Journal of Artificial Intelligence Research 74 (2022) 179-225 Submitted 11/2021; published 05/2022

# Proactive Dynamic Distributed Constraint Optimization Problems

**Khoi D. Hoang** KHOI.HOANG@WUSTL.EDU
*Department of Computer Science and Engineering*
*Washington University in St. Louis*
*Saint Louis, MO 63130, USA*

**Ferdinando Fioretto** FFIORETT@SYR.EDU
*Department of Electrical Engineering and Computer Science*
*Syracuse University*
*Syracuse, NY 13244, USA*

**Ping Hou** PHOU@CS.NMSU.EDU
*Aurora Innovation*
*Pittsburgh, PA 15222, USA*

**William Yeoh** WYEOH@WUSTL.EDU
*Department of Computer Science and Engineering*
*Washington University in St. Louis*
*Saint Louis, MO 63130, USA*

**Makoto Yokoo** YOKOO@INF.KYUSHU-U.AC.JP
*Department of Informatics*
*Kyushu University*
*Fukuoka, 819-0395, Japan*

**Roie Zivan** ZIVANR@BGU.AC.IL
*Department of Industrial Engineering and Management*
*Ben-Gurion University of the Negev*
*Beer Sheva, 849900, Israel*

## Abstract

The Distributed Constraint Optimization Problem (DCOP) formulation is a powerful tool for modeling multi-agent coordination problems. To solve DCOPs in a dynamic environment, Dynamic DCOPs (D-DCOPs) have been proposed to model the inherent dynamism present in many coordination problems. D-DCOPs solve a sequence of static problems by *reacting* to changes in the environment as the agents observe them. Such reactive approaches ignore knowledge about future changes of the problem. To overcome this limitation, we introduce *Proactive Dynamic DCOPs (PD-DCOPs)*, a novel formalism to model D-DCOPs in the presence of exogenous uncertainty. In contrast to reactive approaches, PD-DCOPs are able to explicitly model possible changes of the problem and take such information into account when solving the dynamically changing problem in a *proactive* manner. The additional expressivity of this formalism allows it to model a wider variety of distributed optimization problems. Our work presents both theoretical and practical contributions that advance current dynamic DCOP models: (*i*) We introduce *Proactive Dynamic DCOPs (PD-DCOPs)*, which explicitly model how the DCOP will change over time; (*ii*) We develop exact and heuristic algorithms to solve PD-DCOPs in a *proactive* manner; (*iii*) We provide theoretical results about the complexity of this new

class of DCOPs; and (*iv*) We empirically evaluate both proactive and reactive algorithms to determine the trade-offs between the two classes. The final contribution is important as our results are the first that identify the characteristics of the problems that the two classes of algorithms excel in.

## 1. Introduction

*Distributed Constraint Optimization Problems (DCOPs)* (Modi et al., 2005; Petcu & Faltings, 2005a; Yeoh & Yokoo, 2012; Fioretto et al., 2018) are problems where agents coordinate their value assignments to maximize the sum of the utility functions. The model can be applied to solve a number of multi-agent coordination problems including distributed meeting scheduling (Maheswaran et al., 2004b), sensor and wireless network coordination (Farinelli et al., 2008; Yeoh & Yokoo, 2012), multi-robot coordination (Zivan et al., 2015), smart grid optimization (Kumar et al., 2009; Miller et al., 2012; Fioretto et al., 2017b), coalition structure generation (Ueda et al., 2010), smart home automation (Rust et al., 2016; Fioretto et al., 2017a), and cloud computing applications (Paulos et al., 2019; Hoang et al., 2019).

When DCOPs were introduced more than a decade ago, research efforts were initially focused on the investigation of different algorithmic paradigms to solve the problem, including exact search-based methods (Modi et al., 2005; Gershman et al., 2009; Yeoh et al., 2010; Gutierrez et al., 2011), exact inference-based methods (Petcu & Faltings, 2005a; Vinyals et al., 2011), exact declarative methods (Hatano & Hirayama, 2013; Le et al., 2017), approximate search-based methods (Maheswaran et al., 2004a; Zhang et al., 2005; Zivan et al., 2014; Yu et al., 2017; van Leeuwen & Pawelczak, 2017; Chen et al., 2018; Hoang et al., 2018), approximate inference-based methods (Farinelli et al., 2014; Zivan & Peled, 2012; Zivan et al., 2017; Cohen & Zivan, 2018; Cohen et al., 2020; Hoang et al., 2020), and approximate sampling-based methods (Ottens et al., 2017; Nguyen et al., 2019).

Typically, these DCOP algorithms address and solve a single (static) problem as they assume that the problem does not change over time. However, this assumption limits the capability of DCOP to solve and model the problems in dynamic environments. Thus, researchers have proposed the *Dynamic DCOP (D-DCOP)* model (Petcu & Faltings, 2005b, 2007; Lass et al., 2008; Yeoh et al., 2015), where constraints can change during the problem solving process. Existing D-DCOP algorithms share a common assumption that information on how the problem might change is unavailable. As such, they are all *reactive* algorithms, that is, they are *online* algorithms reacting to the changes of the problem by solving the DCOP every time such changes occur (Petcu & Faltings, 2005b; Sultanik et al., 2009; Yeoh et al., 2015). However, in several applications, the information on how the problem might change is indeed available or predictable within some degree of uncertainty. Therefore, in this article, we are interested in investigating *proactive* D-DCOP algorithms, which are *offline* algorithms that take into account prior knowledge on the evolution of the problem when finding solutions.

Our contributions in this article are the following: (*i*) We introduce *Proactive Dynamic DCOPs (PD-DCOPs)*, which explicitly model how the DCOP might change over time; (*ii*) We develop exact and heuristic algorithms to solve PD-DCOPs in a *proactive* manner; (*iii*) We provide theoretical results about the complexity of this new class of DCOPs; and (*iv*) We empirically evaluate both proactive and reactive algorithms to determine the trade-

offs between the two classes. The final contribution is important as our results are the first that identify the characteristics of the problems that the two classes of algorithms excel in.[1]

The structure of this article is as follows: In Section 2, we provide the background for DCOPs, D-DCOPs, relevant DCOP algorithms, and Markov chains. Next, we present a motivating domain in Section 3, the PD-DCOP model in Section 4 and introduce three approaches to solve PD-DCOPs in Section 5. We then discuss the theoretical properties of PD-DCOPs in Section 6 and related work in Section 7. Finally, we present the experimental results in Section 8 and conclude in Section 9.

## 2. Background

In this section, we provide a brief overview of DCOPs, D-DCOPs, relevant DCOP algorithms, and Markov chains.

### 2.1 DCOPs

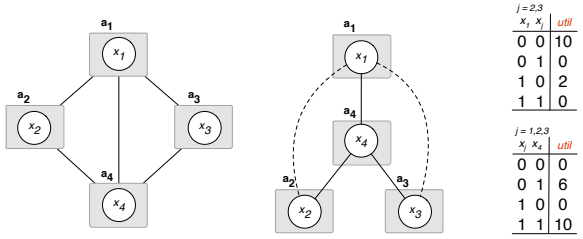
A *Distributed Constraint Optimization Problem (DCOP)* (Modi et al., 2005; Petcu & Faltings, 2005a; Yeoh & Yokoo, 2012; Fioretto et al., 2018) is a tuple $\langle \mathbf{A}, \mathbf{X}, \mathbf{D}, \mathbf{F}, \alpha \rangle$, where:

- $\mathbf{A} = \{a_i\}_{i=1}^{p}$ is a set of *agents*.
- $\mathbf{X} = \{x_i\}_{i=1}^{n}$ is a set of *decision variables*.
- $\mathbf{D} = \{D_x\}_{x \in \mathbf{X}}$ is a set of finite *domains*, where each variable $x \in \mathbf{X}$ takes values from the set $D_x \in \mathbf{D}$.
- $\mathbf{F} = \{f_i\}_{i=1}^{m}$ is a set of *utility functions*, each defined over a set of decision variables: $f_i : \prod_{x \in \mathbf{x}^{f_i}} D_x \rightarrow \mathbb{R}_0^+ \cup \{-\infty\}$, where infeasible configurations have $-\infty$ utilities and $\mathbf{x}^{f_i} \subseteq \mathbf{X}$ is the *scope* of $f_i$.[2]
- $\alpha : \mathbf{X} \rightarrow \mathbf{A}$ is a function that associates each decision variable to one agent.

A *solution* $\sigma$ is a value assignment to a set $\mathbf{x}_\sigma \subseteq \mathbf{X}$ of decision variables that is consistent with their respective domains. The utility $\mathbf{F}(\sigma) = \sum_{f \in \mathbf{F}, \mathbf{x}^f \subseteq \mathbf{x}_\sigma} f(\sigma)$ is the sum of the utilities across all applicable utility functions in $\sigma$. A solution $\sigma$ is *complete* if $\mathbf{x}_\sigma = \mathbf{X}$. The goal of a DCOP is to find an optimal complete solution $\mathbf{x}^* = \text{argmax}_{\mathbf{x}} \mathbf{F}(\mathbf{x})$.

Given a DCOP $P$, $G = (\mathbf{X}, E)$ is the *constraint graph* of $P$, where $\{x, y\} \in E$ iff $\exists f_i \in \mathbf{F}$ such that $\{x, y\} = \mathbf{x}^{f_i}$.[3] A *pseudo-tree* arrangement for $G$ is a *spanning tree* $T = \langle \mathbf{X}, E_T \rangle$ of $G$ such that if $f_i \in \mathbf{F}$ and $\{x, y\} \subseteq \mathbf{x}^{f_i}$, then $x$ and $y$ appear in the same branch of $T$. We use $N(a_i) = \{a_j \in \mathbf{A} \mid \{x_i, x_j\} \in E\}$ to denote the neighbors of agent $a_i$. Figure 1 depicts: (a) the constraint graph of a DCOP with a set of agents $\{a_1, a_2, a_3, a_4\}$, each controlling a variable with domain {0,1}, (b) a pseudo-tree (solid lines identify tree edges connecting parent-children nodes, dotted lines refer to back-edges connecting pseudo-parents and its pseudo-children), and (c) the DCOP utility functions in tabular form.

1. This article extends our previous conference papers (Hoang et al., 2016, 2017) by: (1) combining PD-DCOPs and IPD-DCOPs into a unified model; (2) elaborating on theoretical properties with complete proofs; (3) proposing new scalable algorithms; and (4) presenting more extensive experimental results.
2. The scope of a function is the set of variables that are associated with the function.
3. We assume that the utility functions are binary between two decision variables.




(a) Constraint Graph (b) Pseudo-tree (c) Utility Functions

Figure 1: Example of DCOP

### 2.2 Dynamic DCOPs

A *Dynamic DCOP (D-DCOP)* (Petcu & Faltings, 2005b, 2007; Lass et al., 2008; Yeoh et al., 2015) is defined as a sequence of DCOPs with changes between them. Changes between DCOPs occur over time due to addition or removal of variables, addition or removal of values in the variable's domain, addition or removal of utility functions, and increase or decrease in the utility values. Solving a D-DCOP optimally means finding a utility-maximal solution for each DCOP in the sequence. Therefore, this approach is *reactive* since solving each DCOP in the sequence does not consider future changes. Its advantage is that solving a D-DCOP is no harder than solving $h$ DCOPs, where $h$ is the horizon of the problem. Researchers have used this approach to solve D-DCOPs, where they introduce search- and inference-based approaches that are able to reuse information from previous DCOPs to speed up the search for the solution for the current DCOP (Petcu & Faltings, 2005b; Yeoh et al., 2015). Alternatively, a *proactive* approach predicts future changes in the D-DCOP and finds robust solutions that require little or no changes in the sequence of DCOP solutions despite future changes to the DCOP.

Researchers have also proposed other models for D-DCOPs including a model where agents have deadlines to choose their values (Petcu & Faltings, 2007), a model where agents can have imperfect knowledge about their environment (Lass et al., 2008), and a model where changes in the constraint graph depends on the value assignments of agents (Zivan et al., 2015).

### 2.3 DCOP Algorithms: DPOP, S-DPOP, and MGM

We now introduce three relevant DCOP algorithms that are the main component of several PD-DCOP algorithms.

#### 2.3.1 Distributed Pseudo-tree Optimization Procedure (DPOP)

The *Distributed Pseudo-tree Optimization Procedure (DPOP)* (Petcu & Faltings, 2005a) is a complete *inference algorithm* composed of three phases:

- *Pseudo-tree Generation:* The agents build a pseudo-tree (Hamadi et al., 1998).
- *UTIL Propagation:* Each agent, starting from the leaves of the pseudo-tree, computes the optimal sum of utilities in its subtree for each value combination of variables in its

separator.[4] It does so by adding the utilities of its functions with the variables in its separator and the utilities in the UTIL messages received from its children agents, and projecting out its own variables by optimizing over them.

- *VALUE Propagation:* Each agent, starting from the pseudo-tree root, determines the optimal value for its variables. The root agent does so by choosing the values of its variables from its UTIL computations.

#### 2.3.2 Super-stabilizing DPOP (S-DPOP)

*Super-stabilizing DPOP (S-DPOP)* (Petcu & Faltings, 2005b) is a self-stabilizing extension of DPOP, where the agents restart the DPOP phases when they detect changes in the problem. S-DPOP makes use of information that is not affected by the changes in the problem.

#### 2.3.3 Maximum Gain Message (MGM)

*Maximum Gain Message (MGM)* (Maheswaran et al., 2004a) is a local search algorithm that improves the initial solution in an iterative manner. In MGM, each agent starts with a random assignment to the variables it controls and then sends this initial assignment to its neighbors. After receiving the assignments of all neighbors, the agent searches for all possible values in its domain that can improve the current local constraint utilities and computes the highest improvement in utilities. Then, the agent shares the highest improvement value as the gain information with its neighbors and decides to change the assignment if it has the largest gain in the neighborhood. After changing to the new value, the agent sends messages to the neighbors to inform them of the new assignment. This process repeats until a stopping condition is met.

### 2.4 Markov Chains

We now introduce *Markov chains* and their *stationary distribution*, which are used in one of the approaches to solve PD-DCOPs. A Markov chain (Gallager, 2013) is a sequence of random variables $\langle x^0, x^1, \ldots, x^T \rangle$ that share the same state space, and the transition from $x^{t-1}$ to $x^t$ depends exclusively on the previous state. More formally,

$$\Pr(x^t = j \mid x^{t-1} = i, x^{t-2} = r, \ldots, x^0 = s) = \Pr(x^t = j \mid x^{t-1} = i)$$

for all time steps $t > 0$, where $i, j, r$ and $s$ are the values in the state space. We use Pr to denote the probability measure. A Markov chain is said to be *time-homogeneous* if the transition $P_{ij} = \Pr(x^t = j \mid x^{t-1} = i)$ is identical for all time steps $t$. A time-homogeneous Markov chain converges to a *stationary distribution* $p^*$ when $p^{t-1} \cdot P = p^t = p^*$. The probability distribution $p^t$ is the distribution over all states at time $t$ in the chain, and $P$ is the transition matrix where each element $P_{ij}$ is the transition probability from state $i$ to state $j$.

A state $j$ is said to be *accessible* from $i$, denoted by $i \to j$, if there exists a sequence of $t$-step transitions ($t \geq 1$) such that $\Pr(x^t = j \mid x^0 = i) = P^t_{ij} > 0$. Two states $i$ and $j$

4. The separator of $x_i$ contains all ancestors of $x_i$ in the pseudo-tree that are connected to $x_i$ or one of its descendants.

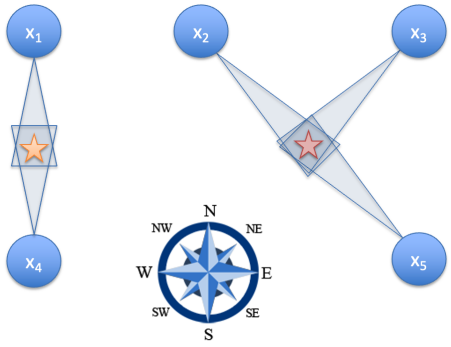



Figure 2: Distributed Radar Coordination and Scheduling Problem

*communicate*, denoted by $i \leftrightarrow j$, if both states are accessible from each other. A *class* $C$ of communicating states is a non-empty set of states where each state $i \in C$ communicates with every other state $j \in C \setminus \{i\}$ but does not communicate with any state $j \notin C$. The *period* of a state $i$, $d(i) = \gcd\{t : P_{ii}^t > 0\}$, is the greatest common divisor (gcd) of the time steps $t$ for which $P_{ii}^t > 0$. The state is said to be *aperiodic* if it has period $d(i) = 1$, and *periodic* if $d(i) > 1$. All states in the same class have the same period. If all states of a Markov chain form a single class, then the chain has the period of the class. A state $i$ is said to be *recurrent* if it is accessible from all states $j$ that are accessible from $i$. In other words, $i \to j$ implies $j \to i$. Otherwise, it is *transient*. All states in the same class are either recurrent or transient. A class of states is said to be *ergodic* if it is both recurrent and aperiodic. A *unichain* is a chain that contains a single recurrent class and may be some transient states. A unichain is *ergodic* if the recurrent class is ergodic.

In this article, we consider Markov chains that are guaranteed to converge to a *unique* stationary distribution $p^*$ given any initial distribution. Specifically, the Markov chain follows one of the following (from strict to loose) conditions: *(i)* $P_{ij} > 0$ for all states $i$ and $j$, *(ii)* all states are in one single ergodic class and they are ergodic, *(iii)* the Markov chain is an ergodic unichain.

## 3. Distributed Radar Coordination and Scheduling Problem

In this section, we motivate our work using the *Distributed Radar Coordination and Scheduling Problem (DRCSP)*, which is based on NetRad, a real-time weather radar sensor system (Deng & An, 2020; Kim et al., 2011; Zink et al., 2005). The main component of the NetRad system is a set of meteorological command and controls (MCCs), where each MCC controls a set of radars with limited sensing range. Instead of operating in "sit and spin" mode, where each radar independently takes 360-degree volume scans, the radars in NetRad are tasked by the MCCs to scan a specific area of interest in a coordinated fashion. For example, in Figure 2, the system with five radars are scanning the area with two weather phenomena, represented as a yellow star and a red star. The MCCs gather moment data from the radars and then generate the best sensing strategy for the radars by collectively solving a distributed coordination and scheduling problem, which is a DRCSP. The goal

of a DRCSP is to find a coordination strategy that maximizes the aggregated utility by scanning the highest-utility phenomena in the area.

While NetRad was originally designed to sense and detect weather phenomena such as tornados, thunderstorms, and hurricanes, it is hard to predict those phenomena in advance so that the system can deliver better sensing strategies. In contrast, precipitation has been widely modeled as stochastic processes (Katz, 1977; Richardson, 1981; Wilks, 1992), and it is known to be associated with many phenomena at locations of interest (Trenberth, 2011; Xu et al., 2012; Moore et al., 2003). Therefore, instead of directly sensing the weather phenomena, the goal of the DRCSP is to generate strategies for the radars such that they best sense the precipitation based on the prediction of the precipitation in the area. Throughout this paper, we will use this problem to motivate the PD-DCOP model.

## 4. PD-DCOP Model

We now describe the *Proactive Dynamic DCOP (PD-DCOP)* model that takes into account the information on how the problem might change dynamically. Formally, a PD-DCOP is a tuple $\langle \mathbf{A}, \mathbf{X}, \mathbf{Y}, \mathbf{D}, \mathbf{\Omega}, \mathbf{F}, p^0_{\mathbf{Y}}, \mathbf{T}, \gamma, h, c, \alpha \rangle$, where:

- $\mathbf{A} = \{a_i\}_{i=1}^{p}$ is a set of *agents*.
- $\mathbf{X} = \{x_i\}_{i=1}^{n}$ is a set of *decision variables*.
- $\mathbf{Y} = \{y_i\}_{i=1}^{m}$ is a set of *random variables*.
- $\mathbf{D} = \{D_x\}_{x \in \mathbf{X}}$ is a set of finite *domains of the decision variables*, where each variable $x \in \mathbf{X}$ takes values from the set $D_x \in \mathbf{D}$.
- $\mathbf{\Omega} = \{\Omega_y\}_{y \in \mathbf{Y}}$ is a set of finite *domains of the random variables*, where each variable $y \in \mathbf{Y}$ takes values from the set $\Omega_y \in \mathbf{\Omega}$.
- $\mathbf{F} = \{f_i\}_{i=1}^{k}$ is a set of *utility functions*, each defined over a mixed set of decision and random variables: $f_i : \prod_{x \in \mathbf{X} \cap \mathbf{x}^{f_i}} D_x \times \prod_{y \in \mathbf{Y} \cap \mathbf{x}^{f_i}} \Omega_y \rightarrow \mathbb{R}_0^+ \cup \{-\infty\}$, where infeasible configurations have $-\infty$ utilities and $\mathbf{x}^{f_i} \subseteq \mathbf{X} \cup \mathbf{Y}$ is the scope of $f_i$. We divide the set of utility functions into two sets: $\mathbf{F_X} = \{f_x\}$, where $\mathbf{x}^{f_x} \cap \mathbf{Y} = \emptyset$, and $\mathbf{F_Y} = \{f_y\}$, where $\mathbf{x}^{f_y} \cap \mathbf{Y} \neq \emptyset$. Note that $\mathbf{F_Y} \cup \mathbf{F_Y} = \mathbf{F}$ and $\mathbf{F_X} \cap \mathbf{F_Y} = \emptyset$.
- $p^0_{\mathbf{Y}} = \{p^0_y\}_{y \in \mathbf{Y}}$ is a set of initial *probability distributions*.
- $\mathbf{T} = \{T_y\}_{y \in \mathbf{Y}}$ is a set of *transition functions*: $T_y : \Omega_y \times \Omega_y \rightarrow [0, 1]$.
- $\gamma \in [0, 1]$ is a *discount factor*.
- $h \in \mathbb{N}$ is a finite *horizon*.
- $c \in \mathbb{R}_0^+$ is a *switching cost*, which is the cost associated with the change in the value of each decision variable from one time step to the next.[5]
- $\alpha : \mathbf{X} \rightarrow \mathbf{A}$ is a function that associates each decision variable to one agent.

Throughout this article, we assume that: *(i)* each agent controls exactly one decision variable and thus use the terms "agent" and "decision variable" interchangeably; and *(ii)* each utility function is associated with at most one random variable. If multiple random variables are associated with a utility function, w.l.o.g., they can be merged into a single variable.

5. For simplicity, we assume that the switching cost is identical across all decision variables.

The goal of a PD-DCOP is to find a sequence of $h+1$ assignments $\mathbf{x}^*$ for all the decision variables in $\mathbf{X}$:

$$\mathbf{x}^* = \underset{\mathbf{x}=\langle \mathbf{x}^0,\dots,\mathbf{x}^h\rangle \in \Sigma^{h+1}}{\operatorname{argmax}} \mathcal{F}^h(\mathbf{x}) \tag{1}$$

$$\mathcal{F}^h(\mathbf{x}) = \underbrace{\sum_{t=0}^{h-1} \gamma^t \left[\mathcal{F}_x^t(\mathbf{x}^t) + \mathcal{F}_y^t(\mathbf{x}^t)\right]}_{\mathbf{P}} - \underbrace{\sum_{t=0}^{h-1} \gamma^t \left[c \cdot \Delta(\mathbf{x}^t, \mathbf{x}^{t+1})\right]}_{\mathbf{Q}} + \underbrace{\tilde{\mathcal{F}}_x(\mathbf{x}^h) + \tilde{\mathcal{F}}_y(\mathbf{x}^h)}_{\mathbf{R}} \tag{2}$$

, where $\Sigma$ is the assignment space for the decision variables of the PD-DCOP. The first term $\mathbf{P}$ refers to the optimization over the first $h$ time steps, with:

$$\mathcal{F}_x^t(\mathbf{x}) = \sum_{f_i \in \mathbf{F_X}} f_i(\mathbf{x}_i) \tag{3}$$

$$\mathcal{F}_y^t(\mathbf{x}) = \sum_{f_i \in \mathbf{F_Y}} \sum_{\omega \in \Omega_{y_i}} f_i(\mathbf{x}_i|_{y_i=\omega}) \cdot p_{y_i}^t(\omega) \tag{4}$$

where $\mathbf{x}_i$ is an assignment for all decision variables in the scope $\mathbf{x}^{f_i}$ of utility function $f_i$; we write $\mathbf{x}_i|_{y_i=\omega}$ to indicate that the random variable $y_i \in \mathbf{x}^{f_i}$ takes on the value $\omega \in \Omega_{y_i}$; $p_{y_i}^t(\omega)$ is the probability of the random variable $y_i$ taking value $\omega$ at time $t$, and is defined as:

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

The second term $\mathbf{Q}$ takes into account the penalty due to changes in decision variables' values during the optimization process, where $\Delta : \Sigma \times \Sigma \to \mathbb{R}_0^+$ is a penalty function that takes into account the difference in the decision variable assignments between two time steps. If one of the assignments is NULL, the penalty function $\Delta$ will return 0.

Lastly, $\mathbf{R}$ refers to the optimization from time step $h$ onward where the solution to the problem at time $h$ remains unchanged for all subsequent problems. Since the nature of discounting in PD-DCOPs is associated with the discount factor $\gamma$, it gives rise to two cases: $\gamma < 1$ and $\gamma = 1$. While the sum of discounted utilities can be optimized using Bellman equation in the former case, we take into account the Markov chain convergence property in the latter case. Thus, we propose two algorithms to optimize $\mathbf{R}$ for two cases $\gamma < 1$ and $\gamma = 1$:

- ***Cumulative Discounted Future Utilities (CDFU)***: In many problems, future utilities are less important than the utility at the current time step (i.e., $\gamma < 1$). Thus, we propose CDFU to optimize $\mathbf{R}$ as the sum of cumulative discounted future utilities. The CDFU algorithm optimizes $\mathbf{R}$ using Equations (6), (7), and (8), which will be introduced in Section 5.
- ***Markov Chain Convergence (MCC)***: In problems where future and current utilities are equally weighted (i.e., $\gamma = 1$), we propose the MCC algorithm that takes into account the convergence property of Markov chains (Gallager, 2013). In this approach, we model each random variable as a Markov chain, and we assume that each Markov chain is

guaranteed to converge to a unique stationary distribution given any initial probability distribution.[6] The MCC algorithm optimizes $\mathbf{R}$ with the stationary distribution of the Markov chains using Equations (9), (10), (11), and (12), which will be introduced in Section 5.

In summary, the goal of a PD-DCOP is to find a value assignment to all the decision variables such that it maximizes the sum of three terms $\mathbf{P}$, $\mathbf{Q}$, and $\mathbf{R}$ (Equation 2). The first term, $\mathbf{P}$, maximizes the sum of cumulative discounted utilities for the functions that do not involve random variables ($\mathcal{F}_x$) and cumulative expected discounted random utilities ($\mathcal{F}_y$) in the first $h$ time steps. The second term, $\mathbf{Q}$, minimizes the cumulative discounted penalty costs incurred by solutions changing over time. The last term, $\mathbf{R}$, maximizes the future utilities for all problems from the time step $h$ onward.

While the PD-DCOP model can be used to capture the presence of exogenous factors in the dynamic aspect of the problem, note that it can also model dynamic changes to the DCOP constraint graph through the transition functions. In particular, the deletion of a constraint will force the random variable associated with that constraint to transit to a 0 utility value for all decision variables; the addition of a constraint can be handled by defining a 0 utility constraint in the model from the start and updating its utility when the constraint is added.

### 4.1 Modeling DRCSP as a PD-DCOP

Since DRCSP is inherently a distributed constraint reasoning problem and PD-DCOPs have the capability to model dynamic and uncertainty events, DRCSP can naturally be modeled using the PD-DCOP formulation. Specifically,

- The set of MCCs is modeled as the set of agents $\mathbf{A} = \{a_i\}_{i=1}^{p}$.
- The set of radars is modeled as the set of decision variables $\mathbf{X} = \{x_i\}_{i=1}^{n}$.
- The precipitation events are modeled as the set of random variables $\mathbf{Y} = \{y_i\}_{i=1}^{m}$.
- The scanning directions of the radars are modeled as the set of decision variable domains $\mathbf{D} = \{D_x\}_{x \in \mathbf{X}}$.
- The set of precipitation level is modeled as the set of random variable domains $\mathbf{\Omega} = \{\Omega_y\}_{y \in \mathbf{Y}}$.
- The set of utility function associated with radars sensing the precipitation is modeled as the set of utility functions $\mathbf{F} = \{f_i\}_{i=1}^{k}$.
- The set of initial distribution of precipitation is modeled as $p^0_{\mathbf{Y}} = \{p^0_y\}_{y \in \mathbf{Y}}$.
- The probability transition of the precipitation is modeled as $\mathbf{T} = \{T_y\}_{y \in \mathbf{Y}}$, where $T_y : \Omega_y \times \Omega_y \to [0, 1]$.
- The discount factor is $\gamma \in [0, 1]$.
- The duration of interest is horizon $h \in \mathbb{N}$.
- The cost of changing the sensing direction of radars (e.g., energy consumption) is modeled as the switching cost $c \in \mathbb{R}^+_0$. Since we assume the sensors are of the same type, the switching cost is identical across all sensors.
- The membership of each radar to a specific MCC is modeled as the mapping function $\alpha : \mathbf{X} \to \mathbf{A}$ between a decision variable and an agent.

6. The conditions for such convergence are discussed in Subsection 2.4.

**Procedure** SolvePD-DCOP()

```
1 if γ < 1 then
2 |   SolveMultiDCOPs(h̄ = h, x^{h̄+1} = null)
3 else
4 |   s^h ← SolveHorizonDCOP()
5 |   SolveMultiDCOPs(h̄ = h − 1, x^{h̄+1} = s^h)
```

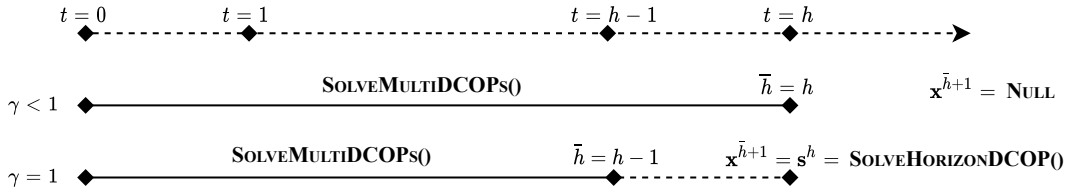



Figure 3: Illustration of the SolvePD-DCOP Procedure

## 5. PD-DCOP Algorithms

We are now ready to describe the two approaches introduced in the previous section to solve PD-DCOPs: *Cumulative Discounted Future Utilities (CDFU)* and *Markov Chain Convergence (MCC)*. A comparison between the two methods is illustrated in Procedure SolvePD-DCOP and Figure 3. Both CDFU and MCC approaches are similar in that they call Procedure SolveMultiDCOPs to solve a number of consecutive DCOPs starting from time step 0. Procedure SolveMultiDCOPs accepts two parameters: $\bar{h}$ and $\mathbf{x}^{\bar{h}+1}$. Parameter $\bar{h}$ indicates the time step of the last DCOP in SolveMultiDCOPs. In other words, SolveMultiDCOPs solves the DCOPs from time step 0 to time step $\bar{h}$. Parameter $\mathbf{x}^{\bar{h}+1}$ indicates the solution to the problem at time step $\bar{h}+1$ if it is not null.[7] The two approaches are different in that one of them calls Procedure SolveHorizonDCOP to solve for the problem at horizon $t = h$ before running SolveMultiDCOPs. In more detail:

- ***Cumulative Discounted Future Utilities (CDFU)***: If $\gamma < 1$, the CDFU approach transforms the problem at time step $h$ and optimizes $\mathbf{R}$ in Equation (2) by computing the cumulative discounted and cumulative discounted expected utilities from horizon $h$ onward:

$$\tilde{\mathcal{F}}_x(\mathbf{x}) = \frac{\gamma^h}{1-\gamma}\mathcal{F}_x^h(\mathbf{x}) \tag{6}$$

$$\tilde{\mathcal{F}}_y(\mathbf{x}) = \sum_{f_i \in \mathbf{F_Y}} \sum_{\omega \in \Omega_{y_i}} \tilde{f}_i(\mathbf{x}_i|_{y_i=\omega}) \cdot p_{y_i}^h(\omega) \tag{7}$$

$$\tilde{f}_i(\mathbf{x}_i|_{y_i=\omega}) = \gamma^h \cdot f_i(\mathbf{x}_i|_{y_i=\omega}) + \gamma \sum_{\omega' \in \Omega_{y_i}} T_{y_i}(\omega, \omega') \cdot \tilde{f}_i(\mathbf{x}_i|_{y_i=\omega'}) \tag{8}$$

  After that, it takes into account the problems from time step 0 to time step $h$ and solve them together by running SolveMultiDCOPs with arguments $\bar{h} = h$ and $\mathbf{x}^{\bar{h}+1} = \text{null}$

7. We do not provide pseudocode for Procedure SolveMultiDCOPs since PD-DCOP algorithms have different ways to implement this procedure.

(lines 1-2). We set $\mathbf{x}^{\bar{h}+1} = \text{NULL}$ since CDFU does not constrain the solution at time step $\bar{h}+1$.

- ***Markov Chain Convergence (MCC)***: If $\gamma = 1$, the MCC approach transforms the problem at $h$ and optimizes $\mathbf{R}$ in Equation (2) by using the stationary distribution of Markov chains in the PD-DCOP:[8]

$$\tilde{\mathcal{F}}_x(\mathbf{x}) = \mathcal{F}_x^h(\mathbf{x}) \tag{9}$$

$$\tilde{\mathcal{F}}_y(\mathbf{x}) = \sum_{f_i \in \mathbf{F_Y}} \sum_{\omega \in \Omega_{y_i}} f_i(\mathbf{x}_i|_{y_i=\omega}) \cdot p^*_{y_i}(\omega) \tag{10}$$

where $p^*_y(\omega)$ is the probability of random variable $y$ having state $\omega$ in the stationary distribution, and $p^*_{y_i}$ is the solution of the following system of linear equations:

$$\sum_{\omega' \in \Omega_{y_i}} p^*_{y_i}(\omega') \cdot T_{y_i}(\omega', \omega) = p^*_{y_i}(\omega) \tag{11}$$

$$\sum_{\omega \in \Omega_{y_i}} p^*_{y_i}(\omega) = 1 \tag{12}$$

After that, the MCC approach solves for the solution $\mathbf{s}^h$ to the problem at horizon $h$ by calling SolveHorizonDCOP (lines 3-4). It then solves the problems from time step 0 to time step $h-1$ by running SolveMultiDCOPs with $\bar{h} = h-1$ and $\mathbf{x}^{\bar{h}+1} = \mathbf{s}^h$ (line 5). While solving the problems from time step 0 to time step $h-1$, SolveMultiDCOPs takes into account the switching cost between the solution at time step $h-1$ and the solution $\mathbf{s}^h$ at time step $h$.

We now describe how the MCC approach solves the problem at time step $h$ by calling SolveHorizonDCOP in more detail. This function solves for the solution at time step $h$ by using the stationary distribution of Markov chains. Since the transition function $T_y \in \mathbf{T}$ of each random variable $y \in \mathbf{Y}$ is independent of the transition functions of other random variables, each random variable in the PD-DCOP forms an independent Markov chain. Furthermore, these Markov chains are *time-homogeneous*–the transition functions are identical for all time steps – and has *finite state spaces*–the domain of each random variable $y$ is a finite set $\Omega_y \in \mathbf{\Omega}$. In this paper, we assume that each Markov chain in PD-DCOPs will converge to a *unique* stationary distribution given any initial distribution. The computation of the unique distribution for each random variable $y$, computed using a system of linear equations (Equations 11 and 12), can be done independently by each agent $a$ that controls the decision variable $x$ that is constrained with random variable $y$. In other words, the computation for random variable $y$ is performed by the agent $a$ such that $\exists\, x \in \mathbf{X}, f \in \mathbf{F_Y} : y \in \mathbf{x}^f \wedge x \in \mathbf{x}^f \wedge \alpha(x) = a$.

Once the stationary distribution of each random variable is found, the agents reformulate the constraints between decision and random variables into constraints between decision variables only. Specifically, for each constraint $f \in \mathbf{F}_Y$ between decision variables

8. When $\gamma = 1$, solving the problem at time step $h$ with stationary distribution will maximize the expected utility from that time step onward (see Theorem 2).

$\mathbf{x}$ and a random variable $y$, the following new constraint is created:

$$F^h(\mathbf{x}) = \sum_{\omega \in \Omega_y} f(\mathbf{x}|_{y=\omega}) \cdot p_y^*(\omega) \tag{13}$$

where $p_y^*(\omega)$ is the probability of random variable $y$ having state $\omega$ in the stationary distribution. Note that the new scope of this new constraint is exclusively the decision variables $\mathbf{x}$. The effect of this post-processing step is that it removes all random variables and reformulates the PD-DCOP into a regular DCOP with exclusively decision variables. After this step, agents will run any off-the-shelf algorithm to solve the regular DCOP.

In summary, the CDFU and MCC approaches are similar in that they run SolveMultiDCOPs($\bar{h}, \mathbf{x}^{\bar{h}+1}$) to solve the problems from time step 0 to time step $\bar{h}$. The key difference is that CDFU runs the function to find solutions from time steps 0 to $h$ while MCC runs the function to find solutions from time steps 0 to $h-1$. To find the solution for time step $h$, MCC runs SolveHorizonDCOP instead.

To implement SolveMultiDCOPs, we propose two approaches: (1) An *Exact* approach that transforms a PD-DCOP into an equivalent DCOP and solves it using any off-the-shelf exact DCOP algorithm, and (2) a *Heuristic* approach that transforms a PD-DCOP into an equivalent dynamic DCOP and solves it using any off-the-shelf dynamic DCOP algorithm. We describe these approaches in Sections 5.1 and 5.2, respectively. Later, in Section 8, we will introduce different PD-DCOP algorithms that are based on these approaches.

### 5.1 Exact Approach

We now describe an exact approach that transforms a PD-DCOP into an equivalent DCOP and solves it using any off-the-shelf DCOP algorithm. Since the transition of each random variable is independent of the assignment of values to decision variables, this problem can be viewed as a Markov chain. Thus, it is possible to collapse an entire PD-DCOP into a single DCOP, where (1) each utility function $F_i$ in this new DCOP captures the sum of utilities of the utility function $f_i \in \mathbf{F}$ across all time steps, and (2) the domain of each decision variable is the set of all possible combinations of values of that decision variable across all time steps. However, this process needs to be done in a distributed manner.

As we mentioned in Section 4, the utility functions are divided into two types: (1) The functions $f_i \in \mathbf{F_X}$ whose scope $\mathbf{x}^{f_i} \cap \mathbf{Y} = \emptyset$ includes exclusively decision variables, and (2) the functions $f_i \in \mathbf{F_Y}$ whose scope $\mathbf{x}^{f_i} \cap \mathbf{Y} \neq \emptyset$ includes one random variable. In both cases, let $\mathbf{x}_i = \langle \mathbf{x}_i^0, \ldots, \mathbf{x}_i^{\bar{h}} \rangle$ denote the vector of value assignments to all *decision* variables in $\mathbf{x}^{f_i}$ for each time step.

Each function $f_i \in \mathbf{F_X}$ whose scope includes only decision variables can be replaced by a function $F_i$:

$$F_i(\mathbf{x}_i) = \sum_{t=0}^{\bar{h}} F_i^t(\mathbf{x}_i^t) \tag{14}$$

where:

$$F_i^t(\mathbf{x}_i^t) = \begin{cases} \dfrac{\gamma^h}{1-\gamma} f_i(\mathbf{x}_i^h) & \text{if } t = \bar{h} \text{ and } \bar{h} = h \\ \gamma^t f_i(\mathbf{x}_i^t) & \text{otherwise} \end{cases} \tag{15}$$

Each function $f_i \in \mathbf{F_Y}$ whose scope includes a random variable can be replaced by a *unary* function $F_i$.[9] The first term is the utility for the first $\bar{h}$ time steps and the second term is the utility for the time step $\bar{h}$:

$$F_i(\mathbf{x}_i) = \sum_{t=0}^{\bar{h}} F_i^t(\mathbf{x}_i^t) \tag{16}$$

where:

$$F_i^t(\mathbf{x}_i^t) = \begin{cases} \gamma^h \sum_{\omega \in \Omega_{y_i}} \tilde{f}_i(\mathbf{x}_i^h|_{y_i=\omega}) \cdot p_{y_i}^h(\omega) & \text{if } t = \bar{h} \text{ and } \bar{h} = h \\ \gamma^t \sum_{\omega \in \Omega_{y_i}} f_i(\mathbf{x}_i^t|_{y_i=\omega}) \cdot p_{y_i}^t(\omega) & \text{otherwise} \end{cases} \tag{17}$$

The function $\tilde{f}_i$ is recursively defined according to Equation (8). Additionally, each decision variable $x_i$ will have a unary function $C_i$:

$$C_i(\mathbf{x}_i) = -\sum_{t=0}^{h-1} \gamma^t \left[ c \cdot \Delta(\mathbf{x}_i^t, \mathbf{x}_i^{t+1}) \right] \tag{18}$$

which captures the cost of switching values across time steps. This collapsed DCOP can then be solved with any off-the-shelf exact DCOP algorithm.

### 5.2 Heuristic Approaches

Since solving PD-DCOPs optimally is PSPACE-hard (see Theorem 1), the exact approach described earlier fails to scale to large problems as we show in our experimental results in Section 8 later. Therefore, heuristic approaches are necessary to solve larger problems of interest. Similar to the exact approach, heuristic approaches solve PD-DCOPs proactively and take into account the discounted utilities and the discounted expected utilities by reformulating constraints in the problem. While the exact approach reformulates the constraints into a single DCOP with decision variables only, our heuristic approaches reformulate the constraints into a dynamic DCOP (specifically, a sequence of $\bar{h}$ DCOPs) with decision variables only.

For each constraint $f_i \in \mathbf{F_X}$ that does not involve a random variable, a new constraint $F_i^t$ is created to capture the discounted utilities for time steps $0 \leq t \leq \bar{h}$. The constraint $F_i^t$ is created by following Equation (15). Similarly, for each constraint $f_i \in \mathbf{F_Y}$ between decision variables $\mathbf{x}$ and a random variable $y$, we compute the constraint $F_i^t$ by following Equation (17). After this pre-processing step, the constraints involve decision variables exclusively, and the problem at each time step has been transformed to a regular DCOP. We now introduce two heuristic approaches: *Local Search* and *Sequential Greedy*.

9. With slight abuse of notation, we use the same notation $F_i$ in Equations (14) and (16) to refer to two different functions in two cases.

**Algorithm 1:** Local Search()

6 $iter \leftarrow 1$
7 $\langle v_i^{0*}, v_i^{1*}, \ldots, v_i^{\bar{h}*} \rangle \leftarrow \langle \text{NULL}, \text{NULL}, \ldots, \text{NULL} \rangle$
8 $\langle v_i^0, v_i^1, \ldots, v_i^{\bar{h}} \rangle \leftarrow$ InitialAssignment()
9 $context \leftarrow \langle (x_j, t, \text{NULL} | \ x_j \in N(a_i), 0 \leq t \leq \bar{h}) \rangle$
10 Send VALUE($\langle v_i^0, v_i^1, \ldots, v_i^{\bar{h}} \rangle$) to all neighbors

**Procedure** CalcGain()

11 $\langle u_i^0, u_i^1, \ldots, u_i^{\bar{h}} \rangle \leftarrow$ CalcUtils($\langle v_i^0, v_i^1, \ldots, v_i^{\bar{h}} \rangle, \mathbf{x}_i^{\bar{h}+1}$)
12 $u^* \leftarrow -\infty$
13 **foreach** $\langle d_i^0, d_i^1, \ldots, d_i^{\bar{h}} \rangle$ in $\times_{t=0}^{\bar{h}} D_{x_i}$ **do**
14 $\quad u \leftarrow$ CalcCumulativeUtil($\langle d_i^0, d_i^1, \ldots, d_i^{\bar{h}} \rangle, \mathbf{x}_i^{\bar{h}+1}$)
15 $\quad$ **if** $u > u^*$ **then**
16 $\quad\quad u^* \leftarrow u$
17 $\quad\quad \langle v_i^{0*}, v_i^{1*}, \ldots, v_i^{\bar{h}*} \rangle \leftarrow \langle d_i^0, d_i^1, \ldots, d_i^{\bar{h}} \rangle$
18 **if** $u^* \neq -\infty$ **then**
19 $\quad \langle u_i^{0*}, u_i^{1*}, \ldots, u_i^{\bar{h}*} \rangle \leftarrow$ CalcUtils($\langle v_i^{0*}, v_i^{1*}, \ldots, v_i^{\bar{h}*} \rangle, \mathbf{x}_i^{\bar{h}+1}$)
20 $\quad \langle \hat{u}_i^0, \hat{u}_i^1, \ldots, \hat{u}_i^{\bar{h}} \rangle \leftarrow \langle u_i^{0*}, u_i^{1*}, \ldots, u_i^{\bar{h}*} \rangle - \langle u_i^0, u_i^1, \ldots, u_i^{\bar{h}} \rangle$
21 **else**
22 $\quad \langle \hat{u}_i^0, \hat{u}_i^1, \ldots, \hat{u}_i^{\bar{h}} \rangle \leftarrow \langle \text{NULL}, \text{NULL}, \ldots, \text{NULL} \rangle$
23 Send GAIN($\langle \hat{u}_i^0, \hat{u}_i^1, \ldots, \hat{u}_i^{\bar{h}} \rangle$) to all neighbors

#### 5.2.1 Local Search Approach

In this section, we propose a local search approach that is inspired by MGM (Maheswaran et al., 2004a), a graphical game-based algorithm that has been shown to be robust in dynamically changing environments. Algorithm 1 shows the pseudocode of the local search approach, where each agent $a_i$ maintains the following data structures:

- *iter* is the current iteration number.
- *context* is a vector of tuples $(x_j, t, v_j^t)$ for all its neighboring variables $x_j \in N(a_i)$. Each of these tuples represents the agent's assumption that variable $x_j$ is assigned value $v_j^t$ at time step $t$.
- $\langle v_i^0, v_i^1, \ldots, v_i^{\bar{h}} \rangle$ is a vector of the agent's current value assignment for its variable $x_i$ at each time step $t$.
- $\langle v_i^{0*}, v_i^{1*}, \ldots, v_i^{\bar{h}*} \rangle$ is a vector of the agent's best value assignment for its variable $x_i$ at each time step $t$.
- $\langle u_i^0, u_i^1, \ldots, u_i^{\bar{h}} \rangle$ is a vector of the agent's utility (utilities from utility functions minus costs from switching costs) given its current value assignment at each time step $t$.
- $\langle u_i^{0*}, u_i^{1*}, \ldots, u_i^{\bar{h}*} \rangle$ is a vector of the agent's best utility given its best value assignment at each time step $t$.
- $\langle \hat{u}_i^{0*}, \hat{u}_i^{1*}, \ldots, \hat{u}_i^{\bar{h}*} \rangle$, which is a vector of the agent's best gain in utility at each time step $t$.

**Procedure** When Receive VALUE($\langle v_s^{0*}, v_s^{1*}, \ldots, v_s^{\bar{h}*} \rangle$)

```
24 foreach t from 0 to h̄ do
25     if v_s^{t*} ≠ NULL then
26         Update (x_s, t, v_s^t) ∈ context with (x_s, t, v_s^{t*})
27 if received VALUE messages from all neighbors in this iteration then
28     CalcGain()
29 iter ← iter + 1
```

**Procedure** When Receive GAIN($\langle \hat{u}_s^0, \hat{u}_s^1, \ldots, \hat{u}_s^{\bar{h}} \rangle$)

```
30 if ⟨û_s^0, û_s^1, ..., û_s^h̄⟩ ≠ ⟨NULL, NULL, ..., NULL⟩ then
31     foreach t from 0 to h̄ do
32         if û_i^t ≤ 0 ∨ û_s^t > û_i^t then
33             v_i^{t*} ← NULL
34 if received GAIN messages from all neighbors in this iteration then
35     foreach t from 0 to h̄ do
36         if v_i^{t*} ≠ NULL then
37             v_i^t ← v_i^{t*}
38     Send VALUE(⟨v_i^{1*}, v_i^{2*}, ..., v_i^{h̄*}⟩) to all neighbors
```

The high-level ideas are as follows: (1) Each agent $a_i$ starts by finding an initial value assignment to its variable $x_i$ for each time step $0 \leq t \leq \bar{h}$ and initializes its context variable *context*. (2) Each agent uses VALUE messages to inform its neighbors of the agent's current assignment and to ensure that it has the current values of its neighboring agents' variables. (3) Each agent computes its current utilities given its current value assignments, its best utilities over all possible value assignments, and its best gain in utilities, and sends this gain in a GAIN message to all its neighbors. (4) Each agent changes the value of its variable for time step $t$ if its gain for that time step is the largest over all its neighbors' gain for that time step, and repeats steps 2 through 4 until a termination condition is met. In more detail:

**Step 1:** Each agent initializes its vector of best values to a vector of NULL values (line 7) and calls InitialAssignment to initializes its current values (line 8). The values can be initialized randomly or according to some heuristic function. We describe later one such heuristic function. Finally, the agent initializes its context, where it assumes that the values for its neighbors is NULL for all time steps (line 9).

**Step 2:** The agent sends its current value assignment in a VALUE message to all neighbors (line 10). When it receives a VALUE message from a neighbor, it updates the context variable with the value assignments in that message (lines 24-26). When it has received VALUE messages from all neighbors in the current iteration, it means that its context now correctly reflects the neighbors' actual values. It then calls CalcGain to start Step 3 (line 28).

**Function** CalcUtils($\langle v_i^0, v_i^1, \ldots, v_i^{\bar{h}} \rangle, \mathbf{x}_i^{\bar{h}+1}$)

39 **foreach** $t$ from 0 to $\bar{h}$ **do**
40  **if** $t = 0$ **then**
41  $c_i^t \leftarrow \gamma^0 \cdot c \cdot \Delta(v_i^0, v_i^1)$
42  **else if** $t = \bar{h}$ **then**
43  $c_i^t \leftarrow \gamma^{\bar{h}-1} \cdot c \cdot \Delta(v_i^{\bar{h}-1}, v_i^{\bar{h}}) + \gamma^{\bar{h}} \cdot c \cdot \Delta(v_i^{\bar{h}}, \mathbf{x}_i^{\bar{h}+1})$
44  **else**
45  $c_i^t \leftarrow \gamma^{t-1} \cdot c \cdot \Delta(v_i^{t-1}, v_i^t) + \gamma^t \cdot c \cdot \Delta(v_i^t, v_i^{t+1})$
46  $u_i^t \leftarrow \sum_{F_j^t | x_i \in \mathbf{x}^{F_j^t}} F_j^t - c_i^t$
47 **return** $\langle u_i^0, u_i^1, \ldots, u_i^{\bar{h}} \rangle$

**Function** CalcCumulativeUtil($\langle v_i^0, v_i^1, \ldots, v_i^{\bar{h}} \rangle, \mathbf{x}_i^{\bar{h}+1}$)

48 $u \leftarrow \sum_{t=0}^{\bar{h}} \sum_{F_j^t | x_i \in \mathbf{x}^{F_j^t}} F_j^t$
49 $c_i \leftarrow 0$
50 **foreach** $t$ from 0 to $\bar{h} - 1$ **do**
51  $c_i \leftarrow c_i + \gamma^t \cdot c \cdot \Delta(v_i^t, v_i^{t+1})$
52 $c_i \leftarrow c_i + \gamma^{\bar{h}} \cdot c \cdot \Delta(v_i^{\bar{h}}, \mathbf{x}_i^{\bar{h}+1})$
53 **return** $u - c_i$

**Step 3:** In the CalcGain procedure, the agent calls CalcUtils to calculate its utility for each time step given its current value assignments and its neighbors' current value assignments recorded in its context (line 11). The utility for a time step $t$ is made out of two components (line 46). The first component is the sum of utilities over all utility functions that involve the agent, under the assumption that the agent takes on its current value and its neighbors take on their values according to its *context*. Specifically, if the scope of the utility function $F_j^t$ involves only decision variables, then $F_j^t(v_i^t, v_j^t)$ is a function of both the agent's current value $v_i^t$ and its neighbor's value $v_j^t$ in its *context* and is defined according to Equation (15). If the scope involves both decision and random variables, then $F_j^t(v_i^t)$ is a unary constraint that is only a function of the agent's current value $v_i^t$ and is defined according to Equation (17). The second component is the cost of switching values from the previous time step $t-1$ to the current time step $t$ and switching from the current time step to the next time step $t+1$. This cost is $c$ if the values in two subsequent time steps are different and 0 otherwise. The variable $c_i^t$ captures this cost (lines 40-45). Note that if $\mathbf{x}_i^{\bar{h}+1} = \text{NULL}$, then $\Delta(v_i^{\bar{h}}, \mathbf{x}_i^{\bar{h}+1}) = 0$. The net utility is thus the utility derived according to the utility functions minus the switching cost (line 46).

The agent then searches over all possible combination of values for its variable across all time steps to find the best value assignment that results in the largest cumulative cost across all time steps (lines 13-17). It then computes the net gain in utility at each time step by subtracting the utility of the best value assignment with the utility of the current value assignment (lines 18-20).

**Step 4:** The agent sends its gains in a GAIN message to all neighbors (line 23). When it receives a GAIN message from its neighbor, it updates its best value $v_i^{t*}$ for time step $t$ to NULL if its gain is non-positive (i.e., $\hat{u}_i^t \leq 0$) or its neighbor has a larger gain (i.e., $\hat{u}_s^t > \hat{u}_i^t$) for that time step (lines 32-33). When it has received GAIN messages from all neighbors in the current iteration, it means that it has identified, for each time step, whether its gain is the largest over all its neighbors' gains. The time steps where it has the largest gain are exactly those time steps $t$ where $v_i^{t*}$ is not NULL. The agent thus assigns its best value for these time steps as its current value and restarts Step 2 by sending a VALUE message that contains its new values to all its neighbors (lines 34-38).

**Heuristics for** InitialAssignment**:** We now introduce a heuristic function to speed up InitialAssignment. We simplify the PD-DCOP into $\bar{h}$ independent DCOPs by assuming that the switching costs are 0 and the constraints with random variables are collapsed into unary constraints similar to the description for our exact approach. Then, one can use any off-the-shelf DCOP algorithm to solve these $\bar{h}$ DCOPs. We initially used DPOP to do this, but our preliminary experimental results show that this approach is computationally inefficient.

However, we observed that these $\bar{h}$ DCOPs do not vary much across subsequent DCOPs as changes are due only to the changes in distribution of values of random variables. Therefore, the utilities in UTIL tables of an agent $a_i$ remain unchanged across subsequent DCOPs if neither it nor any of its descendants in the pseudo-tree are constrained with a random variable. We thus used S-DPOP to solve the $\bar{h}$ DCOPs and the runtimes decreased marginally.

We further optimize this approach by designing a new *pseudo-tree construction heuristic*, such that agents that are constrained with random variables are higher up in the pseudo-tree. Intuitively, this will maximize the number of utility values that can be reused, as they remain unchanged across subsequent time steps. This heuristic, within the Distributed DFS algorithm (Hamadi et al., 1998), assigns a score to each agent $a$ according to heuristic $h_1(a)$:

$$h_1(a) = (1 + I(a)) \cdot |N_y(a)| \tag{19}$$

$$N_y(a) = \{a' | a' \in N(a) \wedge \exists f \in \mathbf{F}, \exists\, y \in \mathbf{Y} : \{a', y\} \in \mathbf{x}^f\} \tag{20}$$

$$I(a) = \begin{cases} 0 & \text{if } \forall f \in \mathbf{F}, \forall y \in \mathbf{Y} : \{a, y\} \notin \mathbf{x}^f \\ 1 & \text{otherwise} \end{cases} \tag{21}$$

It then makes the agent with the largest score the pseudo-tree root and traverses the constraint graph using DFS, greedily adding the neighboring agent with the largest score as the child of the current agent. However, this resulting pseudo-tree can have a large depth, which is undesirable. The popular max-degree heuristic $h_2(a) = |N(a)|$, which chooses the agent with the largest number of neighbors, typically results in pseudo-trees with small depths. We thus also introduce a hybrid heuristic which combines both heuristics and weigh them according to a *heuristic weight* $w$:

$$h_3(a) = w\, h_1(a) + (1 - w)\, h_2(a) \tag{22}$$

### 5.2.2 Sequential Greedy Approach

In addition to the local search approach, we now introduce sequential greedy algorithms to solve PD-DCOPs. We propose two algorithms: FORWARD and BACKWARD. Both

algorithms sequentially solve each DCOP one time step at a time in a greedy manner. However, they differ in how they choose the next problem to solve, where they take into account the switching cost between two problems differently.

**FORWARD:** In general, FORWARD greedily solves each sub-problem in PD-DCOPs one time step at a time starting from the initial time step. In other words, it successively solves the DCOP at each time step starting from $t = 0$ to time step $\bar{h}$. When solving each DCOP, it takes into account the switching cost of changing values from the solution in the previous time step. If the optimal solution $\mathbf{x}^{\bar{h}+1} \neq \text{NULL}$, at the last time step $\bar{h}$, it will take into account the switching cost incurred by changing the solution from $\bar{h}$ to the optimal solution $\mathbf{x}^{\bar{h}+1}$. Specifically, to capture the cost of switching values across time steps, for each decision variable $x \in \mathbf{X}$, the following new unary constraint is created for each time step $0 < t < \bar{h}$:

$$C^t(x) = -c \cdot \Delta(x^{t-1}, x^t) \tag{23}$$

At the last time step $t = \bar{h}$, we add the following constraint:

$$C^{\bar{h}}(x) = \begin{cases} -c \cdot \Delta(x^{\bar{h}-1}, x^{\bar{h}}) & \text{if } \mathbf{x}^{\bar{h}+1} = \text{NULL} \\ -c \cdot \left(\Delta(x^{\bar{h}-1}, x^{\bar{h}}) + \Delta(x^{\bar{h}}, \mathbf{x}_x^{\bar{h}+1})\right) & \text{otherwise} \end{cases} \tag{24}$$

where $\mathbf{x}_x^{\bar{h}+1}$ is the value of variable $x$ in $\mathbf{x}^{\bar{h}+1}$. After adding the switching cost constraints, the agents successively solve each DCOP from time step $t = 0$ onwards using any off-the-shelf DCOP algorithm.

**BACKWARD:** Instead of greedily solving the PD-DCOP one time step at a time forward starting from $t = 0$ towards $\bar{h}$, in the case where the solution at time step $\bar{h} + 1$ is available (i.e., $\mathbf{x}^{\bar{h}+1} \neq \text{NULL}$), one can also greedily solve the problem backwards from $t = \bar{h} + 1$ towards the first time step. The BACKWARD algorithm implements this key difference. At time step $t$, BACKWARD takes into account the switching cost to the solution in the next time step $t + 1$. Specifically, before solving each sub-problem, BACKWARD creates a unary constraint for each time step $0 \leq t < \bar{h}$:

$$C^t(x) = -c \cdot \Delta(x^t, x^{t+1}) \tag{25}$$

Also, BACKWARD creates an additional unary constraint to capture the switching cost between the solution at $\bar{h}$ and the optimal solution $\mathbf{x}^{\bar{h}+1}$:

$$C^{\bar{h}}(x) = -c \cdot \Delta(x^{\bar{h}}, \mathbf{x}_x^{\bar{h}+1}) \tag{26}$$

## 6. Theoretical Results

We now discuss theoretical results of the PD-DCOP model and its algorithms. In Theorem 1, we discuss the complexity of PD-DCOPs in two cases: $h$ is polynomial in $|\mathbf{X}|$ and $h$ is exponential in $|\mathbf{X}|$. In Theorem 2, if the discount factor $\gamma = 1$, we prove that adopting the optimal solution for the stationary distribution at time step $h$ will maximize the sum of future utilities from time step $h$ onward. We then provide the error bounds in Theorem 3, Theorem 4, and Theorem 5. Finally, we discuss the space and time complexities of the local search approach in Theorem 6.

Theorem **1** *Optimally solving a* PD-DCOP *with a horizon that is polynomial (exponential) in* $|\mathbf{X}|$ *is PSPACE-complete (PSPACE-hard).*

Proof: We first consider the case where $h$ is polynomial in $|\mathbf{X}|$. Membership in PSPACE follows from the existence of a naive depth-first search to solve PD-DCOPs, where a non-deterministic branch is created for each complete assignment of the PD-DCOP's decision variables and for each time step $0 \leq t \leq h$. The algorithm requires linear space in the number of variables and horizon length. We reduce the *satisfiability of quantified Boolean formula* (*QSAT*) to a PD-DCOP with 0 horizon. Each existential Boolean variable in the QSAT is mapped to a corresponding decision variable in the PD-DCOP, and each universal Boolean variable in the QSAT is mapped to a PD-DCOP random variable. The domains $D_x$ of all variables $x \in \mathbf{X}$ are the sets of values $\{0, 1\}$, corresponding respectively to the evaluations, *false* and *true*, of the QSAT variables. The initial probability distribution $p_y^0$ of each PD-DCOP random variable $y \in \mathbf{Y}$ is set to as the *uniform* distribution. Each QSAT clause $c$ is mapped to a PD-DCOP utility function $f_c$, whose scope involves all and only the PD-DCOP-corresponding boolean variables appearing in $c$, and such that:

$$f_c(\mathbf{x}^c) = \begin{cases} 1, & \text{if } c(\mathbf{x}^c) = \text{ true} \\ \bot, & \text{otherwise.} \end{cases} \tag{27}$$

where $c(\mathbf{x}^c)$ denotes the instantiation of the values of the variables in $\mathbf{x}^c$ to the truth values of the corresponding literals of $c$. In other words, a clause is satisfied *iff* the equivalent utility function preserves its semantics. The choices for, the switching cost, the discount factor $\gamma$, and the transition function $T_y$, for each $y \in \mathbf{Y}$, of the PD-DCOP, are immaterial. The reduction is linear in the size of the original quantified Boolean formula. The quantified Boolean formula is satisfiable iff the equivalent PD-DCOP has at least one solution $\mathbf{x}$ whose cost $\mathcal{F}(\mathbf{x}) \neq \bot$.

Next, we consider the case where $h$ is exponential in $\mathbf{X}$. In this case, since storing a solution requires space exponential in $|\mathbf{X}|$, solving PD-DCOPs is PSPACE-hard, which concludes the proof. □

Theorem **2** *When* $\gamma = 1$*, from time step* $h$ *onwards, adopting the optimal solution for the stationary distribution, instead of any other solution, will maximize the expected utility from that time step onward.*

Proof: As $p_y^*$ is the stationary distribution of random variable $y$ and it is also the converged distribution of $p_y^t$ when $t \to \infty$:

$$\lim_{t \to \infty} p_y^t = p_y^* \tag{28}$$

$$p_y^* \cdot T = p_y^* \tag{29}$$

After convergence, as $p_y^*$ does not change for every $y \in \mathbf{Y}$, the optimal solution for the successive DCOPs remain the same. Let $h^*$ be the horizon when the stationary distribution converges, $\mathbf{x}^*$ be the optimal solution, $\mathbf{x}'$ be any sub-optimal solution, and $\mathcal{F}^*(\mathbf{x})$ be the quality of solution $\mathbf{x}$ for the DCOP with stationary distribution. As the stationary distribution at $h^*$ is the actual distribution at $h^*$, the solution $\mathbf{x}^*$ is optimal for the DCOP at $h^*$

and also optimal for all DCOPs after $h^*$:

$$\mathcal{F}^*(\mathbf{x}^*) > \mathcal{F}^*(\mathbf{x}') \qquad \forall t \geq h^* \tag{30}$$

The difference in quality between two solutions for DCOPs after $h^*$ is:

$$\Delta_{h^*}^{\infty} = \sum_{t=h^*}^{\infty} \left[\mathcal{F}^*(\mathbf{x}^*) - \mathcal{F}^*(\mathbf{x}')\right] \tag{31}$$

As the difference in solution quality from $h$ to $h^*$ is finite, it is dominated by $\Delta_{h^*}^{\infty} = \infty$. In other words, if we keep the sub-optimal $\mathbf{x}'$ from time step $h$ onward, the accumulated expected utility of $\mathbf{x}'$ is smaller than that of the optimal solution $\mathbf{x}^*$ with the stationary distribution. □

**Error Bounds:** We denote $U^{\infty}$ as the optimal solution quality of a PD-DCOP with an infinite horizon and $U^h$ as the optimal solution quality when the horizon $h$ is finite. Let $F_{\mathbf{y}}(\mathbf{x})$ be the utility of a regular DCOP where the decision variables are assigned $\mathbf{x}$ given values $\mathbf{y}$ of the random variables. We define $F_{\mathbf{y}}^{\Delta} = \max_{\mathbf{x}\in\Sigma} F_{\mathbf{y}}(\mathbf{x}) - \min_{\mathbf{x}\in\Sigma} F_{\mathbf{y}}(\mathbf{x})$ as the maximum loss in solution quality of a regular DCOP for a given random variable assignment $\mathbf{y}$ and $F^{\Delta} = \max_{\mathbf{y}\in\Sigma_{\mathbf{Y}}} F_{\mathbf{y}}^{\Delta}$ where $\Sigma_{\mathbf{Y}} = \prod_{y\in\mathbf{Y}} \Omega_y$ is the assignment space for all random variables.

Theorem **3** *When $\gamma < 1$, the error $U^{\infty} - U^h$ of the optimal solution from solving PD-DCOPs with a finite horizon $h$ instead of an infinite horizon is bounded from above by $\frac{\gamma^h}{1-\gamma} F^{\Delta}$.*

Proof: Let $\hat{\mathbf{x}}^* = \langle \hat{\mathbf{x}}_0^*, \ldots, \hat{\mathbf{x}}_h^*, \hat{\mathbf{x}}_{h+1}^*, \ldots \rangle$ be the optimal solution of PD-DCOPs with infinite horizon $\infty$:

$$U^{\infty} = \sum_{t=0}^{\infty} \gamma^t \left[\mathcal{F}_x^t(\hat{\mathbf{x}}_t^*) + \mathcal{F}_y^t(\hat{\mathbf{x}}_t^*) - c \cdot \Delta(\hat{\mathbf{x}}_t^*, \hat{\mathbf{x}}_{t+1}^*)\right] \tag{32}$$

Ignoring switching costs after time step $h$, an upper bound $U_+^{\infty}$ of $U^{\infty}$ is defined as:

$$U_+^{\infty} = \sum_{t=0}^{h-1} \gamma^t \left[\mathcal{F}_x^t(\hat{\mathbf{x}}_t^*) + \mathcal{F}_y^t(\hat{\mathbf{x}}_t^*) - c \cdot \Delta(\hat{\mathbf{x}}_t^*, \hat{\mathbf{x}}_{t+1}^*)\right] + \sum_{t=h}^{\infty} \gamma^t \left[\mathcal{F}_x^t(\hat{\mathbf{x}}_t^*) + \mathcal{F}_y^t(\hat{\mathbf{x}}_t^*)\right] \tag{33}$$

Let $\mathbf{x}^* = \langle \mathbf{x}_0^*, \ldots, \mathbf{x}_h^* \rangle$ be the optimal solution of the PD-DCOP with a finite horizon $h$:

$$U^h = \sum_{t=0}^{h-1} \gamma^t \left[\mathcal{F}_x^t(\mathbf{x}_t^*) + \mathcal{F}_y^t(\mathbf{x}_t^*) - c \cdot \Delta(\mathbf{x}_t^*, \mathbf{x}_{t+1}^*)\right] + \sum_{t=h}^{\infty} \gamma^t \left[\mathcal{F}_x^t(\mathbf{x}_h^*) + \mathcal{F}_y^t(\mathbf{x}_h^*)\right] \tag{34}$$

For $\hat{\mathbf{x}}^*$, if we change the solution for every DCOP after time step $h$ to $\hat{\mathbf{x}}_h^*$, as $\langle \hat{\mathbf{x}}_0^*, \ldots, \hat{\mathbf{x}}_h^*, \hat{\mathbf{x}}_h^*, \ldots \rangle$, we get an lower bound $U_-^{\infty}$ of $U^h$:

$$U_-^{\infty} = \sum_{t=0}^{h-1} \gamma^t \left[\mathcal{F}_x^t(\hat{\mathbf{x}}_t^*) + \mathcal{F}_y^t(\hat{\mathbf{x}}_t^*) - c \cdot \Delta(\hat{\mathbf{x}}_t^*, \hat{\mathbf{x}}_{t+1}^*)\right] + \sum_{t=h}^{\infty} \gamma^t \left[\mathcal{F}_x^t(\hat{\mathbf{x}}_h^*) + \mathcal{F}_y^t(\hat{\mathbf{x}}_h^*)\right] \tag{35}$$

Therefore, we get $U_-^\infty \leq U^h \leq U^\infty \leq U_+^\infty$.

Next, we compute the difference between the two bounds:

$$U^\infty - U^h \leq U_+^\infty - U_-^\infty \tag{36}$$

$$= \sum_{t=h}^{\infty} \gamma^t \left[ (\mathcal{F}_x^t(\hat{\mathbf{x}}_t^*) + \mathcal{F}_y^t(\hat{\mathbf{x}}_t^*)) - (\mathcal{F}_x^t(\hat{\mathbf{x}}_h^*) + \mathcal{F}_y^t(\hat{\mathbf{x}}_h^*)) \right] \tag{37}$$

Notice that the quantity $(\mathcal{F}_x^t(\hat{\mathbf{x}}_t^*) + \mathcal{F}_y^t(\hat{\mathbf{x}}_t^*)) - (\mathcal{F}_x^t(\hat{\mathbf{x}}_h^*) + \mathcal{F}_y^t(\hat{\mathbf{x}}_h^*))$ is the utility difference between the value assignment $\hat{\mathbf{x}}_t^*$ and $\hat{\mathbf{x}}_h^*$ for a sub-problem in time step $t$, and thus is bounded by the maximum loss of a regular DCOP:

$$(\mathcal{F}_x^t(\hat{\mathbf{x}}_t^*) + \mathcal{F}_y^t(\hat{\mathbf{x}}_t^*)) - (\mathcal{F}_x^t(\hat{\mathbf{x}}_h^*) + \mathcal{F}_y^t(\hat{\mathbf{x}}_h^*)) \leq F^\Delta \tag{38}$$

Thus,

$$U^\infty - U^h \leq U_+^\infty - U_-^\infty \tag{39}$$

$$\leq \sum_{t=h}^{\infty} \gamma^t \left[ \mathcal{F}_x^t(\hat{\mathbf{x}}_t^*) + \mathcal{F}_y^t(\hat{\mathbf{x}}_t^*) - \mathcal{F}_x^t(\hat{\mathbf{x}}_h^*) - \mathcal{F}_y^t(\hat{\mathbf{x}}_h^*) \right] \tag{40}$$

$$\leq \sum_{t=h}^{\infty} \gamma^t F^\Delta \tag{41}$$

$$\leq \frac{\gamma^h}{1-\gamma} F^\Delta \tag{42}$$

which concludes the proof. □

Corollary **1** *Given a maximum acceptable error $\epsilon$, the minimum horizon $h$ is* $\log_\gamma \frac{(1-\gamma)\cdot\epsilon}{F^\Delta}$.

Proof: Following Theorem 3, the error of the optimal solution is bounded above by $\frac{\gamma^h}{1-\gamma} F^\Delta$:

$$\epsilon \leq \frac{\gamma^h}{1-\gamma} F^\Delta \tag{43}$$

$$\frac{(1-\gamma)\cdot\epsilon}{F^\Delta} \leq \gamma^h \tag{44}$$

$$\log_\gamma \frac{(1-\gamma)\cdot\epsilon}{F^\Delta} \leq h \tag{45}$$

Thus, the minimum horizon $h$ is $\log_\gamma \frac{(1-\gamma)\cdot\epsilon}{F^\Delta}$. □

Let $\mathbf{x}^*$ denote the optimal solution for the DCOP with a stationary distribution. We define $\theta_y = \min_{\omega,\omega'} T_y(\omega, \omega')$ as the smallest transition probability between two states $\omega$ and $\omega'$ of the random variable $y$, and $\beta = \prod_{y \in \mathbf{Y}} \theta_y$ as the smallest transition probability between two joint states $\mathbf{y}$ and $\mathbf{y}'$ of all random variables in $\mathbf{Y}$.

Theorem **4** *With $\beta > 0$, when $\gamma = 1$, the error $U^\infty - U^h$ from solving PD-DCOPs with a finite horizon $h$ using MCC approach is bounded from above by $c \cdot |\mathbf{X}| + \sum_{\mathbf{y} \in \Sigma_\mathbf{Y}} \frac{(1-2\beta)^h}{2\beta} \mathcal{F}_\mathbf{y}^\Delta$.*

Proof: First, given a random variable $y$, the following inequality holds if the Markov chain converges to the stationary distribution $p_y^*$ (Gallager, 2013). For a given $\omega \in \Omega_y$:

$$|p_y^*(\omega) - T_y^t(\omega', \omega)| \leq (1 - 2\theta_y)^t \qquad \forall \omega' \in \Omega_y \tag{46}$$

where $T_y^t$ and $T_y^*$ are the stationary transition matrix after $t$ time steps and the stationary transition matrix, respectively:

$$p_y^0 \cdot T_y^t = p_y^t \tag{47}$$
$$p_y^0 \cdot T_y^* = p_y^* \tag{48}$$

For $\omega \in \Omega_y$:

$$p_y^*(\omega) = \sum_{\omega' \in \Omega_y} p_y^0(\omega') \cdot T_y^*(\omega', \omega) \tag{49}$$
$$p_y^t(\omega) = \sum_{\omega' \in \Omega_y} p_y^0(\omega') \cdot T_y^t(\omega', \omega) \tag{50}$$
$$|p_y^*(\omega) - p_y^t(\omega)| = |\sum_{\omega' \in \Omega_y} p_y^0(\omega') \cdot (T_y^*(\omega', \omega) - T_y^t(\omega', \omega))| \tag{51}$$
$$= |\sum_{\omega' \in \Omega_y} p_y^0(\omega') \cdot (p_y^*(\omega) - T_y^t(\omega', \omega))| \tag{52}$$
$$\leq \sum_{\omega' \in \Omega_y} p_y^0(\omega') \cdot |(p_y^*(\omega) - T_y^t(\omega', \omega))| \tag{53}$$
$$\leq \sum_{\omega' \in \Omega_y} p_y^0(\omega') \cdot (1 - 2\theta_y)^t \tag{54}$$
$$\leq (1 - 2\theta_y)^t \tag{55}$$

where $T_y^*(\omega', \omega) = p_y^*(\omega)$ for all $\omega' \in \Omega_y$. Similarly, for $\mathbf{y} \in \Sigma_{\mathbf{Y}}$, we have:

$$\delta_{\mathbf{Y}}^t(\mathbf{y}) = |p_{\mathbf{Y}}^*(\mathbf{y}) - p_{\mathbf{Y}}^t(\mathbf{y})| \leq (1 - 2\beta)^t \tag{56}$$

Then, the solution quality loss for assigning $\mathbf{x}^*$ at time step $t$ is:

$$\mathcal{F}_\Delta^t \leq \sum_{\mathbf{y} \in \Sigma_{\mathbf{Y}}} \delta_{\mathbf{Y}}^t(\mathbf{y}) \cdot \left( \max_{\mathbf{x} \in \Sigma} F_{\mathbf{y}}(\mathbf{x}) - F_{\mathbf{y}}(\mathbf{x}^*) \right) \tag{57}$$
$$\leq \sum_{\mathbf{y} \in \Sigma_{\mathbf{Y}}} (1 - 2\beta)^t \cdot F_{\mathbf{y}}^\Delta \tag{58}$$

Next, let $\bar{\mathbf{x}} = \langle \bar{\mathbf{x}}_0, \ldots, \bar{\mathbf{x}}_h \rangle$ denote the optimal solution of the PD-DCOP using MCC approach with $\bar{\mathbf{x}}_h = \mathbf{x}^*$; $\hat{\mathbf{x}} = \langle \hat{\mathbf{x}}_0, \ldots, \hat{\mathbf{x}}_h \rangle$ denote the optimal solution for the DCOPs from time steps 0 to $h$ without considering the stationary distribution; and $\check{\mathbf{x}} = \langle \check{\mathbf{x}}_0 = \hat{\mathbf{x}}_0, \check{\mathbf{x}}_1 =$

$\hat{\mathbf{x}}_1, \ldots, \check{\mathbf{x}}_{h-1} = \hat{\mathbf{x}}_{h-1}, \check{\mathbf{x}}_h = \bar{\mathbf{x}}_h = \mathbf{x}^*\rangle$. We then have the following solution qualities:

$$U_+ = \sum_{t=0}^{h} \mathcal{F}^t(\hat{\mathbf{x}}_t) - \sum_{t=0}^{h-1} [c \cdot \Delta(\hat{\mathbf{x}}_t, \hat{\mathbf{x}}_{t+1})] \tag{59}$$

$$U = \sum_{t=0}^{h} \mathcal{F}^t(\bar{\mathbf{x}}_t) - \sum_{t=0}^{h-1} [c \cdot \Delta(\bar{\mathbf{x}}_t, \bar{\mathbf{x}}_{t+1})] \tag{60}$$

$$U_- = \sum_{t=0}^{h} \mathcal{F}^t(\check{\mathbf{x}}_t) - \sum_{t=0}^{h-1} [c \cdot \Delta(\check{\mathbf{x}}_t, \check{\mathbf{x}}_{t+1})] \tag{61}$$

Since $\mathbf{x}^*$ is the optimal solution for the PD-DCOP and $\check{\mathbf{x}}_h = \bar{\mathbf{x}}_h = \mathbf{x}^*$, we have $U_- \leq U$. Moreover, as $\check{\mathbf{x}}_t = \hat{\mathbf{x}}_t$ for time steps between 0 and $h-1$, the error bound for time step 0 to time step $h$ is:

$$U_+ - U \leq U_+ - U_- \tag{62}$$

$$\leq \left[\mathcal{F}^h(\hat{\mathbf{x}}_h) - \mathcal{F}^h(\mathbf{x}^*)\right] - [c \cdot \Delta(\hat{\mathbf{x}}_{h-1}, \hat{\mathbf{x}}_h) - c \cdot \Delta(\check{\mathbf{x}}_{h-1}, \mathbf{x}^*)] \tag{63}$$

$$\leq \mathcal{F}^h_\Delta + c \cdot |\mathbf{X}| \tag{64}$$

In addition, from $t = h+1$ to $\infty$, the cumulative error bound is $\sum_{t=h+1}^{\infty} \mathcal{F}^t_\Delta$. Summing up the two error bounds for $0 \to h$ and $h+1 \to \infty$, we get:

$$\mathcal{F}^h_\Delta + c \cdot |\mathbf{X}| + \sum_{t=h+1}^{\infty} \mathcal{F}^t_\Delta = c \cdot |\mathbf{X}| + \sum_{t=h}^{\infty} \mathcal{F}^t_\Delta \tag{65}$$

$$= c \cdot |\mathbf{X}| + \sum_{t=h}^{\infty} \left( \sum_{\mathbf{y} \in \Sigma_{\mathbf{Y}}} \delta^t_{\mathbf{Y}}(\mathbf{y}) \cdot F^\Delta_{\mathbf{y}} \right) \tag{66}$$

$$\leq c \cdot |\mathbf{X}| + \sum_{\mathbf{y} \in \Sigma_{\mathbf{Y}}} \sum_{t=h}^{\infty} (1-2\beta)^t \cdot F^\Delta_{\mathbf{y}} \tag{67}$$

$$\leq c \cdot |\mathbf{X}| + \sum_{\mathbf{y} \in \Sigma_{\mathbf{Y}}} \frac{(1-2\beta)^h}{2\beta} F^\Delta_{\mathbf{y}} \tag{68}$$

which concludes the proof. □

**Upper Bound on Optimal Quality:** We now describe an upper bound on the optimal solution quality $\mathcal{F}^h(\mathbf{x}^*)$. Let $\hat{\mathbf{x}}^* = \langle \hat{\mathbf{x}}^*_0, \ldots, \hat{\mathbf{x}}^*_h \rangle$ be the vector of assignments, where:

$$\hat{\mathbf{x}}^*_t = \begin{cases} \operatorname{argmax}_{\mathbf{x} \in \Sigma} \gamma^t \left[\mathcal{F}^t_x(\mathbf{x}) + \mathcal{F}^t_y(\mathbf{x})\right] & \text{if } 0 \leq t < h \\ \operatorname{argmax}_{\mathbf{x} \in \Sigma} \left[\tilde{\mathcal{F}}_x(\mathbf{x}) + \tilde{\mathcal{F}}_y(\mathbf{x})\right] & \text{otherwise} \end{cases} \tag{69}$$

and

$$\hat{\mathcal{F}}^h(\mathbf{x}) = \sum_{t=0}^{h-1} \gamma^t \left[\mathcal{F}^t_x(\mathbf{x}) + \mathcal{F}^t_y(\mathbf{x})\right] + \tilde{\mathcal{F}}_x(\mathbf{x}) + \tilde{\mathcal{F}}_y(\mathbf{x}). \tag{70}$$

Theorem **5** *The lower and upper bounds of the optimal solution of PD-DCOPs are* $\mathcal{F}^h(\mathbf{x}) \leq \mathcal{F}^h(\mathbf{x}^*) \leq \hat{\mathcal{F}}^h(\hat{\mathbf{x}}^*)$ *for all* $\mathbf{x} \in \Sigma^{h+1}$.

Proof: For any given assignment $\mathbf{x} \in \Sigma^{h+1}$, $\mathcal{F}^h(\mathbf{x})$ is a clear lower bound for $\mathcal{F}^h(\mathbf{x}^*)$.

For the upper bound, let $\mathcal{F}^h_t(\cdot)$ be the $t^{\text{th}}$ component of the $\mathcal{F}^h(\cdot)$, defined as:

$$\mathcal{F}^h_t(\mathbf{x}_t) = \begin{cases} \gamma^t \left[\mathcal{F}^t_x(\mathbf{x}_t) + \mathcal{F}^t_y(\mathbf{x}_t) - [c \cdot \Delta(\mathbf{x}_t, \mathbf{x}_{t+1})\right] & \text{if } 0 \leq t < h \\ \tilde{\mathcal{F}}_x(\mathbf{x}_t) + \tilde{\mathcal{F}}_y(\mathbf{x}_t) & \text{otherwise} \end{cases} \tag{71}$$

with $\mathbf{x}_t$, defined as the $t^{\text{th}}$ value assignment in the PD-DCOP solution $\mathbf{x}$. Similarly, let us denote $\hat{\mathcal{F}}^h_t(\cdot)$ as the $t^{\text{th}}$ component of the $\hat{\mathcal{F}}^h(\cdot)$, defined as:

$$\hat{\mathcal{F}}^h_t(\mathbf{x}_t) = \begin{cases} \gamma^t \left[\mathcal{F}^t_x(\mathbf{x}_t) + \mathcal{F}^t_y(\mathbf{x}_t)\right] & \text{if } 0 \leq t < h \\ \tilde{\mathcal{F}}_x(\mathbf{x}_t) + \tilde{\mathcal{F}}_y(\mathbf{x}_t) & \text{otherwise} \end{cases} \tag{72}$$

It follows that for all $0 \leq t < h$:

$$\begin{aligned} \mathcal{F}^h_t(\mathbf{x}^*_t) &= \gamma^t \left[\mathcal{F}^t_x(\mathbf{x}^*_t) + \mathcal{F}^t_y(\mathbf{x}^*_t) - [c \cdot \Delta(\mathbf{x}_t, \mathbf{x}_{t+1})\right] && (73)\\ &\leq \gamma^t \left[\mathcal{F}^t_x(\mathbf{x}^*_t) + \mathcal{F}^t_y(\mathbf{x}^*_t)\right] && (74)\\ &\leq \max_{\mathbf{x} \in \Sigma} \gamma^t \left[\mathcal{F}^t_x(\mathbf{x}) + \mathcal{F}^t_y(\mathbf{x})\right] && (75)\\ &\leq \gamma^t \left[\mathcal{F}^t_x(\hat{\mathbf{x}}^*_t) + \mathcal{F}^t_y(\hat{\mathbf{x}}^*_t)\right] = \hat{\mathcal{F}}^h_t(\hat{\mathbf{x}}^*_t) && (76) \end{aligned}$$

where $\mathbf{x}^*_t$ (resp. $\hat{\mathbf{x}}^*_t$) is the $t^{\text{th}}$ component of the PD-DCOP solution vector $\mathbf{x}^*$ (resp. $\hat{\mathbf{x}}^*$).

For $t = h$, it follows:

$$\begin{aligned} \mathcal{F}^h_h(\mathbf{x}^*_h) &= \tilde{\mathcal{F}}_x(\mathbf{x}^*_h) + \tilde{\mathcal{F}}_y(\mathbf{x}^*_h) && (77)\\ &\leq \max_{\mathbf{x} \in \Sigma} \left[\tilde{\mathcal{F}}_x(\mathbf{x}) + \tilde{\mathcal{F}}_y(\mathbf{x})\right] = \hat{\mathcal{F}}^h_h(\hat{\mathbf{x}}^*_h) && (78) \end{aligned}$$

Thus, from the two inequalities above, it follows that:

$$\mathcal{F}^h(\mathbf{x}^*) \leq \sum_{t=0}^{h} \hat{\mathcal{F}}^h_t(\mathbf{x}^*_t) = \hat{\mathcal{F}}^h(\hat{\mathbf{x}}^*) \tag{79}$$

which concludes the proof. □

Theorem **6** *An agent's space requirement for the PD-DCOP local search approach is* $O(\mathcal{L} + (h+1)|\mathbf{A}|)$, *where* $O(\mathcal{L})$ *is the agent's space requirement for the* InitialAssignment *function. The time complexity of the local search approach is* $O(D^h)$, *where* $D = \text{argmax}_x |D_x|$.

Proof: In our local search algorithms, each agent first calls the InitialAssignment function to find an initial value assignment to its variable for each time step $0 \leq t \leq h$ (line 8). Thus, the memory requirement of this step is $O((h+1) + \mathcal{L})$ at each agent. Next, each agent performs a local search step (lines 9-10), which is analogous to that of MGM. However, different from MGM, our agents search for tuples of $h+1$ values, one for each

time step in the horizon. Thus, at each iteration, and for each time step $t$, each agent stores a vector of values for its current and best variable assignments for its variable; a vector of the agent's utilities and best utilities given its current value assignments; and a vector of the agent's best gain in utility. In addition, each agent stores the context of its neighbors' values for each time step $t$, which requires $O((h+1) \cdot |N(a_i)|)$ space. Thus, the overall space requirement for our local search algorithm is $O(\mathcal{L} + (h+1)|\mathbf{A}|)$ per agent.

In the local search algorithms, to find the best response in each local search step, in the worst case, each agent enumerates all possible combinations of decision variable domain across all time step $h$. Thus, the time complexity of the local search approach is $O(D^h)$, where $D = \text{argmax}_x |D_x|$ is the largest domain size among all agents. □

Lemma **1** *The solution quality of Local Search approaches is monotonically increasing with respect to the iteration round.*

Proof: In MGM, a variable is allowed to change its value in an iteration only when its gain is higher than its neighbors' gains, and two neighbors are not allowed to change their value in the same iteration. The solution quality of MGM has been proved to monotonically increase with respect to the iteration round (Maheswaran et al., 2004a). Our Local Search approaches such as LS-SDPOP, LS-MGM, and LS-RAND use the same mechanism for variables to change their values at every time step. For a given time step in an iteration, a variable is allowed to change it values only when its gain is the largest among their neighbors' gains for that time step (Procedure When Receive Gain lines 31-33 and 35-37). Therefore, the solution quality of the Local Search approaches is monotonically increasing with respect to the iteration round. □

## 7. Related Work

Aside from the D-DCOPs described in the introduction and background, several approaches have been proposed to proactively solve centralized *Dynamic CSPs*, where value assignments of variables or utilities of constraints may change according to some probabilistic model (Wallace & Freuder, 1998; Holland & O'Sullivan, 2005). The goal is typically to find a solution that is robust to possible changes. Other related models include *Mixed CSPs* (Fargier et al., 1996), which model decision problems under uncertainty by introducing state variables, which are not under control of the solver, and seek assignments that are consistent to any state of the world; and *Stochastic CSPs* (Walsh, 2002; Tarim et al., 2006), which introduce probability distributions that are associated to outcomes of state variables, and seek solutions that maximize the probability of constraint consistencies. While these proactive approaches have been used to solve CSP variants, they have not been used to solve Dynamic DCOPs to the best of our knowledge.

Researchers have also introduced *Markovian D-DCOPs (MD-DCOPs)*, which models D-DCOPs with state variables that are beyond the control of agents (Nguyen et al., 2014). However, they assume that the state is observable to the agents, while PD-DCOPs do not assume the observability of the state and are able to solve the problem even when the state is not observable. Additionally, MD-DCOP agents do not incur a cost for changing values in MD-DCOPs and only a reactive online approach to solving the problem has been proposed thus far.

Another related body of work is *Decentralized Markov Decision Processes (Dec-MDPs)* (Bernstein et al., 2002). In a Dec-MDP, agents can also observe its local state (the global state is the combination of all local states), and the goal of a Dec-MDP is to find a policy that maps each local state to the action for each agent. Thus, like PD-DCOPs, it also solves a sequential decision making problem. However, Dec-MDPs are typically solved in a centralized manner (Bernstein et al., 2002; Becker et al., 2004; Dibangoye et al., 2012, 2013) due to its high complexity – solving Dec-MDPs optimally is NEXP-complete even for the case with only two agents (Bernstein et al., 2002). In contrast, PD-DCOPs are solved in a decentralized manner and its complexity is only PSPACE-hard (see Theorem 1). The reason for the lower complexity is because the solution of PD-DCOPs are *open-loop* policies, which are policies that are not dependent on state observations.

*Decentralized Partially Observable MDPs (Dec-POMDPs)* (Bernstein et al., 2002) is a generalization of Dec-MDPs, where an agent may not accurately observe its local state. Instead, it maintains a belief of its local state. A Dec-POMDP policy thus maps each belief to an action for each agent. Solving Dec-POMDPs is also NEXP-complete (Bernstein et al., 2002) and they are also typically solved in a centralized manner (Hansen et al., 2004; Szer et al., 2005; Seuken & Zilberstein, 2007; Witwicki & Durfee, 2011; Dibangoye et al., 2013; Oliehoek et al., 2013) with some exceptions (Nair et al., 2003). Researchers have also developed a hybrid model, called ND-POMDP (Nair et al., 2005), which is a Dec-POMDP that exploits locality of agent interactions like a DCOP.

In summary, one can view DCOPs and Dec-(PO)MDPs as two ends of a spectrum of offline distributed planning models. In terms of expressivity, DCOPs can only model problems with single time step while Dec-(PO)MDPs can model multiple-time-step problems. However, DCOPs are only NP-hard while Dec-(PO)MDPs are NEXP-complete. PD-DCOPs attempt to balance the trade off between expressivity and complexity by searching for open-loop policies instead of closed-loop policies of Dec-(PO)MDPs. They are thus more expressive than DCOPs at the cost of a higher complexity, yet not as expressive as Dec-(PO)MDPs but also without its prohibitive complexity.

## 8. Experimental Results

In this section, we empirically evaluate our PD-DCOP algorithms. Aside from evaluating the PD-DCOP algorithms in an offline manner, we also evaluate them in an online setting which simulates the environment of many real-life applications. In the online setting, we consider both how long it takes for the algorithms to solve the problem at a given time step and the time they have to adapt the solution that they have just found. As PD-DCOPs can be solved in an offline or an online manner, we report the experimental results for both settings. Our experiments are performed on a 2.1GHz machine with 16GB of RAM using JADE framework (Bellifemine et al., 2005), and the results report the average of 30 independent runs, each with a timeout of 30 minutes.[10]

10. `https://github.com/khoihd/pd-dcop`.

### 8.1 Offline Algorithms

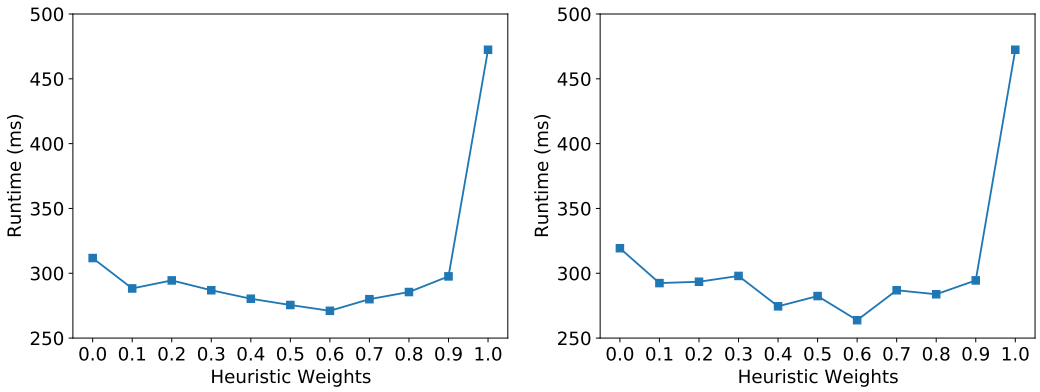
We first evaluate and report the experimental results of our PD-DCOP algorithms in the offline setting. The experiment in the offline setting will shed light on the performance of the PD-DCOP algorithms in the scenario where time and computing resources are generally available. We use the following default configuration: Number of agents and decision variables $|\mathbf{A}| = |\mathbf{X}| = 10$; number of random variables $|\mathbf{Y}| = 0.2 \cdot |\mathbf{X}| = 0.2 \cdot 10 = 2$; domain size $|D_x| = |\Omega_y| = 3$; horizon $h = 4$; and switching cost $c = 50$.[11] The utility values are sampled from the uniform distribution on $[0, 10]$. The initial probability distributions and the transition functions of random variables are randomly generated and normalized. We report solution quality and *simulated runtime* (Sultanik et al., 2008). Specifically, we evaluate the following offline PD-DCOP algorithms:

- *Collapsed DPOP (C-DPOP)*, which uses the exact approach introduced in Subsection 5.1. The C-DPOP algorithm collapses the PD-DCOP into a single DCOP and solves it with DPOP.
- *Local Search S-DPOP (LS-SDPOP)*, which uses the local search approach introduced in Subsection 5.2.1. This algorithm solves for the initial solution for the DCOP at each time step by running S-DPOP and then searches for better solutions.
- *Local Search MGM (LS-MGM)*, which uses the local search approach like LS-SDPOP. However, LS-MGM solves for the initial the solution for the DCOP at each time step by running MGM.
- *Local Search Random (LS-RAND)*, which uses the local search approach like LS-SDPOP and LS-MGM. However, LS-RAND randomly initializes solution for the DCOP at each time step.
- *Forward DPOP (F-DPOP)*, which uses the greedy approach FORWARD introduced in Subsection 5.2.2. F-DPOP sequentially solves the DCOP at each time step with DPOP.
- *Forward MGM (F-MGM)*, which uses the greedy approach FORWARD like F-DPOP. However, F-MGM sequentially solves the DCOP at each time step with MGM.
- *Backward DPOP (B-DPOP)*, which uses the greedy approach BACKWARD introduced in Subsection 5.2.2. B-DPOP sequentially solves the DCOP at each time step with DPOP.
- *Backward MGM (B-MGM)*, which uses the greedy approach BACKWARD like B-DPOP. However, B-MGM sequentially solves the DCOP at each time step with MGM.

#### 8.1.1 Random Networks

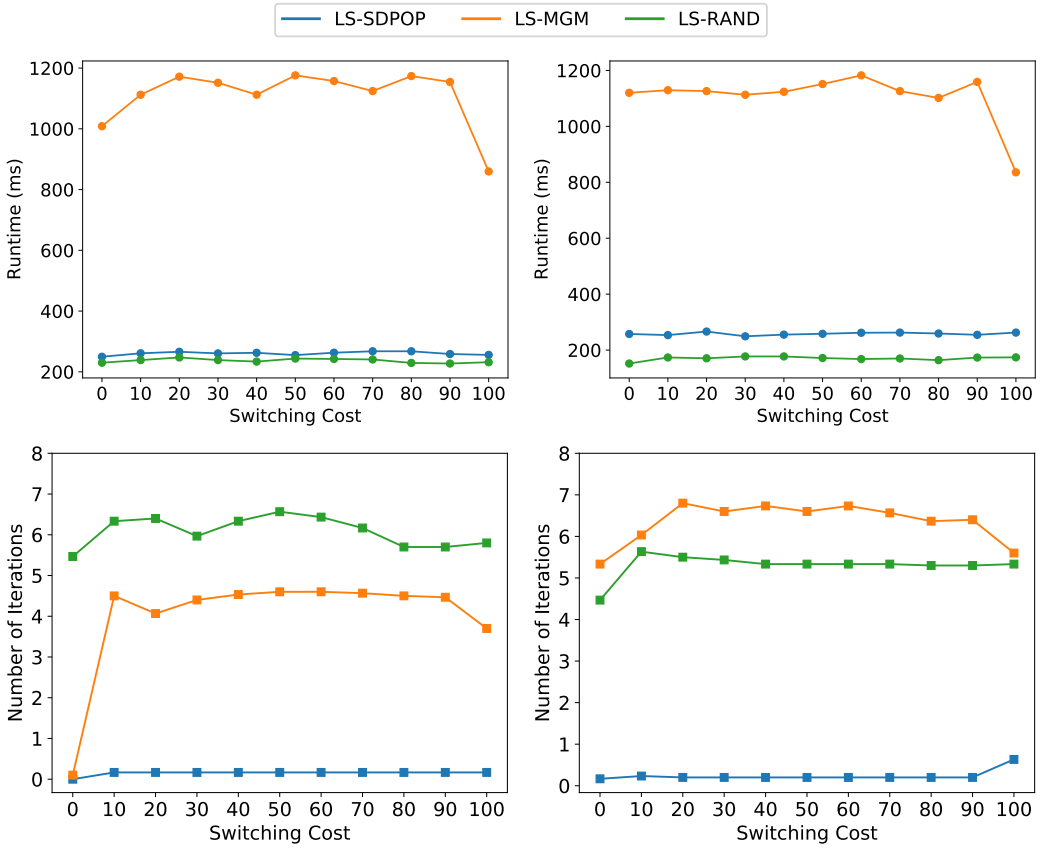
In this experiment, we use random networks with constraint density $p_1 = 0.5$ to evaluate the PD-DCOP algorithms on random instances that do not have a too dense or too sparse topology. As LS-SDPOP reuses information by applying the hybrid heuristic function $h_3$, we vary the heuristic weight $w$ and measure the runtime to evaluate its impact on LS-SDPOP. Figure 4 shows the runtime of LS-SDPOP run on PD-DCOPs with $\gamma = 0.9$ and $\gamma = 1$. At $w = 0$, the heuristic $h_3$ corresponds the max-degree heuristic $h_2$, and at $w = 1$, the heuristic is analogous to our $h_1$ heuristic (see Equation 19). The runtime is high at both extremes for the following reasons: When $w = 0$, LS-SDPOP exploits weakly the reuse of information,

11. The random variables are randomly associated with the utility functions such that each utility function has at most one random variable.




(a) $\gamma = 0.9$ (b) $\gamma = 1$

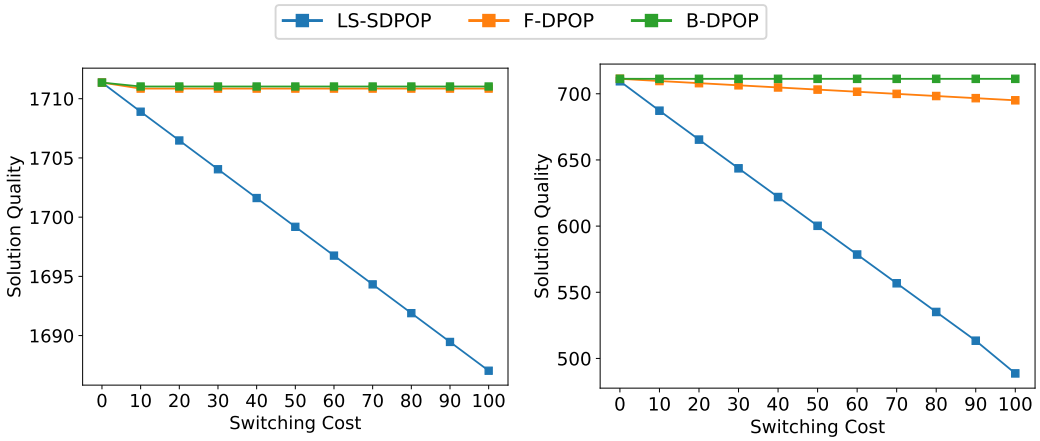
Figure 4: Experimental Results Varying Heuristic Weight




(a) $\gamma = 0.9$ (b) $\gamma = 1$

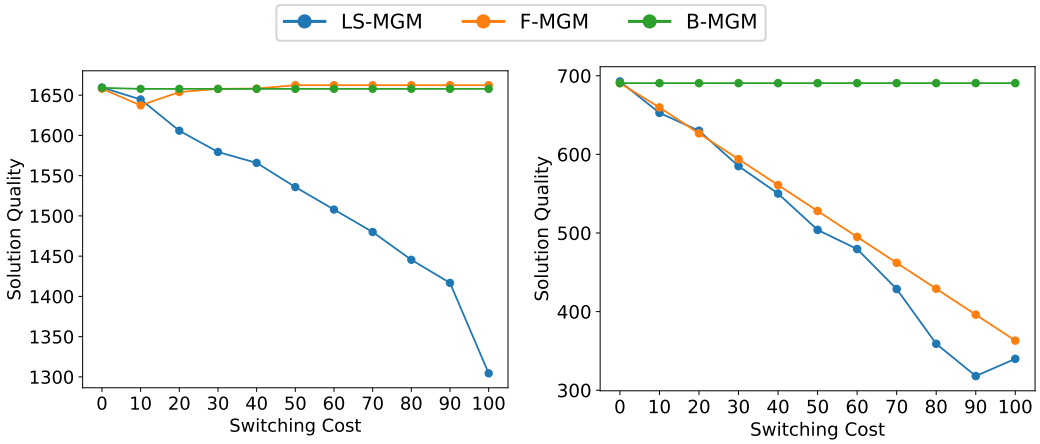
Figure 5: Experimental Results Varying Switching Cost

and when $w = 1$, the resulting pseudo-trees have larger depth, which in turn result in larger runtime. In both cases, the best weight is found at $w = 0.6$, where LS-SDPOP is able to




(a) $\gamma = 0.9$ (b) $\gamma = 1$

Figure 6: Comparison between Sequential Greedy and Local Search for DPOP

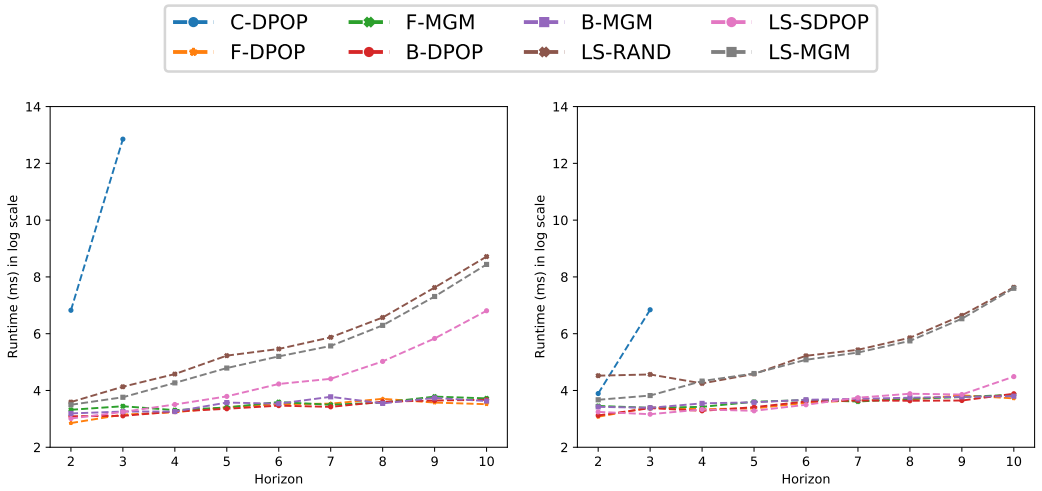



(a) $\gamma = 0.9$ (b) $\gamma = 1$

Figure 7: Comparison between Sequential Greedy and Local Search for MGM

reuse information in the most efficient way and has the smallest runtime. Thus, we set the heuristic weight $w = 0.6$ for LS-SDPOP in the remaining experiments.

Next, we vary the switching cost $c$ from 0 to 100 to evaluate its impact on the following PD-DCOP local search algorithms: LS-SDPOP, LS-MGM and LS-RAND. Figure 5 shows the runtime and the number of iterations that the algorithms take to converge to the final solution. When $c = 0$, the initial solution found by LS-SDPOP is the optimal solution of the PD-DCOP since LS-SDPOP solves the DCOP at each time step optimally by the ignoring the switching cost between them. Thus, it takes 0 iteration for LS-SDPOP to converge since the initial solution is also the final solution. When $c$ increases, LS-SDPOP takes more iterations to converge since it spends more time on searching for a solution that incurs less switching cost. The trend is similar for LS-MGM and LS-RAND in that the number of iterations and the runtime increase with the switching cost. Among three




(a) $\gamma = 0.9$ (b) $\gamma = 1$

Figure 8: Experimental Results Varying Horizon

algorithms, LS-SDPOP requires fewer iterations to converge than LS-RAND and LS-MGM. While LS-SDPOP solves each DCOP optimally, LS-MGM solves each DCOP sub-optimally with MGM and LS-RAND randomly chooses the initial solution for each DCOP. For that reason, LS-SDPOP has the best initial solution and requires the fewest iterations. While LS-MGM is faster than LS-SDPOP in solving for the initial solution and takes fewer iterations to converge than LS-RAND, LS-MGM is slowest among three algorithms. This experiment illustrates the impact of the quality of the initial solution on the number of iterations and the trade-off between the time spent on solving for the initial solution and the time spent on searching for better solutions.

In order to evaluate the impact of switching cost on the solution quality of two different heuristics: Local Search and Sequential Greedy, we vary the switching cost and report the solution quality of the heuristic algorithms. Figure 6 shows the solution quality of LS-SDPOP, F-DPOP and B-DPOP with DPOP as the algorithm solving the DCOP at each time step optimally. The LS-SDPOP algorithm starts by solving the DCOP at each time step without considering the switching cost, and then it locally searches for better solution in an iterative manner. When the switching cost becomes larger, the quality of initial solution found by LS-SDPOP decreases due to the higher cost from the difference in the solutions between two time steps. After solving for the initial solution, LS-SDPOP executes the local search process, which is based on the hill climbing heuristic used in MGM, and the solution will potentially get stuck at local maxima. For that reason, large switching cost has a high impact on the final solution of LS-SDPOP. On the other hand, when sequentially solving for the DCOP at each time step, Sequential Greedy algorithms such as F-DPOP and B-DPOP already take into account the solution of the previously solved DCOP by creating a unary constraint (see Equations 23 - 26). Therefore, while the solution qualities of three algorithms decrease when the switching cost increases, the solution quality of LS-SDPOP decreases more significantly than the solution quality of F-DPOP and B-DPOP. We observe a similar trend in Figure 7 between LS-MGM, F-MGM, and B-MGM. However, the trend

| $\|\mathbf{A}\|$ | C-DPOP | | LS-SDPOP | | LS-MGM | | LS-RAND | | F-DPOP | | F-MGM | | B-DPOP | | B-MGM | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $q$ | $t$ | $q$ | $t$ | $q$ | $t$ | $q$ | $t$ | $q$ | $t$ | $q$ | $t$ | $q$ | $t$ | $q$ | $t$ |
| 5 | — | — | 416 | 41 | 359 | 75 | 401 | 102 | 437 | 35 | 417 | 32 | 439 | 35 | 423 | 31 |
| 10 | — | — | 1699 | 236 | 1535 | 255 | 1620 | 238 | 1710 | 316 | 1677 | 118 | 1711 | 304 | 1677 | 106 |
| 15 | — | — | — | — | 3176 | 444 | 3414 | 386 | — | — | 3575 | 234 | — | — | 3589 | 224 |
| 20 | — | — | — | — | 5614 | 616 | 6066 | 536 | — | — | 6205 | 392 | — | — | 6222 | 376 |
| 25 | — | — | — | — | 8737 | 818 | 9357 | 650 | — | — | 9493 | 553 | — | — | 9524 | 565 |
| 30 | — | — | — | — | 12651 | 1185 | 13166 | 821 | — | — | 13620 | 812 | — | — | 13578 | 841 |
| 35 | — | — | — | — | 16969 | 1452 | 17876 | 1039 | — | — | 18325 | 1013 | — | — | 18303 | 1053 |
| 40 | — | — | — | — | 22237 | 1732 | 23038 | 1121 | — | — | 23602 | 1224 | — | — | 23595 | 1268 |
| 45 | — | — | — | — | 28251 | 1944 | 29159 | 1358 | — | — | 29611 | 1481 | — | — | 29655 | 1523 |
| 50 | — | — | — | — | 33929 | 1987 | 35900 | 1726 | — | — | 36500 | 1736 | — | — | 36454 | 1803 |

Table 1: Varying the Number of Agents on Random Graphs with $\gamma = 0.9$

| $\|\mathbf{A}\|$ | C-DPOP | | LS-SDPOP | | LS-MGM | | LS-RAND | | F-DPOP | | F-MGM | | B-DPOP | | B-MGM | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $q$ | $t$ | $q$ | $t$ | $q$ | $t$ | $q$ | $t$ | $q$ | $t$ | $q$ | $t$ | $q$ | $t$ | $q$ | $t$ |
| 5 | 185 | 346012 | 163 | 32 | 133 | 76 | -17 | 70 | 156 | 39 | 121 | 31 | 185 | 42 | 176 | 33 |
| 10 | — | — | 600 | 238 | 503 | 209 | 393 | 177 | 703 | 322 | 554 | 110 | 711 | 312 | 697 | 113 |
| 15 | — | — | — | — | 1270 | 387 | 931 | 296 | — | — | 1235 | 216 | — | — | 1481 | 218 |
| 20 | — | — | — | — | 2116 | 522 | 1913 | 417 | — | — | 2221 | 365 | — | — | 2569 | 395 |
| 25 | — | — | — | — | 3397 | 661 | 3049 | 482 | — | — | 3513 | 546 | — | — | 3923 | 565 |
| 30 | — | — | — | — | 5022 | 840 | 4392 | 552 | — | — | 5106 | 784 | — | — | 5596 | 832 |
| 35 | — | — | — | — | 6738 | 985 | 5957 | 577 | — | — | 7021 | 932 | — | — | 7544 | 1047 |
| 40 | — | — | — | — | 8548 | 1079 | 7853 | 636 | — | — | 8962 | 1188 | — | — | 9720 | 1284 |
| 45 | — | — | — | — | 10938 | 1146 | 10302 | 752 | — | — | 11250 | 1356 | — | — | 12183 | 1540 |
| 50 | — | — | — | — | 13290 | 1267 | 12865 | 803 | — | — | 14240 | 1614 | — | — | 15000 | 1789 |

Table 2: Varying the Number of Agents on Random Graphs with $\gamma = 1$

tends to fluctuate due to the instability in the quality of solution of each DCOP found by MGM.

We then vary the horizon $h$ from 2 to 10 and compare the runtime of all PD-DCOP algorithms. In this experiment, we set the number of decision variables $|\mathbf{X}| = 5$ and report the runtime in log scale in Figure 8. First, we observe that the exact algorithm C-DPOP has the largest runtime at $h = 2$ and $h = 3$, and it cannot scale to solve problems with horizon $h > 3$. The reason is that C-DPOP collapses all DCOPs into a single DCOP that has the domain size growing exponentially in $h$. The exponential growth in domain size severely affects the runtime of DPOP that is used to solve the collapsed DCOP. When the horizon increases, we observe that local search algorithms (LS-RAND, LS-MGM, and LS-SDPOP) generally run slower than sequential algorithms (F-MGM, B-MGM, F-DPOP, B-DPOP). Among the local search algorithms, LS-SDPOP is faster than both LS-MGM and LS-RAND, and all the sequential algorithms have similar runtimes. This trend is similar for both cases $\gamma = 0.9$ and $\gamma = 1$. The only difference between the two cases is the runtimes of C-DPOP and local search algorithms. When $\gamma = 1$, C-DPOP collapses the DCOPs from time step $t = 0$ to time step $t = h-1$, which is one DCOP fewer compared to when $\gamma = 0.9$. The smaller size of the collapsed DCOP has resulted in significantly smaller runtime for C-DPOP. Similarly, the search space of the local search algorithm is also smaller when $\gamma = 1$.

Finally, we vary the number of agents $|\mathbf{A}|$ from 5 to 50 to evaluate the performance of the algorithms with different number of agents. Table 1 reports solution quality (labeled $q$) and runtime (labeled $t$) of all PD-DCOP algorithms for $\gamma = 0.9$. We observe that C-DPOP times out on all instances due to the large horizon of $h = 4$. On small problems that have $|\mathbf{A}| = 5$ or $|\mathbf{A}| = 10$, DPOP-based algorithms provide solutions with higher quality than those solved by MGM-based algorithms and LS-RAND. However, due to solving the DCOP at each time step optimally, DPOP-based algorithms cannot scale and time out when the number of agents is larger. On the other hand, MGM-based algorithms, which solve each DCOP sub-optimally with MGM, and LS-RAND can scale to solve large problems. In addition, Sequential Greedy algorithms such as F-DPOP and B-DPOP have better solution quality than the Local Search algorithm such as LS-SDPOP due to the large switching cost, which is set at $c = 50$. The similar trend also happens when we compare the solution quality of MGM-based algorithms such as F-MGM, B-MGM, and LS-MGM. We observe the similar result in Table 2 for PD-DCOPs with $\gamma = 1$. The key difference is that C-DPOP can solve problems with $|\mathbf{A}| = 5$ because C-DPOP collapses one fewer DCOP in PD-DCOPs with $\gamma = 1$ compared to PD-DCOPs with $\gamma = 0.9$. Thus, the resulting collapsed DCOP is smaller and it takes less time for C-DPOP to solve.

#### 8.1.2 Dynamic Distributed Meeting Scheduling

Next, we evaluate our PD-DCOP algorithms on dynamic distributed meeting scheduling problems, which are a real world application with a specific network topology. We generate the underlying topology randomly with $p_1 = 0.5$ and use the PEAV formulation (Maheswaran et al., 2004b). In this formulation, we enforce the inequality constraints to ensure that no two meetings can be held at the same time. We vary the number of meetings and allow each meeting to be scheduled in 5 different starting times. If an algorithm fails to find a feasible solution for some instance, we do not report the runtime of that instance.

Tables 3 and 4 report the runtime (labeled $t$) and the percentage of feasible solutions (labeled %SAT) on PD-DCOPs with $\gamma = 0.9$ and $\gamma = 1$, respectively. As we observed from Tables 1 and 2, DPOP-based algorithms return solutions with higher quality than those by LS-RAND and MGM-based algorithms. Similarly, in distributed meeting scheduling problems, DPOP-based algorithms are able to find feasible solutions for many instances, but LS-RAND and MGM-based algorithms fail on most instances. However, it comes at the cost of larger runtime for DPOP-based algorithms since solving each DCOP optimally usually takes longer. Among MGM-based algorithms, LS-MGM find more feasible solutions than F-MGM and B-MGM. Since MGM is not an exact algorithm, the solution for the problem at each time step is usually infeasible. Once MGM cannot find a feasible solution for a problem at some time step, sequential greedy algorithms such as F-MGM and B-MGM do not have a mechanism to improve those infeasible solutions to make them feasible. On the other hand, despite having initial infeasible solutions, LS-MGM can gradually modify the initial solution with local search, and thus it is able to change the initial infeasible solution to a feasible solution. However, LS-MGM is slightly slower than F-DPOP and B-DPOP due to the additional local search step.

| \|A\| | LS-SDPOP | | LS-MGM | | LS-RAND | | F-DPOP | | F-MGM | | B-DPOP | | B-MGM | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | %SAT | $t$ | %SAT | $t$ | %SAT | $t$ | %SAT | $t$ | %SAT | $t$ | %SAT | $t$ | %SAT | $t$ |
| 4 | 100 | 94 | 80 | 229 | 70 | 226 | 100 | 39 | 10 | 33 | 100 | 39 | 10 | 36 |
| 6 | 100 | 150 | 20 | 318 | 26 | 338 | 100 | 112 | 0 | — | 100 | 104 | 0 | — |
| 8 | 100 | 405 | 3 | 380 | 20 | 480 | 100 | 372 | 0 | — | 100 | 360 | 0 | — |
| 10 | 100 | 27062 | 0 | — | 3 | 490 | 100 | 16855 | 0 | — | 100 | 16864 | 0 | — |

Table 3: Results for Distributed Meeting Scheduling Problem with $\gamma = 0.9$

| \|A\| | LS-SDPOP | | LS-MGM | | LS-RAND | | F-DPOP | | F-MGM | | B-DPOP | | B-MGM | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | %SAT | $t$ | %SAT | $t$ | %SAT | $t$ | %SAT | $t$ | %SAT | $t$ | %SAT | $t$ | %SAT | $t$ |
| 4 | 73 | 40 | 23 | 108 | 20 | 135 | 93 | 43 | 6 | 35 | 100 | 42 | 10 | 30 |
| 6 | 76 | 105 | 10 | 210 | 0 | — | 100 | 105 | 0 | — | 100 | 105 | 0 | — |
| 8 | 90 | 345 | 0 | — | 0 | — | 100 | 362 | 0 | — | 100 | 361 | 0 | — |
| 10 | 96 | 27060 | 0 | — | 0 | — | 100 | 17175 | 0 | — | 100 | 16986 | 0 | — |

Table 4: Results for Distributed Meeting Scheduling Problem with $\gamma = 1$

#### 8.1.3 Distributed Radar Coordination and Scheduling

In this experiment, we evaluate our PD-DCOP algorithms on the Distributed Radar Coordination and Scheduling Problem (DRCSP), our motivating application described in Section 3. We use grid networks to represent the DRCSP where sensors are arranged in a rectangular grid. Each sensor has 8 sensing directions and is connected to its four neighboring sensors in the cardinal direction. Those sensors on the edges are connected to three neighboring sensors, and corner sensors are connected to two neighbors. The random variables, which represent the precipitation of the weather phenomena, are randomly placed on the network.

Tables 5 and 6 report the runtime (labeled $t$) and the solution quality (labeled $q$) of PD-DCOP algorithms on DRCSP with $\gamma = 0.9$ and $\gamma = 1$, respectively. Similar to the result of the experiments on random networks and distributed meeting scheduling problems, DPOP-based algorithms achieve higher quality solutions with than those found by LS-RAND and its counterpart MGM-based algorithms. However, the better solution of DPOP-based algorithms comes with a cost of higher runtimes where DPOP-based algorithms run longer than MGM-based algorithms. In addition, due to larger runtime of the DPOP-based algorithms, they time out when solving larger instances with 16, 18, 20 agents. In contrast, MGM-based algorithms successfully finish within the time limit on those larger instances.

### 8.2 Online Algorithms

In this section, we compare our proactive approach and the reactive approach in an online setting in order to identify the characteristics of the problems in that they excel in. In addition, we propose *hybrid approach*, which is a combination of proactive and reactive approaches, and we compare our hybrid approach against the reactive approach. For a fair comparison, we empirically evaluate all approaches in the same online setting. As PD-DCOPs can be solved in an online manner, we compare the following online approaches: FORWARD, HYBRID, and REACT.

| \|**A**\| | LS-SDPOP | | LS-MGM | | LS-RAND | | F-DPOP | | F-MGM | | B-DPOP | | B-MGM | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $q$ | $t$ | $q$ | $t$ | $q$ | $t$ | $q$ | $t$ | $q$ | $t$ | $q$ | $t$ | $q$ | $t$ |
| 4 | 315 | 323 | 283 | 1504 | 120 | 1683 | 327 | 14 | 306 | 19 | 327 | 15 | 305 | 21 |
| 6 | 513 | 499 | 474 | 2097 | 213 | 2296 | 553 | 19 | 523 | 21 | 555 | 20 | 525 | 23 |
| 8 | 843 | 847 | 783 | 3592 | 369 | 3242 | 895 | 62 | 848 | 34 | 897 | 64 | 851 | 34 |
| 10 | 1030 | 1360 | 1010 | 3560 | 444 | 3368 | 1117 | 65 | 1054 | 41 | 1120 | 64 | 1049 | 41 |
| 12 | 1474 | 2686 | 1400 | 4099 | 618 | 4321 | 1519 | 21036 | 1429 | 76 | 1522 | 21332 | 1423 | 72 |
| 14 | 1513 | 55739 | 1525 | 4412 | 707 | 4219 | 1686 | 69511 | 1583 | 52 | 1690 | 69153 | 1599 | 54 |
| 16 | — | — | 1767 | 4854 | 860 | 4478 | — | — | 1848 | 81 | — | — | 1841 | 80 |
| 18 | — | — | 2004 | 5076 | 891 | 4856 | — | — | 2110 | 88 | — | — | 2112 | 80 |
| 20 | — | — | 2332 | 5543 | 1028 | 5269 | — | — | 2436 | 96 | — | — | 2442 | 95 |

Table 5: Results for Distributed Radar Coordination and Scheduling Problem with $\gamma = 0.9$

| \|**A**\| | LS-SDPOP | | LS-MGM | | LS-RAND | | F-DPOP | | F-MGM | | B-DPOP | | B-MGM | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $q$ | $t$ | $q$ | $t$ | $q$ | $t$ | $q$ | $t$ | $q$ | $t$ | $q$ | $t$ | $q$ | $t$ |
| 4 | 136 | 22 | 56 | 226 | 18 | 265 | 134 | 12 | 63 | 18 | 139 | 12 | 130 | 17 |
| 6 | 213 | 103 | 78 | 297 | 31 | 313 | 205 | 20 | 89 | 27 | 238 | 18 | 224 | 20 |
| 8 | 315 | 120 | 153 | 374 | 91 | 392 | 297 | 59 | 122 | 37 | 382 | 58 | 361 | 33 |
| 10 | 401 | 861 | 172 | 456 | 114 | 422 | 405 | 58 | 153 | 43 | 479 | 64 | 451 | 41 |
| 12 | 508 | 1646 | 260 | 511 | 188 | 526 | 525 | 20910 | 232 | 66 | 648 | 20619 | 610 | 70 |
| 14 | 544 | 57709 | 305 | 459 | 210 | 482 | 589 | 69315 | 265 | 53 | 723 | 69086 | 679 | 49 |
| 16 | — | — | 349 | 573 | 243 | 548 | — | — | 332 | 73 | — | — | 792 | 70 |
| 18 | — | — | 393 | 534 | 255 | 532 | — | — | 392 | 79 | — | — | 902 | 84 |
| 20 | — | — | 480 | 587 | 375 | 567 | — | — | 408 | 84 | — | — | 1046 | 85 |

Table 6: Results for Distributed Radar Coordination and Scheduling Problem with $\gamma = 1$

**FORWARD:** Since FORWARD solves the problem at each time step beforehand, it can be used as an offline approach or an online approach. Similar to the offline approach, online FORWARD reformulates the constraints based on the probability distribution of random variables, solves each problem sequentially, and takes into account the switching cost between the problem at the current time step and the problem at the previous time step. In this experiment, we will evaluate FORWARD as a proactive online approach.

**REACT:** REACT waits for each problem to change, observes the realization of random variables, and solves the problem in a *reactive* manner. Similar to FORWARD, REACT takes into account the switching cost between the problem at the current time step and the problem at the previous time step. As REACT observes the problem to change before solving it, REACT is a reactive online approach and cannot be used as an offline approach.

**HYBRID:** While FORWARD solves each problem beforehand and REACT waits for the problem to change before solving it, HYBRID is a combination of the two approaches. Similar to FORWARD, HYBRID greedily solves the problem from the first time step $t = 0$ onwards. The difference is that it will observe the values of the random variables at each time step $t \geq 0$ and using them to retrieve the probability distribution of the random variables

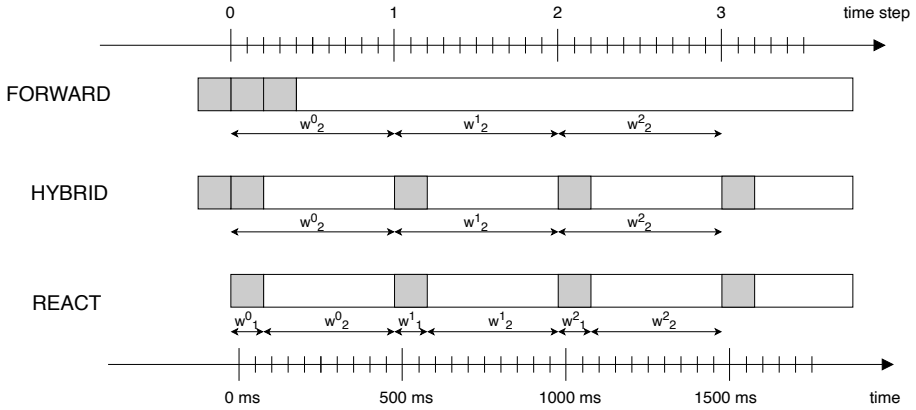



Figure 9: Search Time vs. Solution Adoption Time

in the next time step from the transition function. It then solves the problem for the next time step with the updated probability distributions thereby finding better solutions than the FORWARD algorithm. HYBRID is an online hybrid approach and cannot be used as an offline approach.

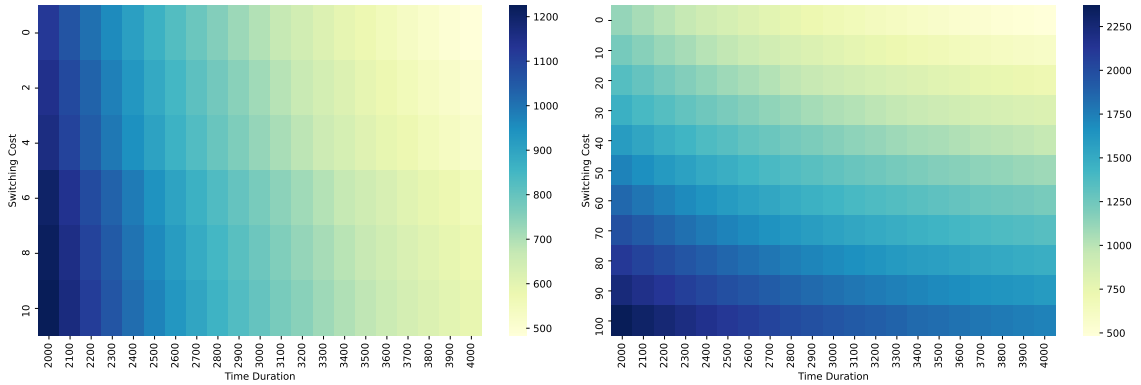
Figure 9 illustrates the time the three approaches spend searching for solutions (denoted by gray rectangles) as well as the time they adopt their solutions (denoted by white rectangles), where the time duration between iterations is 500ms. FORWARD starts searching for optimal solutions before the problem starts, and adopts the solution later. HYBRID solves the first sub-problem at $t = 0$ based on the initial distribution of random variables, which is known a priori. When the problem starts, HYBRID adopts the solution while observing the values of random variables, using the observation to find its solution for the next time step. Finally, REACT solves the problem each time the problem changes.

The *effective utility* $U_{eff}$ of REACT in each time step $t$ is defined as the normalized weighted sum:

$$U_{eff} = \frac{w_1^t \cdot q_{t-1}^t + w_2^t \cdot q_t^t - (w_1^t + w_2^t) \cdot c_{t-1,t}}{w_1^t + w_2^t} \tag{80}$$

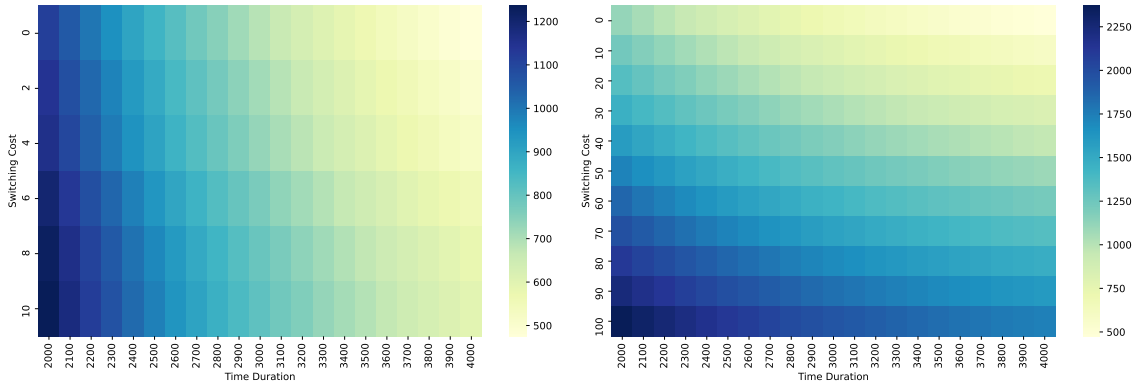
where $w_1^t$ is the duration it spent searching for a solution at time step $t$;[12] $w_2^t$ is the duration it adopted the solution found; $q_{t-1}^t$ is the quality of solution found in the previous time step $t-1$; $q_t^t$ is the quality of solution found in the current time step $t$; and $c_{t-1,t}$ is the switching cost incurred between the two time steps. The effective utility takes into account: (i) the quality $q_{t-1}^t$ of the solution found in the previous time step while the algorithm is searching for a solution in the current time step; (ii) the quality $q_t^t$ of the solution found in the current time step; and (iii) the switching cost $c_{t-1,t}$ incurred by the solutions found in the current time step and the previous time step. For FORWARD and HYBRID, since they find a solution for each time step before the start of that time step, $w_1^t = 0$ for all time steps, and the effective utility takes into account the solution quality of the current time step and the switching cost: $U_{eff} = q_t^t - c_{t-1,t}$. However, the solution found for each time step by the

12. We discretize time into 50ms intervals.




(a) Small Switching Cost Range (b) Large Switching Cost Range

Figure 10: Comparison between F-DPOP and R-DPOP on Random Networks



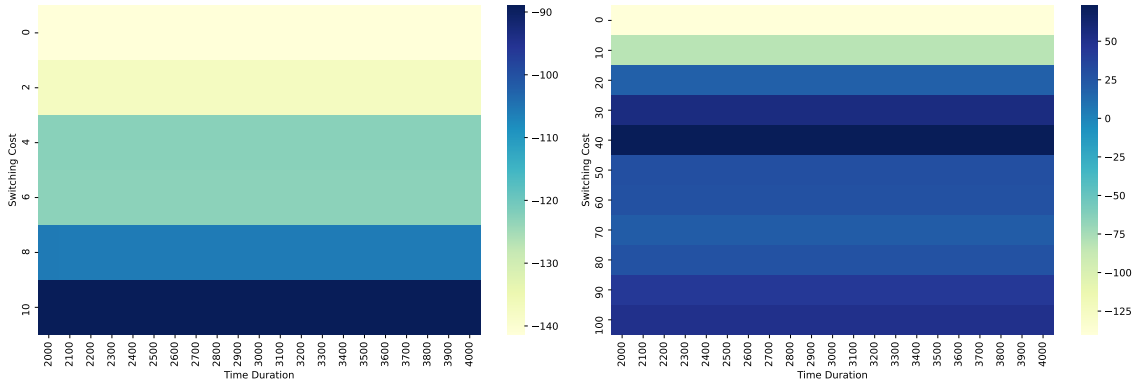

(a) Small Switching Cost Range (b) Large Switching Cost Range

Figure 11: Comparison between Hy-DPOP and R-DPOP on Random Networks

three approaches are likely to differ and we aim to experimentally evaluate the conditions in which one class of algorithms is preferred over the other.

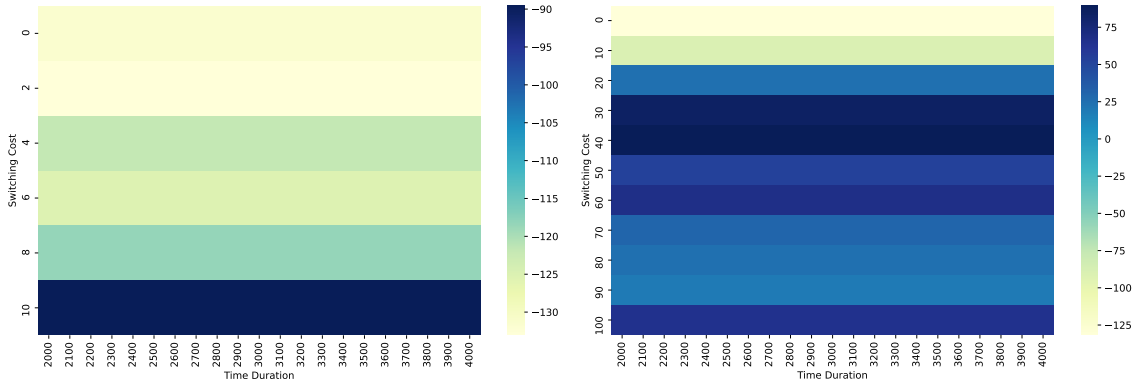
Choosing DPOP and MGM as two algorithms to solve the DCOP at each time step, we evaluate the following algorithms: *Forward DPOP (F-DPOP)*, *Forward MGM (F-MGM)*, *Hybrid DPOP (Hy-DPOP)*,[13] *Hybrid MGM (H-MGM)*, *Reactive DPOP (R-DPOP)*[14], and *Reactive MGM (R-MGM)* by varying two parameters – the time duration between subsequent time steps of the dynamic DCOP (i.e., the time before the DCOP changes) and the switching cost $c$ of the dynamic DCOP. We use the following default configuration: Number of agents and decision variables $|\mathbf{A}| = |\mathbf{X}| = |\mathbf{Y}| = 10$; domain size $|D_x| = |\Omega_y| = 5$; and horizon $h = 10$. We conduct our experiments on random networks with $p_1 = 0.5$ and distributed meeting scheduling problems. We report the average difference in effective utilities (see Equation 80) between a proactive or hybrid algorithm and its reactive counterpart. The

13. We avoid using the acronym H-DPOP as that refers to a different algorithm (Kumar et al., 2008)
14. R-DPOP is the online S-DPOP




(a) Small Switching Cost Range (b) Large Switching Cost Range

Figure 12: Comparison between F-MGM and R-MGM on Random Networks



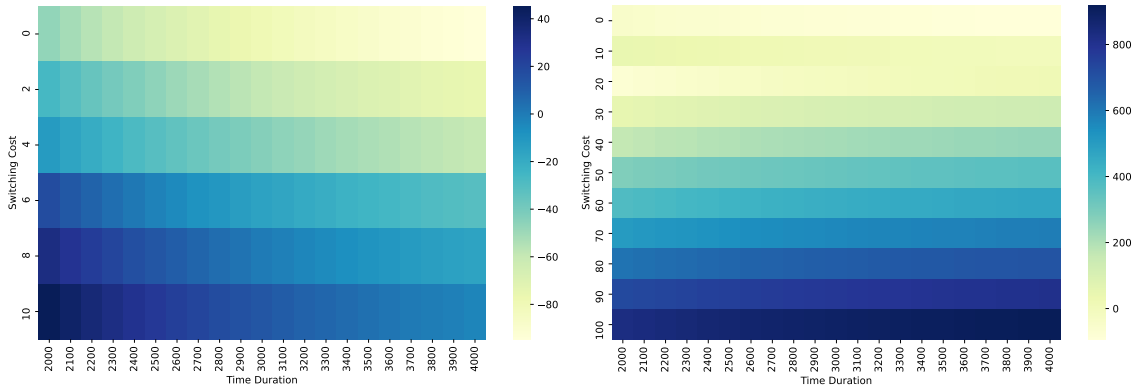

(a) Small Switching Cost Range (b) Large Switching Cost Range

Figure 13: Comparison between H-MGM and R-MGM on Random Networks

difference in effective utilities is the effective utility of the proactive or hybrid algorithm minus the effective utility of the reactive algorithm and divided by the horizon $h$.

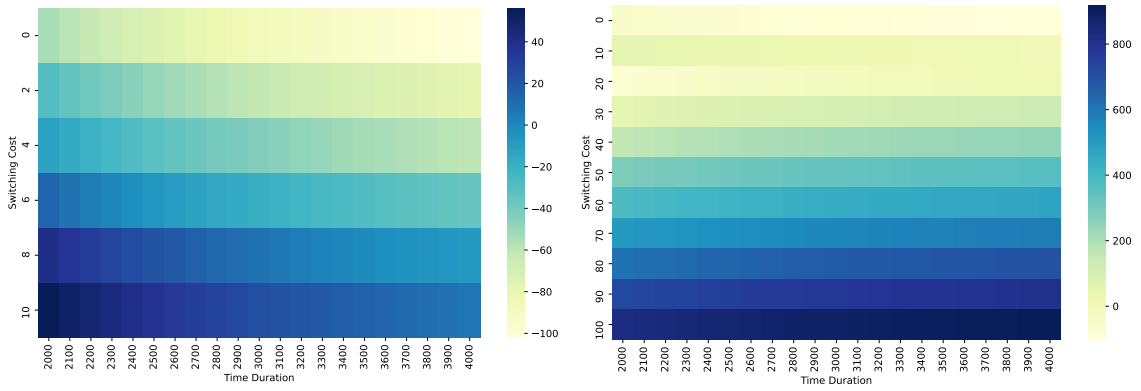
Figures 10(a) and 10(b) compare F-DPOP and R-DPOP with a small switching cost range of $[0, 10]$ and a large switching cost range of $[0, 100]$, respectively. The heatmap shows the average difference in the effective utilities between F-DPOP and R-DPOP. The difference is calculated by subtracting the effective utilities of F-DPOP from those of R-DPOP and divided by the horizon $h$. When the switching cost is 0, R-DPOP is able to find an optimal solution at each time step. However, when the cost increases, it may myopically choose a solution that is good for the current time step but bad for future time steps. Thus, R-DPOP is best when the switching cost is small and deteriorates with larger switching costs. When the time duration between subsequent time steps is small, R-DPOP spends most of the time on searching for the solution and little time on adopting it; vice versa when the time duration is large. Thus, in Figure 10(a) and Figure 10(b), R-DPOP is worst when the time duration is small and improves with longer duration.

We observe a similar trend in Figures 11(a) and 11(b), which show the result comparing Hy-DPOP and R-DPOP, except that the difference is marginally larger. The reason is




(a) Small switching cost range (b) Large switching cost range

Figure 14: Difference in Effective Utilities of F-DPOP minus R-DPOP on Distributed Meeting Scheduling Problems




(a) Small switching cost range (b) Large switching cost range

Figure 15: Comparison between Hy-DPOP and R-DPOP on Distributed Meeting Scheduling Problems

that Hy-DPOP uses its observation of the random variables in the current time step to compute a more accurate probability distribution of random variables for the next time step. By observing and getting better prediction on the values of random variables, Hy-DPOP can find better solutions. Moreover, unlike R-DPOP, Hy-DPOP is able to adopt the solution immediately when the problem changes. Therefore, it combines the strengths of both proactive and reactive algorithms.

We observe similar trends in Figures 12 and 13, where we use MGM instead of DPOP to solve the DCOP at each time step on random networks. Similar to the results in Figures 10 and 11, the reactive approach, which is R-MGM in this case, is best when the switching cost is 0 and deteriorates with larger switching costs. R-MGM is also worst when the time duration is small and improves with longer duration when the switching cost value is small

and vice versa. However, the trend tends to fluctuate due to the instability in the quality of the solutions found by MGM.

We also report our online experimental results comparing F-DPOP and Hy-DPOP against R-DPOP on distributed meeting scheduling problems in Figures 14 and 15. We observe a similar trend as in random networks. The reactive algorithm is best if the switching cost is 0 and its solution quality decreases when the switching cost increases. Also, R-DPOP is worst when the time duration is small and the solution quality increases with longer duration. However, in the distributed meeting scheduling problem, when the switching cost is increasing and becomes much larger, the difference in switching cost dominates the difference in utility. Since R-DPOP switches values between solutions more frequently than F-DPOP and Hy-DPOP, R-DPOP performs worse when the time duration increases and switching cost value is large, which is showed in Figure 14(b) and Figure 15(b). We do not evaluate the online approach that use MGM on distributed meeting scheduling problems since the effective utilities cannot be computed with infeasible solutions found by MGM.

Therefore, for the first time to the best of our knowledge, these experimental results shed light on the identification of characteristics that are well suited for each class of dynamic DCOP algorithms. Reactive algorithms are well suited for problems with small switching costs and that change slowly. In contrast, proactive algorithms are well suited for problems with large switching costs and that change quickly. Our hybrid algorithms combine the strengths of both approaches – it works well in the same type of problems that proactive algorithms work well in and it exploits observations to improve its solutions like reactive algorithms.

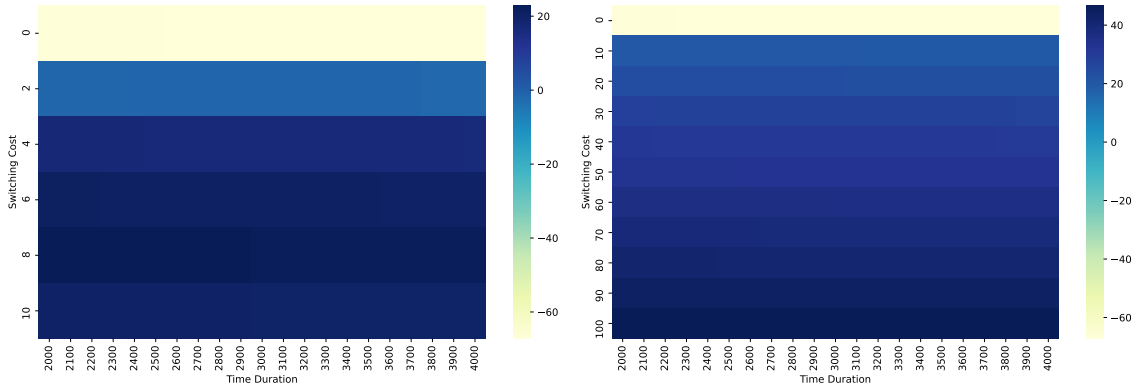
### 8.3 Comparisons with MD-DCOP Algorithms

In the online setting, the states of the random variables are observable by agents and, thus, PD-DCOPs can be modeled as Markovian Dynamic DCOPs (MD-DCOPs) (Nguyen et al., 2014) and solved by MD-DCOP algorithms. One of the key differences between the two models is that agents in PD-DCOPs incur some switching cost by changing solutions between two subsequent time steps while MD-DCOPs do not. In order to integrate switching cost in MD-DCOPs for fair comparisons, we first augment the states of the random variables with the solution of the decision variables in the previous time step and then add the switching cost to the utility function accordingly. Specifically, given a utility function $f_i$ of a PD-DCOP, where its scope contains a random variable $y_i$ with state $\omega_i^t$ in the current time step, and $\mathbf{x}_i^{t-1}$ as the assignment of the decision variables in the previous time step, the state of the random variable $y_i$ in the corresponding MD-DCOP is augmented as $\langle \omega_i^t, \mathbf{x}_i^{t-1} \rangle$. The utility function $f_i'$ of the MD-DCOP now takes into account the switching cost from the previous solution:

$$f_i'(\langle \omega_i^t, \mathbf{x}_i^{t-1} \rangle, \mathbf{x}_i^t) = f_i(\omega_i^t, \mathbf{x}_i^t) - c \cdot \Delta(\mathbf{x}_i^{t-1}, \mathbf{x}_i^t)$$

The transition function $T'$ of the random variable $y_i$ is defined as:

$$T'_{y_i}(\langle \omega_i^t, \mathbf{x}_i^{t-1} \rangle, \langle \omega_i^{t+1}, \mathbf{x}_i^t \rangle) = \begin{cases} T_{y_i}(\omega_i^t, \omega_i^{t+1}) & \text{if } \mathbf{x}_i^{t-1} = \mathbf{x}_i^t \\ 0 & \text{otherwise} \end{cases}$$




(a) Small switching cost range (b) Large switching cost range

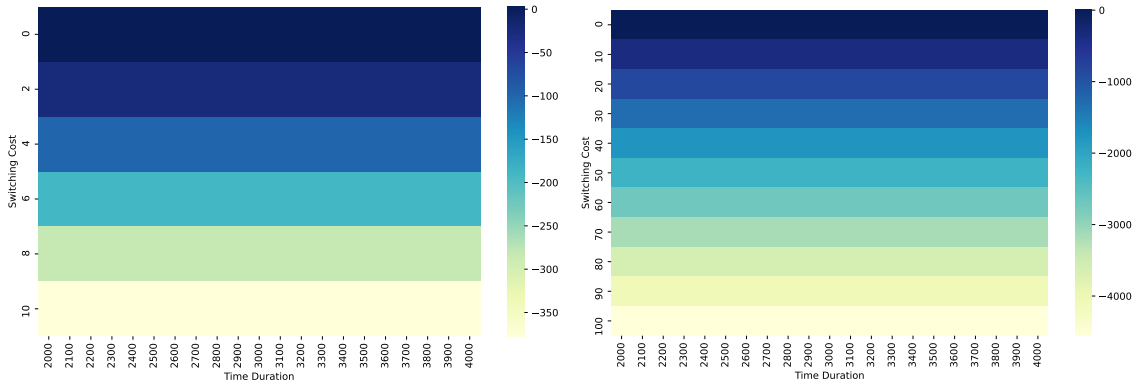
Figure 16: Difference in Effective Utilities of F-DPOP minus Decomposed Distributed R-learning

After this step, the PD-DCOP is now mapped to the MD-DCOP and it can be solved by MD-DCOP algorithms.

In this experiment, we choose F-DPOP and R-DPOP as our representative online algorithms and compare them against Decomposed Distributed R-learning (Nguyen et al., 2014), the best performing MD-DCOP algorithm. We run the experiment on random networks and use the following configuration: Number of agents and variables $|\mathbf{A}| = |\mathbf{X}| = |\mathbf{Y}| = 10$; $p_1 = 0.5$, domain size $|D_x| = |\Omega_y| = 5$. We consider the horizon when the distribution of all random variable has converged and let the algorithms solve the problem for 50 time steps. We report the average difference in effective utility between F-DPOP or R-DPOP and Decomposed Distributed R-learning.

Figures 16(a) and 16(b) compare F-DPOP and Decomposed Distributed R-learning with small switching cost and large switching cost, respectively. The heatmap shows the difference in average effective utility which is computed by subtracting the effective utilities of F-DPOP from those of Decomposed Distributed R-learning. When the switching cost is small, Decomposed Distributed R-learning is able to find better solution where it maps the actual state of the random variables to the final solution. In contrast, since the distribution of random variables have converged, the solution of F-DPOP is identical for these 50 time steps and it ignores the actual states of the random variables. Thus, Decomposed Distributed R-learning is able to take into account the state of the random variables and thus find better solution. However, when the switching cost is higher, the solution found by Decomposed Distributed R-learning is worse than F-DPOP. Since the solutions of F-DPOP between two time steps are identical, F-DPOP incurs no switching cost. In contrast, Decomposed Distributed R-learning still incurs some switching cost due to different states between two time steps and the mapping from the actual state of the random variables. Thus, it returns solutions with worse qualities than solutions of F-DPOP.

Figures 17(a) and 17(b) compare R-DPOP and Decomposed Distributed R-learning with small switching cost and large switching cost, respectively. When the switching cost is 0, R-DPOP is able find the optimal solution at each time step without incurring any switching




(a) Small switching cost range

(b) Large switching cost range

Figure 17: Difference in Effective Utilities of R-DPOP minus Decomposed Distributed R-learning

cost caused by the previous solution. Thus, it is able to find the optimal solution overall and the difference in the average effective utility between R-DPOP and Decomposed Distributed R-learning is marginally positive. However, when the switching cost increases, the solution quality of R-DPOP decreases since the switching cost is now larger and dominates the solution quality found in reactive manner. On the other hand, by integrating the previous solution into its augmented state, Decomposed Distributed R-learning is able to take the solution of the previous time step into account, and thus the difference between R-DPOP and Decomposed Distributed R-learning is smaller and becomes negative.

In summary, our experimental results have identified when a proactive or a reactive approach should be used to solve the problems that are beyond the horizon when the probability distributions of random variables have converged. On one hand, when the switching cost is large or when the computation resource is limited, it is desirable to have the same solution across different time steps that incurs less or even zero switching cost. Thus, FORWARD is a more suitable approach than REACTIVE and R-learning. On the other hand, when the switching cost is small, reactive algorithms such as R-DPOP and R-learning are able to gain higher solution quality by having different solutions at different time steps. When the switching cost increases, a less reactive approach like R-learning is able to avoid aggressively changing solutions such as those provided by R-DPOP, and it is able to gain higher solution quality. However, the downside of R-learning is that it requires a significant number of iterations for training before being able to achieve satisfactory solution qualities.

## 9. Conclusions

In many real-world applications, agents often act in dynamic environments. Thus, the Dynamic DCOP formulation is attractive to model such problems. Existing research has focused at solving such problems *reactively*, thus discarding the information on possible future changes, which is often available in many applications. To cope with this limita-

tion, we proposed *Proactive Dynamic DCOPs* (*PD-DCOPs*), which model the dynamism information in Dynamic DCOPs. In addition, we developed an exact algorithm to solve PD-DCOPs and several heuristic algorithms that can scale to larger and more complex problems. Our theoretical results presented the complexity of PD-DCOPs and the error bound of our approaches. We also empirically evaluated both proactive and reactive algorithms to determine the trade-offs between the two classes. When solving PD-DCOPs online, our new distributed online greedy algorithms FORWARD and HYBRID outperformed reactive algorithms in problems with large switching costs and in problems that change quickly. Our empirical findings on the trade-offs between proactive and reactive algorithms are the first, to the best of our knowledge, that shed light on this important issue. In the future, we plan to study how to combine the proactive and reactive approaches to solve D-DCOPs that change quickly over time. If we can leverage the solution provided by the proactive approach in an efficient way, we might be able to quickly find better solutions without much trade-off and react better in such environment. We also plan to compare and evaluate these approaches in both synthetic settings and real-life applications.

## Acknowledgments

We thank the anonymous reviewers, whose suggestions improved the quality of our paper. This research is partially supported by the United States-Israel Binational Science Foundation (BSF) under award 2018081, the United States National Science Foundation (NSF) under awards 1838364 and 2143706, and the Japan Society for the Promotion of Science (JSPS) under Outline of Grants-in-Aid for Scientific Research (KAKENHI) awards JP20H00609 and JP21H04979. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies, or the United States, Israeli, or Japanese governments.

## References

Becker, R., Zilberstein, S., Lesser, V., & Goldman, C. (2004). Solving transition independent decentralized Markov decision processes. *Journal of Artificial Intelligence Research*, *22*, 423–455.

Bellifemine, F., Bergenti, F., Caire, G., & Poggi, A. (2005). JADE–a Java agent development framework. In *Multi-agent programming*, pp. 125–147.

Bernstein, D., Givan, R., Immerman, N., & Zilberstein, S. (2002). The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, *27*(4), 819–840.

Chen, Z., Wu, T., Deng, Y., & Zhang, C. (2018). An ant-based algorithm to solve distributed constraint optimization problems. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 4654–4661.

Cohen, L., Galiki, R., & Zivan, R. (2020). Governing convergence of max-sum on DCOPs through damping and splitting. *Artificial Intelligence*, *279*.

Cohen, L., & Zivan, R. (2018). Balancing asymmetry in max-sum using split constraint factor graphs. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP)*, pp. 669–687.

Deng, Y., & An, B. (2020). Speeding up incomplete GDL-based algorithms for multi-agent optimization with dense local utilities. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 31–38.

Dibangoye, J. S., Amato, C., & Doniec, A. (2012). Scaling up decentralized MDPs through heuristic search. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 217–226.

Dibangoye, J. S., Amato, C., Doniec, A., & Charpillet, F. (2013). Producing efficient error-bounded solutions for transition independent decentralized MDPs. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 539–546.

Fargier, H., Lang, J., & Schiex, T. (1996). Mixed constraint satisfaction: A framework for decision problems under incomplete knowledge. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 175–180.

Farinelli, A., Rogers, A., & Jennings, N. (2014). Agent-based decentralised coordination for sensor networks using the max-sum algorithm. *Journal of Autonomous Agents and Multi-Agent Systems*, *28*(3), 337–380.

Farinelli, A., Rogers, A., Petcu, A., & Jennings, N. (2008). Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 639–646.

Fioretto, F., Pontelli, E., & Yeoh, W. (2018). Distributed constraint optimization problems and applications: A survey. *Journal of Artificial Intelligence Research*, *61*, 623–698.

Fioretto, F., Yeoh, W., & Pontelli, E. (2017a). A multiagent system approach to scheduling devices in smart homes. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 981–989.

Fioretto, F., Yeoh, W., Pontelli, E., Ma, Y., & Ranade, S. (2017b). A DCOP approach to the economic dispatch with demand response. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 999–1007.

Gallager, R. (2013). *Stochastic Processes: Theory for Applications*. Cambridge University Press.

Gershman, A., Meisels, A., & Zivan, R. (2009). Asynchronous forward-bounding for distributed COPs. *Journal of Artificial Intelligence Research*, *34*, 61–88.

Gutierrez, P., Meseguer, P., & Yeoh, W. (2011). Generalizing ADOPT and BnB-ADOPT. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 554–559.

Hamadi, Y., Bessière, C., & Quinqueton, J. (1998). Distributed intelligent backtracking. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pp. 219–223.

Hansen, E. A., Bernstein, D. S., & Zilberstein, S. (2004). Dynamic programming for partially observable stochastic games. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 709–715.

Hatano, D., & Hirayama, K. (2013). DeQED: An efficient divide-and-coordinate algorithm for DCOP. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 566–572.

Hoang, K. D., Fioretto, F., Hou, P., Yokoo, M., Yeoh, W., & Zivan, R. (2016). Proactive dynamic distributed constraint optimization. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 597–605.

Hoang, K. D., Fioretto, F., Yeoh, W., Pontelli, E., & Zivan, R. (2018). A large neighboring search schema for multi-agent optimization. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP)*, pp. 688–706.

Hoang, K. D., Hou, P., Fioretto, F., Yeoh, W., Zivan, R., & Yokoo, M. (2017). Infinite-horizon proactive dynamic DCOPs. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 212–220.

Hoang, K. D., Wayllace, C., Yeoh, W., Beal, J., Dasgupta, S., Mo, Y., Paulos, A., & Schewe, J. (2019). New distributed constraint reasoning algorithms for load balancing in edge computing. In *Proceedings of the Principles and Practice of Multi-Agent Systems (PRIMA)*, pp. 69–86.

Hoang, K. D., Yeoh, W., Yokoo, M., & Rabinovich, Z. (2020). New algorithms for continuous distributed constraint optimization problems. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, p. 502–510.

Holland, A., & O'Sullivan, B. (2005). Weighted super solutions for constraint programs. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 378–383.

Katz, R. W. (1977). An application of chain-dependent processes to meteorology. *Journal of Applied Probability*, *14*(3), 598–603.

Kim, Y., Krainin, M., & Lesser, V. (2011). Effective variants of the max-sum algorithm for radar coordination and scheduling. In *Proceedings of the International Joint Conferences on Web Intelligence and Intelligent Agent Technologies (WI-IAT)*, pp. 357–364.

Kumar, A., Faltings, B., & Petcu, A. (2009). Distributed constraint optimization with structured resource constraints. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 923–930.

Kumar, A., Petcu, A., & Faltings, B. (2008). H-DPOP: Using hard constraints for search space pruning in DCOP. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 325–330.

Lass, R., Sultanik, E., & Regli, W. (2008). Dynamic distributed constraint reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 1466–1469.

Le, T., Son, T. C., Pontelli, E., & Yeoh, W. (2017). Solving distributed constraint optimization problems with logic programming. *Theory and Practice of Logic Programming*, *17*(4), 634–683.

Maheswaran, R., Pearce, J., & Tambe, M. (2004a). Distributed algorithms for DCOP: A graphical game-based approach. In *Proceedings of the Conference on Parallel and Distributed Computing Systems (PDCS)*, pp. 432–439.

Maheswaran, R., Tambe, M., Bowring, E., Pearce, J., & Varakantham, P. (2004b). Taking DCOP to the real world: Efficient complete solutions for distributed event scheduling. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 310–317.

Miller, S., Ramchurn, S., & Rogers, A. (2012). Optimal decentralised dispatch of embedded generation in the smart grid. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 281–288.

Modi, P., Shen, W.-M., Tambe, M., & Yokoo, M. (2005). ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, *161*(1–2), 149–180.

Moore, J. T., Glass, F. H., Graves, C. E., Rochette, S. M., & Singer, M. J. (2003). The environment of warm-season elevated thunderstorms associated with heavy rainfall over the central united states. *Weather and Forecasting*, *18*(5), 861–878.

Nair, R., Tambe, M., Yokoo, M., Pynadath, D., & Marsella, S. (2003). Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 705–711.

Nair, R., Varakantham, P., Tambe, M., & Yokoo, M. (2005). Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 133–139.

Nguyen, D. T., Yeoh, W., Lau, H. C., Zilberstein, S., & Zhang, C. (2014). Decentralized multi-agent reinforcement learning in average-reward dynamic DCOPs. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 1447–1455.

Nguyen, D. T., Yeoh, W., Lau, H. C., & Zivan, R. (2019). Distributed Gibbs: A linear-space sampling-based DCOP algorithm. *Journal of Artificial Intelligence Research*, *64*, 705–748.

Oliehoek, F., Spaan, M., Amato, C., & Whiteson, S. (2013). Incremental clustering and expansion for faster optimal planning in Dec-POMDPs. *Journal of Artificial Intelligence Research*, *46*, 449–509.

Ottens, B., Dimitrakakis, C., & Faltings, B. (2017). DUCT: An upper confidence bound approach to distributed constraint optimization problems. *ACM Transactions on Intelligent Systems and Technology*, *8*(5), 69:1–69:27.

Paulos, A., Dasgupta, S., Beal, J., Mo, Y., Hoang, K. D., Lyles, J. B., Pal, P., Schantz, R., Schewe, J., Sitaraman, R., Wald, A., Wayllace, C., & Yeoh, W. (2019). A framework for self-adaptive dispersal of computing services. In *IEEE Self-Adaptive and Self-Organizing Systems Workshops*.

Petcu, A., & Faltings, B. (2005a). A scalable method for multiagent constraint optimization. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1413–1420.

Petcu, A., & Faltings, B. (2005b). Superstabilizing, fault-containing multiagent combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 449–454.

Petcu, A., & Faltings, B. (2007). Optimal solution stability in dynamic, distributed constraint optimization. In *Proceedings of the International Conference on Intelligent Agent Technology (IAT)*, pp. 321–327.

Richardson, C. W. (1981). Stochastic simulation of daily precipitation, temperature, and solar radiation. *Water Resources Research*, *17*(1), 182–190.

Rust, P., Picard, G., & Ramparany, F. (2016). Using message-passing DCOP algorithms to solve energy-efficient smart environment configuration problems. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 468–474.

Seuken, S., & Zilberstein, S. (2007). Memory-bounded dynamic programming for DEC-POMDPs. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 2009–2015.

Sultanik, E., Lass, R., & Regli, W. (2008). DCOPolis: A framework for simulating and deploying distributed constraint reasoning algorithms. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 1667–1668.

Sultanik, E., Lass, R., & Regli, W. (2009). Dynamic configuration of agent organizations. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 305–311.

Szer, D., Charpillet, F., & Zilberstein, S. (2005). MAA*: A heuristic search algorithm for solving decentralized POMDPs. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 576–590.

Tarim, S. A., Manandhar, S., & Walsh, T. (2006). Stochastic constraint programming: A scenario-based approach. *Constraints*, *11*(1), 53–80.

Trenberth, K. (2011). Changes in precipitation with climate change. *Climate Research*, *47*(1-2), 123–138.

Ueda, S., Iwasaki, A., & Yokoo, M. (2010). Coalition structure generation based on distributed constraint optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 197–203.

van Leeuwen, C. J., & Pawelczak, P. (2017). CoCoA: A non-iterative approach to a local search (A)DCOP solver. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 3944–3950.

Vinyals, M., Rodríguez-Aguilar, J., & Cerquides, J. (2011). Constructing a unifying theory of dynamic programming DCOP algorithms via the generalized distributive law. *Autonomous Agents and Multi-Agent Systems*, *22*(3), 439–464.

Wallace, R., & Freuder, E. (1998). Stable solutions for dynamic constraint satisfaction problems. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP)*, pp. 447–461.

Walsh, T. (2002). Stochastic constraint programming. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pp. 111–115.

Wilks, D. S. (1992). Adapting stochastic weather generation algorithms for climate change studies. *Climatic Change*, *22*(1), 67–84.

Witwicki, S., & Durfee, E. (2011). Towards a unifying characterization for quantifying weak coupling in Dec-POMDPs. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 29–36.

Xu, W., Zipser, E. J., Chen, Y.-L., Liu, C., Liou, Y.-C., Lee, W.-C., & Jong-Dao Jou, B. (2012). An orography-associated extreme rainfall event during TiMREX: Initiation, storm evolution, and maintenance. *Monthly Weather Review*, *140*(8), 2555–2574.

Yeoh, W., Felner, A., & Koenig, S. (2010). BnB-ADOPT: An asynchronous branch-and-bound DCOP algorithm. *Journal of Artificial Intelligence Research*, *38*, 85–133.

Yeoh, W., Varakantham, P., Sun, X., & Koenig, S. (2015). Incremental DCOP search algorithms for solving dynamic DCOPs. In *Proceedings of the International Conference on Intelligent Agent Technology (IAT)*, pp. 257–264.

Yeoh, W., & Yokoo, M. (2012). Distributed problem solving. *AI Magazine*, *33*(3), 53–65.

Yu, Z., Chen, Z., He, J., & Deng, Y. (2017). A partial decision scheme for local search algorithms for distributed constraint optimization problems. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 187–194.

Zhang, W., Wang, G., Xing, Z., & Wittenberg, L. (2005). Distributed stochastic search and distributed breakout: Properties, comparison and applications to constraint optimization problems in sensor networks. *Artificial Intelligence*, *161*(1–2), 55–87.

Zink, M., Westbrook, D., Abdallah, S., Horling, B., Lakamraju, V., Lyons, E., Manfredi, V., Kurose, J., & Hondl, K. (2005). Meteorological command and control: An end-to-end architecture for a hazardous weather detection sensor network. In *Workshop on End-to-End, Sense-and-Respond Systems, Applications, and Services*. USENIX Association.

Zivan, R., Okamoto, S., & Peled, H. (2014). Explorative anytime local search for distributed constraint optimization. *Artificial Intelligence*, *212*, 1–26.

Zivan, R., Parash, T., Cohen, L., Peled, H., & Okamoto, S. (2017). Balancing exploration and exploitation in incomplete min/max-sum inference for distributed constraint optimization. *Journal of Autonomous Agents and Multi-Agent Systems*, *31*(5), 1165–1207.

Zivan, R., & Peled, H. (2012). Max/min-sum distributed constraint optimization through value propagation on an alternating DAG. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 265–272.

Zivan, R., Yedidsion, H., Okamoto, S., Glinton, R., & Sycara, K. (2015). Distributed constraint optimization for teams of mobile sensing agents. *Journal of Autonomous Agents and Multi-Agent Systems*, *29*(3), 495–536.