information

Article

Proactive Agent Behaviour in Dynamic Distributed Constraint Optimisation Problems

Brighter Agyemang , Fenghui Ren and Jun Yan *

Faculty of Engineering and Information Sciences, University of Wollongong, Wollongong, NSW 2522, Australia; ba233@uowmail.edu.au (B.A.); fren@uow.edu.au (F.R.)
* Correspondence: jyan@uow.edu.au

Abstract: In multi-agent systems, the Dynamic Distributed Constraint Optimisation Problem (D-DCOP) framework is pivotal, allowing for the decomposition of global objectives into agent constraints. Proactive agent behaviour is crucial in such systems, enabling agents to anticipate future changes and adapt accordingly. Existing approaches, like Proactive Dynamic DCOP (PD-DCOP) algorithms, often necessitate a predefined environment model. We address the problem of enabling proactive agent behaviour in D-DCOPs where the dynamics model of the environment is unknown. Specifically, we propose an approach where agents learn local autoregressive models from observations, predicting future states to inform decision-making. To achieve this, we present a temporal experience-sharing message-passing algorithm that leverages dynamic agent connections and a distance metric to collate training data. Our approach outperformed baseline methods in a search-and-extinguish task using the RoboCup Rescue Simulator, achieving better total building damage. The experimental results align with prior work on the significance of decision-switching costs and demonstrate improved performance when the switching cost is combined with a learned model.

Keywords: DCOP; Dynamic DCOP; proactivity; multi-agent systems

Citation: Agyemang, B.; Ren, F.; Yan, J. Proactive Agent Behaviour in Dynamic Distributed Constraint Optimisation Problems. *Information* **2024**, *15*, 255. https://doi.org/10.3390/info15050255

Academic Editors: Katsuhide Fujita

Received: 14 April 2024
Revised: 29 April 2024
Accepted: 30 April 2024
Published: 2 May 2024

1. Introduction

Collaborative Multi-agent Systems (MASs) have been the subject of several studies in the literature due to their ability to model many real-world problems effectively. In such domains, the capabilities of multiple autonomous agents are harnessed to address problems that are otherwise challenging or impossible for single agents to tackle. Some application domains include smart environments [1], mobile sensing [2], disaster management and rescue [3,4], environment monitoring [5,6], traffic light management [7], resource management in microgrids [8], and unmanned air traffic management [9].

The Distributed Constraint Optimisation Problem framework (DCOP) is a well-known framework for investigating MASs due to its ability to model various real-world applications [10]. In the DCOP context, the global objective of the team is decomposed into constraints between agents. The agents then leverage their local interactions to optimise the constraints in the system [11], making DCOP a natural fit for inherently decentralised multi-agent domains [12].

While most extant DCOP studies in MAS focus on agent properties such as reactivity, learning, social abilities, scalability, and stabilisation, proactive agent behaviour is the goal of several approaches [13–16]. In proactive methods, an agent is expected to be goal-driven and not only react to environmental changes. The agent's prediction and reasoning about possible future states of its environment typically result in this goal. For instance, an agent's goal after reasoning may be to maintain the temperature of a building (maintenance goal) or an achievement goal to lead evacuees to safety [17]. Thus, proactive behaviour enables agents to anticipate future changes and adapt accordingly.

Several proactive offline and online DCOP algorithms have recently been discussed [13,18,19]. In these methods, the proposed proactive algorithms do not observe the state and require environmental models to be predefined to enable proactive agent behaviour. However, in complex and dynamic environments, this assumption may not hold, which presents a challenge to applying existing algorithms. Mobile agents typically change their neighbour sets as they move from one region to another in a dynamic environment. Nevertheless, existing approaches do not discuss how dynamic interaction graphs relate to methods for enabling proactive behaviour. In [20], a tabular approach was discussed to maintain statistics about environmental changes and using the statistics in the constraint optimisation process as a means for reasoning about the future. More sophisticated methods are required in complex environments to enable more proactive behaviour than the method for counting changes discussed in [20].

Therefore, we propose an approach to enabling proactive agent behaviour in Dynamic DCOPs (D-DCOPs) where the environment's dynamics are unknown to the agents. Specifically, our main contributions are as follows.

1. The dynamic interaction graph method in [21] is extended to a complete graph setting whilst maintaining the multi-agent hierarchy or variable ordering that enables the execution of DCOP algorithms. This resolves the limitation in [21], where constraints or message-passing can only exist is parent–child relationships.
2. A temporal experience-sharing algorithm, based on dynamic multi-agent connections, is proposed to enable information propagation across agents in different parts of the environment. This experience-sharing algorithm relies on a distance metric defined in the observation space to limit redundant experiences in an agent's buffer due to perceptual aliasing.
3. The temporal experiences are then used to train an autoregressive model to predict the future states of random variables in a constraint function, given the previous observation. This addresses the transition model requirement in Proactive Dynamic DCOP (PD-DCOP) algorithms. We demonstrate how this model can be used with traditional DCOP algorithms to facilitate proactive agent behaviour in dynamic environments.

Our study is organised as follows. We discuss related work in Section 2; the background to our research is then presented in Section 3, our proposed approach is discussed in Section 4, and theoretical properties are given in Section 5. We introduce the experimental setup to evaluate our method in Section 6. In Section 7, we discuss our experiment results and conclude this study in Section 8.

2. Related Work

Proactive agent behaviour is an important agent characteristic in multi-agent systems. As a result, several studies have discussed approaches to enable agents in a collaborative MAS to be proactive in their environment where possible.

One of the earliest studies into proactivity in constraint programming is the work by Wallace and Freuder [20]. The authors discussed methods for tracking environmental changes and incorporating this information into the optimisation process. That way, agents can assign values that are less likely to change in the future. When applied to the dynamic environment, the approaches discussed in the study require agents to conduct a complete observation to gather statistics about changes of interest in the environment. However, agents in dynamic environments typically have partial observations.

Similarly, ref. [14] proposed a framework for finding super solutions that model causes of failures or changes in the environment and reason about the cost of repairs, resulting in the outcome of super solutions. While this method enables agents to reason about the future for value assignments, it requires predefined probabilities or information about change events. Random event probabilities may be challenging or impossible to define in complex dynamic environments.

In a series of studies on the subject, Hoang et al. [13,18,19] proposed the Proactive Dynamic Distributed Constraint Optimisation Problem (PD-DCOP). In the PD-DCOP framework, the random events are modelled as random variables in the environment. The authors proposed exact and approximate methods to solve PD-DCOPs. PD-DCOP approaches are open-loop policies that use the priors of the random variables in the constraint optimisation process, as exemplified by the algorithms proposed by these studies. Since the algorithms discussed do not observe the state of the random variables, their application is challenged in dynamic domains where the priors are unknown, as they cannot learn a model from the environment.

Also, [22] proposed the Markovian D-DCOP (MD-DCOP) framework to introduce Markov Decision Process (MDP) concepts into the Dynamic DCOP (D-DCOP) domain. These concepts include enabling an agent to observe the state of its environment and incorporating the observation into the optimisation process. The authors proposed algorithms that formulate the value assignment constraint between agents as a multi-arm bandit objective using a Q-value function in their work. This Q-value function is maintained by one of the agents in a constraint and is updated using the Q-learning algorithm. Similarly, an RL-based DCOP algorithm was proposed in [23], where the state of an episode maps to a DCOP. In this work, the authors resorted to using the outputs of a genetic algorithm to provide feedback to enable learning the Q-values for acting in the environment. We note that the constraints considered therein are unary constraints incorporated into each agent's state space. Although we consider the observation of the state to be a crucial part of our study, this is to the extent of learning a model to enable agents to predict future conditions and reason about them in a dynamic environment. Additionally, in dynamic environments, the neighbours of an agent may change as it moves in the environment, as opposed to the assumption of a fixed interaction graph used by these existing methods. Indeed, agents may join or leave the environment at arbitrary times (openness).

As noted earlier, agents in a dynamic environment usually have a partial observation of their state. This partial observation challenges agents regarding how they may improve their understanding of the environment and use that information to guide the search for proactive assignments or decisions. Researchers have adopted information-sharing techniques to address the partial observation challenge in such environments. In Multi-Agent Reinforcement Learning (MARL) algorithms, agents learn information sharing over differential communication channels [15,24–26] alongside a policy. In contrast, DCOP-based communication schemes do not require the information-exchange channel to be differentiable and delineate the messages exchanged between agents.

In traditional DCOP algorithms, agents that share a constraint use message-passing schemes to interact. Consequently, methods like the Distributed Depth First Search (DDFS) [27], the Multi-agent Organization with Bounded Edit (Mobed) [28], and the Hybrid Algorithm for Reconstructing Pseudo-trees (HARP) [29] have been used to generate agent interaction hierarchies that facilitate the message-passing algorithms used in D-DCOPs. These agent communication methods assume a fixed interaction graph to generate the multi-agent hierarchies for message-passing. Such algorithms must be better suited in dynamic environments where the agent neighbours may change over time. Accordingly, a new hierarchy-generation method was discussed in [21] to generate and maintain valid multi-agent hierarchies. Our study extends this dynamic hierarchy approach for a neighbourhood with a connected agent. We then leverage this dynamic agent connection to propose a message-passing and experience-sharing scheme.

In summary, this study focuses on developing proactive agent behaviour in D-DCOPs by enabling agents to learn a look-ahead model from the observation of changes in the environment. We leverage existing dynamic agent-connection schemes to develop an experience-sharing method that allows agents to mitigate the challenge of partial observation in a dynamic environment.

3. Background

In this section, we discuss an illustrative scenario where agents can exhibit proactive behaviour, and we formalise the problem of this study.

3.1. RoboCup Rescue Simulation

In Figure 1, we illustrate a search-and-extinguish mission in the RoboCup Rescue Simulation (RCRS) [30], where fire brigade agents (depicted by red circles) are tasked with searching for buildings (shown as polygons) that are on fire and extinguishing them. A building's tendency to be set on fire depends on the material it is made of (wood, steel, or concrete). The environment of these agents is dynamic in that buildings can be set on fire randomly, and those close to already ignited ones may also heat up and eventually be set ablaze. The fire brigade agents can only extinguish a building already on fire. In the illustration, building b_1 is already on fire, and the neighbouring buildings b_2 and b_3 (assumed to be wooden buildings) are heating up. The fire station b_5 is a unique building where an agent can refill its water tank. Buildings that are dark-coloured are entirely burned out. The agents are expected to collaborate in carrying out their tasks. In the environment, the amount of water needed to extinguish a fire in a building is proportional to the temperature of the building. Therefore, an agent proactively approaches a building it anticipates will catch fire in the future and reduces the needed water quantity by extinguishing the fire early. For instance, in the illustration, we depict an agent choosing to be close to building b_3, whereas another is closer to building b_4. However, an available agent is expected to extinguish a fire instead of only anticipating fire at a neighbouring building. In our study, we model this motivating scenario as a D-DCOP and use it to demonstrate the effectiveness of our proposal.

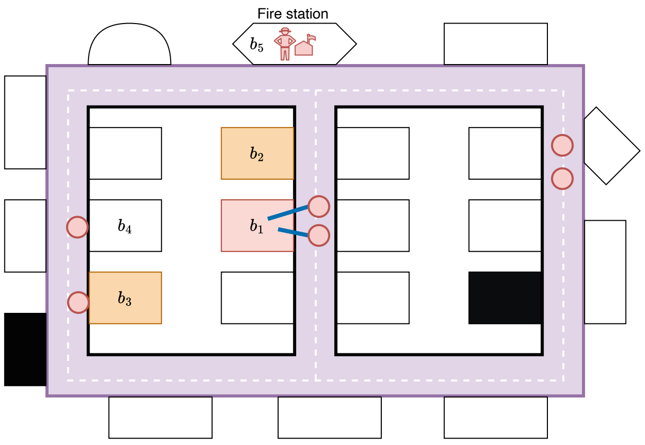

Figure 1. Fire brigade agents engage in a dynamic search-and-extinguish mission. Red circles represent agents, while polygons denote buildings. White shapes, such as b_4, indicate unaffected buildings, while dark ones are fully burnt-out. Buildings b_2 and b_3 are nearly ablaze, while b_1 is actively being extinguished by two nearby agents. Building b_5 serves as a refill station for agents' water tanks. View in colour for optimal comprehension.

3.2. Distributed Constraint Optimisation Problem

In multi-agent systems formulated as DCOPs, agents assign values from a domain to their decision variables to optimise certain constraint functions. It is assumed that the agents are fully cooperative and can observe the environment [10]. Also, the environment is dynamic and deterministic. The DCOP is modelled as a tuple $P = \langle \mathbf{A}, \mathbf{X}, \mathbf{D}, \mathbf{F}, \alpha \rangle$, where:

- $\mathbf{A} = \{a_1, a_2, \dots, a_m\}$ is a finite set of agents,
- $\mathbf{X} = \{x_1, x_2, \dots, x_n\}$ is a finite set of variables,
- $\mathbf{D} = \{D_1, D_2, \dots, D_n\}$ is a set of variable domains such that the domain of $x_i \in \mathbf{X}$ is D_i,
- $\mathbf{F} = \{f_1, f_2, \dots, f_k\}$ is a set of constraint functions defined on $\mathbf{X}$ where each $f_i \in \mathbf{F}$ is defined over a subset $\mathbf{x}^i = \{x_1, x_2, \dots, x_p\} \subseteq \mathbf{X}$, with $p \leq n$, determines the cost of value assignments of the variables in $\mathbf{x}^i$ as $f_i : D_1 \times D_2 \times \dots \times D_p \to \mathbb{R} \cup \{\perp\}$, where

$\perp$ denotes utility for infeasible configurations. Here, the cardinality of $\mathbf{x}^i$ is the arity of f_i. The global cost of the values assigned to variables in $\mathbf{X}$ is $\mathbf{F}_g(\mathbf{X}) = \sum_{i=1}^{k} f_i(\mathbf{x}^i)$,

- $\alpha : \mathbf{X} \rightarrow \mathbf{A}$ is an "onto" function that assigns the control of a variable $x \in \mathbf{X}$ to an agent $\alpha(x)$.

We assume that α assigns only one variable per agent and the use of binary constraint functions. In the multi-variable setting, each variable can 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.

A Current Partial Assignment (CPA) or partial assignment is the assignment of values to a set of variables $\bar{\mathbf{x}}$ such that $\bar{\mathbf{x}} \subset \mathbf{X}$. 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 $f_i(\sigma_{\mathbf{x}^i}) \neq \perp$. The objective of a DCOP is to find a complete assignment that minimises the total cost:

$$\sigma^* := \underset{\sigma \in \mathbf{\Sigma}}{argmin}\, \mathbf{F}_g(\sigma) = \underset{\sigma \in \mathbf{\Sigma}}{argmin} \sum_{f_i \in \mathbf{F}} f_i(\sigma_{\mathbf{x}^i}), \tag{1}$$

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

The D-DCOP is an extension of the DCOP formulation to address dynamic multi-agent environments. D-DCOP is modelled as a sequence of DCOPs, $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_T$. 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 DCOP problem arising at each time step.

Likewise, PD-DCOP extends the D-DCOP to allow the modelling of random variables in the environment that agents do not control. In this context, decision variables refer to variables whose values are assigned by agents. Specifically, the following extensions are introduced to the DCOP framework:

- The set of variables $\mathbf{X}$ is extended to compose random variables $\mathbf{Y} \subseteq \mathbf{X}$ to model stochastic events in the environment (e.g., device malfunctioning, weather, and temperature).
- The domain set $\mathbf{D}$ is also composed of the event spaces for the random variables $\Omega = \{\Omega_y\}_{y \in \mathbf{Y}} \subseteq \mathbf{X}$.
- $h \in \mathbb{N}$ is the horizon of the environment.
- The variables in the $\mathbf{x}^i$ of a constraint function may be a mixed set of decision variables and random variables. We denote the set of constraint functions whose scope contains a random variable as $\mathbf{F_Y} \subseteq \mathbf{F}$, where $f_i^{\mathbf{Y}} \in \mathbf{F_Y}$, which is associated with at most a single random variable.
- $\mathbf{T} = \{T_y\}_{y \in \mathbf{Y}}$ is a set of transition functions $T_y : \Omega_y \times \Omega_y \rightarrow [0,1] \subseteq \mathbb{R}$ for the random variables $y \in \mathbf{Y}$. A transition function specifies the probability that a random variable changes value in future time steps.
- $C \in \mathbb{R}^+$ is a switching cost function that assigns a cost to a decision variable for changing values between time steps. This study views this cost term as a unary constraint function. High switching costs discourage agents from frequently switching values. Hence, the agents have to be purposeful with any potential change in decision in time step 2.
- $\gamma \in [0,1]$ is a discount factor that is used to control the significance of future rewards or costs. We set $\gamma = 1$ in this study.
- $p^0_{\mathbf{Y}} = \{p^0_{y \in \mathbf{Y}}\}$ is a set of priors of the random variables $y \in \mathbf{Y}$.
- The onto function α assigns only decision variables to agents $\alpha : \mathbf{X} \backslash \mathbf{Y} \rightarrow \mathbf{A}$.

The PD-DCOP objective is to find a sequence of assignments σ^* of all the decision variables for all time steps of the environment:

$$\sigma^* := \underset{\sigma = \langle \sigma^0, \ldots, \sigma^h \rangle \in \mathbf{\Sigma}^{h+1}}{argmin} \mathbf{F}_g^h(\sigma) \tag{2}$$

$$\mathbf{F}_g^h(\sigma) = \sum_{t=0}^{h} \gamma^t \left[\sum_{f_i \in \mathbf{F} \setminus \mathbf{F_Y}} f_i(\sigma^t_{\mathbf{x}^i}) + \sum_{f_i^{\mathbf{Y}} \in \mathbf{F_Y}} f_i^{\mathbf{Y}}(\sigma^t_{\mathbf{x}^i}) \right] + \sum_{t=0}^{h-1} \sum_{x \in \mathbf{X} \setminus \mathbf{Y}} \gamma^t C(x^{t-1}, x^t) \tag{3}$$

$$f_i^{\mathbf{Y}}(\sigma^t_{\mathbf{x}^i}) = \sum_{\omega \in \Omega_{y_i}} f_i(\sigma^t_{\mathbf{x}^i} | y_i = \omega) \cdot p^t_{y_i}(\omega) \tag{4}$$

$$p^t_{y_i}(\omega) = \sum_{\omega' \in \Omega_{y_i}} T_{y_i}(\omega', \omega) \cdot p^{t-1}_{y_i}(\omega') \tag{5}$$

Thus, a PD-DCOP solution is an open-loop system (offline) that finds the assignments for each time step in the horizon. In this study, we assume that the transition functions and the priors are unknown to the agents. Hence, the agents depend on observing the changes in the random variables in the environment while operating (online). In determining the assignments σ^t, each agent a_i observes the random variables in the scope of the constraints of a_i, denoted as o^t. Therefore, we reformulate Equation (4) as

$$f_i^{\mathbf{Y}}(\sigma^t_{\mathbf{x}^i}) = f_i\left(\sigma^t_{\mathbf{x}^i} | y_i = M\left(o^t_{y_i}, l\right)\right) \tag{6}$$

where M is a model that takes $o^t_{y_i} \in \Omega_{y_i}$ and predicts the value of y_i after l look-ahead steps. In what follows, we discuss an approach to learning M from the temporal experiences of the agents.

4. Proposed Approach

This section discusses our approach to addressing the problem presented in Section 3.

4.1. Dynamic Multi-Agent Connections

Communication between agents is vital in multi-agent systems. In the DCOP literature, message-passing between agents forms the basis for the optimisation process. One prevalent approach is to derive a hierarchy from the constraint graph representation of the DCOP and use this hierarchy to organise the optimisation process. In [21], a dynamic hierarchy-generation algorithmwas proposed to generate a valid hierarchy in open and dynamic multi-agent environments. However, this proposed dynamic hierarchy-generation method restricts interactions in parent–child relationships only. As a result, nearby agents in other sub-trees of the hierarchy cannot interact in the environment. Thus, only tree-based DCOP methods could be used with the dynamic hierarchy-generation algorithm. In this study, we extend the approach in [21] to enable all nearby agents to establish connections while maintaining the hierarchy. Similar to the existing DCOP literature [10,21,28,31], we assume that agents are uniquely identifiable and cooperate in the environment to optimise the global objective. Also, agents communicate via message-passing, and messages are delivered in the order sent.

It has been shown in [21] that once an agent has selected a neighbour to receive an AddMe message, all other neighbours receive an *AnnounceResponseIgnored* message. In our study, we propose sending a PseudoParentRequest message instead of an *AnnounceResponseIgnored*. When a_j receives a *PseudoParentRequest* from a_i, it adds a_i to its pseudo-children list and sends a *PseudoChildAdded* message to a_i. When a_i receives a *PseudoChildAdded* message, it adds a_j to its pseudo-parents list. Thus, a_i and a_j can establish a link for interaction without affecting their existing parent–child relationships or invalidating the hierarchy with acyclic connections since connection types could be checked to avoid cycles during optimisation. We illustrate this process in Figure 2. In this illustration, we assume that a_3 joins an environment where a_1 is the parent of a_2. a_3 uses the dynamic hierarchy generation procedure to establish a parent–child relationship with a_1, and the above process establishes a pseudo-child link with a_2.

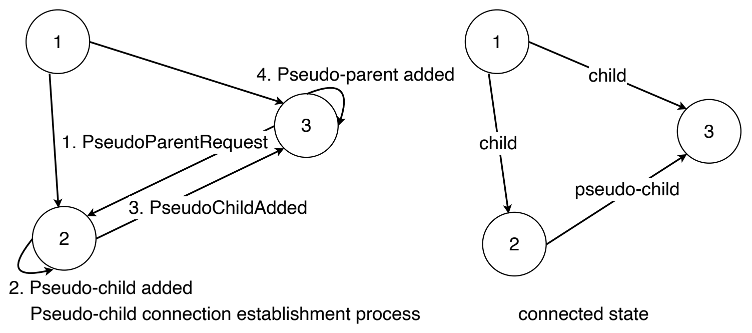

Figure 2. Illustration of the pseudo-child connection process.

4.2. Temporal Experience Sharing

Agents in dynamic environments typically have a partial observation of the environment. Temporal Experience Sharing among agents in a neighbourhood is a practical approach to mitigate the challenges of partial observation [24]. Therefore, we leverage the dynamic agent-connection approach discussed in Section 4.1 to enable agents in a neighbourhood to share their experiences. These experiences show the changes in random events within the environment between time steps. In this study, we model an experience as a tuple $\langle s_{t-1}, s_t \rangle$. Thus, the dynamic interaction graph facilitates the optimisation process (along hierarchy connections) and Temporal Experience Sharing (along all connections) among agents in a neighbourhood.

We outline the temporal experience algorithm of this study in Algorithm 1. We assume that agents can communicate via message-passing and that messages arrive in the order sent despite potential delays. Each agent may execute this algorithm asynchronously in a time step. The main objective is to update the buffer of an agent a_i with experiences from its neighbours N_i. The algorithm requires a list of neighbouring agents involved in the experience-sharing process L and the dynamic interaction graph algorithm Φ.

Further, we assume that each agent stores the experiences in a finite buffer whose underlying data structure is a circular associative array where the key is a globally unique ID of the experience (value). Additionally, we provide a sample size n and a similarity threshold z for managing how shared experiences are added to the agent's buffer. In what follows, we describe the algorithm by referring to the executing agent as i and the sending agent as j.

The Start procedure calls Φ when new agent connections are established, passing the newly connected agents as arguments. If the buffer of i has experiences to share, i sends the keys of B in an *ExperienceHistoryDisclosure* message to all agents in L (lines 1–6). When i receives an *ExperienceHistoryDisclosure* message, i compares the shared keys with its buffer to determine experiences to request and share with j (lines 8–12). In line 13, i sends an *ExperienceSharingWithRequest* message to j. Upon receiving an *ExperienceSharingWithRequest* from j, i first shares the requested experiences with j via an *ExperienceSharing* message (lines 16–18) and then adds the received experiences from j to its buffer by executing the *MergeTemporalExperience* procedure (explained below). Similarly, the experiences shared via the *ExperienceSharing* message are added to B by calling the *MergeTemporalExperience* procedure (lines 22–25). The *SendNeighbourUpdate* procedure is called by i whenever it adds a local experience to its buffer. This procedure shares the latest experience in B with all existing neighbours N_i via an *ExperienceSharing* message (lines 26–28). This approach enables an agent to have varied experiences from unexplored parts of the dynamic environment.

However, the naive addition of all shared experiences to the buffer may populate the buffer with less informative or similar experiences since agents in a neighbourhood may perceive similar events. Consequently, we introduce the *MergeTemporalExperience* procedure to mitigate this scenario. When called, it samples n experiences from its buffer (line 32). The sampled experiences S are compared to the experiences to be added to E using a similarity function φ (line 33) defined over the observation space. This function determines how similar each experience is to existing experiences in the buffer (represented by S). All similar experiences are ignored by i using the similarity threshold z (line 34). The remaining experiences in E are now added to B. If there are no experiences in B, all shared experiences are added to B. We also note that i executes the *MergeTemporalExperience* to add a local experience to B. This approach to incorporating experiences ensures variation in B.

Algorithm 1 Temporal Experience Sharing Algorithm

```
Require: list of agents $L$, experience similarity measure $\varphi$, experience buffer $B$, dynamic interaction graph $\Phi$,
    reference experience sample size $n$, experience similarity threshold $z$
Ensure: Updated experience buffer $B$
 1: procedure START($L$)
 2:     if $B$ contains experiences then
 3:         $k \leftarrow$ keys of $B$
 4:         Send ExperienceHistoryDisclosure{$k$} to agents in $L$
 5:     end if
 6: end procedure
 7: procedure RECEIVEEXPERIENCEHISTORYDISCLOSURE($m_j$)
 8:     $k_j \leftarrow$ shared experience keys from $m_j$
 9:     $k_i \leftarrow$ keys of $B$
10:     $r \leftarrow$ entries in $k_j$ not in $k_i$
11:     $s \leftarrow$ entries in $k_i$ not in $k_j$
12:     $E \leftarrow$ select experiences $s$ from $B$
13:     Send ExperienceSharingWithRequest{$E, r$} to $j$
14: end procedure
15: procedure RECEIVEEXPERIENCESHARINGWITHREQUEST($m_j$)
16:     $r_j \leftarrow$ Get requested experience entries in $m_j$ by $j$
17:     $E \leftarrow$ select experiences $r_j$ from $B$
18:     Send ExperienceSharing{$E$} to $j$
19:     $E \leftarrow$ Shared experiences in $m_j$
20:     MergeTemporalExperiences(E)
21: end procedure
22: procedure RECEIVEEXPERIENCESHARING($m_j$)
23:     $E \leftarrow$ Shared experiences in $m_j$
24:     MergeTemporalExperiences(E)
25: end procedure
26: procedure SENDNEIGHBOURUPDATE( )
27:     $E \leftarrow$ Select last added experience from $B$
28:     Send ExperienceSharing{$E$} to all neighbours in $N_i(\Phi)$
29: end procedure
30: procedure MERGETEMPORALEXPERIENCES($E$)
31:     if $B$ contains experiences then
32:         $S \leftarrow$ Sample $n$ experiences from $B$
33:         $v \leftarrow$ Calculate similarity measures using $\varphi(E, S)$
34:         $E \leftarrow$ Select entries in $E$ whose similarity measures are below $z$
35:     end if
36:     Add all entries in $E$ to $B$
37: end procedure
38: Return $B$
```

4.3. Temporal Experience Modelling

As discussed earlier, we aim to learn a model that can predict possible changes in the D-DCOP and incorporate this prediction in the optimisation process. Consequently, we use the experiences in B to learn this model. Thus, the temporal experience-sharing algorithm is an approach for tracking environmental changes. The agent then uses these experiences to model how the environment evolves. In Algorithm 2, we present a generic framework that we adopt to train the look-ahead model incrementally. Upon startup, i initializes model M_{t-1} and the model performance tracking value ν to 0. The model is trained when more than k experiences are in B (lines 1–3). Once this condition is satisfied, n experiences are sampled from B (line 4). Afterwards, the sampled experiences are used to train a model M_{temp} (line 5). This temporary model is accepted if its evaluation is better than the previous model (lines 6–11). This model is used by i in the optimisation process as specified in Equation (6).

Algorithm 2 Temporal Experience Modelling Algorithm

```
Require: Experience buffer $B$, batch size $n$, pretrained model $M_{t-1}$, previous best score $\nu$, minimum
    number of samples to train $k$, learning algorithm $\Theta$, and performance metric $e$.
Ensure: Model $M_t$
 1: if size($B$) < $k$ then
 2:     Return $M_{t-1}$
 3: end if
 4: $D \leftarrow$ Sample $n$ temporal experiences from $B$
 5: $M_{temp} \leftarrow \Theta(M_{t-1}, D)$
 6: $\nu_{temp} \leftarrow e(M_{temp})$
 7: if $\nu_{temp} > \nu$ then
 8:     $M_t \leftarrow M_{temp}$
 9:     $\nu_{temp} \leftarrow \nu_{temp}$
10: else
11:     $M_t \leftarrow M_{t-1}$
12: end if
13: Return $M_t$
```

5. Theoretical Properties

In this section, we examine the communication and computational complexity of the Temporal Experience Sharing algorithm. We make the following assumptions to analyse the communication complexity:

- In the worst case, all agents in the MAS are within communication range of one another.
- Each agent communicates its experiences with all other agents in the system.
- The communication overhead per message exchange is known and is constant.
- The number of experiences shared in each message exchange is proportional to the size of the experience buffer.

We discuss the communication complexity in terms of the number of agents N and the size of the experience buffer M. The communication complexity of each phase is then as follows.

- Experience Disclosure Phase: In the worst case, each agent sends its experiences to all other agents. Hence, the communication complexity is $O(NM)$.
- Experience Sharing with Request Phase: Each agent requests experiences from other agents. The communication complexity is $O(NM)$ in this case.
- Experience Sharing Phase: Each agent shares experiences with other agents, resulting in $O(NM)$.
- Neighbor Update Phase: Each agent sends the most recent experience to its neighbours with a complexity of $O(N)$.

The asymptotic communication complexity is then $O(NM)$, arising from the need for the agents to exchange experiences.

Similarly, we make the following assumptions to analyse the computational complexity of Algorithm 1:

- We assume a constant communication cost for message exchanges.
- Calculating similarity measures between experiences involves comparing each new experience with a subset of experiences in the buffer. We assume that the computation of the similarity measure is pairwise and requires comparing each new experience with a subset of experiences in the buffer.
- Simple random sampling selects a subset of experiences in the buffer.
- The merging of experiences involves searching for and removing possibly duplicate experiences. This involves iterating through the experiences in the buffer (in the worst case) and comparing them with the new experiences.

In this context, where the size of the buffer is denoted as P, we analyse the computational complexity as follows:

- Message-passing has a constant computational complexity $O(1)$.
- For each new experience, the computational complexity for comparing with all experiences in the buffer is $O(P)$.
- Under simple random sampling, the complexity is linear in the size of the experience buffer $O(P)$.
- Assuming there are Q new experiences to merge, the complexity is $O(PQ)$.

Thus, the asymptotic computational complexity is $O(PQ)$. This is primarily due to the similarity and merging operations.

We note that in real-world scenarios, the agents are usually distributed in the environment and may not have all agents within the communication range. Also, computational techniques such as vectorisation could be used to improve the similarity and merging operations. The merging operation is performed on a subset of the experiences in the buffer, thus reducing the complexity. Therefore, the Temporal Experience Sharing algorithm is feasible in MASs, as we demonstrate in our experiments.

6. Experimental Setup

This section describes our experimental setup to evaluate the proposed approach for enabling proactive agent behaviour in D-DCOP algorithms.

Our experiments used the RCRS platform [30]. The RCRS platform is a server that simulates fire outbreaks, civilian search and rescue, and other common disaster events. Since we consider multi-agent operations in dynamic environments, RCRS is suitable because it creates a partially observable, dynamic, and stochastic environment. Also, the simulation horizon is in discrete time steps. In each time step, agent commands are executed in the environment. Agent programs are implemented as clients that send commands to the simulation server at each time step. There are two types of agents in the RCRS: platoon agents and centre agents. The ambulance team, fire brigade, and police force constitute the platoon agents, and the ambulance centres, fire stations, and police offices make up the centre agents.

In our experiments, we configured the simulation server to follow the motivating domain in Section 3.1. We enabled 12 fire brigade agents and one fire station in a scenario with 36 buildings that could catch fire. The fire brigade agents were randomly placed in the environments, and these starting positions were maintained in all simulations. The simulation server starts with three buildings that are already set on fire. The fire spreads from these three buildings to other buildings if not extinguished. The following are the different states of a building's fieriness:

1. Unburnt
2. Burning slightly
3. Burning more
4. Burning severely
5. Not burning (water damage)
6. Extinguished (minor damage)
7. Extinguished (moderate damage)
8. Extinguished (severe damage)
9. Completely burnt

Also, an extinguished building may be set on fire again if a nearby building is ignited. Therefore, the agents are expected to collaborate to extinguish all fires as quickly as possible or slow down the total building damage in the environment. Slowing the total building damage could give rescue teams additional time to evacuate trapped civilians. We depict the described scenario in Figure 3.

We implemented the agents' program using the RCRS Python API (version 1.0.0). The simulation server sends the observation of each agent to its corresponding program at the beginning of every time step. When an agent receives its observation, it executes its D-DCOP algorithm to determine the appropriate command to send to the server. We set the maximum thinking time of each agent to 5 s. The fire station agent was only used to aggregate simulation metrics from other agents in our experiments. The simulation server and agent programs were run using a MacBook Pro computer with a Ventura 13.6.2 OS, an Apple M1 Pro chip and 16 GB RAM (Apple, Cupertino, CA, USA).

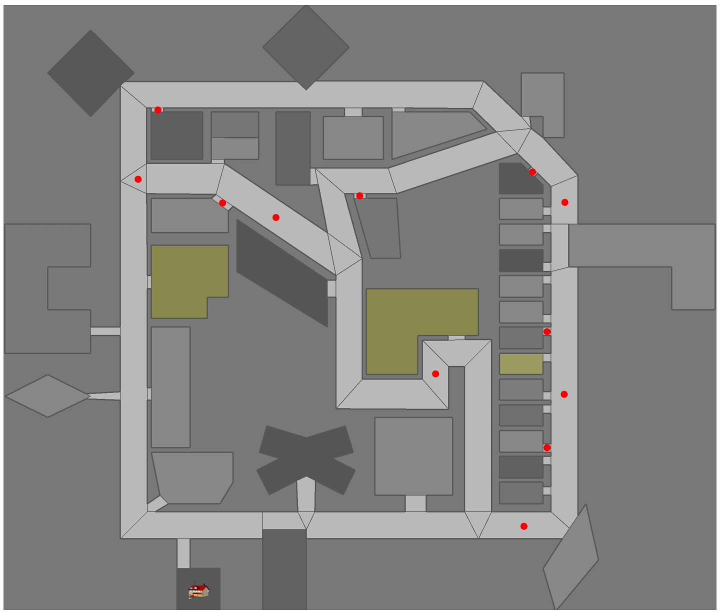

Figure 3. Map of RoboCup Rescue Simulator environment used in our experiments. Buildings already on fire are depicted in yellow, and fire brigade agents are shown as red dots on the map. Agents are randomly placed at the beginning of the simulation.

6.1. Modelling the Search-and-Extinguish Problem as a D-DCOP

Regarding the D-DCOP algorithms, we used the Cooperative Constraint Approximation (COCOA) [32] and Distributed Pseudo-tree Optimisation Problem (DPOP) [33] algorithms. We modelled the search-and-extinguish problem of the RCRS environment as follows:

- Agents: The fire brigade agents in the environment shown in Figure 3 served as the agents in the DCOP. These agents interact with the environment by receiving local observations and sending commands. Agent-to-agent communication was enabled using an implementation of RabbitMQ's AMQP messaging protocol.
- Variables: We mapped each agent to a single decision variable that determines the building selected by the agent to either move to or extinguish the building's fire. The random variable in this setting is the temperature of buildings in the environment.
- Domain: Each agent's domain was the set of all buildings in the environment whose fieriness was either burning severely or lower.
- Constraint functions: Constraints existed between agents that shared parent–child relationships.

We represented each constraint as

$$f_q(d_i, d_j) = f_q^u(d_i) + f_q^c(d_i, d_j) \tag{7}$$

where $d_i \in D_i$, $d_j \in D_j$, f_q^u is the unary constraint of agent i and f_q^c is the coordination constraint agent i shares with agent j. We defined the unary constraint as,

$$f_q^u(d_i) = \begin{cases} 1000 & \text{if } n > 3, \\ C(d_i^{t-1}, d_i) - e(d_i) & \textit{otherwise} \end{cases}$$

where d_i^{t-1} is the building selected in the previous time step, n is the number of agents already assigned to building d_i, C is the decision change cost, and $e(d_i)$ is the exploration factor of d_i. The intuition is to discourage agent i from selecting a building already assigned to more than three other agents by using a high cost. We defined C as

$$C(d_i^{t-1}, d_i) = \begin{cases} c & \text{if } d_i^{t-1} \neq d_i \text{ and } y^{t-1} > 1 \\ 0 & otherwise \end{cases}$$

where y^{t-1} is the fieriness of d_i^{t-1} and c is a cost assigned when the current value being evaluated d_i is different from a burning building selected at the previous time step (switching cost). This encourages agents to consider sticking to the previous selection if the building is burning. Also, we defined $e(d_i)$ as

$$e(d_i) = \sqrt{2|D_i|\frac{ln(t)}{N(d_i)}} \tag{8}$$

to encourage agents to explore the environment. $N(d_i)$ is the number of times the agent has selected d_i. The coordination constraint f_q^c was also defined as

$$f_q^c(d_i, d_j) = \begin{cases} \lambda w \times p & \text{if } d_i = d_j, \\ 0 & otherwise \end{cases}$$

and

$$\lambda = \begin{cases} -1 & \text{if } \tau(d_i) >= T_c, \\ 1 & otherwise, \end{cases}$$

where p is a coordination constant we set to 20 in our experiments, $\tau(d_i)$ is the temperature of d_i, w is the weight of the building calculated as $w(d_i) = \tau(d_i)/T$, and T is the maximum temperature of a building in the environment. T was set to 1000 in all our experiments. T_c is the critical temperature of a building beyond which neighbouring agents are expected to collaborate to move to the building and extinguish the fire; we set $T_c = 400$.

This constraint also encourages agents to move to different buildings when the temperature is below the critical threshold. Here, we expect proactive agents to assign a value based on their predictions. Thus, agents will team up without being proactive only in reaction to critical temperatures.

In contrast, proactive agents use exchanged information and the look-ahead model to determine which buildings are likely to reach critical temperatures and collaborate to extinguish fires ahead of time. We implemented an experience buffer to store shared and local experiences following Section 4.

6.2. Model Training

In the training phase, randomly sampled experiences in the buffer were used to train a model that predicts the next temperature of a building given its current observation. In the search-and-extinguish experiments, an observation from the RCRS server is a tuple of the temperature of each building, the fieriness, the brokenness of the building, the building's material (wood, steel, or concrete), and the fire index (maximum temperature of the set of neighbouring buildings). An experience is a tuple of two consecutive observations by an agent. The sampled experiences were then preprocessed into a dataset, where the temperature of the second observation in each experience was the target for supervised training.

The training procedure is specified in Algorithm 2. We set the k of Algorithm 2 to 100. Thus, a minimum of 100 experiences were required to be in the buffer before the training procedure could be executed. Therefore, each agent ran Algorithm 1 and a D-DCOP algorithm in the environment for 300 time steps, triggering Algorithm 2 every 5 time steps. The XGBoost learning algorithm, with a learning rate of 0.001 and a maximum tree depth of 10, was used in the experiments.

During the evaluation phase, each agent applied its local model in an autoregressive manner for l future steps in the optimisation process, as specified in Equation (6). Thus, the final temperature of a building considered by the coordination constraint was determined in this manner. When $l = 0$, the observed temperature is used without applying the model. We discuss the results of our experiments below

7. Results and Discussion

This section discusses the results of our experiments. We focused on examining agents' decision-making to ascertain whether the agents could exhibit proactive behaviour using our proposed approach. We compare the results of the D-DCOP algorithms (COCOA, DPOP) used in this study with versions that used the look-ahead model developed using our methods (LA-COCOA, LA-DPOP). The LA-COCOA and LA-DPOP methods had $l = 10$, whereas COCOA and DPOP had $l = 0$ (see Equation (6)). We also examined the effect of the switching cost c on agents' behaviour by setting it to 10, 30, and 50. To reduce the impact of randomness on our results, each of the experiments for each value of c was conducted over five random seeds. Thus, 60 different runs were conducted, and each run had 100 time steps.

We show the performance of the methods across the different switching costs in Figure 4. The building damage metric was retrieved from the RCRS server and was the only active metric, since all agents were fire brigade agents. This metric estimates the total structural damage of buildings in the simulation environment: 0 indicates total damage, and 1 indicates no damage. It can be seen that the LA-DPOP consistently outperformed the other methods followed by LA-COCOA.

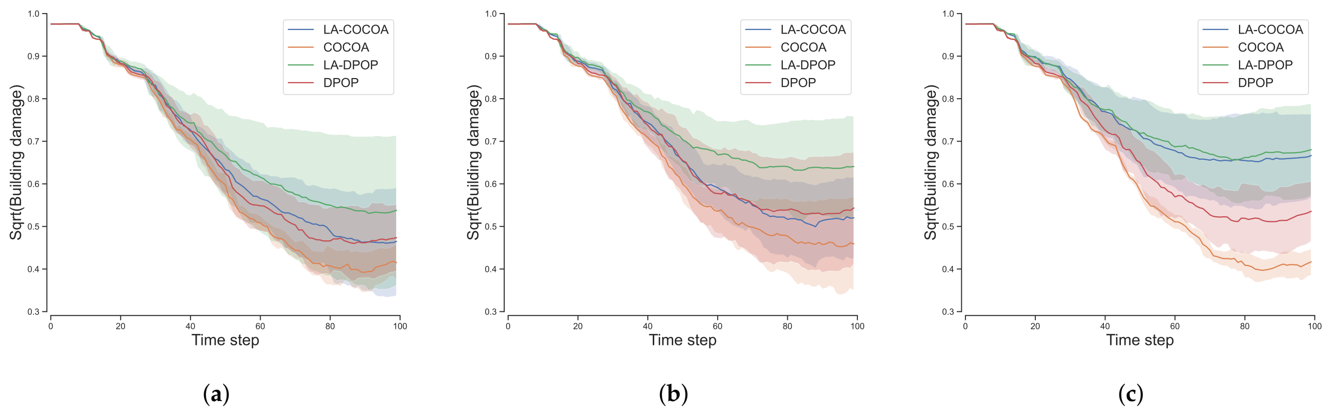

Figure 4. Average performance of fire-brigade agents in the RCRS environment using the different D-DCOP algorithms of the study. LA-COCOA and LA-DPOP represent our proposed methods, whereas COCOA and DPOP represent standard D-DCOP algorithms. (**a**) Switching cost = 10. (**b**) Switching cost = 30. (**c**) Switching cost = 50.

In particular, as the switching cost increased, the performance margins became more pronounced. We observed that combining switching costs with our proposed approach in reasoning could help stabilise agents' decision-making in exhibiting proactive behaviour in an environment. We reckon that the appropriate switching cost depends on the application domain. Likewise, the value of l should be determined to capture periods when the random variables change in the environment. In the RCRS environment, changes to the temperature variable happened slowly over time steps. Hence, we observed that smaller values of l failed to capture enough variance to improve model performance, whereas higher values caused prediction errors to compound.

Since all agents could react to buildings with temperatures beyond the critical threshold (T_c), we focused on temperatures within the range of (0, 400] to investigate all decisions made in this range. Therefore, we divided the range into quartiles and used these regions to categorise the decisions of agents for each building. We show the result of this investigation in Figure 5 for where $c = 50$.

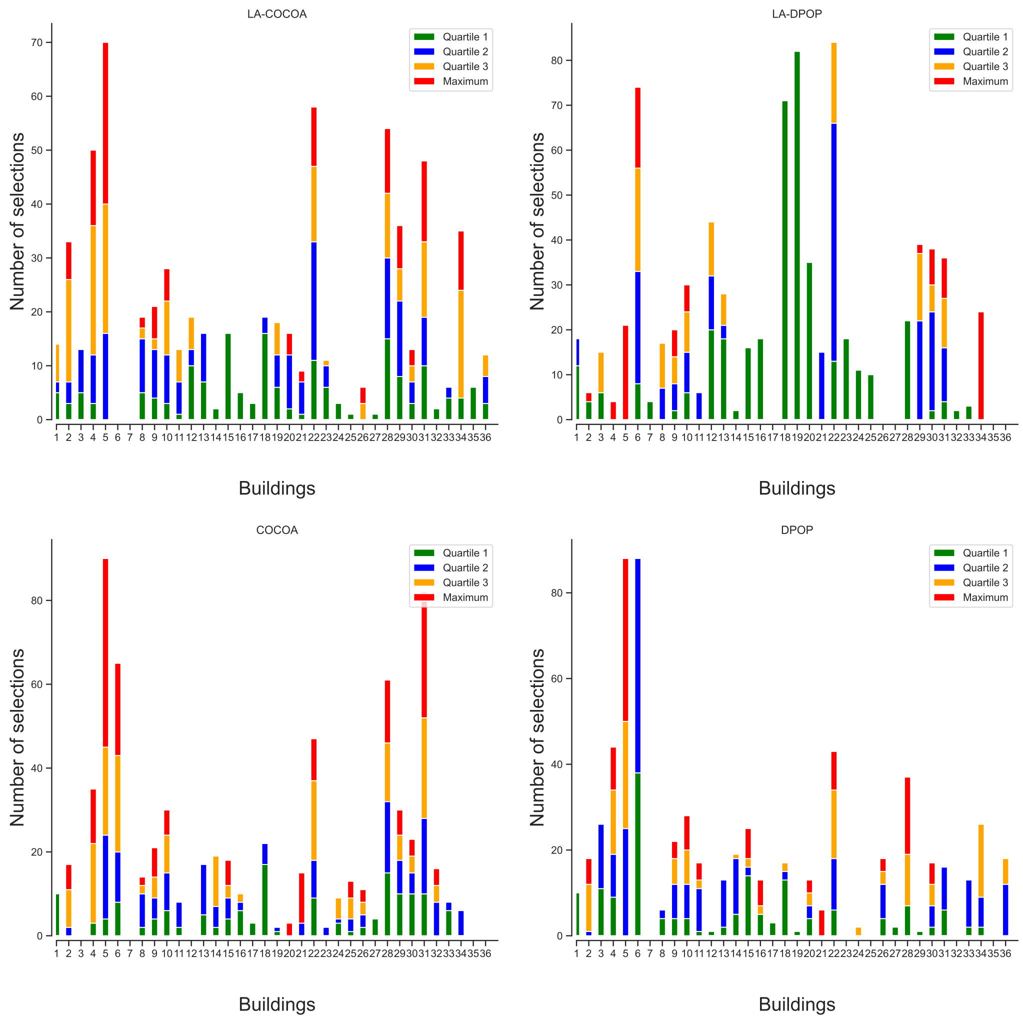

Figure 5. The number of selections each building received when its temperature was in the range (0, 400]. This range is divided into quartiles to indicate the severity of each building's temperature for each selection.

We noticed more building selections whose corresponding temperatures fell in the first quartile in the LA-COCOA and LA-DPOP cases than in the traditional DCOP methods. This observation was more prevalent in the LA-DPOP case and explains the performance of LA-DPOP in Figure 4. Thus, agents often reacted earlier to buildings that were on fire at the initial stages when using our proposed approach than when using the D-DCOP algorithms. The early reaction ensured the agents mostly spent fewer time steps at the buildings.

Overall, our experiment results show that the approach proposed in this study to develop proactive behaviour in agents is effective and enables agents to perform better in dynamic environments. We also show that combining the switching cost and our proactive approach provides better performance in a dynamic environment than relying only on the switching cost. Additionally, proactive D-DCOP algorithms could be developed to rely on the temporal experiences of agents instead of requiring the priors of the dynamic environment.

8. Conclusions

In this study, we have discussed methods enabling proactive agent behaviour in D-DCOPs that do not require prior models of the environment to be provided. We proposed using dynamic multi-agent connections to facilitate the sharing of temporal experiences and learning look-ahead models for decision-making. Our methods were applied to a simulated search-and-extinguish motivating domain to demonstrate their effectiveness. Our results also highlight that using a switching cost, as found in previous studies on PD-DCOPs, could help stabilise proactive agent behaviour in dynamic environments. Further studies could investigate improved cooperative multi-agent learning methods for proactive behaviour in dynamic environments. For instance, learning approaches could be exploited to enable agents to determine experiences to share locally. This will reduce the communication overhead in the proposed approach. Additionally, proactive asynchronous D-DCOP algorithms could be examined to enable fast decision-making in dynamic environments to address the long decision-making times typical of synchronous D-DCOP algorithms. In addition, other applications of the methods proposed in this study could provide further insights into the possibilities and limitations of these methods.

Author Contributions: Conceptualization, B.A.; Methodology, B.A., F.R., and J.Y.; Software, B.A.; Validation, F.R. and J.Y.; formal analysis, B.A., F.R., and J.Y.; writing—original draft preparation B.A.; writing—review and editing, F.R. and J.Y.; supervision, F.R. and J.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This study was supported by the University of Wollongong.

Data Availability Statement: The source codes and data used for our experiments can be found at https://github.com/bbrighttaer/rcrs_ddcop, accessed on 1 May 2024.

Acknowledgments: We would like to thank all reviewers of this study.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

DCOP	Distributed Constraint Optimisation Problem
D-DCOP	Dynamic Distributed Optimisation Problem
PD-DCOP	Proactive Dynamic Distributed Constraint Optimisation Problem
MAS	Multi-Agent System
MDP	Markov Decision Process
DDFS	Distributed Depth First Search
Mobed	Multi-agent Organization with Bounded Edit
HARP	Hybrid Algorithm for Reconstructing Pseudo-trees
RCRS	RoboCup Rescue Simulation
COCOA	Cooperative Constraint Approximation
DPOP	Distributed Pseudo-tree Optimisation Problem
AMQP	Advanced Message Queuing Protocol
LA-COCOA	Lookahead COCOA
LA-DPOP	Lookahead DPOP

References

1. Rust, P.; Picard, G.; Ramparany, F. Using message-passing DCOP algorithms to solve energy-efficient smart environment configuration problems. In Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16), New York, NY, USA, 9–15 July 2016 ; pp. 468–474.
2. Yedidsion, H.; Zivan, R.; Farinelli, A. Applying max-sum to teams of mobile sensing agents. *Eng. Appl. Artif. Intell.* **2018**, *71*, 87–99. [CrossRef]
3. Rybski, P.; Stoeter, S.; Gini, M.; Hougen, D.; Papanikolopoulos, N. Effects of limited bandwidth communications channels on the control of multiple robots. In Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180), Maui, HI, USA, 29 October–3 November 2001; Volume 1, pp. 369–374. [CrossRef]
4. Ramchurn, S.D.; Farinelli, A.; MacArthur, K.S.; Jennings, N.R. Decentralized coordination in RoboCup Rescue. *Comput. J.* **2010**, *53*, 1447–1461. [CrossRef]
5. Padhy, P.; Dash, R.K.; Martinez, K.; Jennings, N.R. 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, Hakodate, Japan, 8–12 May 2006; Association for Computing Machinery: New York, NY, USA, 2006; pp. 1353–1360. [CrossRef]

6. Pujol-Gonzalez, M.; Cerquides, J.; Meseguer, P.; Rodríguez-Aguilar, J.A.; Tambe, M. Engineering the Decentralized Coordination of UAVs with Limited Communication Range. In *Advances in Artificial Intelligence*; Bielza, C., Salmerón, A., Alonso-Betanzos, A., Hidalgo, J.I., Martínez, L., Troncoso, A., Corchado, E., Corchado, J.M., Eds.; Springer: Berlin/Heidelberg, Germany, 2013; pp. 199–208.
7. Junges, R.; Bazzan, A.L.C. 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, Estoril, Portugal, 12–16 May 2008; International Foundation for Autonomous Agents and Multiagent Systems: Richland, SC, USA, 2008; pp. 599–606.
8. 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. In Proceedings of the 19th EPIA Conference on Artificial Intelligence, EPIA 2019, Vila Real, Portugal, 3–6 September 2019; pp. 438–449. [CrossRef]
9. Picard, G. Trajectory Coordination based on Distributed Con-straint Optimization Techniques in Unmanned Air Traffic Management. In Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems, Virtual, 9–13 May 2022; Volume 9, pp. 1065–1073.
10. Fioretto, F.; Pontelli, E.; Yeoh, W. Distributed constraint optimization problems and applications: A survey. *J. Artif. Intell. Res.* **2018**, *61*, 623–698. [CrossRef]
11. Nair, R.; Varakantham, P.; Tambe, M.; Yokoo, M. Networked Distributed POMDPs: A Synthesis of Distributed Constraint Optimization and POMDPs. In Proceedings of the of American Association for Artificial Intelligence, Pittsburgh, PA, USA, 9–13 July 2005; Volume 1, pp. 133–139.
12. Zivan, R.; Yedidsion, H.; Okamoto, S.; Glinton, R.; Sycara, K. Distributed constraint optimization for teams of mobile sensing agents. *Auton. Agents -Multi-Agent Syst.* **2015**, *29*, 495–536. [CrossRef]
13. Hoang, K.D.; Fioretto, F.; Hou, P.; Yokoo, M.; Yeoh, W.; Zivan, R. Proactive dynamic distributed constraint optimization. In Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS, Singapore, 9–13 May 2016; pp. 597–605.
14. Holland, A.; O'Sullivan, B. Weighted super solutions for constraint programs. *Proc. Natl. Conf. Artif. Intell.* **2005**, *1*, 378–383.
15. Jiang, J.; Lu, Z. Learning Attentional Communication for Multi-Agent Cooperation. In *Advances in Neural Information Processing Systems*; Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R., Eds.; Curran Associates, Inc.: Scotland, UK, 2018; Volume 31.
16. 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*, 3807. [CrossRef] [PubMed]
17. Duff, S.; Harland, J.; Thangarajah, J. On proactivity and maintenance goals. *Proc. Int. Conf. Auton. Agents* **2006**, *2006*, 1033–1040. [CrossRef]
18. Hoang, K.D.; Hou, P.; Fioretto, F.; Yeoh, W.; Zivan, R.; Yokoo, M. Infinite-horizon proactive dynamic DCOPs. In Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS, São Paulo, Brazil, 8–12 May 2017; Volume 1, pp. 212–220.
19. Hoang, K.D.; Fioretto, F.; Hou, P.; Yeoh, W.; Yokoo, M.; Zivan, R. Proactive Dynamic Distributed Constraint Optimization Problems. *J. Artif. Intell. Res.* **2022**, *74*, 179–225. [CrossRef]
20. Wallace, R.J.; Freuder, E.C. Stable solutions for dynamic constraint satisfaction problems. In Proceedings of the International Conference on Principles and Practice of Constraint Programming, Pisa, Italy, 26–30 October 1998; Volume 1520, pp. 447–461. [CrossRef]
21. Agyemang, B.; Ren, F.; Yan, J. Distributed Multi-Agent Hierarchy Construction for Dynamic DCOPs in Mobile Sensor Teams. *Hum.-Centric Intell. Syst.* **2023**, *3*, 473–486. [CrossRef]
22. Nguyen, D.T.; Yeoh, W.; Lau, H.C.; Zilberstein, S.; Zhang, C. Decentralized multi-agent reinforcement learning in average-reward dynamic DCOPs. In Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2014, Paris, France, 5–9 May 2014; Volume 2, pp. 1341–1342.
23. Shokoohi, M.; Afsharchi, M.; Shah-Hoseini, H. Dynamic distributed constraint optimization using multi-agent reinforcement learning. *Soft Comput.* **2022**, *26*, 3601–3629. [CrossRef]
24. Xie, S.; Zhang, H.; Yu, H.; Li, Y.; Zhang, Z.; Luo, X. ET-HF: A novel information sharing model to improve multi-agent cooperation. *Knowl.-Based Syst.* **2022**, *257*, 109916. [CrossRef]
25. Sukhbaatar, S.; Szlam, A.; Fergus, R. Learning multiagent communication with backpropagation. In Proceedings of the Advances in Neural Information Processing Systems, Barcelona, Spain, 5–10 December 2016; pp. 2252–2260. Available online: https://dl.acm.org/doi/pdf/10.5555/3157096.3157348 (accessed on 1 May 2024). .
26. Pesce, E.; Montana, G. Learning multi-agent coordination through connectivity-driven communication. *Mach. Learn.* **2023**, *112*, 483–514. [CrossRef]
27. Youssef Hamadi Christian Bessiere, J.Q. Backtracking in Distributed Constraint Networks. In Proceedings of the ECAI 98: 13th European Conference on Artificial Intelligence, Brighton, UK, 23–28 August 1998; pp. 219–223.
28. Sultanik, E.A.; Lass, R.N.; Regli, W.C. Dynamic configuration of agent organizations. In Proceedings of the IJCAI International Joint Conference on Artificial Intelligence, Pasadena, CA, USA, 11–17 July 2009; pp. 305–311.
29. Yeoh, W.; Varakantham, P.; Sun, X.; Koenig, S. Incremental DCOP search algorithms for solving dynamic DCOP problems. In Proceedings of the 2015 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology, WI-IAT 2015, Singapore, 6–9 December 2015; Volume 2, pp. 257–264. [CrossRef]

30. Skinner, C.; Ramchurn, S. The RoboCup Rescue Simulation Platform. In Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems, Toronto, ON, Canada, 9–14 May 2010; International Foundation for Autonomous Agents and Multiagent Systems: Richland, SC, USA 2010; Volume 1, pp. 1647–1648.
31. Sarker, A.; Choudhury, M.; Khan, M.M. A local search based approach to solve continuous DCOPs. In Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS, Virtual, 3–7 May 2021; Volume 2, pp. 1115–1123.
32. Van Leeuwen, C.J.; Pawełczak, P. CoCoA: A non-iterative approach to a local search (A)DCOP Solver. In Proceedings of the 31st AAAI Conference on Artificial Intelligence, AAAI 2017, San Francisco, CA, USA, 4–9 February 2017; pp. 3944–3950. [CrossRef]
33. Petcu, A.; Faltings, B. A Scalable Method for Multiagent Constraint Optimization. In Proceedings of the 19th International Joint Conference on Artificial Intelligence, Scotland, UK, 30 July–5 August 2005; Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 2005; pp. 266–271.