WASHINGTON UNIVERSITY IN ST. LOUIS

McKelvey School of Engineering
Department of Computer Science and Engineering

Dynamic Continuous Distributed Constraint Optimization Problems

by
Khoi Hoang

A dissertation presented to
the McKelvey School of Engineering
of Washington University in
partial fulfillment of the
requirements for the degree
of Doctor of Philosophy

August 2022
St. Louis, Missouri

Table of Contents

List of Figures

List of Tables

Acknowledgments

First and foremost, I would like to express my deep appreciation to my advisor, William Yeoh. You have been very patient, supportive, and provided me with your guidance through my PhD journey. I always feel how fortunate I am to have such a great advisor, and I am sure all of your PhD students will share the same feeling.

I would like to thank Roie Zivan and Makoto Yokoo for your mentorship and for the collaboration throughout the years. I would like to thank the members of my committee, Roman Garnett, Chien-Ju Ho, Alvitta Ottley, Roie Zivan, and Sanmay Das, for your valuable recommendation and constructive feedback to my proposal and dissertation.

I would like to mention my lab-mates, who have shared the best time of my PhD journey, Ferdinando Fioretto, Athena Tabakhi, Ping Hou, Christabel Wallace, Gan Xu, Stylianos Vasileiou, Ashwin Kumar, and Jean Springsteen. My special thank goes to Ferdinando Fioretto for being a great mentor when I started my PhD journey.

I would like to express my gratitude and appreciation to my cousin Uyen Pham, professor Son Tran, and my advisor William Yeoh. Without your tremendous support, I could not have such a great opportunity to study abroad and live in America, a country of opportunity, to pursue my dream.

I would like to thank my cousins, Khoi Nguyen and Phuong Nguyen. Especially Phuong Nguyen, you have always supported me from my first day in the US until now.

I would like to save the last lines for my Mom and Dad. Despite being halfway around the world, you have always showed your support and your love for me. Without your hard work and your unconditional support, I could not achieve what I have today. I would like to express my deep gratitude to you, and I love you, Mom and Dad.

Khoi Hoang

Washington University in St. Louis

August 2022

Dedicated to my parents.

ABSTRACT OF THE DISSERTATION

Dynamic Continuous Distributed Constraint Optimization Problems

by

Khoi Hoang

Doctor of Philosophy in Computer Science

Washington University in St. Louis, 2022

Professor Roman Garnett, Chair

The Distributed Constraint Optimization Problem (DCOP) formulation is a powerful tool to model multi-agent coordination problems that are distributed by nature. The formulation is suitable for problems where the environment does not change over time and where agents seek their value assignment from a discrete domain. However, in many real-world applications, agents often interact in a more dynamic environment and their variables usually require a more complex domain. Thus, the DCOP formulation lacks the capabilities to model the problems in such dynamic and complex environments. To address these limitations, researchers have proposed Dynamic DCOPs (D-DCOPs) to model how DCOPs dynamically change over time and Continuous DCOPs (C-DCOPs) to model DCOPs with continuous variables. The two models address the limitations of DCOPs but in isolation, and thus, it remains a challenge to model problems that have continuous variables and are in a dynamic environment. Therefore, this dissertation investigates a novel formulation that addresses the two limitations of DCOPs together by modeling both dynamic nature of the environment and continuous nature of the variables. Firstly, we propose *Proactive Dynamic DCOPs (PD-DCOPs)* which model and solve DCOPs in dynamic environment in a *proactive* manner. Secondly, we propose several C-DCOP algorithms that are efficient and we provide quality guarantee on their solution. Finally, we propose *Dynamic Continuous DCOP (DC-DCOP)*, a novel formulation that models the DCOPs with continuous variables in a dynamic environment.

Chapter 1

Introduction

Distributed Constraint Optimization Problems (DCOPs) [19, 55, 63] are problems where agents coordinate their value assignments to maximize the aggregate constraint utilities in a *distributed* manner. The model has been applied to solve a wide range of multi-agent coordination problems including distributed meeting scheduling, sensor and wireless network coordination, multi-robot coordination, coalition structure generation, smart grid and smart home automation [10, 17, 21, 22, 33, 34, 35, 37, 45, 49, 53, 54, 67, 77, 85, 95]. Recent advances in the literature improve the state of the art [5, 6, 8, 9, 20, 35, 42, 43, 48, 60, 62, 88, 92]; solve DCOP extensions like Asymmetric DCOPs [12, 13, 50, 93]; and improve key metrics like privacy [27, 28, 74, 75].

Typically, DCOPs assume that the variables are discrete and the environment does not change over time. However, in many multi-agent problems, agents often interact in a more complex and dynamic environment. For example, in distributed sensor networks, targets usually move from one location to another location from time to time, and thus their location keeps changing dynamically over time. To adapt to such dynamic environment, sensors should be equipped with the ability to change their sensing direction when the target moves to a new location. Thus, researchers have proposed *Dynamic DCOPs (D-DCOPs)* [59, 65, 71, 86, 95] that model how the problem evolves during the solving process. These models make a common assumption that information on how the problem might change is unavailable. As such, existing approaches react to the changes in the problem and solve the current problem at hand. However, in several applications, the information on how the problem might change is indeed available, or predictable, within some degree of uncertainty.

Moreover, target location usually corresponds to a wide range of possibilities (i.e., the set of all possible locations in a two-dimensional plane or in a three-dimensional space of the sensor network). With a number of discrete values of sensing direction, the sensors are limited in their capability to capture the target location with high quality. To address this concern, researchers have proposed *Continuous DCOPs (C-DCOPs)*, which extend DCOPs to allow for continuous variables [69]. As variables can now take values from a continuous range, constraint utilities are also extended from tabular forms to functional forms. To solve such problems, researchers have proposed several *Max-Sum (MS)* based-algorithms [18] including *Continuous MS (CMS)* [69], where constraint utility functions are approximated by piecewise linear functions, and *Hybrid CMS (HCMS)* [79], which combines the discrete MS algorithm with continuous non-linear optimization methods. Specifically, agents in HCMS approximate the utility functions with a number of samples that they iteratively improve over time. A key limitation of CMS and HCMS is that they both do not provide quality guarantees on the solutions found. The reason is that they rely on discrete MS as the underlying algorithmic framework, which does not provide quality guarantees on general graphs.

1.1 Overview of Contributions

Since the DCOP formulation lacks the capability to model and solve the problems that are both dynamic and continuous, this dissertation presents a novel formulation that models both dynamic nature of the environment and continuous nature of the variables. Our contribution are outlined as follows:

1. First, we propose *Proactive Dynamic DCOPs (PD-DCOPs)*, which explicitly model how the DCOP might change over time. In addition, we propose several exact and heuristic algorithms that solve PD-DCOPs in a *proactive* manner.
2. Second, we propose several algorithms with quality guarantee to solve C-DCOPs. Our algorithms are the first heuristic C-DCOP algorithms that come with the error bounds for their solution quality. Additionally, we propose an exact algorithm to solve C-DCOPs under a specific setting.
3. Finally, we propose *Dynamic Continuous DCOPs (DC-DCOPs)*, which is a novel formulation that addresses the dynamic nature of the environment and the continuous nature of the variables.

1.2 Research Output

- **Khoi D. Hoang**, Ferdinando Fioretto, Ping Hou, Makoto Yokoo, William Yeoh, and Roie Zivan. "Proactive Dynamic Distributed Constraint Optimization". In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 597-605, 2016.

- **Khoi D. Hoang**, Ping Hou, Ferdinando Fioretto, William Yeoh, Roie Zivan, and Makoto Yokoo. "Infinite-Horizon Proactive Dynamic DCOPs". In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 212-220, 2017.
- **Khoi D. Hoang**, William Yeoh, Makoto Yokoo, and Zinovi Rabinovich. "New Algorithms for Continuous Distributed Constraint Optimization Problems". In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 502–510, 2020.
- **Khoi D. Hoang**, Ferdinando Fioretto, Ping Hou, William Yeoh, Makoto Yokoo, and Roie Zivan. "Proactive Dynamic Distributed Constraint Optimization Problems". *Journal of Artificial Intelligence Research (JAIR)*, pages 179-225, 2022.
- **Khoi D. Hoang** and William Yeoh. "Dynamic Continuous Distributed Constraint Optimization Problems". Submitted to *International Conference on Principles and Practice of Multi-Agent Systems (PRIMA)*, 2022.

1.3 Dissertation Structure

This dissertation is the result of the collaboration with other researchers. Below is the list of the publications and the contribution of the researchers to the work of each chapter.

- **Chapter 3**: The work of this chapter appears in:
  - **Khoi D. Hoang**, Ferdinando Fioretto, Ping Hou, Makoto Yokoo, William Yeoh, and Roie Zivan. "Proactive Dynamic Distributed Constraint Optimization". In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 597-605, 2016.

  - **Khoi D. Hoang**, Ping Hou, Ferdinando Fioretto, William Yeoh, Roie Zivan, and Makoto Yokoo. "Infinite-Horizon Proactive Dynamic DCOPs". In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 212-220, 2017.
  - **Khoi D. Hoang**, Ferdinando Fioretto, Ping Hou, William Yeoh, Makoto Yokoo, and Roie Zivan. "Proactive Dynamic Distributed Constraint Optimization Problems". *Journal of Artificial Intelligence Research (JAIR)*, pages 179-225, 2022.

  Yeoh, Yokoo, and Zivan proposed the idea of PD-DCOPs. Hoang proposed C-DPOP, which is an exact algorithm to solve PD-DCOPs, and proposed local search approaches to solve PD-DCOPs. Hou proposed greedy approaches to solve PD-DCOPs. Yeoh proposed the idea of comparing algorithms in an online setting. Fioretto, Hou, and Hoang collaboratively worked on the theoretical results. Hoang developed, evaluated the algorithms, and collected experimental results. Yeoh, Fioretto, Hou, and Hoang collaboratively wrote the two conference papers, which were published in AAMAS-16 and AAMAS-17. Yeoh and Hoang collaboratively wrote the article that was published in JAIR-22.

- **Chapter 4**: The work of this chapter appears in:
  - **Khoi D. Hoang**, William Yeoh, Makoto Yokoo, and Zinovi Rabinovich. "New Algorithms for Continuous Distributed Constraint Optimization Problems". In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 502–510, 2020.

  Yeoh, Yokoo, and Rabinovich suggested the idea of working on new algorithms for C-DCOPs. Hoang proposed EC-DPOP, an exact algorithm, and C-DSA, a scalable heuristic algorithm. Yeoh proposed AC-DPOP and CAC-DPOP. Yokoo proposed a method to derive the error bound for AC-DPOP. Hoang developed, evaluated the algorithms, and collected experimental results. Yeoh and Hoang collaboratively wrote the paper.

- **Chapter 5**: The work of this chapter appears in:
  - **Khoi D. Hoang** and William Yeoh. "Dynamic Continuous Distributed Constraint Optimization Problems". Submitted to *International Conference on Principles and Practice of Multi-Agent Systems (PRIMA)*, 2022.

Hoang proposed the idea of the DC-DCOP model, proposed algorithms to solve DC-DCOPs, developed the algorithms, ran experiments, and collected experimental results. Hoang and Yeoh collaboratively wrote the paper.

Chapter 2

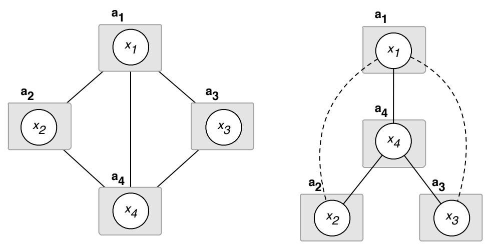
Background and Motivation

In this section, we provide an overview of DCOPs, Dynamic DCOPs, Continuous DCOPs, relevant DCOP algorithms, and Markov chains. We will introduce Distributed Radar Coordination and Scheduling Problem (DRCSP), which is the motivating application for out work.

2.1 Distributed Constraint Optimization Problems

A *Distributed Constraint Optimization Problem (DCOP)* [19, 55, 63, 87] 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 $D_x \in \mathbf{D}$.



(a) Constraint Graph (b) Pseudo-tree

$j = 2,3$

x_1	x_j	util
0	0	10
0	1	0
1	0	2
1	1	0

$j = 1,2,3$

x_j	x_4	util
0	0	0
0	1	6
1	0	0
1	1	10

(c) Utility Functions

Figure 2.1: Example of DCOP

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

A *solution* σ 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 σ. A solution σ 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 $\{x, y\} = \mathbf{x}^{f_i}$.[2] 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 2.1 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]The scope of a function is the set of variables that are associated with the function.
[2]We assume that the utility functions are binary between two decision variables.

2.2 DCOP Algorithms

We now introduce three relevant DCOP algorithms that are the main component of several algorithms for PD-DCOPs, C-DCOPs, and DC-DCOPs.

2.2.1 Distributed Pseudo-tree Optimization Procedure

The *Distributed Pseudo-tree Optimization Procedure (DPOP)* [63] is a complete *inference algorithm* composed of three phases:

- *Pseudo-tree Generation:* The agents build a pseudo-tree [30].
- *UTIL Propagation:* Each agent, starting from the leafs of the pseudo-tree, computes the optimal sum of utilities in its subtree for each value combination of variables in its separator.[3] 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.

2.2.2 Super-stabilizing DPOP

Super-stabilizing DPOP (S-DPOP) [65] 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.

[3]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.

2.2.3 Maximum Gain Message

Maximum Gain Message (MGM) [52] 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.

2.3 Dynamic DCOPs

A *Dynamic DCOP (D-DCOP)* [47, 64, 65, 86] 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 [65, 86]. 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 [64], a model where agents can have imperfect knowledge about their environment [47], and a model where changes in the constraint graph depends on the value assignments of agents [95].

2.4 Continuous DCOPs

The *Continuous DCOP (C-DCOP)* model [69] generalizes the regular discrete DCOP model by modeling the variables as continuous variables. It is defined by a tuple $\langle \mathbf{A}, \mathbf{X}, \mathbf{D}, \mathbf{F}, \alpha \rangle$, where $\mathbf{A}$, $\mathbf{F}$, and α are exactly as defined in DCOPs. The key differences are:

- $\mathbf{X} = \{x_i\}_{i=1}^{n}$ is now a set of *continuous* variables.
- $\mathbf{D} = \{D_x\}_{x \in \mathbf{X}}$ is now a set of *continuous domains*. Each variable $x \in \mathbf{X}$ takes values from the interval $D_x = [LB_x, UB_x]$.

The objective of a C-DCOP is the same as that of DCOPs – to find an optimal complete solution $\mathbf{x}^* = \operatorname{argmax}_{\mathbf{x}} \mathbf{F}(\mathbf{x})$.

2.5 Markov Chains

We now introduce *Markov chains* and the notion of *stationary distribution*, which are used in one of the approaches to solve PD-DCOPs. A Markov chain [25] 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,

$$\Pr(x^t = j \mid x^{t-1} = i, x^{t-2} = r, \ldots, x^0 = s) = \Pr(x^t = j \mid x^{t-1} = i) \tag{2.1}$$

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 *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, $d(i) = \gcd\{t : P^t_{ii} > 0\}$, is the greatest common divisor (gcd) of the time steps t for which $P^t_{ii} > 0$. 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 dissertation, 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.

2.6 Reactive and Proactive Approaches

Since one of our contributions is solving D-DCOPs in a *proactive* manner and later we will compare our proactive approach with the *reactive* approach in Chapter 3, we now introduce the difference between the two approaches in the context of optimization problems. *Dynamic Optimization Problems (DOPs)* [24] are the problems that change dynamically over time, and the goal of DOPs is to find the optimal set of decisions to maximize the overall solution quality. Several approaches have been proposed to solve DOPs including *reactive* and *proactive* approaches [1]. When the environment changes and the problem transitions to a new state, a reactive approach performs certain optimization algorithm to search for a new solution without taking into account future changes from the environment.[4] In contrast, a proactive approach anticipates those changes from the environment through probability distribution and solves the problem in a proactive manner. Proactive approaches are mainly used to improve the robustness of the solution or when the time for reoptimization for each problem is short [40].

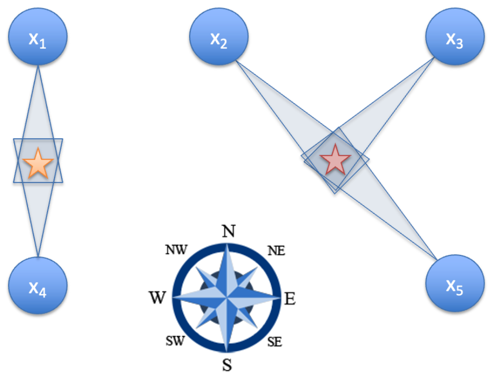


Figure 2.2: Distributed Radar Coordination and Scheduling Problem

2.7 Distributed Radar Coordination and Scheduling Problems

In this section, we motivate our work using the *Distributed Radar Coordination and Scheduling Problems (DRCSPs)*, which are based on NetRad, a real-time weather radar sensor system [11, 44, 90]. 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 2.2, 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.

[4]This approach is referred as No-baseline in [1].

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 [41, 66, 82], and it is known to be associated with many phenomena at locations of interest [56, 76, 84]. 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.

Chapter 3

Proactive Dynamic Distributed Constraint Optimization Problems

In many applications, the information on how the problem might change over time is usually available or predictable within some degree of uncertainty. In this chapter, we introduce a novel formulation that explicitly models how DCOPs change dynamically over time with uncertain events. We develop both exact and heuristic algorithms to solve the dynamic problems in a *proactive* manner. Finally, we provide theoretical results on the complexity of this new class of DCOPs, and empirically evaluate both proactive and reactive algorithms to determine the trade-offs between the two classes.

3.1 Introduction

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 [26, 29, 55, 85], exact inference-based methods [63, 78], exact declarative methods [32, 48], approximate search-based methods [6, 35, 50, 52, 88, 89, 91], approximate inference-based methods [8, 9, 17, 38, 92, 94], and approximate sampling-based methods [60, 62].

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 DCOPs to solve and model the problems in dynamic environments. Thus, researchers have proposed the *Dynamic DCOP (D-DCOP)* model [47, 64, 65, 86], 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 [65, 71, 86]. However, in several applications, the information on how the problem might change is indeed available or predictable within some degree of uncertainty. For example, in DRCSPs, the trajectory of the weather phenomena could be predicted to some degree and the data of the weather phenomena in the past are usually available. Therefore, in this chapter, 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.

Therefore, we *(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.

3.2 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. A PD-DCOP is a tuple $\langle \mathbf{A}, \mathbf{X}, \mathbf{Y}, \mathbf{D}, \boldsymbol{\Omega}, \mathbf{F}, p^0_{\mathbf{Y}}, \mathbf{T}, \gamma, h, c, \alpha \rangle$, 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 $D_x \in \mathbf{D}$.
- $\boldsymbol{\Omega} = \{\Omega_y\}_{y \in \mathbf{Y}}$ is a set of finite *domains of the random variables*, where each variable $y \in \mathbf{Y}$ takes values from the set $\Omega_y \in \boldsymbol{\Omega}$.
- $\mathbf{F} = \{f_i\}_{i=1}^k$ is a set of *utility functions*, each defined over a mixed set of decision and random variables: $f_i : \prod_{x \in \mathbf{X} \cap \mathbf{x}^{f_i}} D_x \times \prod_{y \in \mathbf{Y} \cap \mathbf{x}^{f_i}} \Omega_y \to \mathbb{R}^+_0 \cup \{-\infty\}$, 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*: $T_y : \Omega_y \times \Omega_y \to [0, 1]$.
- $\gamma \in [0, 1]$ 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]
- $\alpha : \mathbf{X} \to \mathbf{A}$ is a function that associates each decision variable to one agent.

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

Throughout this chapter, 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.

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

$$\mathbf{x}^* = \operatorname*{argmax}_{\mathbf{x}=\langle \mathbf{x}^0,\ldots,\mathbf{x}^h\rangle \in \Sigma^{h+1}} \mathcal{F}^h(\mathbf{x}) \tag{3.1}$$

$$\mathcal{F}^h(\mathbf{x}) = \underbrace{\sum_{t=0}^{h-1} \gamma^t \left[\mathcal{F}_x^t(\mathbf{x}^t) + \mathcal{F}_y^t(\mathbf{x}^t)\right]}_{\mathbf{P}} - \underbrace{\sum_{t=0}^{h-1} \gamma^t \left[c \cdot \Delta(\mathbf{x}^t, \mathbf{x}^{t+1})\right]}_{\mathbf{Q}} + \underbrace{\tilde{\mathcal{F}}_x(\mathbf{x}^h) + \tilde{\mathcal{F}}_y(\mathbf{x}^h)}_{\mathbf{R}} \tag{3.2}$$

, where Σ 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:

$$\mathcal{F}_x^t(\mathbf{x}) = \sum_{f_i \in \mathbf{F_X}} f_i(\mathbf{x}_i) \tag{3.3}$$

$$\mathcal{F}_y^t(\mathbf{x}) = \sum_{f_i \in \mathbf{F_Y}} \sum_{\omega \in \Omega_{y_i}} f_i(\mathbf{x}_i|_{y_i=\omega}) \cdot p_{y_i}^t(\omega) \tag{3.4}$$

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 ω at time t, and is defined as:

$$p_{y_i}^t(\omega) = \sum_{\omega' \in \Omega_{y_i}} p_{y_i}^{t-1}(\omega') \cdot T_{y_i}(\omega', \omega) \tag{3.5}$$

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 Δ 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 γ, 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 (3.6), (3.7), and (3.8), which will be introduced in Section 3.3.
- ***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 [25]. 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 (3.9), (3.10), (3.11), and (3.12), which will be introduced in Section 3.3.

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 3.2). The first term,

[6]The conditions for such convergence are discussed in Subsection 2.5.

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, **Q**, minimizes the cumulative discounted penalty costs incurred by solutions changing over time. The last term, **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.

3.3 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 3.1. 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

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

Procedure SolvePD-DCOP()

```
1 if $\gamma < 1$ then
2     SolveMultiDCOPs($\bar{h} = h, \mathbf{x}^{\bar{h}+1} = \text{NULL}$)
3 else
4     $\mathbf{s}^h \leftarrow$ SolveHorizonDCOP()
5     SolveMultiDCOPs($\bar{h} = h - 1, \mathbf{x}^{\bar{h}+1} = \mathbf{s}^h$)
```

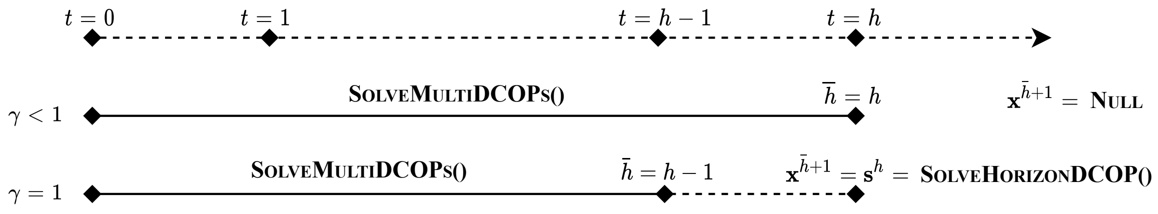


Figure 3.1: Illustration of the SolvePD-DCOP Procedure

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 (3.2) by computing the cumulative discounted and cumulative discounted expected utilities from horizon h onward:

$$\tilde{\mathcal{F}}_x(\mathbf{x}) = \frac{\gamma^h}{1-\gamma}\mathcal{F}_x^h(\mathbf{x}) \tag{3.6}$$

$$\tilde{\mathcal{F}}_y(\mathbf{x}) = \sum_{f_i \in \mathbf{F_Y}} \sum_{\omega \in \Omega_{y_i}} \tilde{f}_i(\mathbf{x}_i|_{y_i=\omega}) \cdot p^h_{y_i}(\omega) \tag{3.7}$$

$$\tilde{f}_i(\mathbf{x}_i|_{y_i=\omega}) = \gamma^h \cdot f_i(\mathbf{x}_i|_{y_i=\omega}) + \gamma \sum_{\omega' \in \Omega_{y_i}} T_{y_i}(\omega, \omega') \cdot \tilde{f}_i(\mathbf{x}_i|_{y_i=\omega'}) \tag{3.8}$$

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 $\mathbf{x}^{\bar{h}+1} = \text{NULL}$

(lines 1-2). We set $\mathbf{x}^{\bar{h}+1} = \textsc{null}$ 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 (3.2) by using the stationary distribution of Markov chains in the PD-DCOP:[8]

$$\tilde{\mathcal{F}}_x(\mathbf{x}) = \mathcal{F}_x^h(\mathbf{x}) \tag{3.9}$$

$$\tilde{\mathcal{F}}_y(\mathbf{x}) = \sum_{f_i \in \mathbf{F_Y}} \sum_{\omega \in \Omega_{y_i}} f_i(\mathbf{x}_i|_{y_i=\omega}) \cdot p^*_{y_i}(\omega) \tag{3.10}$$

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

$$\sum_{\omega' \in \Omega_{y_i}} p^*_{y_i}(\omega') \cdot T_{y_i}(\omega', \omega) = p^*_{y_i}(\omega) \tag{3.11}$$

$$\sum_{\omega \in \Omega_{y_i}} p^*_{y_i}(\omega) = 1 \tag{3.12}$$

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

[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).

$T_y \in \mathbf{T}$ 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 dissertation, 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 3.11 and 3.12), 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 $\mathbf{x}$ and a random variable y, the following new constraint is created:

$$F^h(\mathbf{x}) = \sum_{\omega \in \Omega_y} f(\mathbf{x}|_{y=\omega}) \cdot p^*_y(\omega) \tag{3.13}$$

where $p^*_y(\omega)$ is the probability of random variable y having state ω 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 3.3.1 and 3.3.2, respectively. Later, in Section 3.6, we will introduce different PD-DCOP algorithms that are based on these approaches.

3.3.1 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 3.2, 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:

$$F_i(\mathbf{x}_i) = \sum_{t=0}^{\bar{h}} F_i^t(\mathbf{x}_i^t) \tag{3.14}$$

where:

$$F_i^t(\mathbf{x}_i^t) = \begin{cases} \dfrac{\gamma^h}{1-\gamma} f_i(\mathbf{x}_i^h) & \text{if } t = \bar{h} \text{ and } \bar{h} = h \\ \gamma^t f_i(\mathbf{x}_i^t) & \text{otherwise} \end{cases} \tag{3.15}$$

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 $\bar{h}$:

$$F_i(\mathbf{x}_i) = \sum_{t=0}^{\bar{h}} F_i^t(\mathbf{x}_i^t) \tag{3.16}$$

where:

$$F_i^t(\mathbf{x}_i^t) = \begin{cases} \gamma^h \sum_{\omega \in \Omega_{y_i}} \tilde{f}_i(\mathbf{x}_i^h|_{y_i=\omega}) \cdot p_{y_i}^h(\omega) & \text{if } t = \bar{h} \text{ and } \bar{h} = h \\ \gamma^t \sum_{\omega \in \Omega_{y_i}} f_i(\mathbf{x}_i^t|_{y_i=\omega}) \cdot p_{y_i}^t(\omega) & \text{otherwise} \end{cases} \tag{3.17}$$

[9]With slight abuse of notation, we use the same notation F_i in Equations (3.14) and (3.16) to refer to two different functions in two cases.

The function $\tilde{f}_i$ is recursively defined according to Equation (3.8). Additionally, each decision variable x_i will have a unary function C_i:

$$C_i(\mathbf{x}_i) = -\sum_{t=0}^{h-1} \gamma^t \left[c \cdot \Delta(\mathbf{x}_i^t, \mathbf{x}_i^{t+1}) \right] \tag{3.18}$$

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.

3.3.2 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 3.6 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 (3.15). 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 (3.17). 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*.

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  $context \leftarrow \langle (x_j, t, \text{NULL} \mid x_j \in N(a_i), 0 \leq t \leq \bar{h}) \rangle$
10 Send VALUE($\langle v_i^0, v_i^1, \ldots, v_i^{\bar{h}} \rangle$) to all neighbors
```

Local Search Approach

In this section, we propose a local search approach that is inspired by MGM [52], 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 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     $u \leftarrow$ CalcCumulativeUtil($\langle d_i^0, d_i^1, \ldots, d_i^{\bar{h}} \rangle, \mathbf{x}_i^{\bar{h}+1}$)
15     if $u > u^*$ then
16         $u^* \leftarrow u$
17         $\langle v_i^{0*}, v_i^{1*}, \ldots, v_i^{\bar{h}*} \rangle \leftarrow \langle d_i^0, d_i^1, \ldots, d_i^{\bar{h}} \rangle$
18 if $u^* \neq -\infty$ then
19     $\langle u_i^{0*}, u_i^{1*}, \ldots, u_i^{\bar{h}*} \rangle \leftarrow$ CalcUtils($\langle v_i^{0*}, v_i^{1*}, \ldots, v_i^{\bar{h}*} \rangle, \mathbf{x}_i^{\bar{h}+1}$)
20     $\langle \hat{u}_i^0, \hat{u}_i^1, \ldots, \hat{u}_i^{\bar{h}} \rangle \leftarrow \langle u_i^{0*}, u_i^{1*}, \ldots, u_i^{\bar{h}*} \rangle - \langle u_i^0, u_i^1, \ldots, u_i^{\bar{h}} \rangle$
21 else
22     $\langle \hat{u}_i^0, \hat{u}_i^1, \ldots, \hat{u}_i^{\bar{h}} \rangle \leftarrow \langle$NULL, NULL, $\ldots$, NULL$\rangle$
23 Send GAIN($\langle \hat{u}_i^0, \hat{u}_i^1, \ldots, \hat{u}_i^{\bar{h}} \rangle$) to all neighbors
```

```
Procedure When Receive VALUE($\langle v_s^{0*}, v_s^{1*}, \ldots, v_s^{\bar{h}*} \rangle$)
24 foreach $t$ from 0 to $\bar{h}$ do
25     if $v_s^{t*} \neq$ NULL then
26         Update $(x_s, t, v_s^t) \in$ context with $(x_s, t, v_s^{t*})$
27 if received VALUE messages from all neighbors in this iteration then
28     CalcGain()
29 iter $\leftarrow$ iter + 1
```

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:

```
Procedure When Receive GAIN($\langle \hat{u}_s^0, \hat{u}_s^1, \ldots, \hat{u}_s^{\bar{h}} \rangle$)
30 if $\langle \hat{u}_s^0, \hat{u}_s^1, \ldots, \hat{u}_s^{\bar{h}} \rangle \neq \langle \textit{NULL}, \textit{NULL}, \ldots, \textit{NULL} \rangle$ then
31     foreach $t$ from 0 to $\bar{h}$ do
32         if $\hat{u}_i^t \leq 0 \vee \hat{u}_s^t > \hat{u}_i^t$ then
33             $v_i^{t*} \leftarrow$ NULL
34 if received GAIN messages from all neighbors in this iteration then
35     foreach $t$ from 0 to $\bar{h}$ do
36         if $v_i^{t*} \neq \textit{NULL}$ then
37             $v_i^t \leftarrow v_i^{t*}$
38     Send VALUE($\langle v_i^{1*}, v_i^{2*}, \ldots, v_i^{\bar{h}*} \rangle$) to all neighbors
```

```
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 $\langle u_i^0, u_i^1, \ldots, u_i^{\bar{h}} \rangle$
```

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

```
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 $u - c_i$
```

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).

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 (3.15). 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 (3.17). 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 [30], assigns a score to each agent a according to heuristic $h_1(a)$:

$$h_1(a) = (1 + I(a)) \cdot |N_y(a)| \tag{3.19}$$

$$N_y(a) = \{a' | a' \in N(a) \wedge \exists f \in \mathbf{F}, \exists\, y \in \mathbf{Y} : \{a', y\} \in \mathbf{x}^f\} \tag{3.20}$$

$$I(a) = \begin{cases} 0 & \text{if } \forall f \in \mathbf{F}, \forall y \in \mathbf{Y} : \{a, y\} \notin \mathbf{x}^f \\ 1 & \text{otherwise} \end{cases} \tag{3.21}$$

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:

$$h_3(a) = w\, h_1(a) + (1 - w)\, h_2(a) \tag{3.22}$$

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 $\mathbf{x}^{\bar{h}+1} \neq \textsc{null}$, 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 $0 < t < \bar{h}$:

$$C^t(x) = -c \cdot \Delta(x^{t-1}, x^t) \tag{3.23}$$

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

$$C^{\bar{h}}(x) = \begin{cases} -c \cdot \Delta(x^{\bar{h}-1}, x^{\bar{h}}) & \text{if } \mathbf{x}^{\bar{h}+1} = \textsc{null} \\ -c \cdot \Big(\Delta(x^{\bar{h}-1}, x^{\bar{h}}) + \Delta(x^{\bar{h}}, \mathbf{x}_x^{\bar{h}+1})\Big) & \text{otherwise} \end{cases} \tag{3.24}$$

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., $\mathbf{x}^{\bar{h}+1} \neq$ 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 $0 \leq t < \bar{h}$:

$$C^t(x) = -c \cdot \Delta(x^t, x^{t+1}) \tag{3.25}$$

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

$$C^{\bar{h}}(x) = -c \cdot \Delta(x^{\bar{h}}, \mathbf{x}_x^{\bar{h}+1}) \tag{3.26}$$

3.4 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:

$$f_c(\mathbf{x}^c) = \begin{cases} 1, & \text{if } c(\mathbf{x}^c) = \text{ true} \\ \perp, & \text{otherwise.} \end{cases} \tag{3.27}$$

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 γ, 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 $\mathcal{F}(\mathbf{x}) \neq \perp$.

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. $\square$

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^t_y when $t \to \infty$:

$$\lim_{t\to\infty} p^t_y = p^*_y \tag{3.28}$$

$$p^*_y \cdot T = p^*_y \tag{3.29}$$

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^*:

$$\mathcal{F}^*(\mathbf{x}^*) > \mathcal{F}^*(\mathbf{x}') \qquad \forall t \geq h^* \tag{3.30}$$

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

$$\Delta^{\infty}_{h^*} = \sum_{t=h^*}^{\infty} [\mathcal{F}^*(\mathbf{x}^*) - \mathcal{F}^*(\mathbf{x}')] \tag{3.31}$$

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^{∞} 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 ∞:

$$U^{\infty} = \sum_{t=0}^{\infty} \gamma^t \left[\mathcal{F}_x^t(\hat{\mathbf{x}}_t^*) + \mathcal{F}_y^t(\hat{\mathbf{x}}_t^*) - c \cdot \Delta(\hat{\mathbf{x}}_t^*, \hat{\mathbf{x}}_{t+1}^*) \right] \tag{3.32}$$

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

$$U_+^{\infty} = \sum_{t=0}^{h-1} \gamma^t \left[\mathcal{F}_x^t(\hat{\mathbf{x}}_t^*) + \mathcal{F}_y^t(\hat{\mathbf{x}}_t^*) - c \cdot \Delta(\hat{\mathbf{x}}_t^*, \hat{\mathbf{x}}_{t+1}^*) \right] + \sum_{t=h}^{\infty} \gamma^t \left[\mathcal{F}_x^t(\hat{\mathbf{x}}_t^*) + \mathcal{F}_y^t(\hat{\mathbf{x}}_t^*) \right] \tag{3.33}$$

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:

$$U^h = \sum_{t=0}^{h-1} \gamma^t \left[\mathcal{F}_x^t(\mathbf{x}_t^*) + \mathcal{F}_y^t(\mathbf{x}_t^*) - c \cdot \Delta(\mathbf{x}_t^*, \mathbf{x}_{t+1}^*) \right] + \sum_{t=h}^{\infty} \gamma^t \left[\mathcal{F}_x^t(\mathbf{x}_h^*) + \mathcal{F}_y^t(\mathbf{x}_h^*) \right] \tag{3.34}$$

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^∞_- of U^h:

$$U^\infty_- = \sum_{t=0}^{h-1} \gamma^t \left[\mathcal{F}^t_x(\hat{\mathbf{x}}^*_t) + \mathcal{F}^t_y(\hat{\mathbf{x}}^*_t) - c \cdot \Delta(\hat{\mathbf{x}}^*_t, \hat{\mathbf{x}}^*_{t+1}) \right] + \sum_{t=h}^{\infty} \gamma^t \left[\mathcal{F}^t_x(\hat{\mathbf{x}}^*_h) + \mathcal{F}^t_y(\hat{\mathbf{x}}^*_h) \right] \tag{3.35}$$

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

Next, we compute the difference between the two bounds:

$$\begin{align} U^\infty - U^h &\leq U^\infty_+ - U^\infty_- \tag{3.36} \\ &= \sum_{t=h}^{\infty} \gamma^t \left[(\mathcal{F}^t_x(\hat{\mathbf{x}}^*_t) + \mathcal{F}^t_y(\hat{\mathbf{x}}^*_t)) - (\mathcal{F}^t_x(\hat{\mathbf{x}}^*_h) + \mathcal{F}^t_y(\hat{\mathbf{x}}^*_h)) \right] \tag{3.37} \end{align}$$

Notice that the quantity $(\mathcal{F}^t_x(\hat{\mathbf{x}}^*_t)+\mathcal{F}^t_y(\hat{\mathbf{x}}^*_t))-(\mathcal{F}^t_x(\hat{\mathbf{x}}^*_h)+\mathcal{F}^t_y(\hat{\mathbf{x}}^*_h))$ 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:

$$(\mathcal{F}^t_x(\hat{\mathbf{x}}^*_t) + \mathcal{F}^t_y(\hat{\mathbf{x}}^*_t)) - (\mathcal{F}^t_x(\hat{\mathbf{x}}^*_h) + \mathcal{F}^t_y(\hat{\mathbf{x}}^*_h)) \leq F^\Delta \tag{3.38}$$

Thus,

$$\begin{align} U^\infty - U^h &\leq U^\infty_+ - U^\infty_- \tag{3.39} \\ &\leq \sum_{t=h}^{\infty} \gamma^t \left[\mathcal{F}^t_x(\hat{\mathbf{x}}^*_t) + \mathcal{F}^t_y(\hat{\mathbf{x}}^*_t) - \mathcal{F}^t_x(\hat{\mathbf{x}}^*_h) - \mathcal{F}^t_y(\hat{\mathbf{x}}^*_h) \right] \tag{3.40} \\ &\leq \sum_{t=h}^{\infty} \gamma^t F^\Delta \tag{3.41} \\ &\leq \frac{\gamma^h}{1-\gamma} F^\Delta \tag{3.42} \end{align}$$

which concludes the proof. □

COROLLARY 1 *Given a maximum acceptable error ϵ, 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 $\frac{\gamma^h}{1-\gamma}F^\Delta$:

$$\epsilon \leq \frac{\gamma^h}{1-\gamma}F^\Delta \tag{3.43}$$

$$\frac{(1-\gamma)\cdot\epsilon}{F^\Delta} \leq \gamma^h \tag{3.44}$$

$$\log_\gamma \frac{(1-\gamma)\cdot\epsilon}{F^\Delta} \leq h \tag{3.45}$$

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 ω and ω' 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 $c\cdot|\mathbf{X}| + \sum_{\mathbf{y}\in\Sigma_\mathbf{Y}} \frac{(1-2\beta)^h}{2\beta}\mathcal{F}^\Delta_\mathbf{y}$.*

PROOF: First, given a random variable y, the following inequality holds if the Markov chain converges to the stationary distribution p^*_y [25]. For a given $\omega \in \Omega_y$:

$$|p^*_y(\omega) - T^t_y(\omega',\omega)| \leq (1-2\theta_y)^t \qquad \forall\omega' \in \Omega_y \tag{3.46}$$

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

$$p_y^0 \cdot T_y^t = p_y^t \tag{3.47}$$

$$p_y^0 \cdot T_y^* = p_y^* \tag{3.48}$$

For $\omega \in \Omega_y$:

$$\begin{aligned}
p_y^*(\omega) &= \sum_{\omega' \in \Omega_y} p_y^0(\omega') \cdot T_y^*(\omega', \omega) && (3.49)\\
p_y^t(\omega) &= \sum_{\omega' \in \Omega_y} p_y^0(\omega') \cdot T_y^t(\omega', \omega) && (3.50)\\
|p_y^*(\omega) - p_y^t(\omega)| &= |\sum_{\omega' \in \Omega_y} p_y^0(\omega') \cdot (T_y^*(\omega', \omega) - T_y^t(\omega', \omega))| && (3.51)\\
&= |\sum_{\omega' \in \Omega_y} p_y^0(\omega') \cdot (p_y^*(\omega) - T_y^t(\omega', \omega))| && (3.52)\\
&\leq \sum_{\omega' \in \Omega_y} p_y^0(\omega') \cdot |(p_y^*(\omega) - T_y^t(\omega', \omega))| && (3.53)\\
&\leq \sum_{\omega' \in \Omega_y} p_y^0(\omega') \cdot (1 - 2\theta_y)^t && (3.54)\\
&\leq (1 - 2\theta_y)^t && (3.55)
\end{aligned}$$

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

$$\delta_{\mathbf{Y}}^t(\mathbf{y}) = |p_{\mathbf{Y}}^*(\mathbf{y}) - p_{\mathbf{Y}}^t(\mathbf{y})| \leq (1 - 2\beta)^t \tag{3.56}$$

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

$$\mathcal{F}^t_\Delta \leq \sum_{\mathbf{y}\in\Sigma_\mathbf{Y}} \delta^t_\mathbf{Y}(\mathbf{y}) \cdot \left(\max_{\mathbf{x}\in\Sigma} F_\mathbf{y}(\mathbf{x}) - F_\mathbf{y}(\mathbf{x}^*)\right) \tag{3.57}$$

$$\leq \sum_{\mathbf{y}\in\Sigma_\mathbf{Y}} (1-2\beta)^t \cdot F^\Delta_\mathbf{y} \tag{3.58}$$

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:

$$U_+ = \sum_{t=0}^{h} \mathcal{F}^t(\hat{\mathbf{x}}_t) - \sum_{t=0}^{h-1} [c \cdot \Delta(\hat{\mathbf{x}}_t, \hat{\mathbf{x}}_{t+1})] \tag{3.59}$$

$$U = \sum_{t=0}^{h} \mathcal{F}^t(\bar{\mathbf{x}}_t) - \sum_{t=0}^{h-1} [c \cdot \Delta(\bar{\mathbf{x}}_t, \bar{\mathbf{x}}_{t+1})] \tag{3.60}$$

$$U_- = \sum_{t=0}^{h} \mathcal{F}^t(\check{\mathbf{x}}_t) - \sum_{t=0}^{h-1} [c \cdot \Delta(\check{\mathbf{x}}_t, \check{\mathbf{x}}_{t+1})] \tag{3.61}$$

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:

$$U_+ - U \leq U_+ - U_- \tag{3.62}$$

$$\leq \left[\mathcal{F}^h(\hat{\mathbf{x}}_h) - \mathcal{F}^h(\mathbf{x}^*)\right] - [c \cdot \Delta(\widehat{\mathbf{x}}_{h-1}, \widehat{\mathbf{x}}_h) - c \cdot \Delta(\mathbf{x}_{h-1}, \mathbf{x}^*)] \tag{3.63}$$

$$\leq \mathcal{F}^h_\Delta + c \cdot |\mathbf{X}| \tag{3.64}$$

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

$$\begin{aligned}
\mathcal{F}_{\Delta}^{h} + c \cdot |\mathbf{X}| + \sum_{t=h+1}^{\infty} \mathcal{F}_{\Delta}^{t} &= c \cdot |\mathbf{X}| + \sum_{t=h}^{\infty} \mathcal{F}_{\Delta}^{t} && (3.65)\\
&= c \cdot |\mathbf{X}| + \sum_{t=h}^{\infty} \left(\sum_{\mathbf{y} \in \Sigma_{\mathbf{Y}}} \delta_{\mathbf{Y}}^{t}(\mathbf{y}) \cdot F_{\mathbf{y}}^{\Delta} \right) && (3.66)\\
&\leq c \cdot |\mathbf{X}| + \sum_{\mathbf{y} \in \Sigma_{\mathbf{Y}}} \sum_{t=h}^{\infty} (1-2\beta)^{t} \cdot F_{\mathbf{y}}^{\Delta} && (3.67)\\
&\leq c \cdot |\mathbf{X}| + \sum_{\mathbf{y} \in \Sigma_{\mathbf{Y}}} \frac{(1-2\beta)^{h}}{2\beta} F_{\mathbf{y}}^{\Delta} && (3.68)
\end{aligned}$$

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:

$$\hat{\mathbf{x}}_{t}^{*} = \begin{cases} \operatorname*{argmax}_{\mathbf{x} \in \Sigma} \gamma^{t} \left[\mathcal{F}_{x}^{t}(\mathbf{x}) + \mathcal{F}_{y}^{t}(\mathbf{x}) \right] & \text{if } 0 \leq t < h \\ \operatorname*{argmax}_{\mathbf{x} \in \Sigma} \left[\tilde{\mathcal{F}}_{x}(\mathbf{x}) + \tilde{\mathcal{F}}_{y}(\mathbf{x}) \right] & \text{otherwise} \end{cases} \tag{3.69}$$

and

$$\hat{\mathcal{F}}^{h}(\mathbf{x}) = \sum_{t=0}^{h-1} \gamma^{t} \left[\mathcal{F}_{x}^{t}(\mathbf{x}) + \mathcal{F}_{y}^{t}(\mathbf{x}) \right] + \tilde{\mathcal{F}}_{x}(\mathbf{x}) + \tilde{\mathcal{F}}_{y}(\mathbf{x}). \tag{3.70}$$

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^{th} component of the $\mathcal{F}^h(\cdot)$, defined as:

$$\mathcal{F}^h_t(\mathbf{x}_t) = \begin{cases} \gamma^t \left[\mathcal{F}^t_x(\mathbf{x}_t) + \mathcal{F}^t_y(\mathbf{x}_t) - [c \cdot \Delta(\mathbf{x}_t, \mathbf{x}_{t+1})\right] & \text{if } 0 \leq t < h \\ \tilde{\mathcal{F}}_x(\mathbf{x}_t) + \tilde{\mathcal{F}}_y(\mathbf{x}_t) & \text{otherwise} \end{cases} \tag{3.71}$$

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

$$\hat{\mathcal{F}}^h_t(\mathbf{x}_t) = \begin{cases} \gamma^t \left[\mathcal{F}^t_x(\mathbf{x}_t) + \mathcal{F}^t_y(\mathbf{x}_t)\right] & \text{if } 0 \leq t < h \\ \tilde{\mathcal{F}}_x(\mathbf{x}_t) + \tilde{\mathcal{F}}_y(\mathbf{x}_t) & \text{otherwise} \end{cases} \tag{3.72}$$

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

$$\begin{aligned} \mathcal{F}^h_t(\mathbf{x}^*_t) &= \gamma^t \left[\mathcal{F}^t_x(\mathbf{x}^*_t) + \mathcal{F}^t_y(\mathbf{x}^*_t) - [c \cdot \Delta(\mathbf{x}_t, \mathbf{x}_{t+1})\right] & (3.73) \\ &\leq \gamma^t \left[\mathcal{F}^t_x(\mathbf{x}^*_t) + \mathcal{F}^t_y(\mathbf{x}^*_t)\right] & (3.74) \\ &\leq \max_{\mathbf{x} \in \Sigma} \gamma^t \left[\mathcal{F}^t_x(\mathbf{x}) + \mathcal{F}^t_y(\mathbf{x})\right] & (3.75) \\ &\leq \gamma^t \left[\mathcal{F}^t_x(\hat{\mathbf{x}}^*_t) + \mathcal{F}^t_y(\hat{\mathbf{x}}^*_t)\right] = \hat{\mathcal{F}}^h_t(\hat{\mathbf{x}}^*_t) & (3.76) \end{aligned}$$

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

For $t = h$, it follows:

$$\begin{aligned} \mathcal{F}^h_h(\mathbf{x}^*_h) &= \tilde{\mathcal{F}}_x(\mathbf{x}^*_h) + \tilde{\mathcal{F}}_y(\mathbf{x}^*_h) & (3.77) \\ &\leq \max_{\mathbf{x} \in \Sigma} \left[\tilde{\mathcal{F}}_x(\mathbf{x}) + \tilde{\mathcal{F}}_y(\mathbf{x})\right] = \hat{\mathcal{F}}^h_h(\hat{\mathbf{x}}^*_h) & (3.78) \end{aligned}$$

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

$$\mathcal{F}^h(\mathbf{x}^*) \leq \sum_{t=0}^{h} \hat{\mathcal{F}}_t^h(\mathbf{x}_t^*) = \hat{\mathcal{F}}^h(\hat{\mathbf{x}}^*) \tag{3.79}$$

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 [52]. 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. □

3.5 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 [39, 80]. The goal is typically to find a solution that is robust to possible changes. Other related models include *Mixed CSPs* [16], 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* [73, 81], 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 [59]. 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)* [4]. 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 [2, 4, 14, 15] due to its high complexity – solving Dec-MDPs optimally is NEXP-complete even for the case with only two agents [4]. 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) [4] 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 [4] and they are also typically solved in a centralized manner [15, 31, 61, 68, 72, 83] with some exceptions [57]. Researchers have also developed a hybrid model, called ND-POMDP [58], 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.

3.6 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 [3], and the results report the average of 30 independent runs, each with a timeout of 30 minutes.[10]

3.6.1 Offline Algorithms

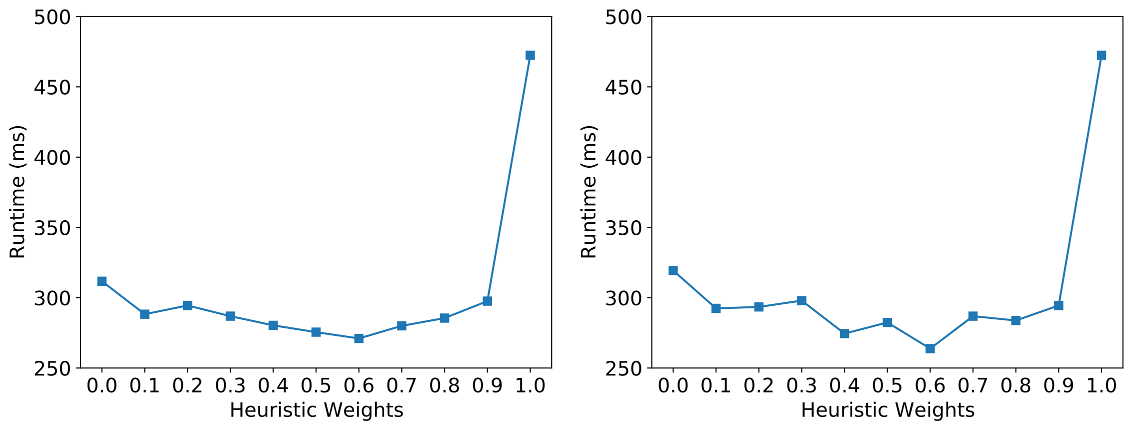
We first evaluate and report the experimental results of our PD-DCOP algorithms in the offline setting. The experiment in the offline setting will shed light on the performance of

[10]`https://github.com/YODA-Lab/PD-DCOP`.

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* [70]. Specifically, we evaluate the following offline PD-DCOP algorithms:

- *Collapsed DPOP (C-DPOP)*, which uses the exact approach introduced in Subsection 3.3.1. 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 3.3.2. 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 53. 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.

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



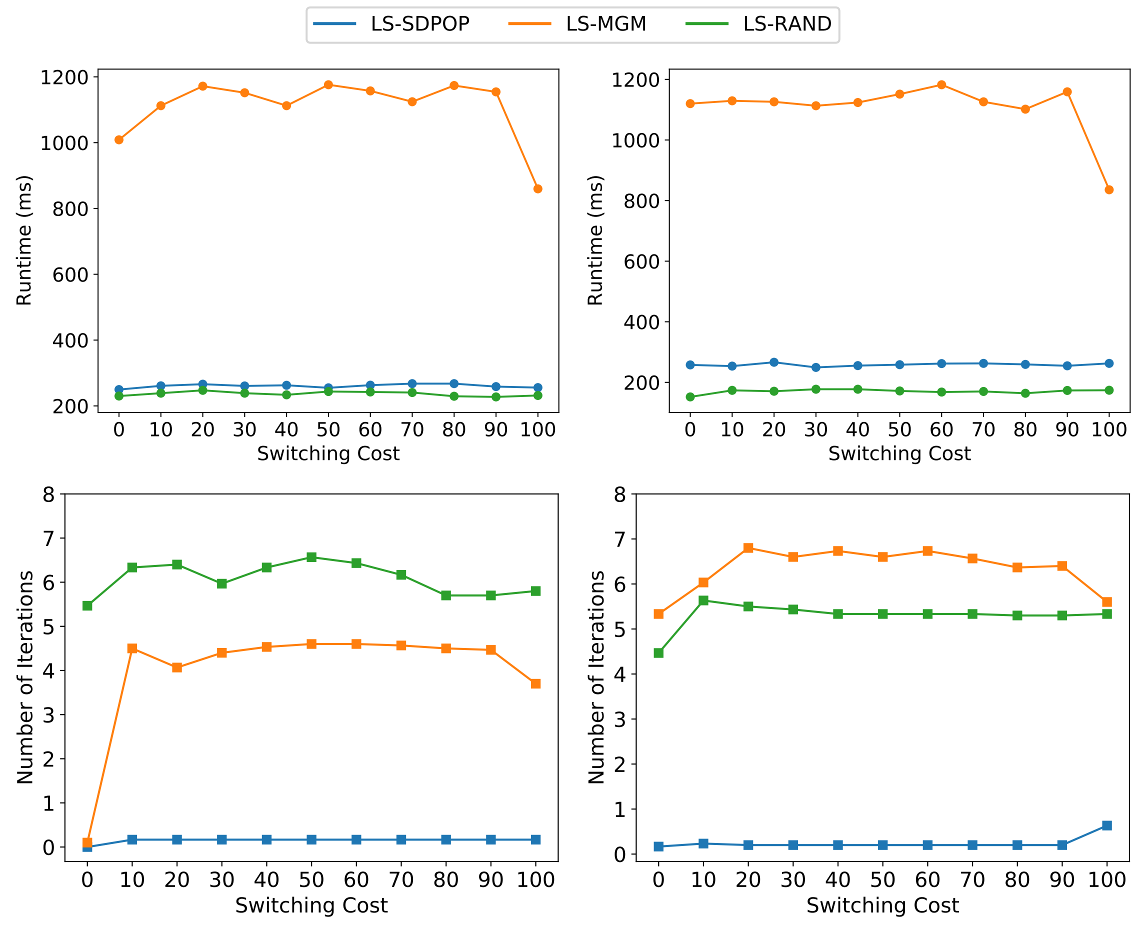
(a) $\gamma = 0.9$ (b) $\gamma = 1$

Figure 3.2: Experimental Results Varying Heuristic Weight

- *Backward DPOP (B-DPOP)*, which uses the greedy approach BACKWARD introduced in Subsection 53. 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.

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 3.2 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 3.19). The runtime is high at both extremes for the following reasons: When $w = 0$, LS-SDPOP exploits weakly the reuse of information, 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

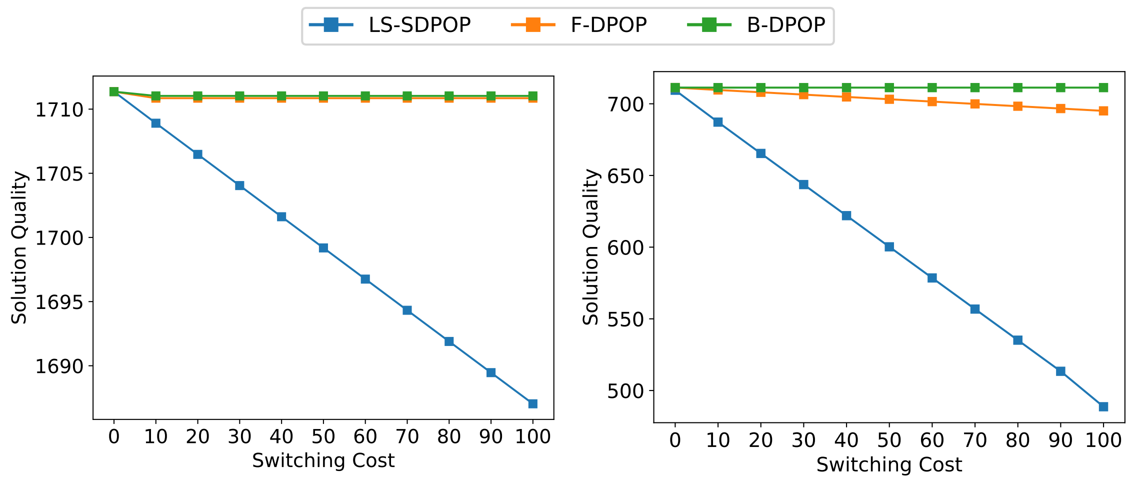


(a) $\gamma = 0.9$ (b) $\gamma = 1$

Figure 3.3: Experimental Results Varying Switching Cost

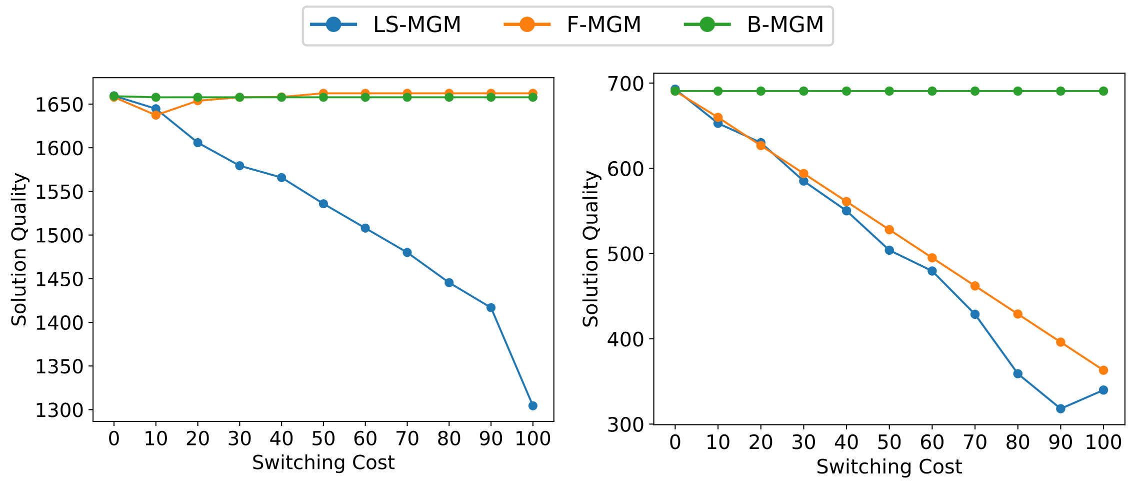
reuse information in the most efficient way and has the smallest runtime. Thus, we set the heuristic weight $w = 0.6$ for LS-SDPOP in the remaining experiments.

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 3.3 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



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

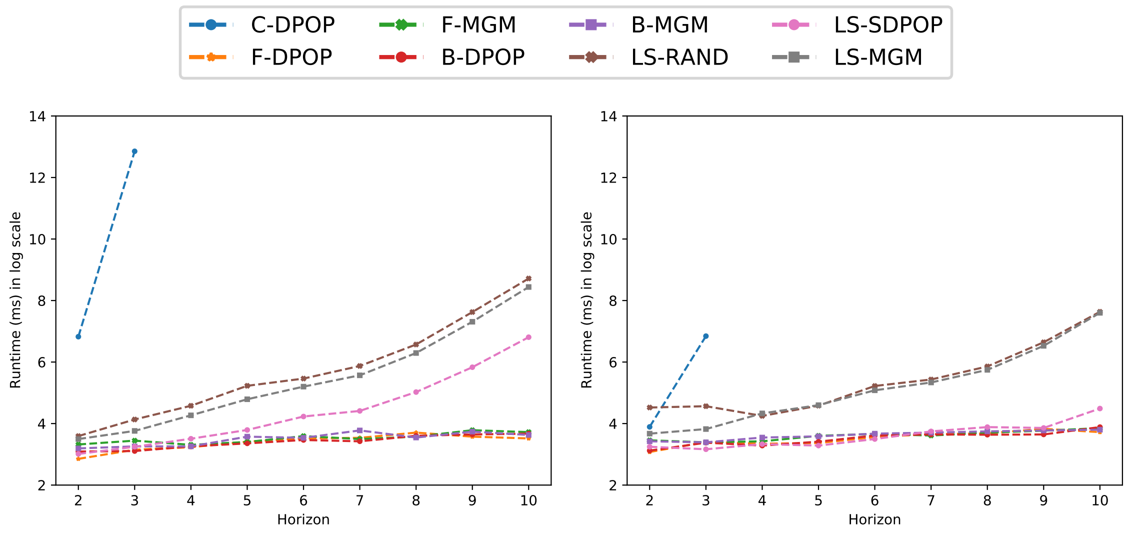
Figure 3.4: Comparison between Sequential Greedy and Local Search for DPOP



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

Figure 3.5: Comparison between Sequential Greedy and Local Search for MGM

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 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



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

Figure 3.6: Experimental Results Varying Horizon

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 3.4 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 3.23 - 3.26). 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 3.5 between LS-MGM, F-MGM, and B-MGM. However, the trend 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 3.6. 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

$\lvert\mathbf{A}\rvert$	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 3.1: Varying the Number of Agents on Random Graphs with $\gamma = 0.9$

$\lvert\mathbf{A}\rvert$	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 3.2: Varying the Number of Agents on Random Graphs with $\gamma = 1$

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 $\lvert\mathbf{A}\rvert$ from 5 to 50 to evaluate the performance of the algorithms with different number of agents. Table 3.1 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 $\lvert\mathbf{A}\rvert = 5$ or $\lvert\mathbf{A}\rvert = 10$, 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 3.2 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.

Dynamic Distributed Meeting Scheduling Problems

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 [53]. 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 3.3 and 3.4 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 3.1 and 3.2, 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

$\|\mathbf{A}\|$	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 3.3: Results for Distributed Meeting Schedling Problems with $\gamma = 0.9$

$\|\mathbf{A}\|$	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 3.4: Results for Distributed Meeting Scheduling Problems with $\gamma = 1$

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.

Distributed Radar Coordination and Scheduling Problems

In this experiment, we evaluate our PD-DCOP algorithms on the Distributed Radar Coordination and Scheduling Problem (DRCSP), our motivating application described in Section 2.7. We use grid networks to represent the DRCSP where sensors are arranged in a rectangular

$\lvert\mathbf{A}\rvert$	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 3.5: Results for Distributed Radar Coordination and Scheduling Problems with $\gamma = 0.9$

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 3.5 and 3.6 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.

$\|\mathbf{A}\|$	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 3.6: Results for Distributed Radar Coordination and Scheduling Problem with $\gamma = 1$

3.6.2 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.

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.

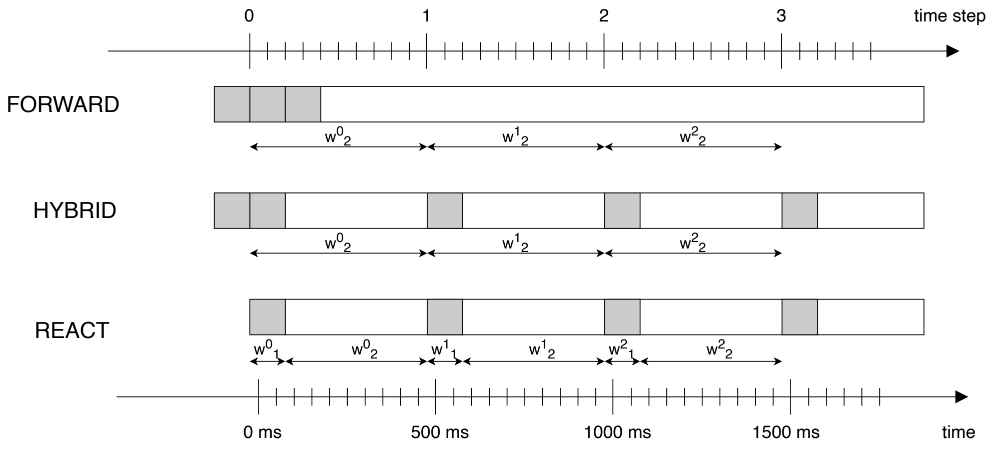


Figure 3.7: Search Time vs. Solution Adoption Time

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 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.

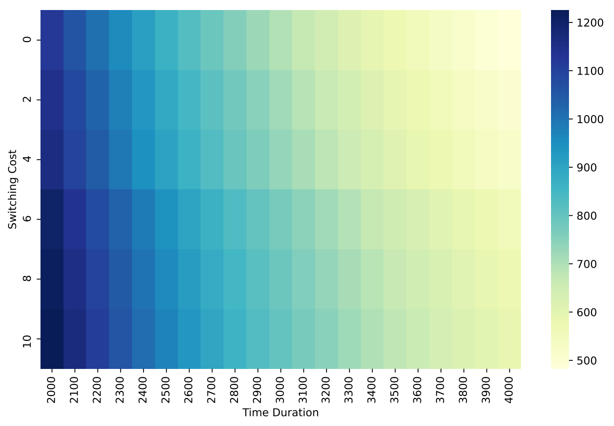
Figure 3.7 illustrates the time the three approaches spend searching for solutions (denoted by gray rectangles) as well as the time they adopt their solutions (denoted by white rectangles), where the time duration between iterations is 500ms. FORWARD starts searching for optimal solutions before the 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:

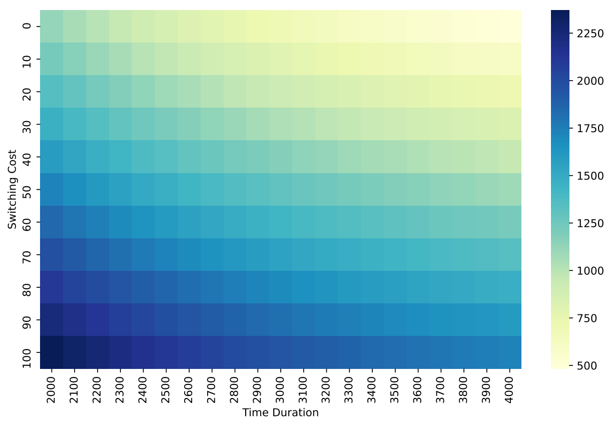
$$U_{eff} = \frac{w_1^t \cdot q_{t-1}^t + w_2^t \cdot q_t^t - (w_1^t + w_2^t) \cdot c_{t-1,t}}{w_1^t + w_2^t} \tag{3.80}$$

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 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.

[12]We discretize time into 50ms intervals.

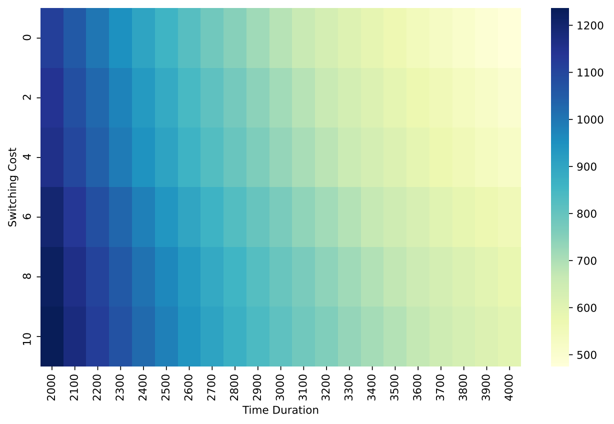


(a) Small Switching Cost Range

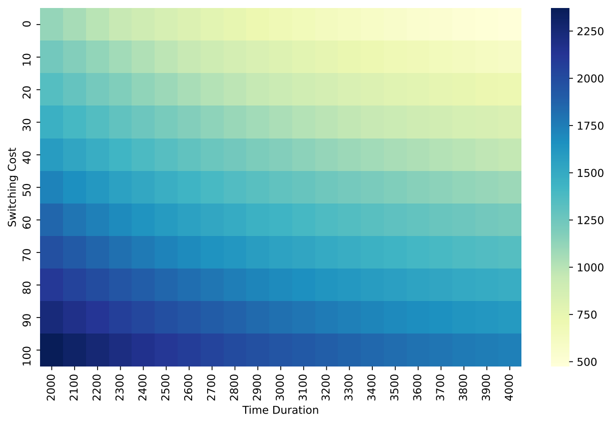


(b) Large Switching Cost Range

Figure 3.8: Comparison between F-DPOP and R-DPOP on Random Networks



(a) Small Switching Cost Range

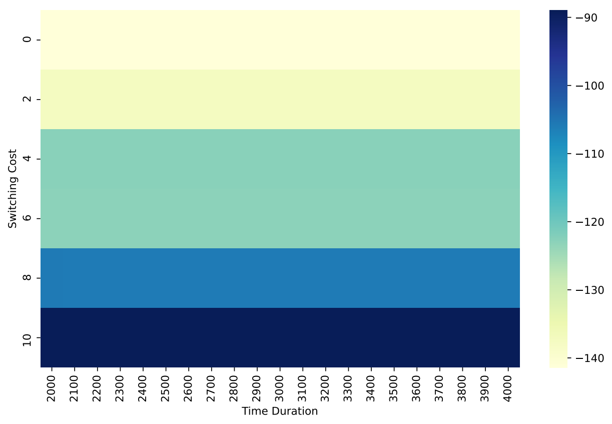


(b) Large Switching Cost Range

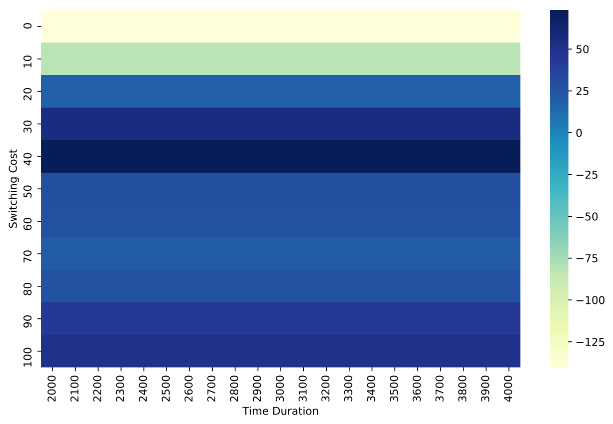
Figure 3.9: Comparison between Hy-DPOP and R-DPOP on Random Networks

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

[13]We avoid using the acronym H-DPOP as that refers to a different algorithm [46]
[14]R-DPOP is the online S-DPOP

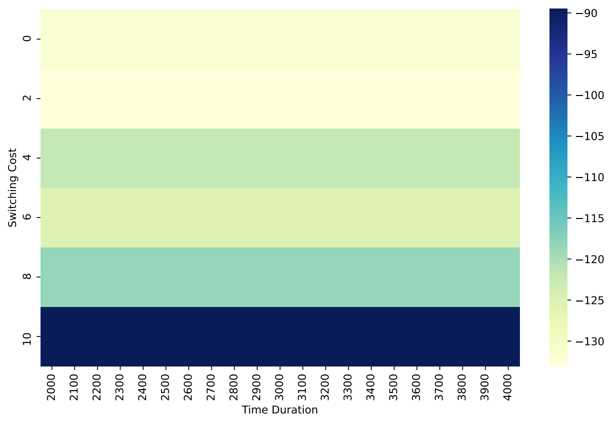


(a) Small Switching Cost Range

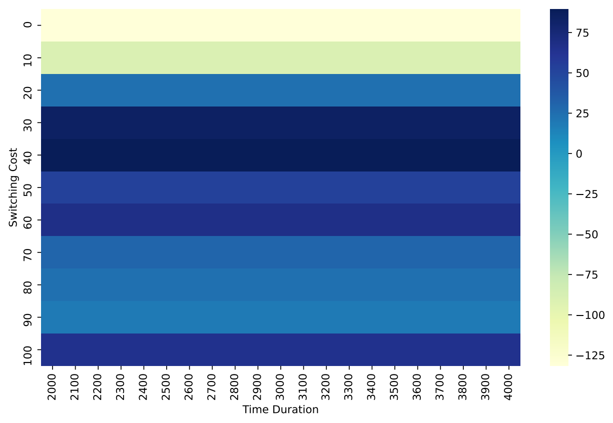


(b) Large Switching Cost Range

Figure 3.10: Comparison between F-MGM and R-MGM on Random Networks



(a) Small Switching Cost Range



(b) Large Switching Cost Range

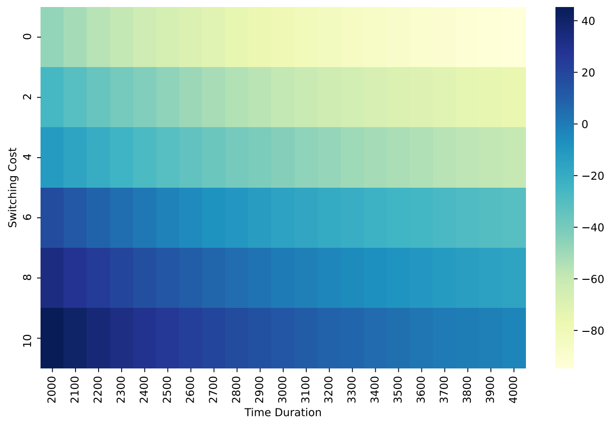
Figure 3.11: Comparison between H-MGM and R-MGM on Random Networks

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 3.80) between a proactive or hybrid algorithm and its reactive counterpart. The 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.

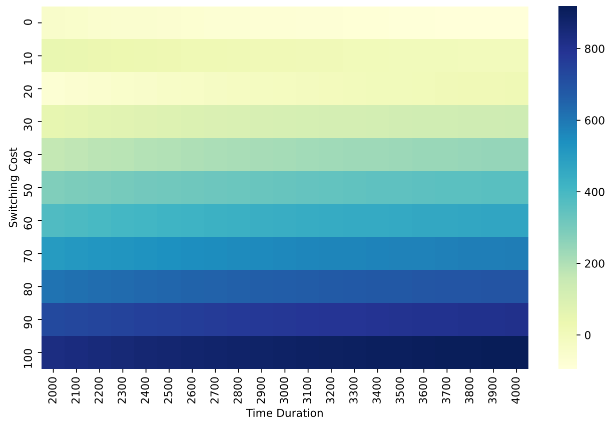
Figures 3.8(a) and 3.8(b) compare F-DPOP and R-DPOP with a small switching cost range of $[0, 10]$ and a large switching cost range of $[0, 100]$, respectively. The heatmap shows the average difference in the effective utilities between F-DPOP and R-DPOP. The difference is calculated by subtracting the 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 3.8(a) and Figure 3.8(b), R-DPOP is worst when the time duration is small and improves with longer duration.

We observe a similar trend in Figures 3.9(a) and 3.9(b), which show the result comparing Hy-DPOP and R-DPOP, except that the difference is marginally larger. The reason is 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 3.10 and 3.11, where we use MGM instead of DPOP to solve the DCOP at each time step on random networks. Similar to the results in Figures 3.8 and 3.9, 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

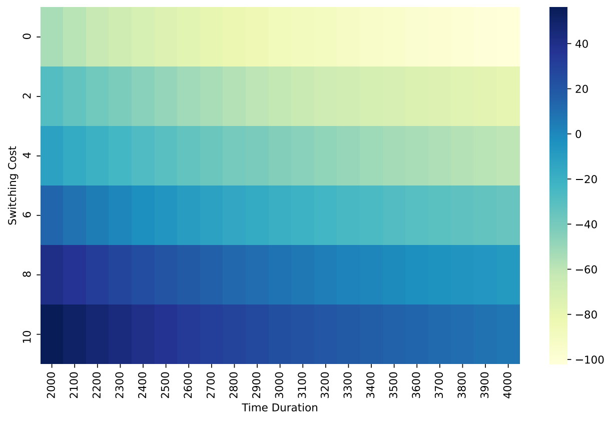


(a) Small switching cost range

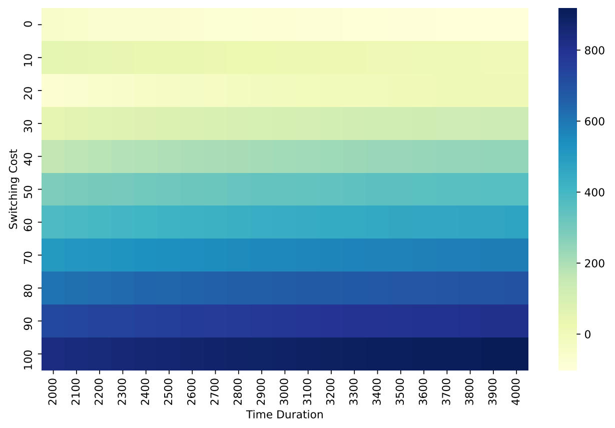


(b) Large switching cost range

Figure 3.12: 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 3.13: Comparison between Hy-DPOP and R-DPOP on Distributed Meeting Scheduling Problems

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 3.12 and 3.13. 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 3.12(b) and Figure 3.13(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.

3.6.3 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) [59] 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 ω_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:

$$f_i'(\langle \omega_i^t, \mathbf{x}_i^{t-1} \rangle, \mathbf{x}_i^t) = f_i(\omega_i^t, \mathbf{x}_i^t) - c \cdot \Delta(\mathbf{x}_i^{t-1}, \mathbf{x}_i^t) \tag{3.81}$$

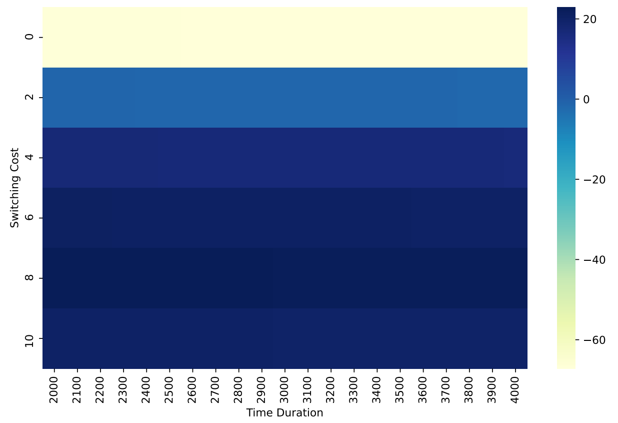
The transition function T' of the random variable y_i is defined as:

$$T_{y_i}'(\langle \omega_i^t, \mathbf{x}_i^{t-1} \rangle, \langle \omega_i^{t+1}, \mathbf{x}_i^t \rangle) = \begin{cases} T_{y_i}(\omega_i^t, \omega_i^{t+1}) & \text{if } \mathbf{x}_i^{t-1} = \mathbf{x}_i^t \\ 0 & \text{otherwise} \end{cases} \tag{3.82}$$

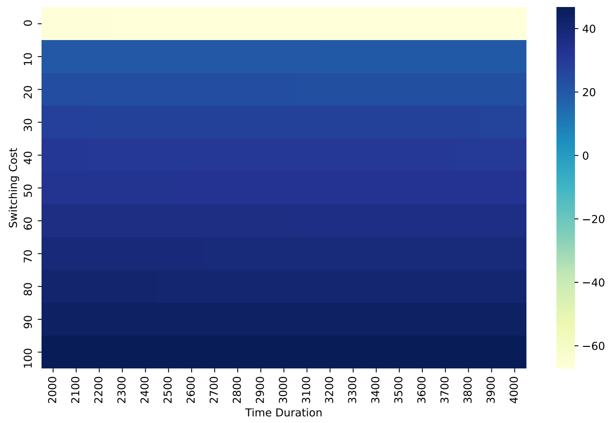
After this step, the PD-DCOP is now mapped to the MD-DCOP and it can be solved by MD-DCOP algorithms.

In this experiment, we choose F-DPOP and R-DPOP as our representative online algorithms and compare them against Decomposed Distributed R-learning [59], 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 3.14(a) and 3.14(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



(a) Small switching cost range

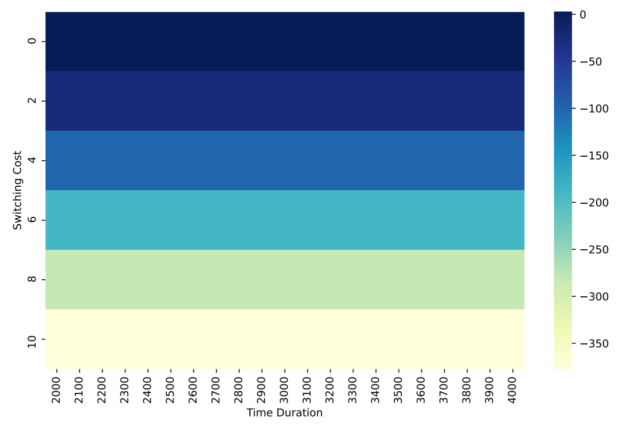


(b) Large switching cost range

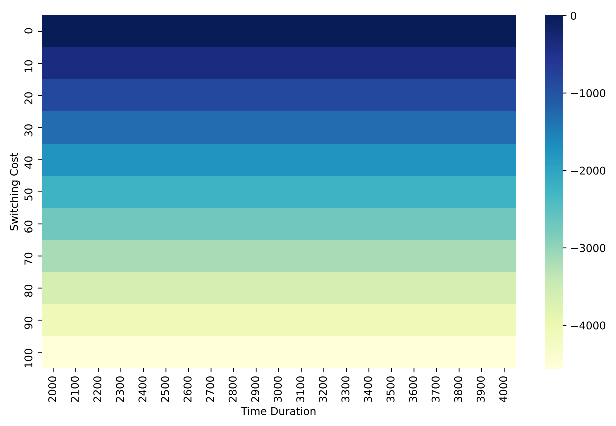
Figure 3.14: Difference in Effective Utilities of F-DPOP minus Decomposed Distributed R-learning

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 3.15(a) and 3.15(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 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



(a) Small switching cost range



(b) Large switching cost range

Figure 3.15: Difference in Effective Utilities of R-DPOP minus Decomposed Distributed R-learning

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.

3.7 Discussions and 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 limitation, in this chapter, 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.

The PD-DCOP formulation makes it possible to model and solve several real-life applications where the prior information is available in dynamic environments. One such application is our motivating DRCSP application. However, by taking into account the prior information, the quality of the solution provided by PD-DCOPs depends significantly on the quality of the information and on the quality of the prediction on how the environment might change over time. Thus, for PD-DCOPs to be deployed successfully in the real world, the prior information should be guaranteed to be reliable to some degree.

Chapter 4

Continuous Distributed Constraint Optimization Problems

In many applications, agents often interact in a complex environment that requires a wide range of possible actions. The DCOP formulation assumes that the domains of the variables are discrete, and it limits the capability of the agents to coordinate efficiently in such environment. To overcome this limitation, researchers have proposed the *Continuous DCOP (C-DCOP)* formulation to model DCOPs with continuous variables and proposed several heuristic algorithms to solve C-DCOPs. However, existing algorithms usually come with some restrictions on the form of constraint utilities and without guarantee on the solution quality. Therefore, in this chapter, we *(i)* propose an exact algorithm to solve a specific subclass of C-DCOPs; *(ii)* propose heuristics algorithms with quality guarantee to solve general C-DCOPs; *(iii)* propose additional C-DCOP algorithms that are more scalable; and *(iv)* empirically show that our algorithms outperform existing state-of-the-art C-DCOP algorithms when given the same communication limitations.

4.1 Introduction

The DCOP formulation has been widely used to model and solve many multi-agent coordination problems where agents choose their actions from a set of possibilities. Typically, DCOPs assume that the variables are discrete and the constraint utilities are represented in tabular form (i.e., a utility is defined for every combination of discrete values of variables). While these assumptions are reasonable in some applications where values of variables correspond to a set of *discrete* possibilities (e.g., the set of tasks that robots can perform in multi-robot coordination problems or the set of coalitions that agents can join in coalition structure generation problems), they make less sense in applications where values of variables correspond to a *continuous* range of possibilities (e.g., the range of orientations a sensor can take in sensor networks or the range of frequencies an agent can choose in wireless networks). For example, in DRCSPs, the weather phenomena could be at any location in the network, and to better sense the phenomena in this case, the sensors should have a wider range of sensing directions.

These limiting assumptions have prompted Stranders *et al.* [69] to propose *Continuous DCOPs (C-DCOPs)*, which extend DCOPs to allow for continuous variables. As variables can now take values from a continuous range, constraint utilities are also extended from tabular forms to functional forms. To solve such problems, Stranders *et al.* [69] extended the discrete *Max-Sum (MS)* algorithm [18] to *Continuous MS (CMS)*, where constraint utility functions are approximated by piecewise linear functions. Voice *et al.* [79] later proposed *Hybrid CMS (HCMS)*, which combines the discrete MS algorithm with continuous non-linear optimization methods. Specifically, agents in HCMS approximate the utility functions with a number of samples that they iteratively improve over time. A key limitation of CMS and HCMS is that they both do not provide quality guarantees on the solutions found. The reason is that they rely on discrete MS as the underlying algorithmic framework, which does not provide quality guarantees on general graphs.

To overcome this limitation, we extend the inference-based *Distributed Pseudo-tree Optimization Procedure (DPOP)* [63] algorithm to three extensions – *Exact Continuous DPOP (EC-DPOP)*; *Approximate Continuous DPOP (AC-DPOP)*; and *Clustered AC-DPOP (CAC-DPOP)*. We also extend the search-based *Distributed Stochastic Algorithm (DSA)* [89] to *Continuous DSA (C-DSA)*. While EC-DPOP provides an exact approach to solve C-DCOPs with linear or quadratic utility functions and are defined over tree-structured graphs, AC-DPOP, CAC-DPOP, and C-DSA solve C-DCOPs approximately with any smooth and differentiable utility functions and without restriction on graph structure. We also provide theoretical properties on the error bounds of AC-DPOP and communication complexities of AC-DPOP, CAC-DPOP, and C-DSA. Finally, we show that these algorithms outperform HCMS in randomly generated instances when given the same communication limitations.

4.2 C-DCOP Algorithms

We now introduce four C-DCOP algorithms: *Exact Continuous DPOP (EC-DPOP)*, *Approximate Continuous DPOP (AC-DPOP)*, and *Clustered AC-DPOP (CAC-DPOP)*, which are based on DPOP; and *Continuous DSA (C-DSA)*, which is based on DSA. These algorithms extend the capability of their original algorithms such that they can solve C-DCOPs with continuous variables and utility functions.

4.2.1 Exact Continuous DPOP

In this section, we propose *Exact Continuous DPOP (EC-DPOP)*, which is an exact algorithm. EC-DPOP solves C-DCOPs that are defined over tree-structured graphs with linear or

```
Function ADD-Functions($f_{pw}, g_{pw}$)
54 Initialize a piecewise function $h_{pw}$
55 $\langle \mathbf{x}, \mathbf{d} \rangle \leftarrow$ GetCommonVariablesAndRanges($f_{pw}, g_{pw}$)
56 foreach domain $d \in \mathbf{d}$ do
57     foreach $f \in f_{pw}$ with domain $d_f$ do
58         foreach $g \in g_{pw}$ with domain $d_g$ do
59             if $d$ is a sub-domain of $f$ and $g$ then
60                 $h \leftarrow f + g$
61                 $d_h = d \cup d_f \cup d_g$
62                 Add $h$ with domain $d_h$ to $h_{pw}$
63 return $h_{pw}$
```

quadratic utility functions. The algorithm extends the two primary operations of DPOP in the UTIL propagation phase – Add and Project. Those modification are modified such that they can be applied to C-DCOPs in the context of continuous variables and real-valued functions.

In the UTIL propagation phase of DPOP, each agent *adds* the utilities in UTIL messages received from its children together with the utilities of constraints that the agent shares with the agents in its separator. Then, it *projects* out its own variable and sends the projected utilities as a UTIL message to its parent. Both of these processes are straightforward as utility functions are represented in tabular form, thereby allowing the agents to enumerate through all possible value combinations, aggregate their corresponding utilities, and optimize over them. However, this process is more complicated in C-DCOPs, where utility functions are represented in functional form. We now describe Add-Functions and Project-Function, which are two EC-DPOP operations extended from the Add and Project operations of DPOP respectively.

ADD-Functions: In EC-DPOP, each UTIL message contains a piecewise function that is derived from the Project-Function operation (described below). The addition of two

```
Function PROJECT-Function($f_{pw}, x_i$)
64 Initialize a piecewise function $h_{pw}$
65 foreach $f \in f_{pw}$ do
66     Solve $\frac{\partial f}{\partial x_i} = 0$ for closed-form solutions $\bar{x}_i = g^*(\mathbf{x})$
67     Compute $\bar{g}(\mathbf{x}) = f(x_i = \bar{x}, \mathbf{x})$
68     Compute $g(\check{\mathbf{x}}) = f(x_i = LB_x, \mathbf{x})$
69     Compute $g(\hat{\mathbf{x}}) = f(x_i = UB_x, \mathbf{x})$
70     Solve $\bar{g}, \check{g},$ and $\hat{g}$ pairwise for intersection range set $\mathbf{r}$
71     foreach $r \in \mathbf{r}$ do
72         Detemine either $\bar{g}, \check{g},$ or $\hat{g}$ is the largest on range $r$
73         Add the function with range $r$ to $h_{pw}$
74 return $h_{pw}$
```

piecewise functions is done by adding their sub-functions, which may have different domains. We will use the following two functions for illustration:

$$f_{12}(x_1, x_2) = \begin{cases} f_{12}^a & \text{if } x_1 \in [0,4], x_2 \in [0,6] \\ f_{12}^b & \text{if } x_1 \in [0,4], x_2 \in [6,10] \\ f_{12}^c & \text{if } x_1 \in [4,10], x_2 \in [0,6] \\ f_{12}^d & \text{if } x_1 \in [4,10], x_2 \in [6,10] \end{cases} \tag{4.1}$$

$$f_{23}(x_2, x_3) = \begin{cases} f_{23}^a & \text{if } x_2 \in [0,3], x_3 \in [0,7] \\ f_{23}^b & \text{if } x_2 \in [0,3], x_3 \in [7,10] \\ f_{23}^c & \text{if } x_2 \in [3,10], x_3 \in [0,7] \\ f_{23}^d & \text{if } x_2 \in [3,10], x_3 \in [7,10] \end{cases} \tag{4.2}$$

When adding two piecewise functions, we first identify the common variables between the two functions and create a new set of atomic ranges for the variables (line 55). For example, when adding the functions f_{12} and f_{23} above, the only common variable is x_2, and the atomic ranges for x_2 are $[0, 3]$, $[3, 6]$, and $[6, 10]$. The ranges of the other variables remain unchanged from their original functions. We then take the Cartesian product of the range sets of all common variables and associate the appropriate function to that range. For example, adding f_{12} and f_{23} will result in f_{123} (line 56-62):

$$f_{123}(x_1, x_2, x_3) = \begin{cases} f_{12}^a + f_{23}^a & \text{if } x_1 \in [0,4], x_2 \in [0,3], x_3 \in [0,7] \\ f_{12}^a + f_{23}^b & \text{if } x_1 \in [0,4], x_2 \in [0,3], x_3 \in [7,10] \\ f_{12}^c + f_{23}^a & \text{if } x_1 \in [4,10], x_2 \in [0,3], x_3 \in [0,7] \\ f_{12}^c + f_{23}^b & \text{if } x_1 \in [4,10], x_2 \in [0,3], x_3 \in [7,10] \\ \dots \end{cases} \tag{4.3}$$

PROJECT-Function: Projecting out a variable x_i from a piecewise function means projecting out x_i from every sub-function $f(x_i, x_{i_1}, \dots, x_{i_k})$:

$$g(x_{i_1}, \dots, x_{i_k}) = \max_{x_i} f(x_i, x_{i_1}, \dots, x_{i_k}) \tag{4.4}$$

First, we solve the following for closed-form solutions (line 66):

$$\frac{\partial f(x_i, x_{i_1}, \dots, x_{i_k})}{\partial x_i} = 0 \tag{4.5}$$

Let $\bar{x}_i = g^*(x_{i_1}, \dots, x_{i_k})$ be the solution to the above equation, one candidate function for g is (line 67):

$$\bar{g}(x_{i_1}, \dots, x_{i_k}) = f(x_i = \bar{x}_i, x_{i_1}, \dots, x_{i_k}) \tag{4.6}$$

We then compute other two candidate functions (line 68-69):

$$\check{g} = f(x_i = LB_{x_i}, x_{i_1}, \ldots, x_{i_k}) \tag{4.7}$$

$$\hat{g} = f(x_i = UB_{x_i}, x_{i_1}, \ldots, x_{i_k}) \tag{4.8}$$

Next, we need to find the intervals where each of the functions $\bar{g}, \check{g}$, and $\hat{g}$ is the largest (line 70-73). Those intervals are the intersections between the three functions and, thus, we solve each of the equations below to find them:

$$\check{g}(x_{i_1}, \ldots, x_{i_k}) = \hat{g}(x_{i_1}, \ldots, x_{i_k}) \tag{4.9}$$

$$\check{g}(x_{i_1}, \ldots, x_{i_k}) = \bar{g}(x_{i_1}, \ldots, x_{i_k}) \tag{4.10}$$

$$\hat{g}(x_{i_1}, \ldots, x_{i_k}) = \bar{g}(x_{i_1}, \ldots, x_{i_k}) \tag{4.11}$$

The result of this process is the set of intervals where either $\bar{g}$, $\check{g}$, or $\hat{g}$ is the largest. The projected function g is the piecewise function that consists of $\bar{g}$, $\check{g}$, and $\hat{g}$ with the intervals that they are the largest in.

Unfortunately, it is not always possible to find closed-form solutions to the partial derivative in Equation (4.5). We discuss below two types of functions – binary linear and quadratic functions – where it is possible to find closed-form solutions. We assume that all coefficients are non-zero.

- Binary linear functions of the form $f(x_i, x_{i_1}) = ax_i + bx_{i_1} + c$. By following the monotonicity property of linear functions, we can find $g(x_{i_1}) = \max_{x_i} f(x_i, x_{i_1})$ at the two extremes:

$$g(x_{i_1}) = \begin{cases} f(x_i = LB_{x_i}, x_{i_1}) & \text{if} \quad a > 0 \\ f(x_i = UB_{x_i}, x_{i_1}) & \text{otherwise} \end{cases} \tag{4.12}$$

- Binary quadratic functions of the form $f(x_i, x_{i_1}) = ax_i^2 + bx_i + cx_{i_1}^2 + dx_{i_1} + ex_ix_{i_1} + f$. We first take the partial derivative and setting it to 0 to find the critical point:

$$\frac{\partial f(x_i, x_{i1})}{\partial x_i} = 0 \tag{4.13}$$

$$\bar{x}_i = \frac{-b - ex_{i_1}}{2a} \tag{4.14}$$

As $\bar{x}_i$ has to belong to the interval $[LB_{x_i}, UB_{x_i}]$, we solve the inequalities below to find the range x_{i_1} as the domain of $\bar{g}(x_{i_1})$:

$$LB_{x_i} \leq \frac{-b - ex_{i_1}}{2a} \leq UB_{xi} \tag{4.15}$$

Example 1: Consider that agent a_1 projects out its variable x_1 from the sub-function $f(x_1, x_2)$:

$$f(x_1, x_2) = -2x_1^2 + 4x_1 + 2x_2^2 + x_2 + 7x_1x_2 - 10 \tag{4.16}$$

where $x_1 \in [-5, 5]$ and $x_2 \in [-10, 10]$. The agent needs to find the piecewise function $g(x_2) = \max_{x_1} f(x_1, x_2)$. The two functions at the bounds of x_1's range are:

$$\breve{g}(x_2) = f(x_1 = -5, x_2) = 2x_2^2 - 34x_2 - 80 \qquad x_2 \in [-10, 10] \tag{4.17}$$

$$\hat{g}(x_2) = f(x_1 = 5, x_2) = 2x_2^2 + 36x_2 - 40 \qquad x_2 \in [-10, 10] \tag{4.18}$$

First, we find the critical point of f by taking the partial derivative:

$$\frac{\partial f(x_1, x_2)}{\partial x_1} = 0 \tag{4.19}$$

$$-4x_1 + 4 + 7x_2 = 0 \tag{4.20}$$

$$x_1 = \frac{7x_2 + 4}{4} \tag{4.21}$$

Since $x_1 \in [-5, 5]$, we need to find the appropriate range of x_2:

$$-5 \leq x_1 \leq 5 \tag{4.22}$$

$$-5 \leq \frac{7x_2 + 4}{4} \leq 5 \tag{4.23}$$

$$\frac{-24}{7} \leq x_2 \leq \frac{16}{7} \tag{4.24}$$

Now, we get the function $\bar{g}(x_2)$ at the critical point $x_1 = \frac{7x_2+4}{4}$:

$$\bar{g}(x_2) = f(x_1 = \frac{7x_2 + 4}{4}, x_2) \tag{4.25}$$

$$= \frac{65}{8}x_2^2 + 8x_2 - 8 \tag{4.26}$$

where $x_2 \in [\frac{-24}{7}, \frac{16}{7}]$.

Next, we will find all intersection points of $\breve{g}, \hat{g}$, and $\bar{g}$ by solving them pairwise. By solving $\breve{g} = \hat{g}$, we have:

$$\breve{g}(x_2) = \hat{g}(x_2) \tag{4.27}$$

$$2x_2^2 - 34x_2 - 80 = 2x_2^2 + 36x_2 - 40 \tag{4.28}$$

$$x_2 = -\frac{4}{7} \tag{4.29}$$

Solving $\check{g} = \bar{g}$:

$$\check{g}(x_2) = \bar{g}(x_2) \tag{4.30}$$

$$2x_2^2 - 34x_2 - 80 = \frac{65}{8}x_2^2 + 8x_2 - 8 \tag{4.31}$$

$$x_2 = -\frac{24}{7} \tag{4.32}$$

Solving $\hat{g} = \bar{g}$:

$$\hat{g}(x_2) = \bar{g}(x_2) \tag{4.33}$$

$$2x_2^2 + 36x_2 - 40 = \frac{65}{8}x_2^2 + 8x_2 - 8 \tag{4.34}$$

$$x_2 = \frac{16}{7} \tag{4.35}$$

After finding all intersection points of the three functions, we combine them with the bounds of x_2's range. This will result in a set of ranges: $[-10, -\frac{24}{7}], [-\frac{24}{7}, -\frac{4}{7}], [-\frac{4}{7}, \frac{16}{7}], [\frac{16}{7}, 10]$. In each range, by choosing an arbitrary point and evaluating the functions $\check{g}, \bar{g}$, and $\hat{g}$, we can determine which one is the largest on that range. Finally, the projection of the utility function $f(x_1, x_2)$ is:

$$g(x_2) = \max_{x_1} f(x_1, x_2) \tag{4.36}$$

$$= \max_{x_1} \left(-2x_1^2 + 4x_1 + 2x_2^2 + x_2 + 7x_1x_2 - 10\right) \tag{4.37}$$

$$= \begin{cases} 2x_2^2 - 34x_2 - 80, & x_2 \in [-10, -\frac{24}{7}] \\ \frac{65}{8}x_2^2 + 8x_2 - 8, & x_2 \in [-\frac{24}{7}, -\frac{4}{7}] \\ \frac{65}{8}x_2^2 + 8x_2 - 8, & x_2 \in [-\frac{4}{7}, \frac{16}{7}] \\ 2x_2^2 + 36x_2 - 40, & x_2 \in [\frac{16}{7}, 10] \end{cases} \tag{4.38}$$

4.2.2 Approximate Continuous DPOP

In general C-DCOPs, it is not always possible to find a closed-form solution to Equation (4.5) (e.g., it is a multivariate equation). Therefore, an approximation approach is desired for C-DCOPs.

In this section, we introduce *Approximate Continuous DPOP (AC-DPOP)*, which is an approximation algorithm that can solve C-DCOPs without any restriction on the functional form of the constraint utilities. AC-DPOP is similar to DPOP in that the algorithm has the same three phases: pseudo-tree generation, UTIL propagation, and VALUE propagation. The pseudo-tree generation phase is identical to that of DPOP, and the UTIL and VALUE propagation phases share some similarities.

We now describe how these two propagation phases work at a high level. In the UTIL propagation phase, like DPOP, agents in AC-DPOP first discretizes the domains of variables and sends up UTIL tables that contain utilities for each value combination of values of separator agents. However, unlike DPOP, agents in AC-DPOP perform local optimization of these values by "moving" them along the gradients of relevant utility functions in order to improve the overall solution quality. As such, the addition and projection operators have to be updated as well. In the VALUE propagation phase, like DPOP, agents in AC-DPOP sends down their best value down to their children in the pseudo-tree. However, unlike DPOP, agents in AC-DPOP may receive values of ancestors that do not map to computed utilities. As such, the agents must perform local interpolation of the utilities value in this phase.

We now describe the algorithm in more detail, where we focus on the UTIL and VALUE propagation phases of the algorithm.

Procedure AC-UTIL($\mathbf{T}_i$)

```
75 if isLeaf() then
76     $V \leftarrow$ DiscretizePPDomain()
77     $V' \leftarrow$ LeafMoveValues($V$)
78     $T_{p_i} \leftarrow$ CreateUtilTable($V'$)
79 else
80     receive Util$_c(T_c)$ from each $a_c \in Children_i$
81     Add additional tuples and interpolate utilities for all $T_c$
82     $UTIL_i \leftarrow \text{Add}\left( \underset{a_s \in Separator_i}{f_i^s} , \underset{a_c \in Children_i}{T_c} \right)$
83     $V' \leftarrow$ NonLeafMovePPValues($UTIL_i$)
84     $T_{pi} \leftarrow$ Interpolate($V', UTIL_i$)
85 send Util$_i(T_{pi})$ to $Parent_i$
```

UTIL Propagation: In this phase, each leaf agent first discretizes the domains of agents in its separator (i.e., its parent and pseudo-parents) and then stores the Cartesian product of these discrete values in set V (line 76). Therefore, each element $v \in V$ is a tuple $\langle v_{i_1}, \ldots, v_{i_k} \rangle$, where v_{i_j} is the value of separator agent a_{i_j}.

Then, for each tuple $v \in V$, the agent "moves" each value v_{i_j} in the tuple along the gradient of each function that is relevant to agent a_{i_j} (line 77). Specifically, the agent updates value v_{i_j} for each separator agent x_{i_j} of the leaf agent x_i:

$$v_{i_j} = v_{i_j} + \alpha \left. \frac{\partial f_{i_j}(x_i, x_{i_j})}{\partial x_{i_j}} \right|_{\text{argmax}_{x_i} f_{i_j}(x_i, x_{i_j} = v_{i_j})}^{v_{i_j}} \tag{4.39}$$

, where $f_{i_j}(x_i, x_{i_j})$ is the utility function between the leaf agent x_i and the separator agent x_{i_j}, and α is the *learning rate* of the algorithm. The agent "moves" the values until they have either converged or a maximum number of iterations is reached. Then, the updated values in V' and their corresponding utilities define the UTIL table that is sent to the parent agent in a UTIL message (line 78).

As in DPOP, each non-leaf agent will first wait for the UTIL messages from each of its children. When all the UTIL messages are received, the agent processes the UTIL tables in the UTIL message from each child. Note that in regular DPOP, the Cartesian product of the values of agents are consistent across the UTIL tables of all children (i.e., if the values of an agent a exists in the Cartesian products of two children, then those values are identical). The reason is because all agents agree on the discretization of the domain of agent a and do not update the value of that agent (such as through Equation (4.39)). Therefore, each agent can easily add up the utilities in the UTIL tables received together with the utilities of constraints between the agent and its separator.

In contrast, since the values of agents are updated according to Equation (4.39) in AC-DPOP, these values may no longer be consistent across different UTIL tables received. To remedy this issue, each agent first adds additional tuples to each UTIL table received such that the Cartesian product of the values of agents are consistent across all the UTIL tables. Then, it approximates the utilities of the newly added tuples by interpolating between the utilities of the existing tuples. Finally, since the UTIL tables are now all consistent, the agent adds up the utilities in the UTIL tables of children together with the utilities of constraints between the agent and its separator in the same way as DPOP. However, if some variables in the separator are missing, the agent will discretize and add their domains to the UTIL table (line 81-82).

After the utilities are added up, similar to leaf agents, the agent x_i will proceed to repeatedly update the values v_{i_j} of the separator a_{i_j} in the updated Cartesian product V using:

$$v_{i_j} = v_{i_j} + \alpha \left. \frac{\partial f_{i_j}(x_i, x_{i_j})}{\partial x_{i_j}} \right|_{\operatorname{argmax}_{x_i} UTIL_i(x_i, v_{i_1}, \ldots, v_{i_k})}^{v_{i_j}} \tag{4.40}$$

where $UTIL_i$ is the utility table that is constructed from the summation of the children's utilities and the utilities of constraints between the agent x_i with its separator set (line 83). The key difference between this Equation (4.40) and the Equation (4.39) used by leaf agents is that the substitution of $f_{i_j}(x_{i_j} = v_{i_j})$ with $UTIL_i(v_{i_1}, \ldots, v_{i_k})$. The reason for this substitution is that the utilities in the UTIL tables of leaf agents are only a function of constraints with their separator agents and the functional form of those constraints are known. Therefore, leaf agents can optimize exactly those functions to get accurate gradients. In contrast, utilities in the UTIL tables of non-leaf agents are also a function of the constraints between its descendant agents and its separator agent, and the functional form of those constraints are not known. They are only represented by samples within the UTIL tables received and are now integrated into the UTIL table of the non-leaf agent. Therefore, in Equation (4.40), the agent approximates its maximum value x_i by choosing the best value of under the assumption that the values of the other separator agents are exactly the same as in the tuple $\langle v_{i_1}, \ldots, v_{i_k} \rangle$ that is being updated.

After these values are all updated, the agent approximates their corresponding utilities by interpolating between known utilities and sends these utilities up to its parent in a UTIL message (line 84). These UTIL messages propagate up to the root agent, which then starts the VALUE phase.

VALUE Propagation: The root agent starts this phase after processing all the UTIL messages received from its children in the UTIL phase. It chooses its best value based on its computed UTIL table and sends this value down to its children. Like in DPOP, each agent will repeat the same process after receiving the values of its parent and pseudo-parents. However, unlike DPOP, an agent may receive the information that its parent or pseudo-parent is taking on a value that doesn't correspond to an existing value in the agent's UTIL table due to the values being moved during the UTIL propagation phase. As a result, the agent will need to approximate the utility for this new value received and it does so by interpolating between known utilities in its UTIL table.

Once all the leaf agents receive VALUE messages from their parents and choose their best values, the algorithm terminates.

Example 2: Given the following constraint functions of the pseudo-tree where x_1 is the parent of the only child x_4; both leaves x_2 and x_3 are the children of x_4 and are the pseudo-children of x_1:

$$f_{13}(x_1, x_3) = 16x_1^2 + 13x_1 + 12x_3^2 + 18x_3 + 9x_1x_3 - 13 \tag{4.41}$$

$$f_{34}(x_3, x_4) = -3x_3^2 + 18x_3 - 8x_4^2 + 8x_4 + 2x_3x_4 + 12 \tag{4.42}$$

Agent x_3 discretizes its domain $[-100, 100]$ into the set of values $V = \{-100, -50, 0, 50, 100\}$, and computes the Cartesian product of x_1 and x_4's values $V \times V = \{\langle -100, -100\rangle, \langle -100, -50\rangle, \dots \langle 0, 0\rangle, \dots\}$. To move the values of x_1 and x_4 in the tuple $\langle 0, 0\rangle$, the agent follows the Equation (4.39):

$$v_{x_1} = v_{x_1} + \alpha\, \frac{\partial f(x_1, x_3)}{\partial x_1}\bigg|_{\text{argmax}_{x_3} f_{13}(x_1 = v_{x_1}, x_3)}^{v_{x_1}} \tag{4.43}$$

$$= 0 + 0.001\,(32x_1 + 9x_3 + 13)\big|_{x_3=100}^{x_1=0} \tag{4.44}$$

$$= 0.913 \tag{4.45}$$

Similarly, x_3 moves value of its parent x_4:

$$v_{x_4} = v_{x_4} + \alpha \left. \frac{\partial f(x_3, x_4)}{\partial x_4} \right|_{\operatorname{argmax}_{x_3} f_{34}(x_3, x_4 = v_{x_4})}^{v_{x_4}} \tag{4.46}$$

$$= 0 + 0.001 \left(-16x_4 + 2x_3 + 8\right)\Big|_{x_3=3}^{x_4=0} \tag{4.47}$$

$$= 0.014 \tag{4.48}$$

Example 3: With the same pseudo-tree from Example 2, let's consider the constraint function $f_{14}(x_1, x_4) = x_1^2 + 19x_1 + 3x_4^2 - 4x_4 + 16x_1x_4 - 8$, where x_4 receives the following UTIL messages from x_3 and x_2:

(a) $\text{UTIL}_{x_3}^{x_4}$

x_1	x_4	Utility
-1.3	-1.4	22.79
2.1	1.8	23.49
2.2	0.4	19.09

(b) $\text{UTIL}_{x_2}^{x_4}$

x_1	x_4	Utility
-2.3	2.7	19.57
-1.4	-0.3	26.38
1.5	0.5	27.20

As one UTIL message doesn't have some or all value tuples from the other UTIL message (e.g., $\text{UTIL}_{x_3}^{x_4}$ doesn't have the tuple $\langle -2.3, 2.7 \rangle$ from $\text{UTIL}_{x_2}^{x_4}$), agent x_4 needs to add these missing tuples and approximate their utilities using local interpolation. Then, x_4 adds up the two UTIL messages, which now have identical value tuples, with the constraint function $f_{14}(x_1, x_4)$. This process results in the table UTIL_{x_4}:

(a) UTIL$_{x_3}^{x_4}$

x_1	x_4	Utility
-1.3	-1.4	22.79
2.1	1.8	23.49
2.2	0.4	19.09
-2.3	2.7	21.91
-1.4	-0.3	22.20
1.5	0.5	20.82

(b) UTIL$_{x_2}^{x_4}$

x_1	x_4	Utility
-2.3	2.7	19.57
-1.4	-0.3	26.38
1.5	0.5	27.20
-1.3	-1.4	25.42
2.1	1.8	25.57
2.2	0.4	26.28

(c) UTIL$_{x_4}$

x_1	x_4	Utility
-2.3	2.7	-93.22
-1.3	-1.4	57.80
-1.4	-0.3	24.13
1.5	0.5	81.52
2.1	1.8	148.37
2.2	0.4	96.97

Now, the agent x_4 moves its parent x_1's values:

$$v_{x_1} = v_{x_1} + \alpha \left. \frac{\partial f(x_1, x_4)}{\partial x_1} \right|_{\text{argmax}_{x_4} UTIL_{x_4}(x_1 = v_{x_1}, x_4)}^{v_{x_1}} \tag{4.49}$$

$$= 2.2 + 0.001 \left. (2x_1 + 16x_4 + 19) \right|_{x_4=1.75}^{x_1=2.2} \tag{4.50}$$

$$= 2.25 \tag{4.51}$$

To find the argmax value, the agent x_4 first creates the value set $\{-99.75, -99.5, \ldots, 99.75\}$ by discretizing its domain $[-100, 100]$. The agent then combines every value with $x_1 = 2.2$ to create the set of tuples $\{\langle 2.2, -99.75\rangle, \langle 2.2, -99.5\rangle \ldots, \langle 2.2, 99.75\rangle\}$. By approximating the utilities with local interpolation from UTIL$_{x_4}$, x_4 chooses the tuple $\langle 2.2, 1.75\rangle$ with the largest utility and thus pick the argmax value as 1.75. This example also illustrates the argmax operation used in VALUE phase.

4.2.3 Clustered Approximate Continuous DPOP

A possible limitation of AC-DPOP is that the number of tuples in the Cartesian product that is propagated in the UTIL messages can be quite large, especially if additional tuples are added to maintain consistency between the UTIL tables of children. In communication-constrained applications, it is preferred that the number and size of messages transmitted between agents to be as small as possible.

With this motivation in mind, we extend AC-DPOP to *Clustered AC-DPOP (CAC-DPOP)*, which bounds the number of tuples sent in UTIL messages to limit the message size. CAC-DPOP is identical to AC-DPOP in every way except that agents choose k representative tuples and their corresponding utilities to be sent up to their parents in UTIL messages. To choose these k representative tuples, we use the k-means clustering algorithm [51] to cluster the tuples and then approximate the utilities of those tuples through interpolation. This approach assumes that tuples that are close to each other will have similar values.

Note that while only k tuples are sent between agents in UTIL messages, each agent still maintains the original unclustered set of tuples in their memory. Thus, when they perform interpolation during the VALUE propagation phase, they will use the utilities of the unclustered set of tuples since they are more accurate than the utilities of the clustered set of tuples.

4.2.4 Continuous DSA

Continuous DSA (C-DSA) is an approximation C-DCOP algorithm that is based on DSA. Similar to DSA, each agent in C-DSA initially chooses a random value and loops over a sequence of steps that improves the solution quality. Agents also stochastically decide to keep their current values or change them to new values. The difference between C-DSA and DSA lies in the way agents choose their values. Instead of choosing from a discrete domain, each C-DSA agent now chooses from a continuous range by computing the maximum of the aggregate utility functions given the current values of neighboring agents. Specifically, after receiving messages containing the current values of neighbors, each agent evaluates the corresponding multivariate utility functions, resulting in a unary function for each constraint. Then, by adding all the unary functions together and computing its maximum, agents choose the value that has the largest gain.

4.3 Theoretical Results

For each reward function $f(x_i, x_{i_1}, \dots, x_{i_k})$ of an agent x_i and its separator agents $x_{i_1}, \dots, x_{i_k}$, assume that agent x_i discretizes the domains of the reward function into hypercubes of size m (i.e., the distance between two neighboring discrete points for the same agent x_{i_j} is m). Let $\nabla f(v)$ denote the gradient of the function $f(x_i, x_{i_1}, \dots, x_{i_k})$ at $v = (v_i, v_{i_1}, \dots, v_{i_k})$:

$$\nabla f(v) = (\frac{\partial f}{\partial x_i}(v_i), \frac{\partial f}{\partial x_{i_1}}(v_{i_1}), \dots, \frac{\partial f}{\partial x_{i_k}}(v_{i_k})) \tag{4.52}$$

Furthermore, let $|\nabla f(v)|$ denote the sum of magnitude:

$$|\nabla f(v)| = |\frac{\partial f}{\partial x_i}(v_i)| + |\frac{\partial f}{\partial x_{i_1}}(v_{i_1})| + \dots + |\frac{\partial f}{\partial x_{i_k}}(v_{i_k})| \tag{4.53}$$

, and assume that $|\nabla f(v)| \leq \delta$ holds for all utility functions in the DCOP and for all v.

THEOREM **7** *The error bound of* discrete DPOP *is* $|\mathbf{F}|m\delta$.

PROOF: First, we prove that the magnitude of the projection of function f is also bounded from above by δ. Let $x_i = v_i$ be the point where:

$$g(x_{i_1}, \ldots, x_{i_k}) = f(x_i = v_i, x_{i_1}, \ldots, x_{i_k}) \tag{4.54}$$

$$= \max_{x_i} f(x_i, x_{i_1}, \ldots, x_{i_k}) \tag{4.55}$$

Then, assume that $|\nabla g(v)| > \delta$ for all v. Let $v' = (v_i, v_{i_1}, \ldots, v_{i_k})$ and $v'_{-i} = (v_{i_1}, \ldots, v_{i_k})$, then:

$$|\nabla f(v')| = |\frac{\partial f}{\partial x_i}(v_i)| + |\frac{\partial f}{\partial x_{i_1}}(v_{i_1})| + \ldots + |\frac{\partial f}{\partial x_{i_k}}(v_{i_k})| \tag{4.56}$$

$$\geq |\frac{\partial f}{\partial x_{i_1}}(v_{i_1})| + \ldots + |\frac{\partial f}{\partial x_{i_k}}(v_{i_k})| \tag{4.57}$$

$$= |\nabla g(v'_{-i})| \tag{4.58}$$

$$> \delta \tag{4.59}$$

This contradicts our assumption that $|\nabla f(v)| \leq \delta$ for all v.

The error bound of each function is then $m\delta$ because each hypercube is of size m and the magnitude of the gradient within each hypercube is at most δ. As the error may be accumulated each time an agent sums up utility functions, the total error bound for a problem is thus $|\mathbf{F}|m\delta$, where $|\mathbf{F}|$ is the number of utility functions in the problem. □

THEOREM 8 *The error bound of* AC-DPOP *is* $|\mathbf{F}|(m + |\mathbf{A}|k\alpha\delta)\delta$, *where* k *is the number of times each agent "moves" values of its separator by calling Equation* (4.39) *or* (4.40)

PROOF: After each "move" by either Equation (4.39) or (4.40), the maximum size of the hypercubes increases by $\alpha\delta$, where α is the learning rate. Since each agent performs this update only k times, the largest increase in the size of the hypercube is $k\alpha\delta$. Finally, since the value of an agent can be updated by any of its children or pseudo-children, the total increase in the size of the hypercube is thus $|\mathbf{A}|k\alpha\delta$, where $|\mathbf{A}|$ is the number of agents in the problem. Therefore, this combined with the proof of the bound for discrete DPOP, the error bound is thus $|\mathbf{F}|(m + |\mathbf{A}|k\alpha\delta)\delta$. □

Theorem **9** *In a binary constraint graph $G = (\mathbf{X}, E)$, the number of messages of* HCMS *and* C-DSA *with k iterations is $4k|E|$ and $2k|E|$, respectively. The number of messages of* discrete DPOP*,* AC-DPOP*, and* CAC-DPOP *is* $2|\mathbf{X}|$*.*

Proof: HCMS has the same number of messages as that of the Max-Sum algorithm [18]. Every edge of the constraint graph has two variable nodes and one function node and, thus, it takes 4 messages per edge in one iteration. The total number of messages in HCMS is thus $4k|E|$. On the other hand, C-DSA requires 2 messages per edge in one iteration and, thus, requiring $2k|E|$ messages in total.

The number of messages required by AC-DPOP and CAC-DPOP is identical to that of DPOP – each agent sends one UTIL message to its parent and one VALUE message to each of its children in the pseudo-tree. Since pseudo-trees are spanning trees, the number of messages is thus $2|\mathbf{X}|$. □

Theorem **10** *The message size complexity of* discrete DPOP*,* AC-DPOP*, and* CAC-DPOP *is $O(d^w)$, $O((d|\mathbf{X}|)^w)$, and* $\max\{|\mathbf{A}|, k\}$*, respectively, where d is the number of points used by each agent to discretize the domain of its separator agents, w is the induced width of the pseudo-tree, and k is the number of clusters used by CAC-DPOP.*

Proof: For DPOP, the message size complexity is $O(d^w)$ [63]. For AC-DPOP, as the values of an agent are "moved" by their children and pseudo-children, in the worst case, all the values are unique and the maximum number of such values is $O(d|\mathbf{X}|)$. The message sizes are then similar to discrete DPOP with $O(d|\mathbf{X}|)$ values per agent. Therefore, its message size complexity is $O((d|\mathbf{X}|)^w)$. For CAC-DPOP, the message size complexity of UTIL messages is $O(k)$ since only the utilities of the centroids of k clusters are sent. And the message size complexity of VALUE messages is $O(|\mathbf{A}|)$, such as in a fully-connected graph where an agent sends the values of every agent from the root of the pseudo-tree down to itself in a VALUE message to its child. Therefore, the message complexity of the algorithm is the $O(\max\{|\mathbf{A}, k\})$. □

4.4 Related Work

Researchers have proposed several algorithms to solve C-DCOPs. One of such algorithms is Continuous Max-Sum (CMS) [69], which is based on Max-Sum [18], a belief propagation algorithm. To represent the constraints, CMS uses multivariate continuous piecewise linear functions (CPLFs) and later encodes the n-ary CPLFs as n-simplexes. To add two CPLFs, CMS partitions the domains of the two functions and then finds the simplexes that make up the resulting summation function. To project a CPLF, the function is projected onto the corresponding plane and the result is the upper envelope of the simplexes. However, CMS is not suitable for the problems where constraint functions are smooth, and it does not provide quality guarantee for the solution.

Later, Voice *et al.* [79] proposed Hybrid Continuous Max-Sum (HCMS) to solve C-DCOPs with differentiable functions. Instead of working directly on continuous domains, HCMS first discretizes the domain into a number of initial discrete points and then uses continuous non-linear optimization techniques such as gradient method and Newton method to optimize the marginal function at each variable. However, similar to CMS, HCMS does not provide solution quality guarantee.

Recently, Choudhury *et al.* [7] proposed *Particle Swarm Optimization Based Functional DCOP (PFD)* which is based on *Particle Swarm Optimization (PSO)* technique. While being an iterative and a heuristic algorithm, PFD shares the same limitation with CMS and HCMS that they do not provide guarantee for their solutions. Besides, Fransman *et al.* [23] proposed *Bayesian DPOP (B-DPOP)*, an Bayesian optimization based algorithm, to solve C-DCOPs. While B-DPOP guarantees that it will eventually converge to the global optimum for Lipschitz-continuous objective functions, it does not provide guarantees on intermediate solutions prior to convergence.

4.5 Experimental Results

We empirically evaluate *EC-DPOP*, *AC-DPOP*, *CAC-DPOP*, and *C-DSA*[15] against (discrete) *DPOP* and *HCMS* on random trees, random graphs, and the Distributed Radar Coordination and Scheduling Problem (DRCSP), which was introduced in Section 2.7. We adapt the (discrete) DPOP algorithm to solve C-DCOPs by discretizing the continuous domain into discrete representative points.[16]

We measure the quality of solutions, simulated runtimes [70] as well as the number of messages taken by the algorithm. Since HCMS and C-DSA are iterative algorithms that may take a long time and a large number of messages before converging, in order for fair comparisons, we initially planned to terminate the two algorithms after it sends as many messages as the DPOP-variants. However, in a single iteration, HCMS requires more messages than the DPOP-variants, and C-DSA requires the exact number of messages as the DPOP-variants.

[15]We use DSA-B and set $p = 0.6$.

[16]`https://github.com/YODA-Lab/Continuous_DCOPs`.

$\vert\mathbf{A}\vert$	HCMS (time)	C-DSA (time)	DPOP (time)	AC-DPOP (time)				EC-DPOP (time)
				5	10	15	20	
10	129 (129)	177 (34)	220 (170)	330 (710)	356 (735)	374 (753)	404 (774)	518 (406)
20	306 (190)	468 (49)	541 (270)	795 (1.3s)	870 (1.4s)	947 (1.4s)	1008 (1.4s)	—
30	436 (290)	678 (74)	766 (404)	1128 (1.9s)	1230 (2.0s)	1331 (2.0s)	1414 (2.1s)	—
40	636 (358)	1137 (191)	1104 (620)	1587 (3.0s)	1728 (3.0s)	1876 (3.0s)	1980 (3.1s)	—
50	832 (393)	1294 (420)	1456 (741)	2109 (3.8s)	2316 (3.6s)	2533 (3.6s)	2687 (3.8s)	—

Table 4.1: Varying the Number of Agents on Random Trees with Three Initial Discrete Points

$\vert\mathbf{A}\vert$	HCMS (time)	C-DSA (time)	DPOP (time)	AC-DPOP (time)				CAC-DPOP (time)			
				5	10	15	20	5	10	15	20
15	265 (206)	523 (38)	522 (321)	710 (822)	763 (925)	824 (949)	891 (1.0s)	639 (1.2s)	697 (1.3s)	715 (1.3s)	787 (1.5s)
20	345 (308)	842 (43)	865 (6.3s)	1171 (14.7s)	1285 (17.6s)	1334 (32.7s)	1407 (41.7s)	1006 (2.4s)	1017 (2.5s)	975 (2.7s)	973 (2.9s)
25	439 (500)	1108 (64)	—	—	—	—	—	1040 (8.2s)	1101 (10.6s)	1024 (11.9s)	1027 (13.8s)
30	506 (605)	1400 (164)	—	—	—	—	—	1513 (47.6s)	1682 (99.5s)	1597 (96.3s)	1656 (178.0s)

Table 4.2: Varying the Number of Agents on Random Graphs with $p_1 = 0.2$ and Three Initial Discrete Points

	(a) Random Trees with $\vert\mathbf{A}\vert = 20$			(b) Random Graphs with $\vert\mathbf{A}\vert = 20$ and $p_1 = 0.2$			
Points	HCMS (time)	DPOP (time)	AC-DPOP (time)	HCMS (time)	DPOP (time)	AC-DPOP (time)	CAC-DPOP (time)
1	0 (174)	0 (238)	254 (572)	0 (263)	15 (220)	428 (613)	431 (1.5s)
3	306 (190)	541 (270)	870 (1.4s)	345 (308)	865 (6.3s)	1285 (17.6s)	1017 (2.5s)
9	554 (291)	990 (369)	1133 (1.1s)	706 (552)	—	—	1272 (1185.9s)

Table 4.3: Varying the Number of Discretized Points

We thus let HCMS and C-DSA terminate after one iteration. We did not report the actual number of messages since they could be trivially computed via Theorem 9.

Tables 4.1 and 4.2 show the reported solution qualities in a unit of 1,000 and simulated runtimes in milliseconds and seconds (ending with *s*) on random trees and graphs, respectively, where we vary the number of agents $|\mathbf{A}|$ and every algorithm discretizes the domains of variables into three points. We also vary the number of times AC-DPOP and CAC-DPOP agents "move" a point (by calling Equations (4.39) or (4.40)) from 5 to 20. Tables 4.3(a) and 4.3(b) show the results on random trees and graphs, respectively, where we set the number of agents $|\mathbf{A}|$ to 20 and vary the number of discrete points from 1 to 9. In all our experiments, we set the domain of each agent to be in the range $[-100, 100]$. We generate

$\lvert\mathbf{A}\rvert$	HCMS (time)	C-DSA (time)	AC-DPOP (time)				CAC-DPOP (time)			
			5	10	15	20	5	10	15	20
15	796 (33)	758 (22)	1022 (124)	982 (178)	975 (203)	1016 (268)	987 (144)	923 (182)	955 (203)	1014 (246)
20	888 (40)	1012 (24)	1311 (204)	1347 (355)	1364 (453)	1394 (573)	1237 (171)	1256 (251)	1219 (322)	1278 (396)
25	1186 (47)	1371 (26)	1842 (240)	1928 (423)	1849 (548)	1829 (724)	1676 (177)	1812 (286)	1777 (351)	1790 (454)
30	1567 (52)	1704 (25)	2282 (432)	2372 (899)	2353 (1.1s)	2354 (1.5s)	2036 (247)	2057 (444)	2145 (570)	2135 (766)

Table 4.4: Varying the Number of Agents on Distributed Radar Coordination and Scheduling Problems with Three Initial Discrete Points

utility functions that are binary quadratic functions, where the signs and coefficients are randomly chosen. Our experiments were performed on a 2.10GHz machine with 8GB of RAM. Results are averaged over 20 runs, each with a timeout of 30 minutes.

Random Trees: We omit the results of CAC-DPOP from Table 4.1 since it finds identical solutions to AC-DPOP on trees – there is no need to perform any clustering on trees since an agent does not receive utilities for value combinations of its parent from its children since there are no backedges in the pseudo-tree.

Not surprisingly, EC-DPOP finds the best solution since it is an exact algorithm. However, it could only solve the smallest of instances – due to memory limitations, the agents could not store the necessary number of piecewise functions to accurately represent the utility functions after adding functions and projecting out variables. In general, AC-DPOP finds better solutions than DPOP, C-DSA and HCMS but at the cost of higher runtimes. AC-DPOP finds better solutions than DPOP because AC-DPOP spends more time on updating the value of representative points before propagating up the pseudo-tree. In contrast, the values chosen by DPOP is fixed from the start. HCMS performs poorly because a single iteration is insufficient for it to converge to a good solution. Interestingly, a single iteration is sufficient for C-DSA to find solutions that are comparable in quality to those found by DPOP. Additionally, as expected, the quality of solutions found by AC-DPOP improves with increasing number of times points are "moved" by the algorithm.

We omit the results of EC-DPOP from Table 4.3(a) as it failed to solve these instances and we omit the results of CAC-DPOP because it finds identical solutions to AC-DPOP on trees. We do not include C-DSA in the experiment because it does not discretize the domains. Not surprisingly, the quality of solutions found by all the three algorithms and their runtimes increase with increasing number of points. The reason is that the agents can more accurately represent the utility function with more points.

Random Networks: The trends in Table 4.2 are similar to those in random trees, except that CAC-DPOP finds solutions with qualities between that of AC-DPOP and DPOP. The reason is that CAC-DPOP clusters the points into k clusters and only propagate a representative point from each cluster. Therefore, the k points represent the utility functions less accurately than the full number of unclustered points that AC-DPOP uses. However, this reduced number of points propagated also improves the scalability of CAC-DPOP, where it is able to solve problems larger problems than AC-DPOP and DPOP.

The trends in Table 4.3(b) are again similar to that in random trees, except that both AC-DPOP and DPOP ran out of memory with 9 points. Interestingly, CAC-DPOP also finds better solutions than AC-DPOP when they use only 1 point.

Distributed Radar Coordination and Scheduling Problems: Not surprisingly, the trends in Table 4.4 for the Distributed Radar Coordination and Scheduling Problems (DRC-SPs) are similar to those in random trees and random networks. By representing the constraints more accurately, AC-DPOP finds better solutions than C-DSA and HCMS but it comes with the cost of higher runtimes. By clustering the points into k clusters, CAC-DPOP only propagates the representative points during the solving process, and it results in smaller runtimes than AC-DPOP. However, the tradeoff is that the constraints are now less accurately represented, and thus the solution quality from CAC-DPOP is slightly worse than those from AC-DPOP.

4.6 Discussions and Conclusions

Motivated by applications where agents choose their values from continuous ranges, researchers have proposed C-DCOPs to model continuous variables. However, existing methods suffer from the limitation that they do not provide quality guarantees. In this chapter, we remedied this limitation by introducing *(i)* EC-DPOP, which finds exact solutions for C-DCOPs with linear or quadratic utility functions and with tree-structure graphs; *(ii)* AC-DPOP, which finds error-bounded solutions for general C-DCOPs; *(iii)* CAC-DPOP, which limits the message size of AC-DPOP to a user-defined parameter k; and *(iv)* C-DSA, which is a scalable local search C-DCOP algorithm. Experimental results showed that our algorithms find better solutions than HCMS, an existing state-of-the-art algorithm, when given the same communication limitations. Moreover, these algorithms combined extend the applicability of DCOPs to more applications that require quality guarantees on the solutions found as well as those that require limited communication capabilities.

While our algorithms advance the state of the art of C-DCOPs, to apply the algorithms to solve real-world applications, we should take into account the scenarios on which algorithm should be used. For applications that have tree-structure topology and the constraints can be modeled as binary linear or quadratic functions, EC-DPOP could be used to find the optimal solution for small problems. When the guarantee on the solution quality is required, AC-DPOP is an appropriate algorithm in this case since AC-DPOP provides an error bound on the solution quality. For those applications that seek a quick solution, CAC-DPOP is a more scalable algorithm than AC-DPOP with trade-off on quality guarantee, and C-DSA is appropriate for anytime solution.

Chapter 5

Dynamic Continuous Distributed Constraint Optimization Problems

In many coordination problems, agents often require a wide range of actions and the environment usually keeps changing over time. Distributed Constraint Optimization Problems (DCOPs) have been widely employed to solve many multi-agent problems, but they lack the capability to model the problems in such dynamic and complex environment. While *Dynamic DCOPs (D-DCOPs)* and *Continuous DCOPs (C-DCOPs)* have been proposed to model DCOPs in dynamic environments and DCOPs with continuous variables, respectively, the two models were proposed in isolation and could only address one limitation at a time. Therefore, in this chapter, we propose *Dynamic Continuous DCOPs (DC-DCOPs)*, a novel formulation that models both dynamic nature of the environment and continuous nature of the variables, which are inherent in many multi-agent problems. In addition, we introduce several greedy algorithms to solve DC-DCOPs and discuss their theoretical properties. Finally, we empirically evaluate the algorithms in random networks and in distributed weather sensor network application.

5.1 Introduction

For many coordination problems, DCOPs are a suitable model where agents need to coordinate their value assignments to maximize the aggregate constraint utilities. In these problems, DCOPs assume that the domains of variables are discrete and the environment does not change over time. However, in many distributed multi-agent problems, agents often interact in a more dynamic and complex environment. For example, in distributed sensor networks, targets usually move from one location to another location from time to time, and thus their location keeps changing dynamically over time. To adapt to such dynamic environment, sensors should be augmented with the capability to change their sensing direction accordingly. To address this concern, researchers have proposed *Dynamic DCOPs (D-DCOPs)* [47, 64, 65] that models how the problem evolves during the solving process. In addition, to better sense the targets of interest, whose locations correspond to a wide range of possibilities (i.e., the set of all possible locations in a two-dimensional plane or a three-dimensional space of the sensor network), the sensors should be equipped with a continuous range of sensing directions. Therefore, researchers have introduced *Continuous DCOPs (C-DCOPs)* [69] that model continuous variables with a bounded domain and represent the constraints in functional form.

While D-DCOPs and C-DCOPs have been proposed to address the two limiting assumptions of DCOPs, the two models only address these assumptions in isolation. Thus, it remains a challenge to model and solve the problems that *both* have the dynamism nature in the environment and continuous nature of the variables. For example, in DRCSPs, the weather phenomena will keep moving over time and has a wide range of possible locations in the sensor network. Therefore, in this chapter, we propose *Dynamic Continuous DCOPs (DC-DCOPs)*, a novel formulation that models both dynamic environment and continuous variables, which are present in many multi-agent problems. In addition, we introduce several greedy algorithms to solve DC-DCOPs and discuss their theoretical properties. Finally, we empirically evaluate the algorithms in random networks and in distributed radar coordination and sensing problem.

5.2 DC-DCOP Model

A *Dynamic Continuous DCOP (DC-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 continuous *domains* of the decision variables. Each variable $x \in \mathbf{X}$ takes values from the interval $D_x = [LB_x, UB_x]$.
- $\mathbf{\Omega} = \{\Omega_y\}_{y \in \mathbf{Y}}$ is a set of continuous *domains* of the random variables. Each variable $y \in \mathbf{Y}$ takes values from the interval $\Omega_y = [LB_y, UB_y]$.
- $\mathbf{F} = \{f_i\}_{i=1}^k$ is a set of *utility functions*, each defined over a mixed set of decision and random variables: $f_i : \prod_{x \in \mathbf{X} \cap \mathbf{x}^{f_i}} D_x \times \prod_{y \in \mathbf{Y} \cap \mathbf{x}^{f_i}} \Omega_y \to \mathbb{R}^+_0 \cup \{-\infty\}$, where infeasible configurations have $-\infty$ rewards 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: $\mathbf{F}_{\mathbf{X}} = \{f_x\}$, where $\mathbf{x}^{f_x} \cap \mathbf{Y} = \emptyset$, and $\mathbf{F}_{\mathbf{Y}} = \{f_y\}$, where $\mathbf{x}^{f_y} \cap \mathbf{Y} \neq \emptyset$. Note that $\mathbf{F}_{\mathbf{X}} \cup \mathbf{F}_{\mathbf{Y}} = \mathbf{F}$ and $\mathbf{F}_{\mathbf{X}} \cap \mathbf{F}_{\mathbf{Y}} = \emptyset$.
- $p^0_{\mathbf{Y}} = \{p^0_y\}_{y \in \mathbf{Y}}$ is a set of initial *probability density functions* of random variable y.
- $\mathbf{T} = \{T_y\}_{y \in \mathbf{Y}}$ is a set of *transition functions*, where each transition function is a conditional density function $T_y : \Omega_y \times \mathcal{P}(\Omega_y) \to [0, 1]$ specifying the transition from a value $d_y \in \Omega_y$ to a subset of Ω_y.
- $\gamma \in [0, 1]$ is a *discount factor*.
- $h \in \mathbb{N}$ is a finite *horizon*.

- $\mathbf{C} = \{c_x\}_{x \in \mathbf{X}}$ is a set of *switching cost functions*, each defined over a set of decision variables: $c_x : D_x \times D_x \to \mathbb{R}_0^+$. Each switching cost function c_x models the cost associated with the change in the value of the decision variable x from one time step to the next.
- $\alpha : \mathbf{X} \to \mathbf{A}$ is a function that associates each decision variable to one agent.

Throughout this chapter, we assume that each agent controls exactly one decision variable and that each utility function is associated with at most one random variable.[17] The goal of a DC-DCOP is to find a sequence of $h+1$ assignments $\mathbf{x}^*$ for all the decision variables in $\mathbf{X}$:

$$\mathbf{x}^* = \underset{\mathbf{x}=\langle \mathbf{x}^0,\ldots,\mathbf{x}^h\rangle \in \Sigma^{h+1}}{\operatorname{argmax}} \mathcal{F}^h(\mathbf{x}) \tag{5.1}$$

$$\mathcal{F}^h(\mathbf{x}) = \underbrace{\sum_{t=0}^{h} \gamma^t \left[\mathcal{F}_x^t(\mathbf{x}^t) + \mathcal{F}_y^t(\mathbf{x}^t)\right]}_{\mathbf{P}} - \underbrace{\sum_{t=0}^{h-1} \gamma^t \left[C_{\mathbf{x}}(\mathbf{x}^t, \mathbf{x}^{t+1})\right]}_{\mathbf{Q}} \tag{5.2}$$

where Σ is the assignment space for the decision variables of the DC-DCOP. The first term $\mathbf{P}$ refers to the optimization over $h+1$ time steps, with:

$$\mathcal{F}_x^t(\mathbf{x}) = \sum_{f_i \in \mathbf{F_X}} f_i(\mathbf{x}_i) \tag{5.3}$$

$$\mathcal{F}_y^t(\mathbf{x}) = \sum_{f_i \in \mathbf{F_Y}} \int_{\Omega_{y_i}} f_i(\mathbf{x}_i, y_i) \cdot p_{y_i}^t(y_i) dy_i \tag{5.4}$$

where $\mathbf{x}_i$ is an assignment for all the variables in the scope $\mathbf{x}^{f_i}$ of the function f_i; $p_{y_i}^t$ is the probability density function of the random variable y_i at time step t, and defined as:

$$p_{y_i}^t(y_i) = \int_{\Omega_{y_i}} p_{y_i}^{t-1}(y_i) \cdot T(y_i, \Omega_{y_i}) dy_i \tag{5.5}$$

[17]If multiple random variables are associated with a utility function, w.l.o.g., they can be merged into a single variable.

The second term $\mathbf{Q}$ considers the penalty due to changes in decision variables' values during the optimization process:

$$C_{\mathbf{x}}(\mathbf{x}^t, \mathbf{x}^{t+1}) = \sum_{x \in \mathbf{X}} c_x(x^t, x^{t+1}) \tag{5.6}$$

is a penalty function that takes into account the difference in the decision variable assignments between two time steps.

5.3 DC-DCOP Algorithms

We now introduce our DC-DCOP algorithms, which are built upon two sequential greedy Dynamic DCOP algorithms: Forward and Backward [36]. The two algorithms have been applied to solve the Dynamic DCOPs where each sub-problem is a *discrete* DCOP. However, in DC-DCOPs, the sub-problem at every time step is a *Continuous* DCOP. Thus the original version of Forward and Backward cannot be applied to solve DC-DCOPs. In this section, we propose a new version of the two algorithms that can address and solve the C-DCOP at every time step.

5.3.1 Forward

In general, Forward greedily solves each sub-problem in DC-DCOPs one time step at a time starting from the first time step. In other words, it successively solves the C-DCOP at each time step starting from $t = 0$ to $t = h$. When solving each C-DCOP, it takes into account the switching cost incurred by changing the solution from time step $t-1$ to the optimal solution at time step t. Specifically, before solving the C-DCOP at each time step, the agents run a pre-processing step, where they (1) reformulate the constraint between decision and random variables, and (2) capture the cost of switching values between time steps in new unary constraints of decision variables. For each constraint $f_i \in \mathbf{F}_Y$ between decision variables $\mathbf{x}_i$ and a random variable y_i, the following new constraint is created for each time step $0 \leq t \leq h$:

$$F_i^t(\mathbf{x}_i) = \int_{\Omega_{y_i}} f_i(\mathbf{x}_i, y_i) \cdot p_{y_i}^t(y_i) dy_i \tag{5.7}$$

where $p_{y_i}^t(\cdot)$ is the probability density function of random variable y_i at time step t.

After reformulating the constraints between decision variables and random variable, the agents create a new constraint to capture the cost of switching values across time steps. Specifically, for each decision variable $x \in \mathbf{X}$, the following new unary constraint is created for each time step $0 < t \leq h$:

$$C_x^t(x^t) = -c_x(x^{t-1}, x^t) \tag{5.8}$$

After adding the switching cost constraints, the agents successively solve each C-DCOP from time step $t = 0$ onwards using any off-the-shelf C-DCOP algorithm.

We use the following off-the-shelf C-DCOP algorithms to solve the problem at each time step: *AC-DPOP*, *CAC-DPOP*, *HCMS*, and *C-DSA* [38]. AC-DPOP, CAC-DPOP, and HCMS are *inference*-based algorithms, while C-DSA is a *local search* algorithm. AC-DPOP solves C-DCOPs by first discretizing the domains of the variables into initial discrete values and then using gradient methods to move the values of the parent and psedo-parent variables in order to better approximate the constraint utilities. CAC-DPOP is a variant of AC-DPOP that reduces the memory and time consumption of AC-DPOP by clustering the values of the agents before sending them up the pseudo-tree. Instead of using pseudo-tree, HCMS uses a factor graph to represent C-DCOPs and gradually adjusts agents' values over a number of iterations. Finally, C-DSA is a continuous *stochastic* algorithm, where each agent communicates their assignment with neighboring agents and stochastically determines to keep the current assignment or change to a better one.

5.3.2 Backward

Instead of solving the DC-DCOP one time step at a time forward starting from $t = 0$ towards h, one can also greedily solve the problem backwards from $t = h$ towards the first time step. Similar to Forward, before solving the C-DCOP at each time step, agents in Backward run a pre-processing step to reformulate the constraint between decision and random variables, and to capture the switching cost between two time steps. To reformulate the constraints between decision variables and a random variable, the agents calls Equation (5.7) and create a new constraints for each time steps $0 \leq t \leq h$. However, the key difference between Backward and Forward is how the agents compute the new switching cost constraint at each time step t. Specifically, when solving the C-DCOP at time step t, instead of taking into account the switching cost between time step t and time step $t-1$, agents in Backward takes into account the switching cost between time step t and time step $t+1$.

Specifically, before solving each sub-problem, Backward creates a unary constraint for each time step $0 \leq t < h$:

$$C_x^t(x^t) = -c_x(x^t, x^{t+1}) \tag{5.9}$$

After adding the switching cost constraints and the reformulated constraints between decision variables and a random variable, the agents successively solve each C-DCOP from time step $t = h$ backward using any off-the-shelf C-DCOP algorithm. Similar to Forward, we use AC-DPOP, CAC-DPOP, HCMS, and C-DSA to solve the C-DCOP at each time step. We will empirically evaluate both greedy versions of these C-DCOP algorithms in the experimental result section and will also discuss their theoretical properties in Section 5.4.

5.4 Theoretical Results

Error Bounds: We denote U^∞ as the optimal solution quality of a DC-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 C-DCOP where the decision variables are assigned $\mathbf{x}$ given values $\mathbf{y}$ of the random variables. We define $F^\Delta_{\mathbf{y}} = \max_{\mathbf{x}\in\Sigma} F_{\mathbf{y}}(\mathbf{x}) - \min_{\mathbf{x}\in\Sigma} F_{\mathbf{y}}(\mathbf{x})$ as the maximum loss in solution quality of a regular DCOP for a given random variable assignment $\mathbf{y}$ and $F^\Delta = \max_{\mathbf{y}\in\Sigma_{\mathbf{Y}}} F^\Delta_{\mathbf{y}}$ where $\Sigma_{\mathbf{Y}} = \prod_{y\in\mathbf{Y}} \Omega_y$ is the assignment space for all random variables.

Theorem 11 *When $\gamma < 1$, the error $U^\infty - U^h$ of the optimal solution from solving DC-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 DC-DCOPs with infinite horizon ∞:

$$U^\infty = \sum_{t=0}^{\infty} \gamma^t \left[\mathcal{F}^t_x(\hat{\mathbf{x}}^*_t) + \mathcal{F}^t_y(\hat{\mathbf{x}}^*_t) - C_{\mathbf{x}}(\hat{\mathbf{x}}^*_t, \hat{\mathbf{x}}^*_{t+1})\right] \tag{5.10}$$

Ignoring switching costs after time step h, an upper bound U^∞_+ of U^∞ is defined as:

$$U^\infty_+ = \sum_{t=0}^{h-1} \gamma^t \left[\mathcal{F}^t_x(\hat{\mathbf{x}}^*_t) + \mathcal{F}^t_y(\hat{\mathbf{x}}^*_t) - C_{\mathbf{x}}(\hat{\mathbf{x}}^*_t, \hat{\mathbf{x}}^*_{t+1})\right] \tag{5.11}$$

$$+ \sum_{t=h}^{\infty} \gamma^t \left[\mathcal{F}^t_x(\hat{\mathbf{x}}^*_t) + \mathcal{F}^t_y(\hat{\mathbf{x}}^*_t)\right] \tag{5.12}$$

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

$$U^h = \sum_{t=0}^{h-1} \gamma^t \left[\mathcal{F}_x^t(\mathbf{x}_t^*) + \mathcal{F}_y^t(\mathbf{x}_t^*) - C_{\mathbf{x}}(\mathbf{x}_t^*, \mathbf{x}_{t+1}^*) \right] \tag{5.13}$$

$$+ \sum_{t=h}^{\infty} \gamma^t \left[\mathcal{F}_x^t(\mathbf{x}_h^*) + \mathcal{F}_y^t(\mathbf{x}_h^*) \right] \tag{5.14}$$

For $\hat{\mathbf{x}}^*$, if we change the solution for every C-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 a lower bound U_-^∞ of U^h:

$$U_-^\infty = \sum_{t=0}^{h-1} \gamma^t \left[\mathcal{F}_x^t(\hat{\mathbf{x}}_t^*) + \mathcal{F}_y^t(\hat{\mathbf{x}}_t^*) - C_{\mathbf{x}}(\hat{\mathbf{x}}_t^*, \hat{\mathbf{x}}_{t+1}^*) \right] \tag{5.15}$$

$$+ \sum_{t=h}^{\infty} \gamma^t \left[\mathcal{F}_x^t(\hat{\mathbf{x}}_h^*) + \mathcal{F}_y^t(\hat{\mathbf{x}}_h^*) \right] \tag{5.16}$$

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

Next, we compute the difference between the two bounds:

$$U^\infty - U^h \leq U_+^\infty - U_-^\infty \tag{5.17}$$

$$= \sum_{t=h}^{\infty} \gamma^t \left[(\mathcal{F}_x^t(\hat{\mathbf{x}}_t^*) + \mathcal{F}_y^t(\hat{\mathbf{x}}_t^*)) - (\mathcal{F}_x^t(\hat{\mathbf{x}}_h^*) + \mathcal{F}_y^t(\hat{\mathbf{x}}_h^*)) \right] \tag{5.18}$$

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 C-DCOP:

$$(\mathcal{F}_x^t(\hat{\mathbf{x}}_t^*) + \mathcal{F}_y^t(\hat{\mathbf{x}}_t^*)) - (\mathcal{F}_x^t(\hat{\mathbf{x}}_h^*) + \mathcal{F}_y^t(\hat{\mathbf{x}}_h^*)) \leq F^\Delta \tag{5.19}$$

Thus,

$$\begin{aligned}
U^{\infty} - U^{h} &\leq U^{\infty}_{+} - U^{\infty}_{-} && (5.20)\\
&\leq \sum_{t=h}^{\infty} \gamma^{t} \left[\mathcal{F}^{t}_{x}(\hat{\mathbf{x}}^{*}_{t}) + \mathcal{F}^{t}_{y}(\hat{\mathbf{x}}^{*}_{t}) - \mathcal{F}^{t}_{x}(\hat{\mathbf{x}}^{*}_{h}) - \mathcal{F}^{t}_{y}(\hat{\mathbf{x}}^{*}_{h})\right] && (5.21)\\
&\leq \sum_{t=h}^{\infty} \gamma^{t} F^{\Delta} && (5.22)\\
&\leq \frac{\gamma^{h}}{1-\gamma} F^{\Delta} && (5.23)
\end{aligned}$$

which concludes the proof. □

Error Bounds from C-DCOP Algorithms: For each reward function $f(x_i, x_{i_1}, \ldots, x_{i_k})$ of an agent x_i and its separator agents $x_{i_1}, \ldots, x_{i_k}$, assume that agent x_i discretizes the domains of the reward function into hypercubes of size m (i.e., the distance between two neighboring discrete points for the same agent x_{i_j} is m). Let $\nabla f(v)$ denote the gradient of the function $f(x_i, x_{i_1}, \ldots, x_{i_k})$ at $v = (v_i, v_{i_1}, \ldots, v_{i_k})$:

$$\nabla f(v) = (\frac{\partial f}{\partial x_i}(v_i), \frac{\partial f}{\partial x_{i_1}}(v_{i_1}), \ldots, \frac{\partial f}{\partial x_{i_k}}(v_{i_k})) \qquad (5.24)$$

Furthermore, let $|\nabla f(v)|$ denote the sum of magnitude:

$$|\nabla f(v)| = |\frac{\partial f}{\partial x_i}(v_i)| + |\frac{\partial f}{\partial x_{i_1}}(v_{i_1})| + \ldots + |\frac{\partial f}{\partial x_{i_k}}(v_{i_k})| \qquad (5.25)$$

Assume that $|\nabla f(v)| \leq \delta$ holds for all utility functions in the DCOP and for all v.

Theorem **12** *The error of* AC-DPOP-*based algorithms is bounded above by* $h \cdot |\mathbf{F}|(m + |\mathbf{A}|k\alpha\delta)\delta + (h-1) \cdot \Theta|\mathbf{A}|$, *where k is the number of times each agent "moves" values of its separator, and* $\Theta = \max_{x \in \mathbf{X}} c_x(v, v')$ *is the maximum of the bounded switching cost functions.*

Proof: From Theorem 8, the error bound of solving a C-DCOP using AC-DPOP algorithm is $|\mathbf{F}|(m + |\mathbf{A}|k\alpha\delta)\delta$. In DC-DCOPs, there are h C-DCOPs, each is a sub-problem at every time step. Without taking into account the switching cost, the error bound of AC-DPOP-based algorithms (e.g., Forward-AC-DPOP and Backward-AC-DPOP) is $h \cdot |\mathbf{F}|(m+|\mathbf{A}|k\alpha\delta)\delta$. Given $\Theta = \max_{x,x'} c(x, x')$ as the maximum value of switching cost between two time steps, and there are at most $h - 1$ switching times between h time steps from $|\mathbf{A}|$ agents, the upper bound is then $h \cdot |\mathbf{F}|(m + |\mathbf{A}|k\alpha\delta)\delta + (h - 1) \cdot \Theta|\mathbf{A}|$. □

Theorem **13** *In a binary constraint graph $G = (\mathbf{X}, E)$, the number of messages of* HCMS-*based algorithms and* C-DSA-*based algorithms with k iterations is $h \cdot 4k|E|$ and $h \cdot 2k|E|$, respectively. The number of messages of* AC-DPOP-*based algorithms and* CAC-DPOP-*based algorithms is $h \cdot 2|\mathbf{X}|$.*

Proof: From Theorem 9, the number of messages of *HCMS*-based algorithms and *C-DSA*-based algorithms with k iterations is $4k|E|$ and $2k|E|$, respectively. The number of messages of *AC-DPOP*-based algorithms and *CAC-DPOP*-based algorithms is $2|\mathbf{X}|$. Since solving a DC-DCOP is equivalent to solving h C-DCOPs, each at a time step, the number of messages is as h times as the number of messages need to solve a single C-DCOP, which concludes the proof. □

5.5 Related Work

As discussed in Section 3.5, several approaches and algorithms have been proposed to solve related constraint models with discrete variables including centralized Dynamic CSPs [39, 80], Mixed CSPs [16], and Stochastic CSPs [73, 81]. Another Dynamic DCOP variant is Markovian D-DCOPs (MD-DCOPs) [59]. MD-DCOPs assume that the state space is discrete and observable to the agents, while DC-DCOPs do not assume the observability of the state. While these approaches have been proposed to solve the models with discrete state space, they have not been used to solve Dynamic DCOPs with continuous state space, to the best of our knowledge.

In Section 4.4, we discussed several C-DCOP algorithms such as Continuous Max-Sum [69], Hybrid Continuous Max-Sum [79], Particle Swarm Optimization Based Functional DCOP (PFD) [7], and Bayesian DPOP (B-DPOP) [23]. While those algorithms were proposed to solve C-DCOPs in a static setting, they have not been used to solve C-DCOPs in dynamic environments.

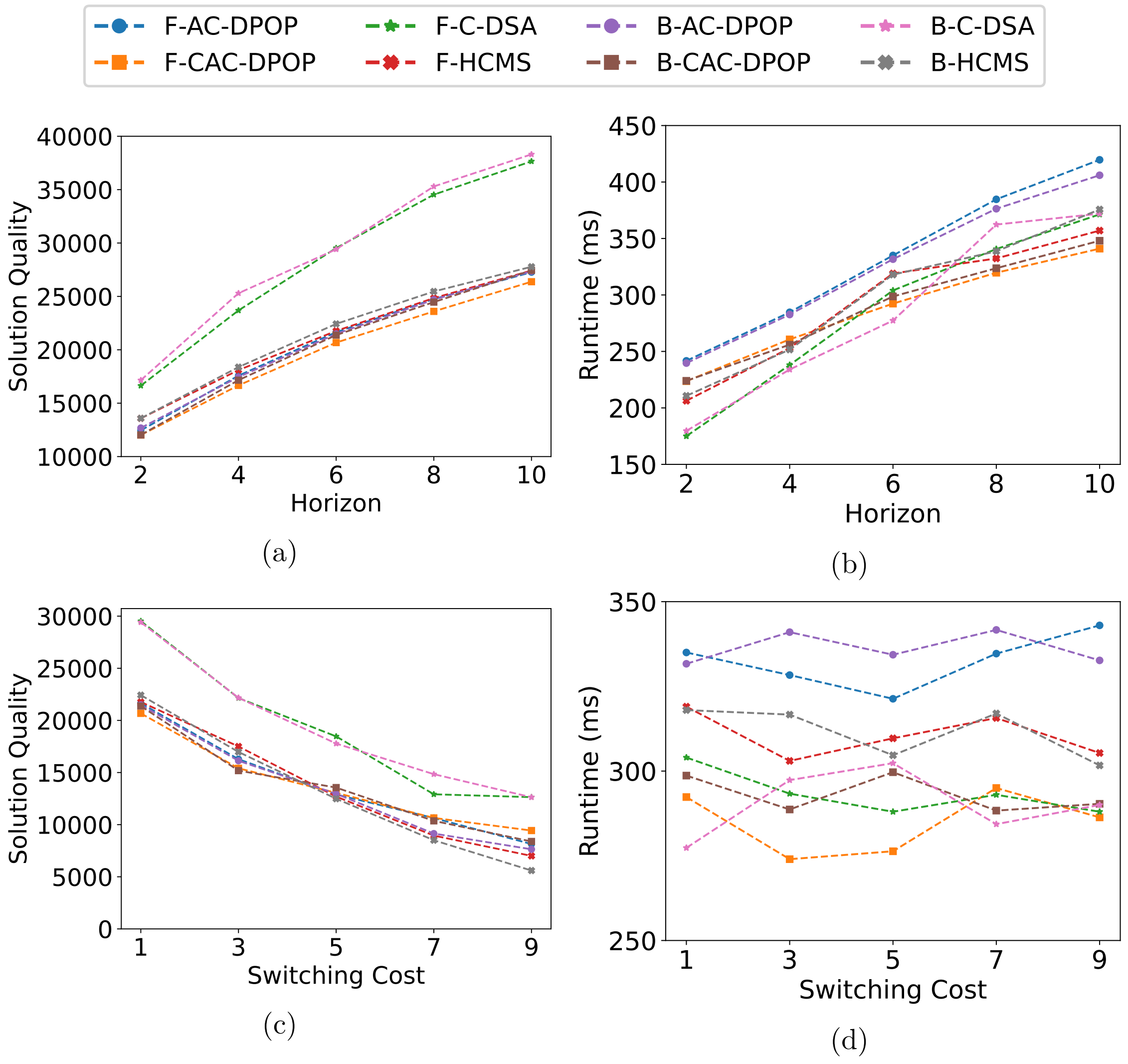


Figure 5.1: Experimental Results Varying Horizon and Switching Cost on Sparse Random Networks

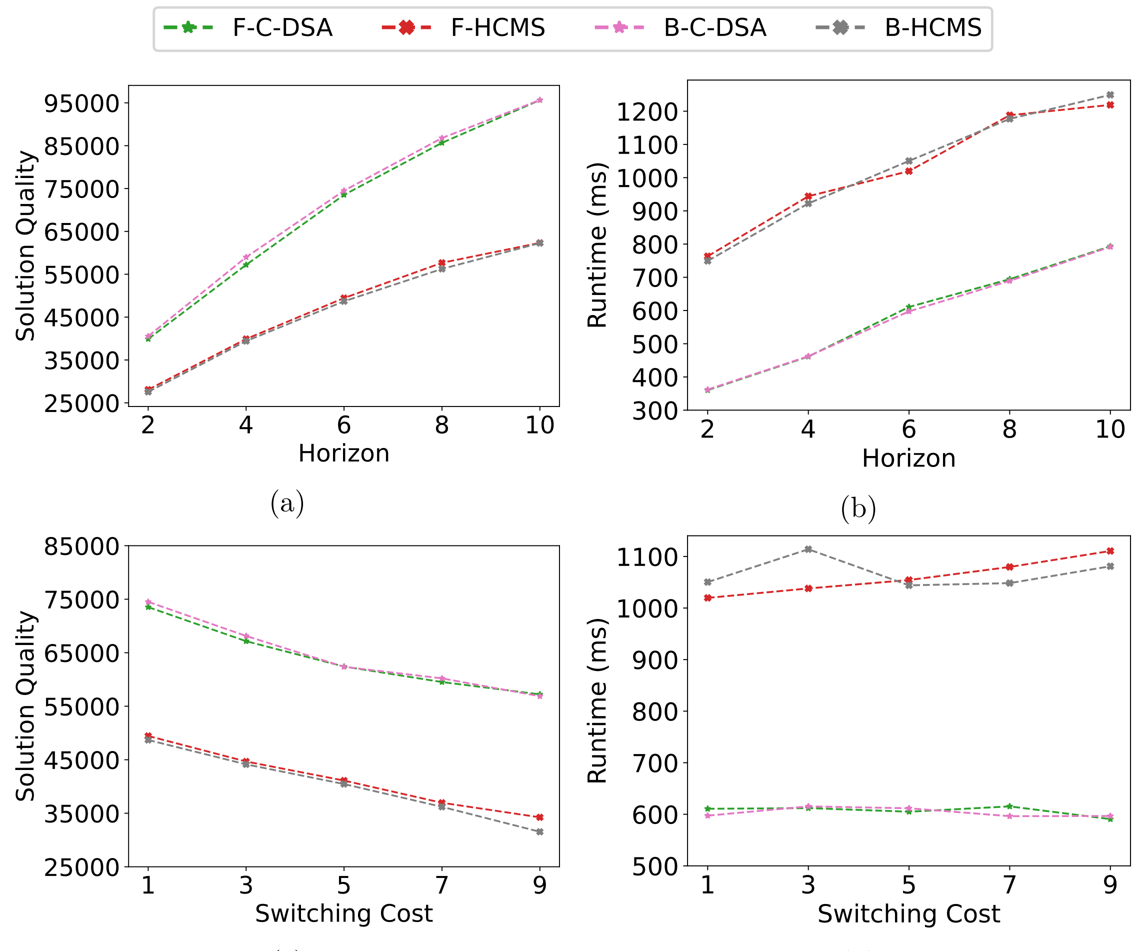


Figure 5.2: Experimental Results Varying Horizon and Switching Cost on Dense Random Networks

5.6 Experimental Results

We empirically evaluate the following DC-DCOP algorithms: *Forward*- and *Backward*-versions of AC-DPOP, CAC-DPOP, C-DSA, and HCMS on random networks and distributed sensor network problems. For those algorithms with *Forward*- version, we use the prefix F-, and for those algorithms with *Backward*- version, we use the prefix B-. Our experiments are performed on a 2.1GHz machine with 16GB of RAM using JADE framework [3]. We report solution quality and simulated runtime [70] averaged over 30 independent runs, each with a timeout of 30 minutes. We use the following default configuration: Number of agents and random variables $|\mathbf{A}| = |\mathbf{X}| = |\mathbf{Y}| = 12$; domains of decision and random variables $D_x = D_y = [-10, 10]$; discount factor $\gamma = 0.9$; horizon $h = 6$; switching cost function $c(x, x') = c \cdot (x - x')^2$ with the default cost $c = 1$. We set the number of discrete points to 3 for AC-DPOP-, CAC-DPOP-, and HCMS-based algorithms. For all algorithms, we set the number of iterations as 20.[18]

We first vary the horizon h to evaluate the performance of the algorithms with different horizon length. Figures 5.1(a) and 5.1(b) show the solution quality and runtime with horizon varying from 2 to 10 on sparse networks with $p_1 = 0.2$. When the horizon increases, both F-C-DSA and B-C-DSA produce the highest solution quality and outperform all other algorithms. The reason is that C-DSA-based algorithms do not depend on a number of initial discrete points and thus they are free to explore the search space. Interestingly, when the horizon becomes longer, their runtime is as small as other algorithms. This result shows that while C-DSA-based algorithms have the best solution quality, they do not come with the cost of higher runtime. Since AC-DPOP takes the longest time to solve each single

[18]For AC-DPOP- and CAC-DPOP-based algorithms, that is the number of iterations to move the values of parent and pseudo-parent variables. For HCMS- and C-DSA-based algorithms, it is the number of iterations to perform the local search.

$\lvert\mathbf{A}\rvert$	F-AC-DPOP		B-AC-DPOP		F-CAC-DPOP		B-CAC-DPOP		F-C-DSA		B-C-DSA		F-HCMS		B-HCMS	
	q	t	q	t	q	t	q	t	q	t	q	t	q	t	q	t
8	20789	285	20467	280	20778	284	20656	281	25824	280	26354	272	18668	285	18792	285
12	27269	420	27356	406	26378	341	27445	348	37653	371	38306	372	27423	357	27780	376
16	42473	111276	43327	119863	37708	874	39005	870	59390	542	59296	545	43511	611	44373	591
20	–	–	–	–	65345	3697	65075	3690	100725	690	102904	709	73269	751	74140	760
24	–	–	–	–	70058	49275	68965	43269	134213	774	134964	782	93427	958	94610	956
28	–	–	–	–	88135	493444	88156	524903	176429	884	175571	887	120822	1247	120917	1259
32	–	–	–	–	–	–	–	–	224743	1014	227258	1015	162918	1498	163627	1623

Table 5.1: Varying the Number of Agents on Sparse Random Networks with $p_1 = 0.2$

$\lvert\mathbf{A}\rvert$	F-CAC-DPOP		B-CAC-DPOP		F-C-DSA		B-C-DSA		F-HCMS		B-HCMS	
	q	t	q	t	q	t	q	t	q	t	q	t
8	30359	6604	31158	6229	47747	436	47594	418	33166	684	32958	674
12	–	–	–	–	95133	782	96722	791	62861	1289	62767	1273
16	–	–	–	–	160255	1273	160713	1289	109128	2222	110232	2227
20	–	–	–	–	217548	1603	215767	1601	148709	3243	149640	3284
24	–	–	–	–	281505	1842	283987	1836	196808	4508	196467	4453
28	–	–	–	–	375106	1972	374697	1986	255482	5565	252306	5643
32	–	–	–	–	466945	2138	463594	2121	313125	6891	318854	6860

Table 5.2: Varying the Number of Agents on Dense Random Networks with $p_1 = 0.7$

C-DCOP [38], F-AC-DPOP and B-AC-DPOP are the slowest algorithms across different horizon length. Similarly, on dense networks with $p_1 = 0.7$, Figures 5.2(a) and 5.2(b) show that both versions of C-DSA again outperform the HCMS-based algorithms in terms of solution quality with smaller runtime. We do not include AC-DPOP- and CAC-DPOP-based algorithms in Figure 5.2 since they time out on the dense networks.

Figures 5.1(c) and 5.1(d) show the result of varying the switching cost c in the switching cost function $c \cdot (x - x')^2$. The result shows that the solution quality of all algorithms decreases when the switching cost increases. If there is no switching cost (i.e., $c = 0$), the optimal solution of DC-DCOP consists of the optimal solution of the C-DCOP at each time step. However, with higher switching cost, the solution quality found by algorithms is likely to decreases due to the higher penalty incurred by different solutions across time steps. We also

$\lvert\mathbf{A}\rvert$	F-AC-DPOP		B-AC-DPOP		F-CAC-DPOP		B-CAC-DPOP		F-C-DSA		B-C-DSA		F-HCMS		B-HCMS	
	q	t	q	t	q	t	q	t	q	t	q	t	q	t	q	t
4	4750	108	5053	104	4750	171	5052	168	7302	131	7432	136	4344	127	4406	125
8	19166	275	21155	263	19287	296	19208	298	24958	252	25045	253	17415	282	17969	282
12	31809	387	31916	387	31116	411	30773	418	40014	392	40912	404	28357	373	28892	358
16	37842	507	38427	493	36007	450	36615	469	50069	459	51274	458	36454	412	36265	411
20	58987	696	60088	695	51418	570	51265	567	70953	513	71888	514	52402	444	52201	465

Table 5.3: Varying the Number of Agents for Distributed Radar Coordination and Scheduling Problems

observe that C-DSA-based algorithms have the best solution quality, which is consistent with the result on dense graph reported in Figures 5.2(c) and 5.2(d).

Finally, we vary the number of agents $|\mathbf{A}|$ (and thus the number of decision $|\mathbf{X}|$ and random variables $|\mathbf{Y}|$) of the problems from 8 to 32 with horizon $h = 10$. Table 5.1 tabulates the solution quality (denoted by q) and simulated runtime (denoted by t in ms) of the algorithms on sparse networks with $p_1 = 0.2$. Since AC-DPOP takes the longest time to solve the C-DCOP at each time step, both F-AC-DPOP and B-AC-DPOP can only solve small instances with 8, 12 and 16 agents and time out with larger instances. CAC-DPOP, which is the clustering version of AC-DPOP, reduces the memory used in the UTIL phrase and is more scalable to solve C-DCOPs with more number of agents [38]. Thus both F-CAC-DPOP and B-CAC-DPOP are able to solve instances with more number of agents than AC-DPOP-based algorithms and only time out with 32 agents. On the other hand, since C-DSA and HCMS are more scalable, it takes less time for them to solve each individual C-DCOP, and their DC-DCOP algorithms are able to solve all instances with much smaller runtime than AC-DPOP and CAC-DPOP. Interestingly, HCMS-based algorithms report a slightly larger runtime than C-DSA-based algorithms. Similar to the results from Figure 5.1, C-DSA-based algorithms report the highest solution quality than those from AC-DPOP-, CAC-DPOP-, and HCMS-based algorithms on different numbers of agents.

Table 5.2 shows the result varying agents on a dense random networks with $p_1 = 0.7$. Since both F-AC-DPOP and B-AC-DPOP time out with 8 agents, we do not include these algorithms in the table. While CAC-DPOP-based algorithms are able to solve the instances with 8 agents, they time out on larger instances. Both C-DSA- and HCMS- based algorithms are able to scale to solve larger instances with incremental runtime. While C-DSA-based algorithms outperform HCMS-based algorithms on all instances, it also takes them less time than the counterpart algorithms.

Distributed Radar Coordination and Scheduling Problems: We evaluate our DC-DCOP algorithms on Distributed Radar Coordination and Scheduling Problems (DRCSPs), which are introduced in Section 2.7. Table 5.3 shows the quality solution and runtime (in ms) of DC-DCOP algorithms on DRCSP with agents varying from 4 to 20. Both AC-DPOP- and CAC-DPOP-based algorithms run from smaller (4 radars) to larger instances (20 radars) without timeout and have slightly higher solution quality than HCMS-based algorithms. However, both versions of C-DSA outperform all other algorithms by providing the best solution quality from small to large instances. In addition, C-DSA-based algorithms have smaller runtime than AC-DPOP- and CAC-DPOP-based since C-DSA is a local search algorithm on C-DCOP. However, on grid network problems, HCMS-based algorithms execute faster than C-DSA-based algorithms.

5.7 Discussions and Conclusions

In many real-world applications, agents often act in a complex and dynamic environment. While DCOPs have been widely used to solve several multi-agent problems, the formulation lacks the capability to model the dynamic and continuous nature in complex environments. While researchers have proposed D-DCOPs to model how the environment changes over time and C-DCOPs to model the continuous domain of decision variables, they can only address these limitations of DCOPs in isolation. Thus, it remains a challenge to model more complex problems with continuous variables in a dynamic environment. Therefore, in this chapter, we introduced Dynamic Continuous DCOPs (DC-DCOPs), which *both* model the dynamic environment and model decision variables with continuous domain. To solve DC-DCOPs, we proposed several sequential greedy algorithms that can use any off-the-shelf C-DCOP algorithms to solve DC-DCOPs and we discussed their theoretical properties. Finally, we evaluated our algorithms on random networks and on distributed sensor network problems, which are our motivating application for this line of work.

To apply DC-DCOPs to solve real-world applications, it remains a challenge to model the initial probability distribution and the transition function of the random variables in the problem. Unlike PD-DCOPs, where random variables have discrete domain, the random variables in DC-DCOPs have continuous domain, and that could make the transition function complicated in some cases. For example, in our motivating DRCSP application, how complicated the transition function is depends on the type of weather phenomena and on the data about the phenomena in the past. Since DC-DCOPs take into account the prior information on how the problem might change, the reliability of the prior information will affect the solution quality of the DC-DCOP model.

Chapter 6

Conclusions and Future Work

In many real-world applications, agents often act in a dynamic and complex environment. While DCOPs have been a powerful tool to model many multi-agent systems, the formulation lacks the capability to model and solve the problems in such environments. Over the years, researchers have introduced several DCOP variants, but they were proposed in isolation and haven't fully solved the problems in environments that are both dynamic and complex. To address the above concerns, this dissertation makes the following four contributions:

- In Chapter 3, we proposed PD-DCOPs, which explicitly model how DCOPs change in dynamic environments with prior information. We developed an exact algorithm to solve PD-DCOPs and several heuristic algorithms that can scale to larger and more complex problems. 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.

By introducing the PD-DCOP model and algorithms, we made a significant contribution to several DCOP applications in dynamic environment, especially for those where the prior information on how the problem might change is available or predictable. Our experimental results showed that for those applications that have high switching cost or change quickly over time, proactive algorithms delivers better solution than the reactive counterpart.

- In Chapter 4, we proposed several algorithms to solve C-DCOPs, which model DCOPs in complex environments where agents take values from a continuous domain. Existing C-DCOP methods suffer from the limitation that they do not provide quality guarantees. We remedied this limitation by introducing *(i)* EC-DPOP, which finds exact solutions for C-DCOPs with linear or quadratic utility functions and with tree-structure graphs; *(ii)* AC-DPOP, which finds error-bounded solutions for general C-DCOPs; *(iii)* CAC-DPOP, which limits the message size of AC-DPOP to a user-defined parameter k; and *(iv)* C-DSA, which is a scalable local search C-DCOP algorithm. Experimental results showed that our algorithms find better solutions than HCMS, an existing state-of-the-art algorithm, when given the same communication limitations. Moreover, these algorithms combined extend the applicability of DCOPs to more applications that require quality guarantees on the solutions found as well as those that require limited communication capabilities.

  By introducing several C-DCOP algorithms, we advanced the state of the art and contributed different options for solving many real-world C-DCOP applications. For those problems that have constraint graph with tree structure, our EC-DPOP algorithm guarantees an optimal solution for binary linear or quadratic constraint functions.

When a guarantee on solution quality is required, our AC-DPOP provides an error bound for their approximate solution. For those applications that need a solution in a timely manner, one can either use CAC-DPOP with tradeoff on the theoretical bound or use C-DSA when a solution is needed at any time.

- In Chapter 5, we introduced DC-DCOP, a formulation that models the problems in a dynamic and complex environment. DC-DCOPs model how the problem changes dynamically over time by taking into account the prior information and model the variables with continuous domain. To solve DC-DCOPs, we proposed several sequential greedy algorithms that can use any off-the-shelf C-DCOP algorithms to solve DC-DCOPs and we discussed their theoretical properties. Finally, we evaluated our algorithms on random networks and on distributed radar coordination and scheduling network problems, which are our motivating application for this line of work.

  By introducing DC-DCOPs, we made the contribution to solve many applications which are in dynamic environment and require continuous variables. However, it remains a challenge to model the initial probability distribution and the transition function of the continuous random variables, which depend significantly on the application and on the data of the event in the past.

This dissertation demonstrates that one can model and solve multi-agent problems in a dynamic and complex environment. There are several directions that can extend the current work. First, D-DCOPs have been only solved by either reactive approach or proactive approach. However, combining the two approaches together might result in higher solution quality, especially in those problems 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. Secondly, while DC-DCOPs approach and model C-DCOPs in dynamic environments in a proactive manner, a reactive approach might provide better solution in some cases. For example, in the environment that changes quickly or the cost of changing solution is small, a reactive approach might provide better solution than a proactive approach. For those problems where the prior information is not available, solving the problem beforehand is not suitable and thus, a reactive approach might be preferred. Finally, after reactive approaches have been introduced to solve C-DCOPs in dynamic environment, we can combine both reactive and proactive approaches by leveraging the initial solution provided by the proactive approach and quickly searching for a solution in a reactive manner for those problems where continuous domains are required.

References

[1] Manuel Blanco Abello, Zbignew Michalewicz, and Lam Thu Bui. “A Reactive-Proactive Approach for Solving Dynamic Scheduling with Time-Varying Number of Tasks.” In: *2012 IEEE Congress on Evolutionary Computation*. IEEE. 2012, pp. 1–10.

[2] Raphen Becker, Shlomo Zilberstein, Victor Lesser, and Claudia Goldman. “Solving Transition Independent Decentralized Markov Decision Processes.” In: *Journal of Artificial Intelligence Research* 22 (2004), pp. 423–455.

[3] Fabio Bellifemine, Federico Bergenti, Giovanni Caire, and Agostino Poggi. “JADE–A Java Agent Development Framework.” In: *Multi-Agent Programming*. 2005, pp. 125–147.

[4] Daniel Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. “The Complexity of Decentralized Control of Markov Decision Processes.” In: *Mathematics of Operations Research* 27.4 (2002), pp. 819–840.

[5] Ziyu Chen, Yanchen Deng, and Tengfei Wu. “An Iterative Refined Max-Sum_AD Algorithm via Single-Side Value Propagation and Local Search.” In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2017, pp. 195–202.

[6] Ziyu Chen, Tengfei Wu, Yanchen Deng, and Cheng Zhang. “An Ant-Based Algorithm to Solve Distributed Constraint Optimization Problems.” In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 2018, pp. 4654–4661.

[7] Moumita Choudhury, Saaduddin Mahmud, and Md. Mosaddek Khan. “A Particle Swarm Based Algorithm for Functional Distributed Constraint Optimization Problems.” In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 2020, pp. 7111–7118.

[8] Liel Cohen, Rotem Galiki, and Roie Zivan. “Governing Convergence of Max-Sum on DCOPs through Damping and Splitting.” In: *Artificial Intelligence* 279 (2020), p. 103212.

[9] Liel Cohen and Roie Zivan. “Balancing Asymmetry in Max-Sum Using Split Constraint Factor Graphs.” In: *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP)*. 2018, pp. 669–687.

[10] Francesco Delle Fave, Alex Rogers, Zhe Xu, Salah Sukkarieh, and Nicholas Jennings. "Deploying the Max-Sum Algorithm for Decentralised Coordination and Task Allocation of Unmanned Aerial Vehicles for Live Aerial Imagery Collection." In: *Proceedings of the International Conference on Robotics and Automation (IEEE ICRA)*. 2012, pp. 469–476.

[11] Yanchen Deng and Bo An. "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)*. 2020, pp. 31–38.

[12] Yanchen Deng, Ziyu Chen, Dingding Chen, Xingqiong Jiang, and Qiang Li. "PT-ISABB A Hybrid Tree-Based Complete Algorithm to Solve Asymmetric Distributed Constraint Optimization Problems." In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2019, pp. 1506–1514.

[13] Yanchen Deng, Ziyu Chen, Dingding Chen, Wenxin Zhang, and Xingqiong Jiang. "AsymDPOP Complete Inference for Asymmetric Distributed Constraint Optimization Problems." In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 2019, pp. 223–230.

[14] Jilles Steeve Dibangoye, Christopher Amato, and Arnaud Doniec. "Scaling up Decentralized MDPs through Heuristic Search." In: *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*. 2012, pp. 217–226.

[15] Jilles Steeve Dibangoye, Christopher Amato, Arnaud Doniec, and François Charpillet. "Producing Efficient Error-Bounded Solutions for Transition Independent Decentralized MDPs." In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2013, pp. 539–546.

[16] Hélène Fargier, Jérôme Lang, and Thomas Schiex. "Mixed Constraint Satisfaction: A Framework for Decision Problems under Incomplete Knowledge." In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 1996, pp. 175–180.

[17] Alessandro Farinelli, Alex Rogers, and Nicholas Jennings. "Agent-Based Decentralised Coordination for Sensor Networks Using the Max-Sum Algorithm." In: *Journal of Autonomous Agents and Multi-Agent Systems* 28.3 (2014), pp. 337–380.

[18] Alessandro Farinelli, Alex Rogers, Adrian Petcu, and Nicholas Jennings. "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)*. 2008, pp. 639–646.

[19] Ferdinando Fioretto, Enrico Pontelli, and William Yeoh. "Distributed Constraint Optimization Problems and Applications: A Survey." In: *Journal of Artificial Intelligence Research* 61 (2018), pp. 623–698.

[20] Ferdinando Fioretto, Enrico Pontelli, William Yeoh, and Rina Dechter. "Accelerating Exact and Approximate Inference for (Distributed) Discrete Optimization with GPUs." In: *Constraints* 23.1 (2018), pp. 1–43.

[21] Ferdinando Fioretto, William Yeoh, and Enrico Pontelli. “A Multiagent System Approach to Scheduling Devices in Smart Homes.” In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2017, pp. 981–989.

[22] Ferdinando Fioretto, William Yeoh, Enrico Pontelli, Ye Ma, and Satishkumar Ranade. “A DCOP Approach to the Economic Dispatch with Demand Response.” In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2017, pp. 999–1007.

[23] Jeroen Fransman, Joris Sijs, Henry Dol, Erik Theunissen, and Bart De Schutter. “Bayesian-DPOP for Continuous Distributed Constraint Optimization Problems.” In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2019, pp. 1961–1963.

[24] Haobo Fu, Peter R Lewis, Bernhard Sendhoff, Ke Tang, and Xin Yao. “What Are Dynamic Optimization Problems?” In: *2014 Congress on Evolutionary Computation*. IEEE. 2014, pp. 1550–1557.

[25] Robert Gallager. *Stochastic Processes: Theory for Applications*. Cambridge University Press, 2013.

[26] Amir Gershman, Amnon Meisels, and Roie Zivan. “Asynchronous Forward-Bounding for Distributed COPs.” In: *Journal of Artificial Intelligence Research* 34 (2009), pp. 61–88.

[27] Tal Grinshpoun and Tamir Tassa. “P-SyncBB: A Privacy Preserving Branch and Bound DCOP Algorithm.” In: *Journal of Artificial Intelligence Research* 57 (2016), pp. 621–660.

[28] Tal Grinshpoun, Tamir Tassa, Vadim Levit, and Roie Zivan. “Privacy Preserving Region Optimal Algorithms for Symmetric and Asymmetric DCOPs.” In: *Artificial Intelligence* 266 (2019), pp. 27–50.

[29] Patricia Gutierrez, Pedro Meseguer, and William Yeoh. “Generalizing ADOPT and BnB-ADOPT.” In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 2011, pp. 554–559.

[30] Youssef Hamadi, Christian Bessière, and Joël Quinqueton. “Distributed Intelligent Backtracking.” In: *Proceedings of the European Conference on Artificial Intelligence (ECAI)*. 1998, pp. 219–223.

[31] Eric A. Hansen, Daniel S. Bernstein, and Shlomo Zilberstein. “Dynamic Programming for Partially Observable Stochastic Games.” In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 2004, pp. 709–715.

[32] Daisuke Hatano and Katsutoshi Hirayama. “DeQED: An Efficient Divide-and-Coordinate Algorithm for DCOP.” In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 2013, pp. 566–572.

[33] Khoi D. Hoang, Ferdinando Fioretto, Ping Hou, William Yeoh, Makoto Yokoo, and Roie Zivan. “Proactive Dynamic Distributed Constraint Optimization Problems.” In: *Journal of Artificial Intelligence Research* 74 (2022), pp. 179–225.

[34] Khoi D. Hoang, Ferdinando Fioretto, Ping Hou, Makoto Yokoo, William Yeoh, and Roie Zivan. "Proactive Dynamic Distributed Constraint Optimization." In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS).* 2016, pp. 597–605.

[35] Khoi D. Hoang, Ferdinando Fioretto, William Yeoh, Enrico Pontelli, and Roie Zivan. "A Large Neighboring Search Schema for Multi-Agent Optimization." In: *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP).* 2018, pp. 688–706.

[36] Khoi D. Hoang, Ping Hou, Ferdinando Fioretto, William Yeoh, Roie Zivan, and Makoto Yokoo. "Infinite-Horizon Proactive Dynamic DCOPs." In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS).* 2017, pp. 212–220.

[37] Khoi D. Hoang, Christabel Wayllace, William Yeoh, Jacob Beal, Soura Dasgupta, Yuanqiu Mo, Aaron Paulos, and Jon Schewe. "New Distributed Constraint Reasoning Algorithms for Load Balancing in Edge Computing." In: *Proceedings of the Principles and Practice of Multi-Agent Systems (PRIMA).* 2019, pp. 69–86.

[38] Khoi D. Hoang, William Yeoh, Makoto Yokoo, and Zinovi Rabinovich. "New Algorithms for Continuous Distributed Constraint Optimization Problems." In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS).* 2020, pp. 502–510.

[39] Alan Holland and Barry O'Sullivan. "Weighted Super Solutions for Constraint Programs." In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI).* 2005, pp. 378–383.

[40] Yaochu Jin and Jürgen Branke. "Evolutionary Optimization in Uncertain Environments - A Survey." In: *IEEE Transactions on Evolutionary Computation* 9.3 (2005), pp. 303–317.

[41] Richard W Katz. "An Application of Chain-Dependent Processes to Meteorology." In: *Journal of Applied Probability* 14.3 (1977), pp. 598–603.

[42] Md. Mosaddek Khan, Long Tran-Thanh, and Nicholas R. Jennings. "A Generic Domain Pruning Technique for GDL-Based DCOP Algorithms in Cooperative Multi-Agent Systems." In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS).* 2018, pp. 1595–1603.

[43] Md. Mosaddek Khan, Long Tran-Thanh, William Yeoh, and Nicholas R. Jennings. "A Near-Optimal Node-to-Agent Mapping Heuristic for GDL-Based DCOP Algorithms in Multi-Agent Systems." In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS).* 2018, pp. 1613–1621.

[44] Yoonheui Kim, Michael Krainin, and Victor Lesser. "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).* Vol. 2. 2011, pp. 357–364.

[45] Akshat Kumar, Boi Faltings, and Adrian Petcu. "Distributed Constraint Optimization with Structured Resource Constraints." In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS).* 2009, pp. 923–930.

[46] Akshat Kumar, Adrian Petcu, and Boi Faltings. "H-DPOP: Using Hard Constraints for Search Space Pruning in DCOP." In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI).* 2008, pp. 325–330.

[47] Robert Lass, Evan Sultanik, and William Regli. "Dynamic Distributed Constraint Reasoning." In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI).* 2008, pp. 1466–1469.

[48] Tiep Le, Tran Cao Son, Enrico Pontelli, and William Yeoh. "Solving Distributed Constraint Optimization Problems with Logic Programming." In: *Theory and Practice of Logic Programming* 17.4 (2017), pp. 634–683.

[49] Thomas Léauté and Boi Faltings. "Coordinating Logistics Operations with Privacy Guarantees." In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI).* 2011, pp. 2482–2487.

[50] Cornelis Jan van Leeuwen and Przemyslaw Pawelczak. "CoCoA: A Non-Iterative Approach to a Local Search (A)DCOP Solver." In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI).* 2017, pp. 3944–3950.

[51] James MacQueen. "Some Methods for Classification and Analysis of Multivariate Observations." In: *Proceedings of the Berkeley Symposium on Mathematical Statistics and Probability.* 1967, pp. 281–297.

[52] Rajiv Maheswaran, Jonathan Pearce, and Milind Tambe. "Distributed Algorithms for DCOP: A Graphical Game-Based Approach." In: *Proceedings of the Conference on Parallel and Distributed Computing Systems (PDCS).* 2004, pp. 432–439.

[53] Rajiv Maheswaran, Milind Tambe, Emma Bowring, Jonathan Pearce, and Pradeep Varakantham. "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).* 2004, pp. 310–317.

[54] Sam Miller, Sarvapali Ramchurn, and Alex Rogers. "Optimal Decentralised Dispatch of Embedded Generation in the Smart Grid." In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS).* 2012, pp. 281–288.

[55] Pragnesh Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. "ADOPT: Asynchronous Distributed Constraint Optimization with Quality Guarantees." In: *Artificial Intelligence* 161.1–2 (2005), pp. 149–180.

[56] James T Moore, Fred H Glass, Charles E Graves, Scott M Rochette, and Marc J Singer. "The Environment of Warm-Season Elevated Thunderstorms Associated with Heavy Rainfall over the Central United States." In: *Weather and Forecasting* 18.5 (2003), pp. 861–878.

[57] Ranjit Nair, Milind Tambe, Makoto Yokoo, David Pynadath, and Stacy Marsella. "Taming Decentralized POMDPs: Towards Efficient Policy Computation for Multiagent Settings." In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 2003, pp. 705–711.

[58] Ranjit Nair, Pradeep Varakantham, Milind Tambe, and Makoto Yokoo. "Networked Distributed POMDPs: A Synthesis of Distributed Constraint Optimization and POMDPs." In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 2005, pp. 133–139.

[59] Duc Thien Nguyen, William Yeoh, Hoong Chuin Lau, Shlomo Zilberstein, and Chongjie Zhang. "Decentralized Multi-Agent Reinforcement Learning in Average-Reward Dynamic DCOPs." In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 2014, pp. 1447–1455.

[60] Duc Thien Nguyen, William Yeoh, Hoong Chuin Lau, and Roie Zivan. "Distributed Gibbs: A Linear-Space Sampling-Based DCOP Algorithm." In: *Journal of Artificial Intelligence Research* 64 (2019), pp. 705–748.

[61] Frans Oliehoek, Matthijs Spaan, Christopher Amato, and Shimon Whiteson. "Incremental Clustering and Expansion for Faster Optimal Planning in Dec-POMDPs." In: *Journal of Artificial Intelligence Research* 46 (2013), pp. 449–509.

[62] Brammert Ottens, Christos Dimitrakakis, and Boi Faltings. "DUCT: An Upper Confidence Bound Approach to Distributed Constraint Optimization Problems." In: *ACM Transactions on Intelligent Systems and Technology* 8.5 (2017), 69:1–69:27.

[63] Adrian Petcu and Boi Faltings. "A Scalable Method for Multiagent Constraint Optimization." In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 2005, pp. 1413–1420.

[64] Adrian Petcu and Boi Faltings. "Optimal Solution Stability in Dynamic, Distributed Constraint Optimization." In: *Proceedings of the International Conference on Intelligent Agent Technology (IAT)*. 2007, pp. 321–327.

[65] Adrian Petcu and Boi Faltings. "Superstabilizing, Fault-Containing Multiagent Combinatorial Optimization." In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 2005, pp. 449–454.

[66] Clarence W Richardson. "Stochastic Simulation of Daily Precipitation, Temperature, and Solar Radiation." In: *Water Resources Research* 17.1 (1981), pp. 182–190.

[67] Pierre Rust, Gauthier Picard, and Fano Ramparany. "Using Message-Passing DCOP Algorithms to Solve Energy-Efficient Smart Environment Configuration Problems." In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 2016, pp. 468–474.

[68] Sven Seuken and Shlomo Zilberstein. "Memory-Bounded Dynamic Programming for DEC-POMDPs." In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 2007, pp. 2009–2015.

[69] Ruben Stranders, Alessandro Farinelli, Alex Rogers, and Nicholas Jennings. “Decentralised Coordination of Continuously Valued Control Parameters Using the Max-Sum Algorithm.” In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2009, pp. 601–608.

[70] Evan Sultanik, Robert Lass, and William Regli. “DCOPolis: A Framework for Simulating and Deploying Distributed Constraint Reasoning Algorithms.” In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2008, pp. 1667–1668.

[71] Evan Sultanik, Robert Lass, and William Regli. “Dynamic Configuration of Agent Organizations.” In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 2009, pp. 305–311.

[72] Daniel Szer, François Charpillet, and Shlomo Zilberstein. “MAA*: A Heuristic Search Algorithm for Solving Decentralized POMDPs.” In: *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*. 2005, pp. 576–590.

[73] S Armagan Tarim, Suresh Manandhar, and Toby Walsh. “Stochastic Constraint Programming: A Scenario-Based Approach.” In: *Constraints* 11.1 (2006), pp. 53–80.

[74] Tamir Tassa, Tal Grinshpoun, and Avishay Yanai. “A Privacy Preserving Collusion Secure DCOP Algorithm.” In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 2019, pp. 4774–4780.

[75] Tamir Tassa, Tal Grinshpoun, and Roie Zivan. “Privacy Preserving Implementation of the Max-Sum Algorithm and Its Variants.” In: *Journal of Artificial Intelligence Research* 59 (2017), pp. 311–349.

[76] Kevin Trenberth. “Changes in Precipitation with Climate Change.” In: *Climate Research* 47.1-2 (2011), pp. 123–138.

[77] Suguru Ueda, Atsushi Iwasaki, and Makoto Yokoo. “Coalition Structure Generation Based on Distributed Constraint Optimization.” In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 2010, pp. 197–203.

[78] Meritxell Vinyals, Juan Rodríguez-Aguilar, and Jesús Cerquides. “Constructing a Unifying Theory of Dynamic Programming DCOP Algorithms via the Generalized Distributive Law.” In: *Journal of Autonomous Agents and Multi-Agent Systems* 22.3 (2011), pp. 439–464.

[79] Thomas Voice, Ruben Stranders, Alex Rogers, and Nicholas Jennings. “A Hybrid Continuous Max-Sum Algorithm for Decentralised Coordination.” In: *Proceedings of the European Conference on Artificial Intelligence (ECAI)*. 2010, pp. 61–66.

[80] Richard Wallace and Eugene Freuder. “Stable Solutions for Dynamic Constraint Satisfaction Problems.” In: *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP)*. 1998, pp. 447–461.

[81] Toby Walsh. “Stochastic Constraint Programming.” In: *Proceedings of the European Conference on Artificial Intelligence (ECAI)*. 2002, pp. 111–115.

[82] Daniel S Wilks. “Adapting Stochastic Weather Generation Algorithms for Climate Change Studies.” In: *Climatic Change* 22.1 (1992), pp. 67–84.

[83] Stefan Witwicki and Edmund Durfee. “Towards a Unifying Characterization for Quantifying Weak Coupling in Dec-POMDPs.” In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS).* 2011, pp. 29–36.

[84] Weixin Xu, Edward J Zipser, Yi-Leng Chen, Chuntao Liu, Yu-Chieng Liou, Wen-Chau Lee, and Ben Jong-Dao Jou. “An Orography-Associated Extreme Rainfall Event During TiMREX: Initiation, Storm Evolution, and Maintenance.” In: *Monthly Weather Review* 140.8 (2012), pp. 2555–2574.

[85] William Yeoh, Ariel Felner, and Sven Koenig. “BnB-ADOPT: An Asynchronous Branch-and-Bound DCOP Algorithm.” In: *Journal of Artificial Intelligence Research* 38 (2010), pp. 85–133.

[86] William Yeoh, Pradeep Varakantham, Xiaoxun Sun, and Sven Koenig. “Incremental DCOP Search Algorithms for Solving Dynamic DCOPs.” In: *Proceedings of the International Conference on Intelligent Agent Technology (IAT).* 2015, pp. 257–264.

[87] William Yeoh and Makoto Yokoo. “Distributed Problem Solving.” In: *AI Magazine* 33.3 (2012), pp. 53–65.

[88] Zhepeng Yu, Ziyu Chen, Jingyuan He, and Yancheng Deng. “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).* 2017, pp. 187–194.

[89] Weixiong Zhang, Guandong Wang, Zhao Xing, and Lars Wittenberg. “Distributed Stochastic Search and Distributed Breakout: Properties, Comparison and Applications to Constraint Optimization Problems in Sensor Networks.” In: *Artificial Intelligence* 161.1–2 (2005), pp. 55–87.

[90] Michael Zink, David Westbrook, Sherief Abdallah, Bryan Horling, Vijay Lakamraju, Eric Lyons, Victoria Manfredi, Jim Kurose, and Kurt Hondl. “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, 2005.

[91] Roie Zivan, Steven Okamoto, and Hilla Peled. “Explorative Anytime Local Search for Distributed Constraint Optimization.” In: *Artificial Intelligence* 212 (2014), pp. 1–26.

[92] Roie Zivan, Tomer Parash, Liel Cohen, Hilla Peled, and Steven Okamoto. “Balancing Exploration and Exploitation in Incomplete Min/Max-Sum Inference for Distributed Constraint Optimization.” In: *Journal of Autonomous Agents and Multi-Agent Systems* 31.5 (2017), pp. 1165–1207.

[93] Roie Zivan, Tomer Parash, Liel Cohen-Lavi, and Yarden Naveh. “Applying Max-Sum to Asymmetric Distributed Constraint Optimization Problems.” In: *Journal of Autonomous Agents and Multi-Agent Systems* 34.1 (2020), pp. 1–29.

[94] Roie Zivan and Hilla Peled. “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)*. 2012, pp. 265–272.

[95] Roie Zivan, Harel Yedidsion, Steven Okamoto, Robin Glinton, and Katia Sycara. “Distributed Constraint Optimization for Teams of Mobile Sensing Agents.” In: *Journal of Autonomous Agents and Multi-Agent Systems* 29.3 (2015), pp. 495–536.