Human-Centric Intelligent Systems (2023) 3:473–486
https://doi.org/10.1007/s44230-023-00044-0

RESEARCH ARTICLE

Distributed Multi-Agent Hierarchy Construction for Dynamic DCOPs in Mobile Sensor Teams

Brighter Agyemang[1] · **Fenghui Ren**[1] · **Jun Yan**[1]

Received: 8 May 2023 / Accepted: 29 August 2023 / Published online: 12 September 2023

Abstract
Coordinating multiple agents to optimize an objective has several real-world applications. In areas such as disaster rescue, environment monitoring and the like, mobile agents may be deployed to work as a team to achieve a joint goal. Recently, multi-agent problems involving mobile sensor teams have been formalized in the literature as DCOP_MSTs. Under this class of problems, DCOP algorithms are applied to enable agents to coordinate the assignment of their physical locations as they jointly optimize the team objective. In DCOP_MSTs, the environment is dynamic, and agents may leave or join the environment at random times. As a result, a predefined interaction topology or graph may not be useful over the problem horizon. Therefore, there is a need to study methods that could facilitate agent-to-agent interaction in such open and dynamic environments. Existing methods require reconstructing the entire graph upon detecting changes in the environment or assume a predefined interaction graph. In this study, we propose a dynamic multi-agent hierarchy construction algorithm that can be used by DCOP_MST algorithms that require a pseudo-tree for execution. We evaluate our proposed method in a simulated target detection case study to show the effectiveness of the proposed approach in large agent teams.

Keywords DCOP · D-DCOP · Multi-agent systems · Multi-agent coordination · DCOP_MST

Abbreviations

DCOP	Distributed constrained optimization problem
DCOP_MST	Distributed constrained optimization problem for mobile sensor teams
MST	Mobile sensor teams
DynDisMHG	Dynamic distributed multi-agent hierarchy generation
D-DCOP	Dynamic distributed constrained optimization problem
DDFS	Distributed depth first search
Mobed	Multi-agent organization with bounded edit distance
HARP	Hybrid algorithm for reconstructing pseudo-trees
DCSP	Distributed constraint satisfaction problem
CPA	Current partial assignment
DIGCA	Distributed interaction graph construction algorithm
CoCoA	Cooperative constraint approximation
DPOP	Distributed pseudo-tree optimization problem
AMQP	Advanced message queuing protocol
MAS	Multi-agent system
UTM	Unmanned aircraft system traffic management
UAV	Unmanned aerial vehicle

✉ Brighter Agyemang
ba233@uowmail.edu.au

Fenghui Ren
fren@uow.edu.au

Jun Yan
jyan@uow.edu.au

[1] School of Computing and Information Technology, University of Wollongong, 2 Northfields Ave., Wollongong, NSW 2500, Australia

1 Introduction

In many distributed real-world problems, multi-agent methods offer key advantages over centralized solutions due to their ability to scale over time, locality of interactions, leveraging expertise of different agents, and the like. To this end, multi-agent collaboration has been studied from different perspectives in recent years. Some interesting application

domains include power systems, mobile sensing [1], disaster management [2], environment monitoring [3, 4], traffic light management [5], and resource management in microgrids [6].

In the Mobile Sensor Team (MST) domain, agents must collaboratively select their physical positions to monitor specific points of interest (targets). The MST environment typically changes over time. Some of these changes may be external to the agent, such as changes to target coverage requirements, addition or removal of targets over time, and weather condition. Likewise, there can be changes inherent to agents, such as device malfunctioning and disconnection, neighbourhood change due to movement, and adding a new agent.

Due to the peculiarity of the MST domain, Zivan et al. have formalized this class of problems as DCOP_MSTs [7]. Thus, dynamic DCOP algorithms may enable agents to coordinate to select positions in the environment to detect or monitor targets. Most DCOP algorithms require the availability of a multi-agent hierarchy or pseudo-tree (used interchangeably in this study) to execute [8]. Such hierarchies enable paralleling computations that are at different subtrees of the hierarchy. In static DCOP algorithms, pseudo-trees are constructed from a predefined interaction graph.[1] However, for DCOP_MST class of problems, predefined interaction structures may not be helpful since the environment is open and dynamic. Here, an open environment refers to a multi-agent environment where an agent may leave or join the environment at arbitrary times. A dynamic environment refers to an environment that evolves.

In open and dynamic environments, changes to the neighborhood of an agent affect how the optimization algorithm executes. For instance, an agent in an MST problem with a limited communication range may experience frequent neighbor changes as it keeps changing its position. In such a scenario, the agent may be out of range of neighbors in a predefined pseudo-tree and must coordinate with new neighbors in the current DCOP [1]. Likewise, the neighbor set of an agent changes when an agent it shares a constraint with becomes unreachable (e.g. due to power failure). Another real-world manifestation of open and dynamic environments is the Unmanned Aircraft System Traffic Management (UTM) [9, 10]. In such civil air spaces, the number of Unmanned Aerial Vehicles (UAVs) may vary and these UAVs have to address several coordination issues such as trajectory de-confliction with different neighbors as the environment changes. In [11], a DCOP framework is discussed for this trajectory de-confliction problem in the UTM domain. All these scenarios, and the like, contribute to the challenges that affect the coordination of agents when traditional DCOP algorithms are applied to open and dynamic environments.

A basic solution has been to restart the pseudo-tree construction algorithm or interaction graph each time there is a change to enable the application of DCOP algorithms [1, 12, 13]. While guaranteed to re-establish a valid pseudo-tree, such an approach causes unaffected parts of the hierarchy to execute reconstruction procedures needlessly. Due to possibly significant differences between hierarchies, there is also the challenge of reusing information from previous hierarchies in the optimization process.

Therefore, there is a need to address how agents interact in a dynamic environment to facilitate the application of algorithms that require pseudo-trees for execution. In [14], this problem is called the Dynamic Distributed Multi-agent Hierarchy Generation (DynDisMHG) problem. Existing DynDisMHG approaches such as Distributed Depth-First Search (DDFS) [15, 16] and the Multi-agent Organization with Bounded Edit Distance (Mobed) depend on the availability of an interaction graph to operate [12, 17, 18]. In domains where defining the interaction graph beforehand is challenging or impossible, agents must be equipped to generate and maintain the multi-agent hierarchy dynamically. Our focus in this study is to address this gap in the domain.

This study extends our earlier work on the Ad-hoc Distributed Multi-agent Hierarchy generation problem [19] to the MST domain using the DCOP_MST formulation. Specifically, our main contributions are as follows:

1. We propose a dynamic multi-agent hierarchy construction algorithm for open and dynamic environments. Due to the challenges mentioned above, the proposed approach dispenses with the requirement of a predefined interaction graph requirement before a multi-agent hierarchy can be constructed.
2. Using a simulated target-detection application domain, we discuss how our proposal can be used in an MST environment framed as a DCOP_MST.
3. We apply our proposed approach to the DPOP and CoCoA algorithms (which, respectively, belong to the inference-based and local search categories) to demonstrate the feasibility of using our proposal with existing DCOP proposals.

In what follows, we discuss related work in Sect. 2. Section 3 introduces this study's background and problem formulation. Section 4 discusses our proposed approach. Section 5 introduces the experiment setup and then discusses the results. We draw our conclusions in Sect. 6.

[1] The interaction graph discussed in this study equates to a constraint graph in static DCOP literature.

2 Related Work

The constraint or interaction graph is a standard approach to represent a DCOP in the literature. It helps to depict the constraints between nodes and underscore the locality of interactions. The decomposition of the global objective into local constraints makes DCOP suitable to model multi-agent problems where the team objective consists of constraints between agents. Each agent can then be represented as a node in the interaction graph, with an edge indicating a shared constraint between agents.

Several DCOP algorithms in the literature rely on an ordering derived from the interaction graph to execute. In [20–22], the authors apply a DDFS algorithm [15, 16] to construct a pseudo-tree to enable the usage of their respective DCOP algorithms for agents to coordinate their value assignments. The DDFS algorithm depends on a predefined constraint or interaction graph to determine the set of acquaintances of each node for the ordering procedure. Likewise, a decentralized pseudo-tree construction that can exploit the problem structure has been proposed [23]. While the pseudo-tree construction is usually seen as a preprocessing step of the optimization process, given the interaction graph, other proposals also construct the pseudo-tree as part of the optimization process [24, 25].

On the other hand, most real-world multi-agent environments are dynamic and therefore present a challenge when conventional DCOP algorithms are applied. In response to this challenge, researchers have proposed algorithms that solve each change in the environment as a separate DCOP [12, 13]. These methods rely on restarting the pseudo-tree construction algorithm each time a change is detected to enable the re-run of the optimization algorithm on a valid hierarchy. In [1], the authors discussed applying the Max-Sum algorithm to MSTs and relied on restarting an iterative algorithm discussed in [26] to construct a factor graph after each change. With every reconstruction of the pseudo-tree is the challenge of maintaining reusable information from previous solutions. Also, the newly constructed hierarchy could have significant variance from the previous one and limit the ability to reuse search information between time steps. Moreover, if the time scale of changes in the environment is smaller than the time taken for the disrupted hierarchy to stabilize, the optimization process may fail to execute.

Other studies have considered methods that remove the need to address the DynDisMHG problem in mulit-agent systems by adopting non-hierarchy-based graph structures. For instance, in [27], Darko et al. proposed a DCOP framework for traffic incident management. In their work, the authors assumed that an incident response vehicle could communicate with all other vehicles in the network. This removed the need to address the DynDisMHG problem introduced by such a dynamic environment. However, in complex environments with several agents, such a constraint graph structure may not be feasible or expensive to construct. Also, the complete graph structure is unable to exploit the domain structure (e.g. locality of interactions). In [28], the authors extended the CoCoA [29] algorithm to the continuous domain. The proposed C-CoCoA method was compared to several baselines using different constraint graph structures. In their experiments, hierarchy-based (or tree-based) versions of all algorithms discussed in the paper mostly performed better their non-hierarchy-based graph types such as Sparse, Dense, and Scale-Free networks. Thus, using hierarchy-based methods to facilitate multi-agent collaboration can enable optimization algorithms to achieve better performance.

Due to the challenges above and the likes, Sultanik et al. [14] proposed the Multi-agent Organization with Bounded Edit Distance (Mobed) for maintaining multi-agent hierarchies. The proposed algorithm ensures that the difference between hierarchies is minimal. With Mobed, only affected parts of the hierarchy are reconstructed. By attaining minimal edit distance, search information reuse between DCOPs is encouraged in dynamic environments. Mobed requires a different hierarchy construction algorithm (e.g. DDFS) to run as an initial step. Also, Mobed cannot be used in environments where the interaction graph cannot be specified a priori.

In [17], Yeoh et. discussed the significance of information reuse between DCOPs in a dynamic environment. In particular, the authors proposed an algorithm enabling the reuse of contexts when using the ADOPT algorithm. In addition, they discussed the Hybrid Algorithm for Reconstructing Pseudo Trees (HARP). In its execution, HARP is used to detect agents affected in the new DCOP in a distributed manner. These affected agents then execute the DDFS algorithm to reconstruct their local subtree.

Thus, even though HARP and Mobed discuss new techniques for improving the hierarchy construction, HARP still relies on DDFS to operate, albeit at the subtree level. In contrast, Mobed requires a method such as DDFS to construct the first hierarchy. Also, due to the assumption that there is an expected interaction graph in DCOP proposals, [8, 30–32], DDFS-based pseudo-tree construction is typical in the literature. However, in open and dynamic environments, such as those modeled as DCOP_MSTs, it may be challenging or impossible to specify an exhaustive interaction graph beforehand since several random events could influence the environment and agent organization or order. Indeed, Zivan et al. reported that fixed interaction graphs are ineffective

in dynamic environments [7]. To this end, we investigate a multi-agent hierarchy generation problem where a fixed interaction graph cannot be used over the horizon and propose a valid hierarchy generation and maintenance method.

3 Problem Formulation and Background

3.1 Problem Formulation

We extend the formulation in [14] to the domain where no fixed interaction graph is guaranteed to be consistent across the horizon of the Multi-agent System (MAS). Let A_t be an ordered set of agents in a multi-agent system at time t that gives rise to an unordered, labeled, rooted tree. We call the tree $T = \langle A, \pi : A \rightarrow A \rangle$ a multi-agent hierarchy. π is a function that, given an agent $a_j \in A$, specifies the parent of a_j already in the hierarchy. The neighbor set of a_j is denoted as N_j. We denote the children set of a_j as $C_j \subseteq N_j$. The hierarchy is valid if, after adding or removing agent a_i to the hierarchy, it continues to be acyclic after a finite number of steps. Such a hierarchy enables agents in disjoint parts of the tree to execute in parallel.

3.2 Dynamic DCOP

We consider the operation of the ad-hoc multi-agent hierarchy in a dynamic DCOP context. In multi-agent systems formulated as DCOPs, agents assign values from a domain to their decision variables to optimize certain constraint functions. It is assumed that the agents are fully cooperative and can fully observe the environment [8, 28]. Also, the environment is dynamic and deterministic. The DCOP is modeled as a tuple $P = \langle \mathbf{A}, \mathbf{X}, \mathbf{D}, \mathbf{F}, \alpha \rangle$, where:

- $\mathbf{A} = \{a_1, a_2, ..., a_m\}$ is a finite set of agents,
- $\mathbf{X} = \{x_1, x_2, ..., x_n\}$ is a finite set of variables,
- $\mathbf{D} = \{D_1, D_2, ..., D_n\}$ is a set of variable domains such that, the domain of $x_i \in \mathbf{X}$ is D_i,
- $\mathbf{F} = \{f_1, f_2, ..., f_K\}$ is a set of constraint functions defined on $\mathbf{X}$ where each $f_k \in \mathbf{F}$ is defined over a subset $\mathcal{X}_k = \{x_1^k, x_2^k, ..., x_p^k\}$, with $p \leq n$, determines the cost of value assignments of the variables in $\mathcal{X}_k$ as $f_k : D_1 \times D_2 \times ... \times D_p \rightarrow \mathbb{R} \cup \{\perp\}$. Here, the cardinality of $\mathcal{X}_k$ is the arity of f_k. The total cost of the values assigned to variables in $\mathbf{X}$ is $\mathbf{F}_g(\mathbf{X}) = \sum_{k=1}^{K} f_k(\mathcal{X}_k)$,
- $\alpha : \mathbf{X} \rightarrow \mathbf{A}$ is an onto function that assigns the control of a variable $x \in \mathbf{X}$ to an agent $\alpha(x)$.

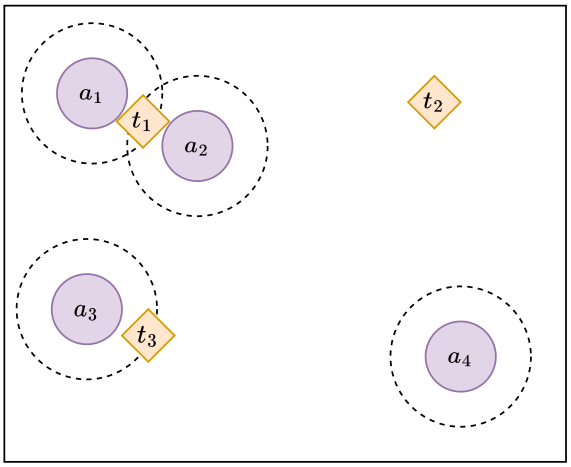


Fig. 1 Mobile sensor team target detection illustration

We assume that α assigns only one variable per agent[2] and the use of binary constraint functions. A Current Partial Assignment (CPA) or partial assignment is the assignment of values to a set of variables $\bar{x}$ such that $\bar{x} \subset \mathbf{X}$. A complete assignment σ is when all variables in $\mathbf{X}$ are assigned a value. A constraint function $f_k \in \mathbf{F}$ is satisfied if $f_k(\sigma_{\mathcal{X}_k}) \neq \perp$. The objective of a DCOP is to find a complete assignment that minimizes the total cost:

$$\sigma^* := \underset{\sigma \in \mathbf{\Sigma}}{argmin}\, \mathbf{F}_g(\sigma) = \underset{\sigma \in \mathbf{\Sigma}}{argmin} \sum_{f_k \in \mathbf{F}} f_k(\sigma_{\mathcal{X}_k}), \tag{1}$$

where $\mathbf{\Sigma}$ is the set of all possible complete assignments.

The Dynamic DCOP (D-DCOP) is an extension of the DCOP formulation to address dynamic multi-agent environments. D-DCOP is modeled as a sequence of DCOPs, $\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_T$. Here, $\mathcal{D}_t = \langle \mathbf{A}^t, \mathbf{X}^t, \mathbf{D}^t, \mathbf{F}^t, \alpha^t \rangle$ where $1 \leq t \leq T$. D-DCOP aims to solve the arising DCOP problem at each time step.

We consider adding an agent, removing an agent, and constraint function modification as events that transition the environment from one DCOP to another. Each agent a_i first has to resolve its local neighbor list and parent–child associations before it can solve the DCOP collaboratively with its identified neighbors and relations. DynDisMHG algorithms (such as the proposal of this study) enable DCOP algorithms that rely on pseudo-trees to operate.

[2] Each variable could be represented as an aggregation of sub-variables where its domain constitutes the cartesian product of the domains of all the sub-variables. We use agent and variable interchangeably since an agent controls only one variable.

3.3 Motivating Domain

In Fig. 1, we illustrate a target detection problem for a Mobile Sensing Team (MST). In this environment, agents (denoted by solid circles) are to collaborate and detect targets in the environment. Thus, the variable of interest in the environment is the agent's position, and its domain is the possible locations it can move into in a single step. In the DCOP_MST formulation, targets (denoted by diamond shapes) may also be mobile in the environment. Here, we assume the targets are static in the environment (e.g. military installations, illegal mining sites, or potential natural disaster sites). As an agent moves in the environment, its domain, neighbour set, constraints, and utilities may be affected. Each target defines the constraint that applies to agents within its region. In Fig. 1, we assume that the sensing range and mobility range are equal (denoted by broken circles). These two properties may be differ in the real-world and does not limit the proposed method. Here, the environment is manifested as a grid world, and the coverage requirement of each target is for an agent to be in the same cell it occupies.

We assume agents have a perfect reputation model[3] and detecting a target with teammates yields a higher score than detecting the target alone. In the environment, each change is represented as a DCOP that the agents have to optimize. Such changes include adding an agent in the environment, removing an agent, and constraint change. As an agent moves in the environment from target to target, it changes its constraint, as defined in the DCOP_MST model. Agents in close proximity construct a multi-agent hierarchy or pseudo-tree and then execute an optimization algorithm to coordinate their value assignments (positions to move into) to detect targets. In Fig. 1, agents a_i, a_2 and a_3 are likely to construct a hierarchy due to their proximity, whereas a_4 will be an isolated agent since it is far from the rest of the team in the given scenario. We refer the reader to [7] for a comprehensive discussion on DCOP_MSTs.

4 Proposed Approach

4.1 Distributed Interaction Graph Construction Algorithm

This section discusses our proposed approach for the multi-agent hierarchy construction problem in open and dynamic environments. We make the following assumptions in this study:

1. Each agent has a globally unique ID that determines its index in the ordered set in 3.1.
2. The agents behave cooperatively to the extent necessary for optimizing the global objective.
3. Agents communicate via message-passing. In this message-passing approach, each message contains the sender's ID and address that the receiver can use to send a response. Other information could be included in the payload when needed.
4. Communication messages are guaranteed to be delivered and in the order sent, despite possible delays.
5. We assume that the agent is able to detect other agents in communication range $\mathcal{U}$ in each time step.

We use a state machine to track the algorithm's execution, manage concurrent procedure calls, and prevent deadlocks.

The pseudocode of the proposed algorithm is presented in Algorithm 1. Our key objective is to enable an agent to dynamically discover and order its neighbors without requiring a predefined interaction graph. The algorithm ensures that an agent seeking connection has a single insertion point under consideration to avoid conflicting edges in the hierarchy. Also, unreachable neighbors of an agent are removed, and where a parent is removed, the agent can reconnect to an existing hierarchy. Since no interaction graph is used, a new agent to be added to the hierarchy does not know of potential neighbors. We address this challenge using a message broadcasting scheme to discover nearby agents. In our discussion, agent a_i (referenced as i) is the agent that executes a procedure asynchronously. Agent a_j (referenced as j) is an agent interacting with i in the environment.

Algorithm 1 is executed by i in every time step (note that each time step represents a new DCOP in DCOP_MST). The agent is set to an inactive state on initialization, and other initial properties are set (lines 1–5). The children set and parent properties ($\pi(i)$ and C_i) are set when the agent starts the algorithm for the first time (lines 6–9).

The Connect function is the primary procedure and is scheduled as a process that regularly executes within a time step (or based on a schedule that may depend on the application domain). When called, it first ensures that the agent is in an inactive state, has an agent that can serve as a parent (see Sect. 4.3), and does not have a parent (line 9). Once the connection conditions are satisfied, the agent broadcasts an Announce message in the environment (lines 11, 12) and waits for a period, condition, or timeout before proceeding (line 13). During this waiting period, an available agent in the environment receives the Announce message and responds by sending an AnnounceResponse message (lines 27–31) if it is a potential parent. Before an agent can send

[3] In a real-world setting, a perfect reputation model may not hold in a DCOP_MST. However, since we focus on the hierarchy construction problem, this does not limit our work but enables us to focus on the study's objective.

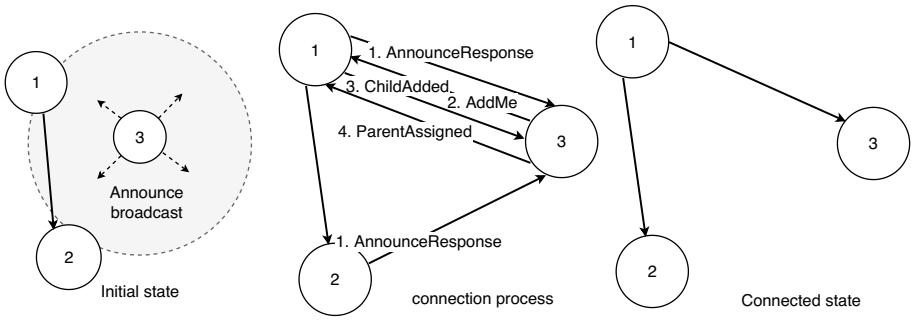


Fig. 2 An illustration of how an agent gets connected using the proposed algorithm. An un-annotated line denotes already established connection and an annotated line depicts a step in the connection process. When Agent 3 wants to connect/re-connect to the current interaction graph, (1) it broadcasts Announce message to agents in range (2) Agents 1 and 2 respond to Agent 3 with AnnounceResponse, (3) Agent 3 selects agent 1 and sends an AddMe message, (4) Agent 1 responds with a ChildAdded message, (5) Agent 3 then sends a ParentAssigned message

an AnnounceResponse, it must be inactive, and a response determinant function $\phi : A \rightarrow \mathbb{B}$ must be TRUE. The function ϕ may be defined based on the application domain (we discuss one such function in Sect. 4.3).

When i receives an AnnounceResponse message, it adds the sending agent to a response list if i is inactive (lines 32–36). After the waiting period (line 13), i selects an agent in the response list using ϑ (line 14). This selection ensures that a single point of insertion is considered for connection. ϑ could be designed to first examine each agent in $\mathcal{L}$ to determine a potential neighbour or uniformly sample from $\mathcal{L}$. Agent i then sends an AddMe message to the selected agent j and goes into an active state while it waits to hear from j (lines 15–17). Since other agents may be expecting AddMe message from i, these agents (except j selected to receive AddMe message) receive AnnounceResponseIgnored message from i (lines 18–20). The response list is cleared before ending the execution of the connect function (line 20). When the condition for broadcasting Announce messages fails but the timeout for the optimization elapses, the D-DCOP algorithm is started by i (lines 23–25).

When i receives an AddMe message from j, it adds j to its children and sends a ChildAdded message to j if i is inactive (lines 37–40). Otherwise, it replies with an AlreadyActive message to j (lines 41–43). When an AlreadyActive message is received by i, it sets its state to inactive (lines 56–58) to enable the next call to the Connect procedure to satisfy the state condition on line 11.

After receiving a ChildAdded message from j, i must be in an active state and without a parent to proceed (line 46). If so, it assigns j as its parent and becomes inactive (lines 47, 48). Agent i then sends a ParentAssigned message to j (line 49). Subsequently, i calls the D-DCOP computation since its neighbour set has changed (line 50). On the other hand, i calls the D-DCOP algorithm when it receives a ParentAssigned message (lines 53, 55) since it has a new child in the hierarchy.

Also, when i receives an AnnounceResponseIgnored message from j, it updates the register for tracking all agents that have sent such messages (lines 59–60). When all other agents that i expect AddMe messages from ignore i's AnnounceResponse messages and i has not assigned a value to its decision variable, it initiates the D-DCOP algorithm (lines 59–61). This scenario may happen when, as i moves in the environment, it comes in contact with another agent j that satisfies ϕ_i but j picks a different agent to send an AddMe instead of i.

While invoking the D-DCOP algorithm for every agent connection is possible, we place a further check to ensure all possible connections within a time step have been established before i executes the optimization algorithm. Therefore, if i comes into contact with j that satisfies ϕ_i and hence assumes that j would be sending an Announce message but j already has a parent, i will never trigger its optimization procedure (if i always waits for all potential connections to be established before optimization begins). The timeout condition on line 23 ensures this deadlock is broken for the agent to call the D-DCOP algorithm.

Algorithm 1 Distributed Interaction Graph Construction Algorithm (DIGCA)

Input: DCOP algorithm, stateTimeout
Output: Selected value v for decision variable of i in current time step

$v \leftarrow null$
$state \leftarrow$ INACTIVE
$\mathcal{L} \leftarrow \emptyset$
$\mathcal{U} \leftarrow$ determine agents in communication range
$\mathcal{B} \leftarrow \emptyset$
if on start up **then**
  $C_i \leftarrow \emptyset$
  $\pi(i) \leftarrow null$
end if
procedure CONNECT(i) ▷ Called by i to find a neighbor to connect to
  if $\pi(i)$ is null and hasPotentialParent($\mathcal{U}$) and $state =$ INACTIVE **then**
    Publish Announce$\{i\}$
    Wait for AnnounceResponse from available agents for a period
    $j \leftarrow \vartheta(\mathcal{L})$ ▷ Select an agent from the list of respondents, if any
    if j is found **then**
      Send AddMe$\{i\}$ to j
      $state \leftarrow$ ACTIVE
      for a in $\mathcal{L}/j$ **do**
        Send AnnounceResponseIgnored$\{i\}$ to a
      end for
    end if
    $\mathcal{L} \leftarrow \emptyset$
  else if execTimeout **then**
    Start D-DCOP algorithm to determine v
  end if
end procedure
procedure RECEIVEANNOUNCE(j) ▷ When i gets an Announce message from j
  if $state$ is INACTIVE and $\phi_i(j)$ **then**
    Send AnnounceResponse$\{i\}$ to j
  end if
end procedure
procedure RECEIVEANNOUNCERESPONSE(j) ▷ When i gets AnnounceResp from j
  if $state$ is INACTIVE **then**
    Add j to $\mathcal{L}$
  end if
end procedure
procedure RECEIVEADDME(j) ▷ When i receives AddMe message from j
  if $state$ is INACTIVE **then**
    Add j to C_i
    Send ChildAdded$\{i\}$ to j
  else
    Send AlreadyActive$\{i\}$ to j
  end if
end procedure

4.2 Agent Connection Maintenance

In Algorithm 1, i can connect to another agent in the environment or the multi-agent hierarchy described in Sect. 3.1. Next, the question we address is how it discovers agents in its neighbour set that are currently unreachable and updates its registers accordingly. In a domain where i can perceive all agents in its neighbourhood at every time step, i can easily remove agents that are no longer in communication range.

Also, agents can use a keep-alive message-passing approach in Algorithm 2 to address this question.

Similar to Algorithm 1, the procedures in Algorithm 2 are executed asynchronously. First, the inspectNeighbors and sendKeepAlive procedures are executed by *i* as two background processes called at regular periods like the Connect procedure of Algorithm 1. Agent *i* maintains a list of neighbours to keep alive (line 1). The sendKeepAlive procedure sends a message to all its neighbours to inform them of its availability when called (lines 2–5). When *i* receives a KeepAlive message, it adds the sender to the keep-alive message list *P* (lines 7–10). When the inspectNeighbors procedure is executed by *i*, it removes any neighbour *j* not found in the keep-alive list (lines 13–21). If *j* (a removed agent) was the parent, *i* goes into an inactive state. This state change is necessary to enable the connect procedure of Algorithm 1 to find a new parent. If a change in neighbourhood is detected, *i* starts the associated D-DCOP computation (lines 27–29).

Since the execution order is a property of the D-DCOP algorithm, *i* will be able to know whether to execute or forward computations to its parent or children. While we focus on D-DCOP computation with the multi-agent hierarchy, other distributed computations that use hierarchies could also be applied. Due to the asynchronous execution environment, there could be a quick succession of computation invocations, and therefore, such an environment may require an abort procedure. Another approach could be to defer the execution of computations till all graph-associated message handling are complete. We adpot the latter in this study.

Algorithm 2 Disconnected Neighbor Removal Algorithm

Input: N_i,DCOP algorithm

1: $P \leftarrow \emptyset$ ▷ keep alive list
2: **procedure** SENDKEEPALIVE(j) ▷ Called by i to inform j of its availability
3: **for** neighbor in N_i **do**
4: Send KeepAlive$\{i\}$ to j
5: **end for**
6: **end procedure**
7: **procedure** RECEIVEKEEPALIVE(j) ▷ When i gets a KeepAlive message from j
8: **if** j not in P **then**
9: Add j to P
10: **end if**
11: **end procedure**
12: **procedure** INSPECTNEIGHBORS ▷ Called regularly by i to remove unreachable neighbors
13: $affected \leftarrow$ False
14: **for** j in N_i **do**
15: **if** j not in P **then**
16: **if** $j == \pi(i)$ **then**
17: $\pi(i) \leftarrow null$
18: $state \leftarrow$ INACTIVE
19: **else**
20: remove j from C_i
21: **end if**
22: clear all information about j in i's registers
23: $affected \leftarrow$ True
24: **end if**
25: **end for**
26: $P \leftarrow \emptyset$
27: **if** $affected$ **then**
28: start DCOP algorithm
29: **end if**
30: **end procedure**

We illustrate how an agent connects to another agent in Fig. 2. This illustration assumes a successful connection process on the first try for didactic purposes. However, we note that an agent may issue multiple connect calls in the environment, and the frequency of such calls is application dependent.

4.3 Response Determinant Heuristic

Notice that using ϑ ensures that i considers only a single insertion point during the connection process. Nonetheless, since j could also be broadcasting Announce messages, there is a need for a mechanism that avoids cyclic connections in the hierarchy. To this end, the result of $\phi_i : A \rightarrow \mathbb{B}$ must be consistent and independent of the problem horizon. We define an instance of ϕ in equation 2 based on the ordered set of agents mentioned in Sect. 3.1.

$$\phi_i(j) = \begin{cases} \text{TRUE} & \text{if } i < j, \\ \text{FALSE} & \text{Otherwise} \end{cases} \tag{2}$$

This definition has the intuition that the agent with the lowest index will be the root. In other words, given two agents i and j, the agent with the lower order is a potential parent to the agent with the higher order (and vice-versa). We consider two extreme cases in the generation of the multi-agent hierarchy. The first is a situation where the hierarchy tends towards a chain. Such a scenario may be helpful for synchronized computations but undesirable when parallelism is desired. The second is when all agents are connected to one root (tree of depth 1). While this encourages parallelism, removing the root causes all other agents to be affected and the need to invoke the connection algorithm. To balance these extremes, we use a max-degree heuristic to limit the possible neighbours and define ϑ to weight agents with a single or no child higher when selecting from the response list.

5 Experiments and Results

In this section we discus the experiments and analyze the results.

5.1 Setting

We conducted our experiments based on the motivating domain in Sect. 3.3. We implemented the simulation environment as a grid world where a cell can contain agents and targets. In our experiments, we used a 5-by-5 grid environment whose horizon is composed of discrete time steps. The environment waits for a complete assignment in each time step before transitioning to a new time step. An agent may be added to a cell randomly (add event) or removed from the environment (remove event). Existing agents, however, can perform a single move operation in the step. Legitimate actions of an agent in a cell include directions of movement that can lead the agent to an adjacent cell (i.e. maximum of 8 actions). Thus, the list of possible actions may change as the agent moves from cell to cell. For instance, an agent in cell 1–1 can only move to the right, down, and right-down (or bottom right) cells, whereas an agent in cell 2–2 can move to the left, right, up, down, left-up, right-up, left-down, and right-down cells. In each time step, these legitimate actions constitute the domain from which an agent can select a value (i.e. the domain is dynamic) using the optimization process. As mentioned, the goal is to detect static targets randomly placed in the environment. Each target defines a constraint where detection by multiple agents yields higher rewards than detection by a single agent.

Aside from the proposed algorithm, we based on [15] on implementing a dynamic DDFS algorithm as a baseline in our experiments. Since DDFS assumes a predefined interaction graph, we assume that the agents within communication range in a time step form a fully connected graph. The agents use this information to execute the DDFS algorithm. We used Cooperative Constraint Approximation (CoCoA) [29] and Distributed Pseudo-tree Optimization Problem (DPOP) as the DCOP algorithms. These DCOP algorithms only run when all possible connections are established or the hierarchy construction algorithm timeouts as explained in Sect. 4.

Regarding the execution environment, we conducted the experiments on a 16-gigabyte RAM computer using an Intel i7-6700 CPU and running Ubuntu 20.04.5 LTS. The simulation system had five main components: an event generation function that randomly generates agent addition and removal events to be executed as DCOPs in the environment, the grid world environment that runs generated events and receives agent registration when the agent is first added (referred to as AgentRegistration message), a runner component that receives agent addition and removal messages from the simulation environment and starts/remove agents, and agent component. Agents were implemented as threads, and each agent ran an instance of a hierarchy construction algorithm (DIGCA or DDFS) and DCOP algorithm (CoCoA or DPOP). We allowed each active agent in the environment to execute the Connect function as often as possible in each time step. The fifth component of the simulation setup is the communication layer. We used the RabbitMQ Advanced Message Queuing Protocol (AMQP) message broker as the communication layer. We also developed an auxiliary graph visualization component to monitor the hierarchy generation during the simulation. The source codes of our experiments can be found at https://github.com/bbrighttaer/ddcop-dynagraph.

Table 1 Main message types used in our experiments

Code	Message Type	Algorithm	Description
AR	Agent Registration	-	Sent by a new agent to the simulation environment to indicate its presence
AM	Announce Message	DIGCA	Broadcasted by an agent that wants to find a parent to connect to
ARM	Announce Response Message	DIGCA	Sent by an agent that receives an AM from a potential child agent
AIM	Announce Response Ignored Message	DIGCA	Sent by an agent to another agent that was not selected to receive an ADM
ADM	Add-me Message	DIGCA	Sent by an agent to another agent that as been selected to be the sending agent's parent
CAM	Child Added Message	DIGCA	Sent by a parent agent to a child agent to confirm addition as a child
PAM	Parent Assigned Message	DIGCA	Sent by a child agent to its parent to confirm assignment as a parent
AAM	Already Active Message	DIGCA	Sent by an agent that is unable to respond to an ADM
USM	Update State Message	CoCoA	Sent to neighbor agents to inform them about the sending agent's status/state
IM	Inquiry Message	CoCoA	Sent by an agent to its neighbors to solicit CMs, as part of the local search procedure
CM	Cost Message	CoCoA	Sent by an agent that as received an IM to the sender of the IM
ER	Execution Request Message	CoCoA	Due to dynamic environment, we introduced this message type to enable an agent that is not the root to request the start of the optimization from its parent
DVM	DDFS Value Message	DDFS	Sent to an agent that can either be parent or a pseudo-parent
DPM	DDFS Position Message	DDFS	Sent to an agent that can either be a child or pseudo-child
DCM	DDFS Child Message	DDFS	Sent by a child agent to its parent to indicate establishment of a parent–child connection
DPCM	DDFS Pseudo-Child Message	DDFS	Sent to an agent that has been selected as a pseudo-parent
DND	DDFS Neighbor Data	DDFS	In the dynamic DDFS implementation, we use this message type to help agents in close proximity to share degree information among themselves to enable the use of the max-degree heuristic
UM	Util Message	DPOP	Sent by a child agent to a parent to report utility values in the child agent's subtree
VM	Value Message	DPOP	Sent by a parent agent to a child agent to report the value selected by the parent agent

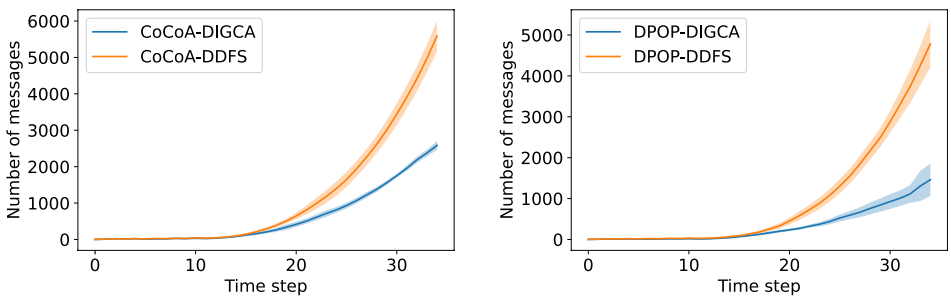


Fig. 3 Number of messages exchanged in the environment

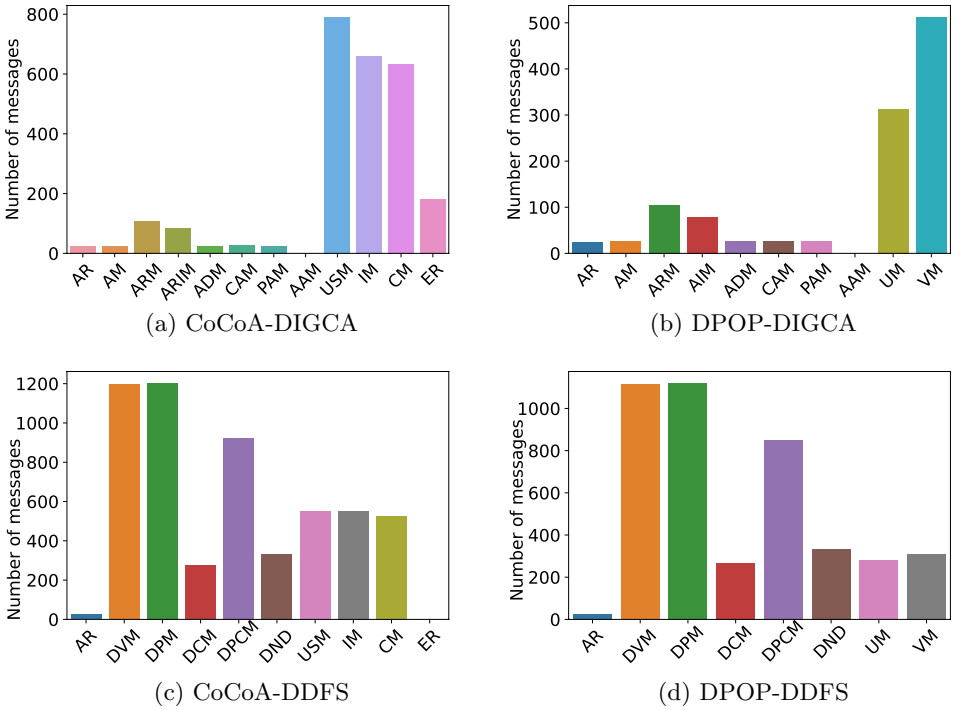


Fig. 4 Number of messages exchanged in the environment by message types

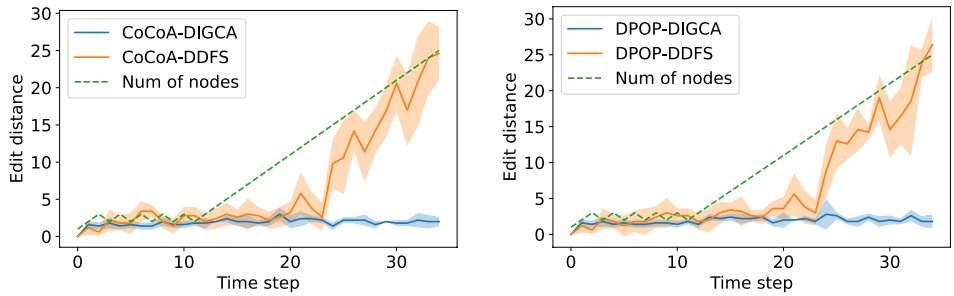


Fig. 5 Average edit distance recorded per time step. Dashed line depicts number of agents at each time step

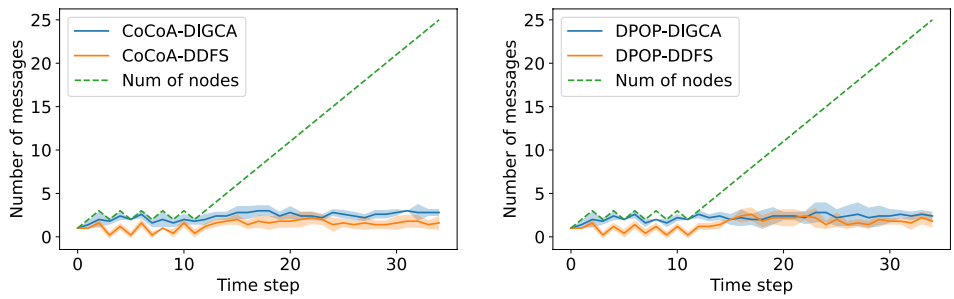
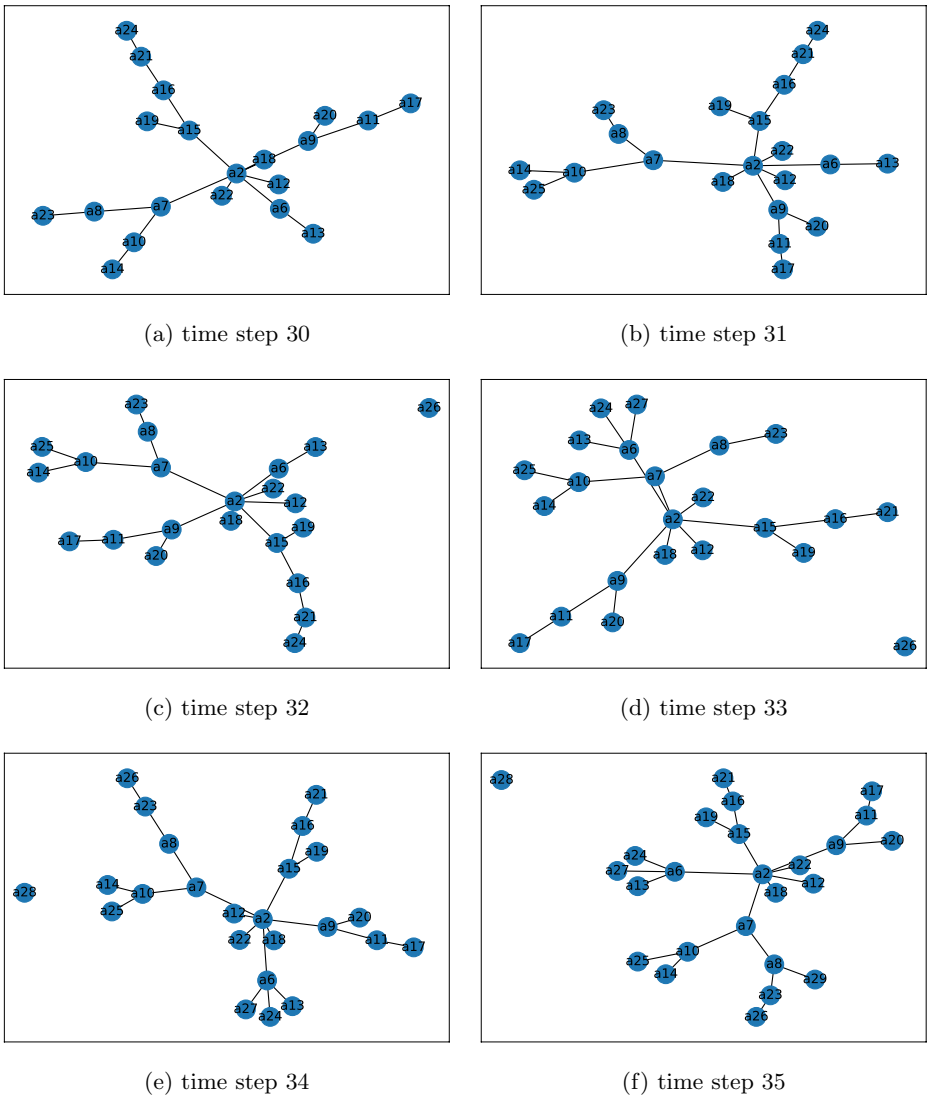


Fig. 6 Average number of components in the graph constructed in each time step. Dashed line depicts number of agents at each time step

5.2 Results

5.2.1 Complexity Analysis

We now analyze the interaction complexity (number of messages) when an agent makes a Connect call to connect to another agent. Since the complexity of the optimization depends on the DCOP algorithm in use, it is not factored in this analysis. Assuming there are m agents within the communication range of i, all m agents would receive the Announce message. Hence, in the worst case, the connect procedure's complexity is $O(m)$. Using Eq. 2, the worst case would be when i has the highest index, which implies that it will receive $O(m)$ AnnounceResponse messages. Since only one insertion point is selected by i, the worst-case complexity of AddMe interaction is $O(1)$. On the other hand,



(a) time step 30 (b) time step 31

(c) time step 32 (d) time step 33

(e) time step 34 (f) time step 35

Fig. 7 Sample connected components (of agents) in the environment for the last 6 time steps of a run

assuming i receives AddMe messages from m new agents (meaning i is the lowest index in the communication range), sending ChildAdded or AlreadyActive messages as replies has an asymptotic complexity of $O(m)$. Thus, the interaction complexity of each agent in a time step is linear in terms of the number of agents in its local area in the worst case. It is, therefore, feasible to use the proposed approach with optimization algorithms capable of exploiting local interactions for agents to collaborate in a MAS.

5.2.2 Empirical Analysis

Firstly, we used the event generation component to randomly generate 35 scenarios of 30 add-agent events and 5 remove-agent events. We maintained this sequence of events throughout our experiments to enable a fair comparison. The results report averages of 5 runs of each experiment using 5 different random number seeds. The main message types that agents exchanged in the experiments are listed in Table 1.

We show the accumulated number of messages exchanged by the agents in the environment using different combinations of hierarchy construction and DCOP algorithms in Fig. 3. In both DCOP algorithms, our proposed approach used about half the number of messages used by the baseline method at the end of the horizon. This performance results from DDFS being restarted in each time step, as has been done in previous studies that adopt DDFS in dynamic environments. DIGCA, on the contrary, maintains unaffected connections in the hierarchy and only agents in affected parts exchange connection-related messages.

In Fig. 4, where we show the breakdown of the number of messages tally, we gain deeper insights into the contribution of each message type. Using Table 1 as a guide, we notice that the message types of the proposed approach contributed far less to its overall number of messages exchanged, as

shown in Fig. 3. Instead, the DCOP-related messages were the main contributors to the overall number of messages in the case of DIGCA. In contrast, the DDFS baseline recorded more messages for its message types compared to associated DCOP message types. This reveals the efficiency of our proposed approach in terms of the number of messages it takes to establish a multi-agent hierarchy.

We also studied some properties of the graph constructed in each time step to understand how each hierarchy transformed. Here, the desiderata are to reduce the number of changes between multi-agent hierarchy updates and have agents close to one another form a valid hierarchy. Therefore, we measure each time step's average edit distance and the number of connected components. The edit distance measures hierarchy perturbations between time steps. The results are presented in Figs. 5 and 6. The broken line shows the event type executed in each time step - a rise indicates an add-agent event, and a dip indicates a remove-agent event. The best behaviour is to see the edit distance minimized after adding an agent to the environment.

In both the baseline and proposed approach experiments, we observed that as more agents were added to the environment, agents in different parts of the environment formed hierarchies. This observation explains why the average number of connected components recorded across the time steps in Fig. 6 is mostly below 5 even though several agents were in the environment.

Interestingly, while the baseline method outperformed the proposed approach in ensuring a minimal number of connected components, it performed poorly regarding its edit distance. The performance of DDFS on the number of connected components metrics is due to its ability to use updated agent neighbourhood information to reconstruct the hierarchy in each time step. DIGCA, on the other hand, maintains previous unaffected connections, enabling DCOP algorithms that need to reuse information from the previous time step to do so. This property of DIGCA explains why it outperformed the baseline on the edit distance metric. Thus, DDFS is more suitable for environments with few agents, whereas the proposed method works well in large agent sizes.

Our experiments show that our proposed method is feasible for facilitating the application of pseudo-tree-based DCOP algorithms and other multi-agent hierarchy-based optimization methods in an open and dynamic environment. We show sample multi-agent hierarchies constructed by the proposed method in Fig. 7.

6 Conclusion

In this paper, we have discussed the ad-hoc distributed multi-agent hierarchy generation problem. We have also proposed a distributed algorithm for constructing and maintaining a stable multi-agent hierarchy for interaction when collaborating in a dynamic environment. Our proposed approach addresses a vital issue in multi-agent operations in open and dynamic environments. Unlike existing methods, DIGCA does not require an existing interaction graph or reconstruction of the entire multi-agent hierarchy when changes are detected. We compared our proposed approach to a dynamic variant of the DDFS algorithm. Our method's effectiveness in domains with a high number of agents has been shown using a grid world simulation environment and examining the behaviour of the hierarchy construction method across all time steps. An aspect of our work that could be probed further is how Abort schemes could be incorporated to enable already initiated optimization processes to be terminated when necessary. Also, since real-world communication systems may not always guarantee the delivery of messages, further research is needed on how to address the DynDisMHG problem in unstable communication settings. We are also leveraging DIGCA to propose robust multi-agent coordination algorithms for open and dynamic environments.

Acknowledgements The authors would like to thank all reviewers of the manuscript for their inputs.

Author Contributions BA contributed in concept, experiments, and writing. FR and JY assisted with report improvement and review, as well as provided guidance on writing. All authors examined the results and gave final approval to the manuscript's final version.

Funding Not applicable

Availability of data and materials The source codes and related materials can be found at https://github.com/bbrighttaer/ddcop-dynagraph

Declarations

Conflict of interest The authors have no competing interests to declare that are relevant to the content of this article.

Consent for publication The authors hereby consent to publication of the work.

References

1. Yedidsion H, Zivan R, Farinelli A. Applying max-sum to teams of mobile sensing agents. Eng Appl Artif Intell. 2018;71:87–99. https://doi.org/10.1016/j.engappai.2018.02.017.

2. Rybski PE, Stoeter SA, Gini M, Hougen DF, Papanikolopoulos N. Effects of limited bandwidth communications channels on the control of multiple robots. In: Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180), 2001;1:369–3741. https://doi.org/10.1109/IROS.2001.973385
3. Padhy P, Dash RK, Martinez K, Jennings NR. A utility-based sensing and communication model for a glacial sensor network. In: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems. AAMAS '06, pp. 1353–1360. Association for Computing Machinery New York, NY, USA 2006. https://doi.org/10.1145/1160633.1160885 .
4. Pujol-Gonzalez M, Cerquides J, Meseguer P, Rodríguez-Aguilar JA, Tambe M. Engineering the decentralized coordination of uavs with limited communication range. In: Bielza C, Salmerón A, Alonso-Betanzos A, Hidalgo JI, Martínez L, Troncoso A, Corchado E, Corchado JM, editors. Advances in artificial intelligence. Berlin: Springer; 2013. p. 199–208.
5. Junges R, Bazzan ALC. Evaluating the performance of dcop algorithms in a real world, dynamic problem. In: Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2. AAMAS '08, pp. 599–606. International Foundation for Autonomous Agents and Multiagent Systems Richland, SC. 2008.
6. Lezama F, Munoz de Cote E, Farinelli A, Soares J, Pinto T, Vale Z. Distributed constrained optimization towards effective agent-based microgrid energy resource management. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 11804 LNAI(July 2017), 2019;438–449. https://doi.org/10.1007/978-3-030-30241-2_37
7. Zivan R, Yedidsion H, Okamoto S, Glinton R, Sycara K. Distributed constraint optimization for teams of mobile sensing agents. Auton Agent Multi-Agent Syst. 2015;29(3):495–536. https://doi.org/10.1007/s10458-014-9255-3.
8. Fioretto F, Pontelli E, Yeoh W. Distributed constraint optimization problems and applications: a survey. J Artif Intell Res 2018;61:623–698 https://doi.org/10.1613/jair.5565arXiv:1602.06347
9. Federal Aviation Agency: UTM Concept of Operations Version 2.0. Technical report 2020. https://www.faa.gov/sites/faa.gov/files/2022-08/UTM_ConOps_v2.pdf
10. Capitán C, Pérez-León H, Capitán J, Castaño Á, Ollero A. Unmanned aerial traffic management system architecture for U-space in-flight services. Appl Sci. 2021;11:9. https://doi.org/10.3390/app11093995.
11. Picard G. Trajectory coordination based on distributed constraint optimization techniques in unmanned air traffic management. Ifaamas. 2022;9:1065–73.
12. Petcu A, Faltings B. Superstabilizing, fault-containing distributed combinatorial optimization. Proc Natl Conf Artif Intell. 2005;1:449–54.
13. Petcu A, Faltings B. Optimal solution stability in dynamic, distributed constraint optimization. Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IAT. 2007;2007:321–7. https://doi.org/10.1109/IAT.2007.11.
14. Sultanik EA, Lass RN, Regli WC. Dynamic configuration of agent organizations. IJCAI International Joint Conference on Artificial Intelligence, 2009;305–311
15. Youssef Hamadi Christian Bessiere JQ. Backtracking in distributed constraint networks. ECAI, 1998;219–223
16. Collin Z, Dolev S. Self-stabilizing depth-first search. Inf Process Lett. 1994;49(6):297–301. https://doi.org/10.1016/0020-0190(94)90103-1.
17. Yeoh W, Varakantham P, Sun X, Koenig S. Incremental DCOP search algorithms for solving dynamic DCOP problems. Proceedings - 2015 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology, WI-IAT 2015 2015;2:257–264. https://doi.org/10.1109/WI-IAT.2015.114
18. Yeoh W, Varakantham P, Sun X, Koenig S. Incremental DCOP Search Algorithms for Solving Dynamic DCOPs (Extended Abstract). Artificial Intelligence (Aamas), 2011;1069–1070
19. Agyemang B, Ren F, Yan J. Distributed interaction graph construction for dynamic DCOPs in cooperative multi-agent systems. arXiv 2022. https://doi.org/10.48550/ARXIV.2212.03461 . https://arxiv.org/abs/2212.03461
20. Modi PJ, Shen WM, Tambe M, Yokoo M. Adopt: Asynchronous distributed constraint optimization with quality guarantees. Artif Intell. 2005;161(1–2):149–80. https://doi.org/10.1016/j.artint.2004.09.003.
21. Hoang K.D, Fioretto F, Hou P, Yokoo M, Yeoh W, Zivan R. Proactive dynamic distributed constraint optimization. Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS, 597–605 (2016)
22. Hoang KD, Hou P, Fioretto F, Yeoh W, Zivan R, Yokoo M. Infinite-horizon proactive dynamic DCOPs. Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS. 2017;1:212–20.
23. Chechetka A, Sycara K. A decentralized variable ordering method for distributed constraint optimization. Proceedings of the International Conference on Autonomous Agents, 2005;1385–1386. https://doi.org/10.1145/1082473.1082746
24. Silaghi MC, Yokoo M. Distributed constraint reasoning. Encycl Artif Intell. 2011;2:507–13. https://doi.org/10.4018/978-1-59904-849-9.ch077.
25. Zivan R, Meisels A. Dynamic ordering for asynchronous backtracking on DisCSPs. In: Beek P, editor. Principles and practice of constraint programming - CP 2005. Berlin: Springer; 2005. p. 32–46.
26. Stranders R, Farinelli A, Rogers A, Jennings NR. Decentralised coordination of mobile sensors using the max-sum algorithm. IJCAI International Joint Conference on Artificial Intelligence, 299–304 (2009)
27. Darko J, Park H. Proactive distributed constraint optimization of heterogeneous incident vehicle teams. CoRR **abs/2207.11132** 2022. https://doi.org/10.48550/arXiv.2207.11132arXiv:2207.11132
28. Sarker A, Choudhury M, Khan MM. A local search based approach to solve continuous DCOPs. Aamas 21, 2021;1127–1135.
29. Van Leeuwen CJ, Pawełczak P. CoCoA: A non-iterative approach to a local search (A)DCOP Solver. 31st AAAI Conference on Artificial Intelligence, AAAI 2017, 3944–3950 (2017) https://doi.org/10.1609/aaai.v31i1.11125
30. Hoang K.D, Yeoh W, Yokoo M, Rabinovich Z. New algorithms for continuous distributed constraint optimization problems. Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2020-May(Aamas), 2020;502–510.
31. Barambones J, Imbert R, Moral C. Applicability of multi-agent systems and constrained reasoning for sensor-based distributed scenarios: a systematic mapping study on dynamic DCOPs. Sensors. 2021;21:11. https://doi.org/10.3390/s21113807.
32. Hoang KD, Fioretto F, Hou P, Yeoh W, Yokoo M, Zivan R. Proactive dynamic distributed constraint optimization problems. J Artif Intell Res. 2022;74:179–225. https://doi.org/10.1613/jair.1.13499.