EasyChair Preprint

№ 3054

Anytime and Efficient Coalition Formation with Spatial and Temporal Constraints

Luca Capezzuto, Danesh Tarapore and Sarvapali D. Ramchurn

April 15, 2020

Anytime and Efficient Coalition Formation with Spatial and Temporal Constraints

Luca Capezzuto[0000−0003−4404−0998], Danesh Tarapore[0000−0002−3226−6861], and Sarvapali D. Ramchurn[0000−0001−9686−4302]

ECS Centre for Machine Intelligence, University of Southampton, UK
{luca.capezzuto,d.s.tarapore,sdr1}@soton.ac.uk

Abstract The *Coalition Formation with Spatial and Temporal constraints Problem* (CFSTP) is a multi-agent task allocation problem where the agents are few and cooperative, the tasks are many, spatially distributed, with deadlines and workloads, and the objective is to find a schedule that maximises the number of completed tasks. The current state-of-the-art CFSTP solver, the *Coalition Formation with Look-Ahead* (CFLA) algorithm, has two main limitations. First, its time complexity is quadratic with the number of tasks and exponential with the number of agents, which makes it not efficient. Second, 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. Motivated by this, we propose an extension of CFLA, which we call *Coalition Formation with Improved Look-Ahead* (CFLA2). Since CFLA2 inherits the limitations of CFLA, we also develop a novel algorithm to solve the CFSTP, the first to be anytime, efficient and approximate, which we call *Cluster-based Coalition Formation* (CCF). We empirically show that, in settings where the look-ahead technique is highly effective, CCF completes up to 20% (resp. 10%) more tasks than CFLA (resp. CFLA2) while being up to four orders of magnitude faster. Our results affirm CCF as the new state-of-the-art algorithm to solve the CFSTP.

Keywords: RoboCup rescue simulation · coalition formation · spatial and temporal constraints · XD [ST-MR-TA] · anytime · efficient

1 Introduction

According to the Global Risks Report 2020 [26], natural disasters and human-made environmental disasters are in the top 5 risks in terms of likelihood and in the top 10 risks in terms of impact. The reason is that they are caused by and are the cause of other crucial issues, such as extreme weather events, biodiversity loss and ecosystem collapse, water and food crises, failure of climate-change mitigation and adaptation, failure of regional or global governance, and profound social instability. Therefore, a key component of modern society is *disaster response* [1], that is, the act of reducing or eliminating the consequences of a disaster [4].

In the field of *Multi-Agent Systems* (MASs), one of the most important projects promoting research on disaster response is the RoboCup rescue simulation [15].

Accepted at EUMAS 2020

By reproducing the aftermath of an earthquake in a city, this simulation allows testing coordination approaches that could be enacted by first responders in such situations. In this work, we are interested in a class of task allocation problems that can be generated by the RoboCup rescue simulation, namely, those in which ambulances have to find and rescue victims trapped under rubble, and fire brigades have to extinguish fires. This class of problems has been characterised by Ramchurn et al. [24] as *Coalition Formation with Spatial and Temporal constraints Problem* (CFSTP)[1]. In the CFSTP, agents (i.e., ambulances or fire brigades) have to decide which sequences of tasks (i.e., victims or fires) they are going to execute (i.e., save or extinguish). Their decision is influenced by how tasks are located in the disaster area, how much time it is required 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 conditions, and considering that there could be many more tasks than agents, it is crucial that agents cooperate with each other by forming coalitions [27] (i.e., grouping together). Hence, the objective of the CFSTP is to schedule the right coalitions (e.g., ambulances with the largest capability) to the right tasks (e.g., sites with the most victims) to ensure that as many tasks as possible are completed.

In this paper, our interest is in algorithms that solve the CFSTP *efficiently* (i.e, that are in the complexity class P [22]) and are *anytime* (i.e., which can return partial solutions if they are interrupted before completion). The reason is that being efficient and anytime is a desirable feature of real-world applications. To date, approaches based on the distributed Max-Sum algorithm [8] have proven to be among the most effective at solving the CFSTP, as well as many other problems [9]. The variants relevant to our scope are *Fast Max-Sum* (FMS) [24], *Bounded Fast Max-Sum* (BFMS) [21], and *Binary Max-Sum* (BinaryMS) [23]. FMS is anytime and provides optimal solutions in exponential time, but it cannot solve general CFSTP instances. This limitation is removed in BFMS, but at the cost of losing the anytime property, and providing only approximate solutions. On the other hand, BinaryMS is efficient, but not anytime, and it requires a pre-processing phase with exponential run-time to solve general CFSTP instances. Other multi-agent approaches make use of social insects [7], automated negotiation [10,12,31] and evolutionary computation [32], but without considering the anytime property. In the field of multi-robot systems, the CFSTP is also known as *Cross-schedule Dependent Single-Task Multi-Robot Time-extended Assignment* (XD [ST-MR-TA]) [18]. To date, the approaches proposed to solve this equivalent formulation utilise linear programming [2,16,17], automated negotiation [19] and memetic algorithms [20]. However, like the above multi-agent approaches that are not based on Max-Sum, they are not anytime.

Against this background, we focus on the current state-of-the-art algorithm that solves the CFSTP, namely, the *Coalition Formation with Look-Ahead* (CFLA)

[1] We use the definitions of *coalition* and *coalition formation* given by [14,25,27]. Hence, a coalition is a flat and task-oriented organisation of agents, short-lived and dissolved when no longer needed, while coalition formation is a consequence of the emergent behaviour [11] of the MAS, rather than inter-agent coordination (as in game theory).

algorithm [25]. Our rationale is that CFLA is anytime and, even though its computational time is exponential in the worst case, thanks to its design [25, Section 6] and the performance of current computers, on average it can solve problems with hundreds of agents and thousands of tasks in minutes.

Our contribution includes: an explanation of CFSTP and CFLA that is clearer, more concise and more detailed than [25]; an extension of CFLA that minimises its limitations; a novel anytime, efficient and approximate algorithm to solve the CFSTP, which outperforms both CFLA and our extension.

The rest of the paper is organised as follows. In Section 2, we formalise the CFSTP model. Section 3 is dedicated to discuss and enhance the CFLA algorithm, which culminates in the CFLA2 algorithm. Given that CFLA2 keeps the core limitations of CFLA, Section 4 presents our novel algorithm. Section 5 reports our empirical evaluation, and Section 6 concludes.

2 The CFSTP model

In this section, we first give our terminology, then characterise coalition allocations and values, and finally give the constraints and objective function of the CFSTP.

2.1 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[2]. Let L_V and L_A be respectively the set of all possible task and agent locations, not necessarily disjoint. Hence, more than one agent or task can be at the same location. Time t is discrete, that is, $t \in \mathbb{N}$, each problem starts at $t = 0$ and agents travel or execute tasks in measurable time units. The time units needed by an agent to travel from one location to another are given by $\rho : A \times (L_A \cup L_V) \times L_V \to \mathbb{N}$. Unlike [25], we put A in the domain of ρ to characterise agents with different speeds[3]. 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 Section 2.3. Hence, workloads can only be positive, and some tasks might have a deadline of zero[4].

We denote the location of agent a at time t by $l_a^t \in L_A \cup L_V$, 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 $d_{max} = \max_{v \in V} d_v$.

2.2 Coalition allocations

Agents are cooperative [30] and can work together to complete a task. Let $Part(A)$ be the set of partitions of A. A subset of agents $C \in Part(A)$ is called a

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

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

[4] In other words, a problem might have tasks that cannot be completed in time, independently of the algorithm chosen to solve it.

coalition. At time t, the rationale for allocating coalition C to task v is that C can complete v in the lowest time possible. 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 $T = \{\tau_t^{a \to v}\}_{a \in A,\, v \in V,\, t \in [0,\, d_{max}]}$ and contains all the combinatorially different 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. The *set of all coalition allocations* is denoted by $\Gamma = \left\{\tau_t^{C \to v} \,|\, C = \{a \mid \tau_t^{a \to v} \in T\}\right\}$. Similar to T, Γ contains all the combinatorially different coalition allocations. Given a set of agent allocations $T' \subseteq T$, the *set of coalition allocations corresponding to* T' is denoted by $\Theta(T') = \left\{\tau_t^{C \to v} \,|\, C = \{a \mid \tau_t^{a \to v} \in T'\}\right\}$. An agent allocation $\tau_t^{a \to v}$ can also be denoted as a *singleton coalition allocation* $\tau_t^{\{a\} \to v}$.

2.3 Coalition values

Each coalition has a *value*, given by the function $u : Part(A) \times V \to \mathbb{R}^+$. Unlike [25], 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 $u(C, v)$.

2.4 Constraints

There are three constraint types: structural, temporal and spatial. Structural constraints require that each task v can be allocated to only one coalition at a time. This is characterised by the following sets:

$$\forall v \in V, \Gamma_v = \left\{\Gamma' \subseteq \Gamma : \tau_t^{C_1 \to v}, \tau_t^{C_2 \to v} \in \Gamma' \implies C_1 = C_2\right\} \tag{1}$$

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:

$$\Delta(v, \Gamma) = \begin{cases} 1, & \text{if } \exists\, t \leq d_v : \sum_{t' \leq t,\, \tau_{t'}^{C \to v} \in \Gamma_v} u(C, v) \geq w_v \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

Equation 2 utilises Γ_v (Equation 1) to count only well-formed coalition allocations.

Spatial constraints require that an agent will not start working on a task before reaching it. This is characterised as follows:

$$\forall a \in A,\, \forall v \in V, \forall t \leq d_v,\; s_a^v \geq t + \rho(a, l_a^t, l_v) \tag{3}$$

$$\forall a \in A, \forall v_1, v_2 \in V,\; f_a^{v_1} + \rho(a, l_{v_1}, l_{v_2}) \leq s_a^{v_2} \tag{4}$$

A set of agent allocations $T' \subseteq T$ such that $\Theta(T')$ satisfies Equation 2 is called *legal*. A set of coalition allocations $\Gamma' \subseteq \Gamma$ that satisfies Equations 2, 3 and 4 is called *feasible*. Consequently, at 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 $C_1 \cap C_2 = \emptyset$.

2.5 Objective function

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

$$\arg\max_{\Gamma' \subseteq \Gamma} \sum_{v \in V} \Delta(v, \Gamma'), \text{ subject to Equations 3 and 4} \tag{5}$$

Since, for each agent a, we might need to consider all the possible task allocations until d_{max}, the time complexity of Equation 5 is $O(|A| \cdot |V|! \cdot (d_{max})^{|V|})$.

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 $0 \leq k \leq |V|$. Moreover, Γ' is called a *partial solution* if $k > 0$ and an *optimal solution*[5] if $k = |V|$. Hence, the argument of the maxima in Equation 5 is a solution with the highest degree.

Ramchurn et al. [25] proved that the CFSTP is NP-hard [22], and a generalisation of the Team Orienteering Problem [3], which is a generalisation of the Travelling Salesman Problem [29]. As we said in Section 1, CFLA is the current state-of-the-art CFSTP solver. In the next section, we show how it can be improved.

3 Coalition Formation with improved Look-Ahead

We now present the *Coalition Formation with improved Look-Ahead* (CFLA2), an extension of the CFLA algorithm [25]. More precisely, its look-ahead phase (Section 3.4) has two modifications that, as we shall see in Section 5, enhance the overall performance.

The concept of CFLA2 is the same as CFLA, but for completeness we briefly report it in Section 3.1. 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 Section 4.

CFLA and CFLA2 have the same four phases, but [25] describes them in three algorithms. For readability purposes, we describe them in four algorithms.

3.1 The concept of CFLA2

CFLA2 is a centralised, anytime and greedy algorithm that approximates Equation 5 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 (Section 3.2).
2. For each task v, choosing the best coalition C (Section 3.3).
3. For each task v, doing a 1-step look-ahead (Section 3.4) 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 (Section 3.5).

We detail them below.

[5] Optimal solutions might not exist (see Footnote 4 in Section 2.1).

Algorithm 1: getLegalAgentAllocations (Phase 1 of CFLA2)

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

1 $L_t \leftarrow \emptyset$
2 **for** $a \in A^t_{free}$ **do** `// for each free agent` a
3 **for** $v \in V_{unc}$ **do** `// for each uncompleted task` v
4 **if** $t + \rho(a, l^t_a, l_v) \leq d_v$ **then** `// if` a `can reach` v `at` t `within` d_v
5 $L_t \leftarrow L_t \cup \{\tau^{a \to v}_{t'}\}_{t+\rho(a,l^t_a,l_v) \leq t' \leq d_v}$

Algorithm 2: ECF (Phase 2 of CFLA2)

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

1 $A^t_v \leftarrow$ define from L_t the agents that can reach v at t within d_v
2 $C^*_v \leftarrow \emptyset$ `// the ECF coalition`
3 $t^*_v \leftarrow d_v + 1$ `// time at which` C^*_v `completes` v
4 $i \leftarrow 1$
5 **while** $i \leq |A^t_v|$ *and* $C^*_v = \emptyset$ **do**
6 **for** $C \in$ *all combinations of* i *agents in* A^t_v **do**
7 **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**
8 $t_{minmax} \leftarrow \min_{t_{max}} \left(w_v - \sum_{\tau^{C \to v}_{t'} \in \Gamma_v,\ C' \subseteq C,\ t' \in [t, t_{max}]} u(C, v) \right)$
9 **if** $t_{minmax} < t^*_v$ **then**
10 $t^*_v \leftarrow t_{minmax}$
11 $C^*_v \leftarrow C$
12 $i \leftarrow i + 1$

3.2 Phase 1: defining the legal agent allocations

At time t, Algorithm 1 determines which free agents[6] (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.

3.3 Phase 2: Selecting the best coalition for each 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)[7] coalition C^*_v that can be allocated to v. 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 the possible coalitions of each size (line 6). When the procedure finds a coalition C

[6] That is, agents who neither are travelling to nor working on a task.
[7] This logic is adapted from the *Earliest-Deadline-First* (EDF) scheduling [28].

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 $\delta_v \leftarrow 0$
2 $f_v \leftarrow$ time at which C_v^* completes v
3 **for** $v_2 \in V_{unc} \setminus \{v\}$ **do**
4 **if** $d_{v_2} \geq d_v$ **then**
5 $A_{free}^{f_v} \leftarrow$ agents that are free at f_v `// derived from` C_v^* `and` T
6 $A^{d_{v_2}} \leftarrow$ select from $A_{free}^{f_v}$ the agents that can reach v_2 within d_{v_2}
7 $i \leftarrow 1$
8 **while** $i \leq |A^{d_{v_2}}|$ **do**
9 **for** $C \in$ *all combinations of* i *agents in* $A^{d_{v_2}}$ **do**
 `// if` C `can complete` v_2
10 **if** $\sum_{\tau_t^{C' \to v} \in \Gamma_v,\ C' \subseteq C,\ t \in [f_v, d_{v_2}]} u(C, v) \geq w_v$ **then**
11 $\delta_v \leftarrow \delta_v + 1 + (1 - \eta_{v_2})$
12 $i \leftarrow |A^{d_{v_2}}|$ `// break external loop too`
13 **break**
14 $i \leftarrow i + 1$

that can complete v within its deadline (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 $8-11$).

Algorithm 2 is more concise than the original formulation [25, Algorithm 2]. In particular, we clarify that the minimum coalition size has to be determined by iterating through the subsets of the combinations[8] of A_v^t, which is the set of free agents that can reach v at time t.

3.4 Phase 3: defining the degree of each task

Given a task v, Algorithm 3 does a 1-step look-ahead[9] to define its degree δ_v (Section 3.1). Similarly to Algorithm 2, it checks how many tasks can be completed after the completion of v (line 8).

Algorithm 3 differs from the original look-ahead phase [25, 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. In fact, as defined in Section 3.1, δ_v represents the number of tasks that can be completed only after the completion of v, not also those that are completed before that. 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 normalisation

[8] To date, the most efficient technique to enumerate all such combinations is the Gray binary code [6, Section 7.2.1.1].
[9] Which can be seen as a brute force phase.

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
```

of w_{v_2} in the interval $[w_{min}, w_{max}]$, with w_{min} and w_{max} being respectively the minimum and maximum task workloads. Hence, δ_v is also a measure of how much total workload is left after the completion of v. When δ_v is maximised (line 12 of Algorithm 4), it leads to the remaining tasks with the smallest workloads, thus increasing the probability of completing more.

3.5 Phase 4: overall procedure of CFLA2

Algorithm 4 shows the overall procedure. It runs in iterations until all tasks are completed or the latest deadline is expired. At each time t, it updates the set of legal agent allocations (line 8). Then, it determines which task to allocate to which coalition (lines $9-18$). If no other tasks can be allocated, the algorithm stops early (line 19). Algorithm 4 can be seen as a myopic approach [24], in which a long-term problem (to allocate all tasks) is divided into a number of short-term problems (to allocate a task with the highest degree at each time t).

3.6 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. For each $s \leq |A_v^t|$, all possible coalitions of size s could be examined (line 6), which requires $O(2^{|A|})$ time in case $A_v^t = A$. 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, and its loop at 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 the previous algorithms, Algorithm 4 has a time complexity of

$$O\left(d_{max} \cdot (\alpha + |V| \cdot (\beta + \gamma))\right) = O\left((d_{max} \cdot |V|)^2 \cdot 2^{|A|}\right) \tag{6}$$

Therefore, despite having a lower complexity than an optimal CFSTP solver (Section 2.5), CFLA2 has a run-time that increases quadratically with the number of tasks and exponentially with the number of agents. This makes the algorithm not efficient, hence not suitable for systems with limited computational resources. Other limitations are as follows:

1. It can allocate only one task per time [25, Section 7]. More formally, at each time, if one or more tasks are allocable, the worst- and 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 necessarily ensure that uncompleted tasks can all be successfully allocated in future. This is particularly relevant in an open MAS[10], where there is no certainty of having further uncompleted tasks.
3. The more the tasks can be grouped by degree, the more the look-ahead phase becomes a costly random choice. In other words, at time t, if some tasks $V_t^{'} \subseteq V$ have all maximum degree, then Algorithm 4 selects v^* randomly from $V_t^{'}$. Hence, the larger $V_t^{'}$ is, the less relevant Algorithm 3 becomes.
4. In Algorithm 4, all tasks have the same weight. That is, tasks with earlier deadlines might not be allocated before tasks with later deadlines. This is independent of the order in which the uncompleted tasks are elaborated (line 9). In fact, the computation of δ_{max} (line 12) would not be affected.

These limitations prevent CFLA2 from scoring higher percentages of completed tasks. Because of them, we decided to develop a new CFSTP solver that is anytime, efficient and approximate. We present it in the next section.

4 Cluster-based Coalition Formation

The *Cluster-based Coalition Formation* (CCF) is a centralised, anytime and greedy algorithm that operates at the agent level, rather than at the coalition level. It is divided into two phases:

[10] Here, we mean open as in *open system* [13]. Therefore, in an open MAS, at any time agents can join in or out, and new tasks can appear.

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

```
   Input: time t, agent a
 1 v_a^t ← (NIL, NIL) // array of indices 0 and 1
 2 t_min ← (d_max + 1, d_max + 1) // like above
 3 d_min ← (d_max + 1, d_max + 1) // like above
 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
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]
```

1. For each agent a, defining the closest and most urgent uncompleted task that can be allocated to a.
2. For each task v, defining the minimum coalition of agents to which v has to be allocated.

We describe Algorithm 5, which is used in the first phase, in Section 4.1 and Algorithm 6, which does the two phases, in Section 4.2.

4.1 Selecting the best task for each agent

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 execution ($v_a^t[1]$). This ensures that an agent becomes free only when no other tasks are allocable and uncompleted.

Algorithm 5 does not enforce constraints on the workloads. As we shall see in Section 4.2, it is Algorithm 6 that does it, by allocating a task v to a coalition C only when C has the minimum size and can complete v within d_v.

4.2 Overall procedure of CCF

The overall procedure is described in Algorithm 6. The repeat-until structure 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.

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

Input: tasks V, agents A, task locations L_V, initial agent locations L_A, 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
    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
    $C^t_v \leftarrow$ 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** Π^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
        $\varphi_v \leftarrow 0$ // amount of w_v done at λ_i
        for $j \leftarrow 1$ **to** $i-1$ **do**
          $C_j \leftarrow$ first j agents in Π^t_v
          $\varphi_v \leftarrow \varphi_v + (\lambda_{j+1} - \lambda_j) \cdot u(C_j \cup C^t_v, v)$
        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
      $\Gamma' \leftarrow \Gamma' \cup \Theta(T_v)$ // add $\Theta(T_v)$ (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*

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 $7-8$), 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 during the arrival to v of the first $i-1$ agents in Π_v^t (defined at line 19). After that, if there are agents 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 (Section 4.1).

4.3 Analysis and discussion

The approach of CCF 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. 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 $O(|V| \cdot |A|^2)$ 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 [5]; the loop at line 21 runs in $O(|A|^2)$ time. Since the `repeat-until` structure is executed at most d_{max} times, the time complexity of Algorithm 6 is $O(d_{max} \cdot |V| \cdot |A|^2)$. CCF does not have the limitations of CFLA2 because:

1. It can allocate at least one task per time. More formally, at each time, if one or more tasks are allocable, CCF guarantees to find a partial solution with degree $1 \leq k \leq |A|$.
2. Each agent is always assigned to the allocable task that is closest and with the earliest deadline.
3. It runs in polynomial time and does not have a look-ahead phase. Thus, it is efficient and can be used in open systems.

Theorem 1. *CCF is correct.*

Proof. We prove by induction on time t.

At $t = 0$, a task v is selected for each agent a such that v is allocable, the most urgent and closest to a (Section 4.1). This implies that the agent allocation $\tau_0^{a \to v}$ is legal (Section 2.4). Then, Phase 2 of Algorithm 6 (Section 4.2) allocates v to a only if it exists a coalition C such that $|C|$ is minimum, $\tau_0^{C \to v}$ is feasible (Section 2.4) 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 we said above, CCF can allocate between 1 and $|A|$ tasks at each time. However, its greedy approach does not allow to define the quality of the partial solution it converges to[11], independently of the problem being solved.

In the current literature, no algorithm that solves the CFSTP is simultaneously anytime, efficient and approximate (Section 1). Consequently, CCF is the first to have such properties.

5 Empirical evaluation

We implemented CFLA, CFLA2 and CCF in Java[12], and replicated the experimental setup of [25] because we wanted to evaluate how well CFLA2 and CCF 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[13]; *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.

5.1 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 between two points is given by the Manhattan distance[14].
- 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 $k \in U(1, 2)$. Hence, coalition values depend only on the number of agents involved, and all tasks have the same difficulty.
- Deadlines $d_v \in U^I(5, 600)$ and workloads $w_v \in U^I(10, 50)$.

Unlike [25], 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.

[11] That is, we cannot characterise the degree of the partial solution obtained by CCF before running Algorithm 6.

[12] https://git.soton.ac.uk/cmi/gopal/cfstp

[13] See Section 2.1.

[14] Also known as taxicab metric or ℓ_1 norm.

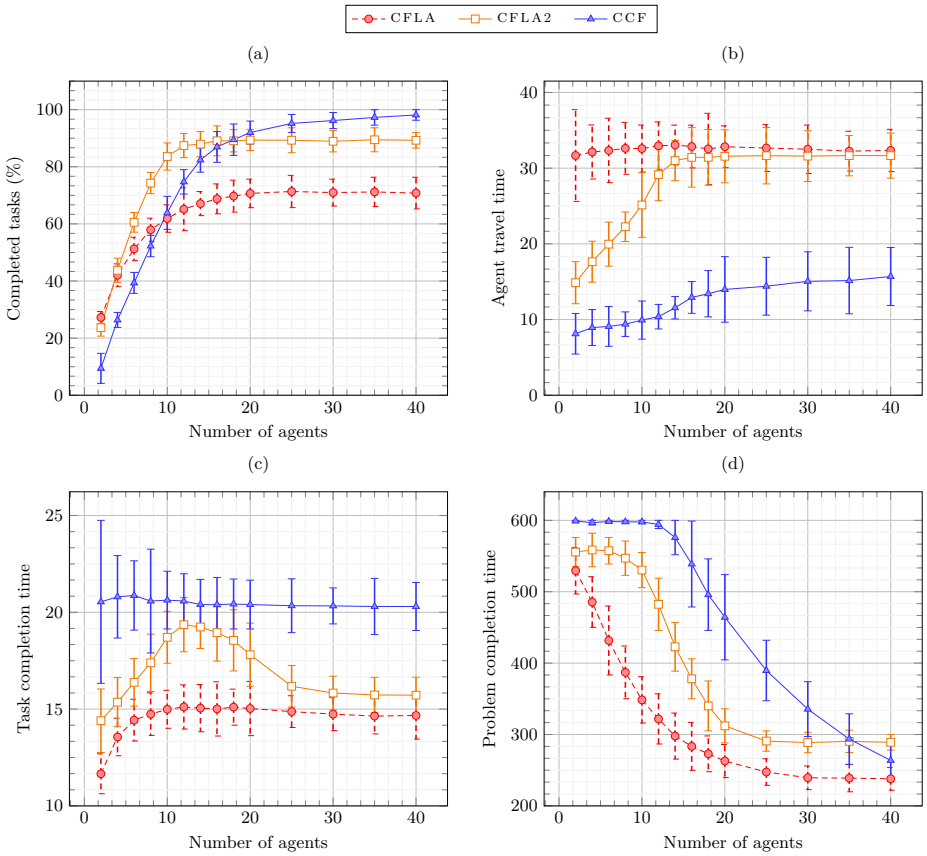

Figure 1. Tests on instances with 300 tasks and up to 40 agents.

5.2 Results

In terms of completed tasks (Figure 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 CCF. CFLA is outperformed by CFLA2 in all instances except those with 2 agents, and by CCF 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 tasks are likely to be grouped by degree[15]. CFLA2 has a similar trend to CFLA because it has the same limitations, but it performs better thanks to its improved look-ahead phase.

Regarding agent travel times (Figure 1b), it can be seen that CCF is up to three times faster than CFLA and CFLA2. This is due to Algorithm 5, which allocates tasks to agents also based on their proximity. To explain why agent travel times

[15] See Limitation 3 described in Section 3.6.

increase with all algorithms[16], 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 scenario, this happens because the initial locations of the agents are random.

In general, task completion times (Figure 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 CCF has the smallest agent travel times and allocates tasks to the smallest coalitions, it consequently has the highest task completion times. Therefore, in CCF, agents work the highest amount of times, and the number of tasks attempted at any one time is the greatest.

The problem completion times (Figure 1d) are in line with the task completion times (Figure 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 decrease significantly from 20 agents up is linked to their performance (see the discussion on Figure 1a above). On the other hand, the fact that the times of CCF decrease more consistently than those of CFLA and CFLA2 indicates that CCF is the most efficient asymptotically. In other words, CCF is likely to solve large-scale problems in fewer time units than CFLA and CFLA2.

In terms of computational times, CCF is significantly faster than CFLA and CFLA2. For example, in instances with 40 agents and 300 tasks, on average[17] CCF 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, thanks to which the look-ahead phase elaborates fewer tasks.

6 Conclusions

In this paper, we proposed two novel approximate algorithms to solve the CFSTP. The first is CFLA2, an improved version of CFLA, and the second is CCF, which is the first to be anytime and efficient. CFLA2 can be used in place of CFLA for small or offline problems, while CCF provides a baseline for benchmarks with large-scale problems. Given that it significantly outperforms CFLA and is more applicable than CFLA2, we can consider CCF to be the new state-of-the-art algorithm to solve the CFSTP.

The limitation of CCF is that it cannot define the quality of of its approximation (Section 4.3). In particular, the fact that it maximises the agent working times (Section 5) implies that some agents may take longer to complete some tasks and therefore might not work on others. Thus, if an optimal solution exists, CCF cannot guarantee to obtain it.

[16] This behaviour is also reported, but not explained, in [25].

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

Future work aims at developing the first anytime and optimal algorithm to solve the CFSTP. We also want to create distributed versions of CCF and our future algorithm, to define a large-scale benchmark from real-world datasets and to test on hard problems generated with the RoboCup rescue simulation.

Acknowledgments

We thank Mohammad Divband Soorati, Ryan Beal and the anonymous reviewers for their helpful comments and suggestions. This research is sponsored by the AXA Research Fund. Danesh Tarapore acknowledges support from a EPSRC New Investigator Award grant (EP/R030073/1).

References

1. Alexander, E.D.: 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. International Journal of Production Research **56**(16), 5522–5540 (2018)
3. Chao, I.M., Golden, B.L., Wasil, E.A.: The team orienteering problem. European Journal of Operational Research **88**(3), 464–474 (1996)
4. Coppola, D.P.: Introduction to International Disaster Management. Elsevier (2006)
5. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT press, 3rd edn. (2009)
6. Donald, K.E.: The Art of Computer Programming, Volume 4, Fascicle 2: Generating All Tuples and Permutations. Pearson Education (2005)
7. Dos Santos, F., Bazzan, A.L.C.: Towards efficient multiagent task allocation in the robocup rescue: a biologically-inspired approach. AAMAS **22**(3), 465–486 (2011)
8. 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 Conference on Autonomous Agents and Multiagent Systems. vol. 2, pp. 639–646 (2008)
9. Fioretto, F., Pontelli, E., Yeoh, W.: Distributed constraint optimization problems and applications: A survey. JAIR **61**, 623–698 (2018)
10. Gallud, X., Selva, D.: Agent-based simulation framework and consensus algorithm for observing systems with adaptive modularity. Syst. Eng. **21**(5), 432–454 (2018)
11. Gell-Mann, M.: The Quark and the Jaguar: Adventures in the Simple and the Complex. Macmillan (1995)
12. Godoy, J., Gini, M.: Task allocation for spatially and temporally distributed tasks. In: Proceedings of the 12th International Conference on Intelligent Autonomous Systems. pp. 603–612. Springer (2013)
13. Hewitt, C.: The challenge of open systems, pp. 383–395. Cambridge University Press (1990)
14. Horling, B., Lesser, V.: A survey of multi-organizational paradigms. The Knowledge Engineering Review **19**(4), 281–316 (2005)
15. Kitano, H., Tadokoro, S., Noda, I., Matsubara, H., Takahashi, T., Shinjou, A., Shimada, S.: Robocup rescue: Search and rescue in large-scale disasters as a domain for autonomous agents research. In: International Conference on Systems, Man, and Cybernetics. vol. 6, pp. 739–743. IEEE (1999)

16. Koes, M., Nourbakhsh, I., Sycara, K.: Constraint optimization coordination architecture for search and rescue robotics. In: Proceedings of International Conference on Robotics and Automation. pp. 3977–3982. IEEE (2006)
17. Korsah, G.A.: Exploring Bounded Optimal Coordination for Heterogeneous Teams with Cross-Schedule Dependencies. Ph.D. thesis, Carnegie Mellon University (2011)
18. Korsah, G.A., Stentz, A., Dias, M.B.: A comprehensive taxonomy for multi-robot task allocation. The International Journal of Robotics Research **32**(12), 1495–1512 (2013)
19. Krizmancic, M., Arbanas, B., Petrovic, T., Petric, F., Bogdan, S.: Cooperative aerial-ground multi-robot system for automated construction tasks. IEEE Robotics and Automation Letters **5**(2) (2020)
20. Liu, C., Kroll, A.: Memetic algorithms for optimal task allocation in multi-robot systems for inspection problems with cooperative tasks. Soft Computing **19**(3), 567–584 (2015)
21. Macarthur, K., Farinelli, A., Ramchurn, S., Jennings, N.R.: Efficient, superstabilizing decentralised optimisation for dynamic task allocation environments. In: International Workshop on Optimization in Multi-Agent systems (OPTMAS). pp. 25–32 (2010)
22. Papadimitriou, C.H.: Computational Complexity. Pearson (1993)
23. Pujol-Gonzalez, M., Cerquides, J., Farinelli, A., Meseguer, P., Rodriguez-Aguilar, J.A.: Efficient inter-team task allocation in robocup rescue. In: Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems. pp. 413–421. International Foundation for Autonomous Agents and Multiagent Systems (2015)
24. Ramchurn, S.D., Farinelli, A., Macarthur, K.S., Jennings, N.R.: Decentralized coordination in robocup rescue. The Computer Journal **53**(9), 1447–1461 (2010)
25. Ramchurn, S.D., Polukarov, M., Farinelli, A., Truong, C., Jennings, N.R.: Coalition formation with spatial and temporal constraints. In: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems. vol. 3, pp. 1181–1188 (2010)
26. Schwab, K.: The global risks report. Tech. rep., World Economic Forum (2020), https://www.weforum.org/reports/the-global-risks-report-2020
27. Shehory, O., Kraus, S.: Methods for task allocation via agent coalition formation. Artificial Intelligence **101**(1-2), 165–200 (1998)
28. Stankovic, J.A., Spuri, M., Ramamritham, K., Buttazzo, G.C.: Deadline scheduling for real-time systems: EDF and related algorithms, vol. 460. Springer Science & Business Media (2013), reprint of the original 1998 edition
29. Tsiligirides, T.: Heuristic methods applied to orienteering. Journal of the Operational Research Society **35**(9), 797–809 (1984)
30. Weiss, G. (ed.): Multiagent systems. MIT press, second edn. (2013)
31. Ye, D., Zhang, M., Sutanto, D.: Self-adaptation-based dynamic coalition formation in a distributed agent network: A mechanism and a brief survey. IEEE Transactions on Parallel and Distributed Systems **24**(5), 1042–1051 (2013)
32. 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)