33. Stankovic JA, Spuri M, Ramamritham K, Buttazzo GC. Deadline scheduling for real-time systems: EDF and related algorithms, vol. 460. Springer Science & Business Media; 2013. Reprint of the original 1998 edition.
34. Tarapore D, Groß R, Zauner KP. Sparse robot swarms: moving swarms to real-world applications. Front Robot AI. 2020;7:83. https://doi.org/10.3389/frobt.2020.00083.
35. Taylor ME, Jain M, Jin Y, Yokoo M, Tambe M. When should there be a "Me" in "Team"? distributed multi-agent optimization under uncertainty. In: Proceedings of the 9th international conference on autonomous agents and multiagent systems: (AAMAS '10). International Foundation for Autonomous Agents and Multiagent Systems, Richland; 2010. p. 109–16

36. Taylor ME, Jain M, Tandon P, Yokoo M, Tambe M. Distributed on-line multi-agent optimization under uncertainty: balancing exploration and exploitation. Adv Complex Syst. 2011;14(03):471–528.
37. Tsiligirides T. Heuristic methods applied to orienteering. J Oper Res Soc. 1984;35(9):797–809.
38. Weiss G, editor. Multiagent systems. 2nd ed. MIT Press; 2013.
39. Ye D, Zhang M, Sutanto D. Self-adaptation-based dynamic coalition formation in a distributed agent network: a mechanism and a brief survey. IEEE Trans Parallel Distrib Syst. 2013;24(5):1042–51.
40. Zhang W, Wang G, Xing Z, Wittenburg L. Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. Artif Intell. 2005;161(1–2):55–87.
41. Zhou J, Zhao X, Zhang X, Zhao D, Li H. Task allocation for multi-agent systems based on distributed many-objective evolutionary algorithm and greedy algorithm. IEEE Access. 2020;8:19306–19318. https://doi.org/10.1109/ACCESS.2020.2967061.
42. Zilberstein S. Using anytime algorithms in intelligent systems. AI Mag. 1996;17(3):73.