Proactive Dynamic Distributed Constraint Optimization*

Khoi D. Hoang†, Ferdinando Fioretto†, Ping Hou†, Makoto Yokoo⋆, William Yeoh†, Roie Zivan◇
†Department of Computer Science, New Mexico State University, USA
{khoang, ffiorett, phou, wyeoh}@cs.nmsu.edu
⋆Department of Informatics, Kyushu University, Japan
yokoo@inf.kyushu-u.ac.jp
◇Department of Industrial Engineering and Management, Ben Gurion University of the Negev, Israel
zivanr@cs.bgu.ac.il

ABSTRACT

Current approaches that model dynamism in DCOPs solve a sequence of static problems, *reacting* to changes in the environment as the agents observe them. Such approaches thus ignore possible predictions on future changes. To overcome this limitation, we introduce *Proactive Dynamic DCOPs* (*PD-DCOPs*), a novel formalism to model dynamic DCOPs in the presence of exogenous uncertainty. In contrast to reactive approaches, PD-DCOPs are able to explicitly model the possible changes to the problem, and take such information into account *proactively*, when solving the dynamically changing problem. The additional expressivity of this formalism allows it to model a wider variety of distributed optimization problems. Our work presents both theoretical and practical contributions that advance current dynamic DCOP models: (*i*) we introduce the PD-DCOP model, which explicitly captures dynamic changes of the DCOP over time; (*ii*) we discuss the complexity of this new class of DCOPs; and (*iii*) we develop both exact and approximation algorithms with quality guarantees to solve PD-DCOPs *proactively*.

Keywords

Distributed Constraint Optimization; DCOP; Dynamic DCOP

1. INTRODUCTION

Distributed Constraint Optimization Problems (*DCOPs*) are problems where agents need to coordinate their value assignments to maximize the sum of the resulting constraint utilities [17, 34]. DCOPs have emerged as one of the prominent multi-agent architectures to govern the agents' autonomous behavior in distributed optimization problems. The model represents a powerful approach to the description and solution of many practical problems, serving several applications such as distributed scheduling, coordination of unmanned air vehicles, smart grid electricity networks, and sensor networks [15, 11, 28, 32, 13, 16, 3, 7, 36]. In many distributed problems of interest, agents interact in complex, uncertain, and dynamic environments. For example, in distributed meeting scheduling, participants could change their preferences and priorities over time. In disaster management, new information (e.g., weather forecasts, priorities on buildings to evacuate) typically becomes available in an incremental manner. Thus, the information flow modifies the environment over time. Unfortunately, the classical DCOP paradigm is unable to model problems that change over time.

Consequently, researchers have introduced *Dynamic DCOPs* (*D-DCOPs*) [23, 24, 12, 33], where utility functions can change during the problem solving process. These models make the 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. We provide one such example, a distributed meeting scheduling problem, as our motivating example in Section 3.

Therefore, in this paper, (*i*) we introduce *Proactive Dynamic DCOPs* (*PD-DCOPs*), which explicitly model how the DCOP will change over time; (*ii*) we discuss the complexity of this new class of DCOPs; and (*iii*) we develop exact and approximation algorithms with quality guarantees to solve PD-DCOPs *proactively*.

*The team from NMSU is partially supported by NSF grants 1345232 and 1540168. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies, or the U.S. government. Makoto Yokoo is partially supported by JSPS KAKENHI Grant Number 24220003.

Appears in: *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016), J. Thangarajah, K. Tuyls, C. Jonker, S. Marsella (eds.), May 9–13, 2016, Singapore.*

2. BACKGROUND

We now provide background on the regular and dynamic DCOPs as well as the regular and super-stabilizing DPOP algorithms.

2.1 DCOPs

A *Distributed Constraint Optimization Problem* (*DCOP*) 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* and each variable $x \in \mathbf{X}$ takes values from the set $D_x \in \mathbf{D}$; $\mathbf{F} = \{f_i\}_{i=1}^{m}$ is a set of *utility functions*, each defined over a set of decision variables: $f_i : \prod_{x \in \mathbf{x}^{f_i}} D_x \to \mathbb{R}^+ \cup \{\bot\}$, where $\mathbf{x}^{f_i} \subseteq \mathbf{X}$ is *scope* of f_i and $\bot$ is a special element used to denote that a given combination of values for the variables in $\mathbf{x}^{f_i}$ is not allowed; and $\alpha : \mathbf{X} \to \mathbf{A}$ is a function that associates each decision variable to one agent.

A *solution* σ is a value assignment for a set $\mathbf{x}_\sigma \subseteq \mathbf{X}$ of 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 the applicable utility functions in σ. A solution σ is *complete* if $\mathbf{x}_\sigma = \mathbf{X}$. The goal is to find an optimal complete solution $\mathbf{x}^* = \operatorname{argmax}_{\mathbf{x}} \mathbf{F}(\mathbf{x})$.

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 $\{x, y\} = \mathbf{x}^{f_i}$. A *DFS*

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 $N(a_i) = \{a_j \in \mathcal{A} \mid \{x_i, x_j\} \in E\}$ to denote the neighbors of agent a_i.

2.2 Dynamic DCOP

A *Dynamic DCOP* (*D-DCOP*) is defined as a sequence of DCOPs with changes between them, without an explicit model for how the DCOP will change over time. Solving a D-DCOP optimally means finding a utility-maximal solution for each DCOP in the sequence. Therefore, this approach is *reactive* since it 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 [23, 33]. Alternatively, a *proactive* approach predicts future changes in the D-DCOP and finds robust solutions that require little or no changes despite future changes.

Researchers have also proposed other models for D-DCOPs including a model where agents have deadlines to choose their values [24], a model where agents can have imperfect knowledge about their environment [12], and a model where changes in the constraint graph depends on the value assignments of agents [36].

2.3 DPOP and S-DPOP

The *Distributed Pseudo-tree Optimization Procedure* (*DPOP*) [22] is a complete *inference algorithm* composed of three phases:

- *Pseudo-tree Generation:* The agents build a pseudo-tree [8].
- *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.[1] 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.

Super-stabilizing DPOP (*S-DPOP*) [23] 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. MOTIVATING DOMAIN

We now introduce a distributed dynamic meeting scheduling problem, which will serve as a representative domain to motivate our work. In a distributed meeting scheduling problem [15], a set of $\mathbf{M}$ *weekly* meetings need to be scheduled between members of an organization (e.g., employees of a company; students, faculty members, and staff of a university), taking restrictions in their availability as well as their time and location preferences into account. For a meeting $m \in \mathbf{M}$, A_m is the set of attendees to the meeting, s_m is the start time of the meeting, d_m is its duration, and l_m is its location. Typically, the attendees commit to a meeting time. However, certain exogenous factors may affect the meeting time preferences of some participants. For instance, depending on the location and time of the meeting, traffic conditions may cause delays that should be taken into account, as they could cause cascading effects.

In a typical DCOP formulation, meeting starting times are modeled as decision variables of the agents. Thus, for a given meeting m, each agent $a \in A_m$, controls a pair of decision variables, x_{s_m}, x_{l_m}, whose values represent possible start times s_m and locations l_m for the meeting m. All agents participating in meeting m need to agree upon a given meeting time and location. This condition can be modeled by imposing an equality constraints on the values of the variables x_{s_m}, and x_{l_m} controlled by the agents in A_m. Agent preferences on the time and location for a given meeting can be modeled as unary constraints involving the variable describing the desired meeting. The goal is to find a feasible *weekly* schedule that maximizes the utility over all attendees.

Typical weekly meeting schedules may be adjusted based on changes of the meeting participants' preferences. To address such a requirement, one can use a Dynamic DCOP formulation, where a new DCOP problem, representing the scheduling problem for a *single* week, can be modeled according to the previous formulation and solved, as soon as some agent's meeting preference changes. However, these formulations exhibit several limitations: (*i*) they fail to capture the presence of exogenous factors (e.g., traffic conditions) in the dynamic aspect of the problem, and (*ii*) they do not take account the inconvenience of the participants to change their schedule, when the preference of some agents are updated. Our proposed PD-DCOP model alleviates these limitations by acting proactively during the problem resolution, which allows us to make a step forward towards more refined dynamic solutions.

4. PD-DCOP MODEL

A *Proactive Dynamic DCOP* (*PD-DCOP*) is a tuple $\langle \mathbf{A}, \mathbf{X}, \mathbf{D}, \mathbf{F}, h, \mathbf{T}, c, \gamma, p^0_{\mathbf{Y}}, \alpha \rangle$, where:

- $\mathbf{A} = \{a_i\}_{i=1}^{p}$ is a set of *agents*.
- $\mathbf{X} = \{x_i\}_{i=1}^{n}$ is a mixed set of *decision* and *random variables*. To differentiate between decision variables and random variables, we use $\mathbf{Y} \subseteq \mathbf{X}$ to denote the set of random variables that model uncontrollable stochastic events (e.g., traffic, weather, malfunctioning devices).
- $\mathbf{D} = \{D_x\}_{x \in \mathbf{X}}$ is a set of finite *domains*. Each variable $x \in \mathbf{X}$ takes values from the set $D_x \in \mathbf{D}$. We also use $\Omega = \{\Omega_y\}_{y \in \mathbf{Y}} \subseteq \mathbf{D}$ to denote the set of event spaces for the random variables (e.g., different traffic conditions, weather conditions, or stress levels to which a device is subjected to) such that each $y \in \mathbf{Y}$ takes values in Ω_y.
- $\mathbf{F} = \{f_i\}_{i=1}^{m}$ is a set of *reward functions*, each defined over a mixed set of decision variables and random variables: $f_i : \prod_{x \in \mathbf{x}^{f_i}} D_x \rightarrow \mathbb{R}^+ \cup \{\bot\}$, where $\mathbf{x}^{f_i} \subseteq \mathbf{X}$ is *scope* of f_i and $\bot$ is a special element used to denote that a given combination of values for the variables in $\mathbf{x}^{f_i}$ is not allowed.
- $h \in \mathbb{N}$ is a finite *horizon* in which the agents can change the values of their variables.
- $\mathbf{T} = \{T_y\}_{y \in \mathbf{Y}}$ is the set of *transition functions* $T_y : \Omega_y \times \Omega_y \rightarrow [0, 1] \subseteq \mathbb{R}$ for the random variables $y \in \mathbf{Y}$, describing the probability for a random variable to change its value in successive time steps. For a time step $t > 0$, and values $\omega_i \in \Omega_y, \omega_j \in \Omega_y$, $T_y(\omega_i, \omega_j) = P(y^t = \omega_j \mid y^{t-1} = \omega_i)$, where y^t denotes the value of the variable y at time step t, and P is a probability measure. Thus, $T_y(\omega_i, \omega_j)$ describes the probability for the random variable y to change its value from ω_i at a time step $t-1$ to ω_j at a time step t. Finally, $\sum_{\omega_j \in \Omega_y} T_y(\omega_i, \omega_j) = 1$ for all $\omega_i \in \Omega_y$.
- $c \in \mathbb{R}^+$ is a *switching cost*, which is the cost associated with the change in the value of a decision variable between time steps.

[1] 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.

- $\gamma \in [0, 1)$ is a *discount factor*, which represents the decrease in the importance of rewards/costs over time.
- $p^0_{\mathbf{Y}} = \{p^0_y\}_{y \in \mathbf{Y}}$ is a set of initial *probability distributions* for the random variables $y \in \mathbf{Y}$.
- $\alpha : \mathbf{X} \setminus \mathbf{Y} \rightarrow \mathbf{A}$ is a function that associates each decision variable to one agent. We assume that the random variables are not under the control of the agents and are independent of decision variables. Thus, their values are solely determined according to their transition functions.

Throughout this paper, we refer to decision (resp. random) variables as with the letter x (resp. y). We also assume that each agent controls exactly one decision variable (thus, α is a bijection), and that each reward function $f_i \in \mathbf{F}$ associates with at most one random variable y_i.[2]

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

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

$$\mathcal{F}^h(\mathbf{x}) = \sum_{t=0}^{h-1} \gamma^t \left[\mathcal{F}^t_x(\mathbf{x}^t) + \mathcal{F}^t_y(\mathbf{x}^t)\right] \tag{2}$$

$$- \sum_{t=0}^{h-1} \gamma^t \left[c \cdot \Delta(\mathbf{x}^t, \mathbf{x}^{t+1})\right] \tag{3}$$

$$+ \tilde{\mathcal{F}}_x(\mathbf{x}^h) + \tilde{\mathcal{F}}_y(\mathbf{x}^h) \tag{4}$$

where Σ is the assignment space for the decision variables of the PD-DCOP, at each time step. Equation (2) refers to the optimization over the first h time steps, with:

$$\mathcal{F}^t_x(\mathbf{x}) = \sum_{f_i \in \mathbf{F} \setminus \mathbf{F}_Y} f_i(\mathbf{x}_i) \tag{5}$$

$$\mathcal{F}^t_y(\mathbf{x}) = \sum_{f_i \in \mathbf{F}_Y} \sum_{\omega \in \Omega_{y_i}} f_i(\mathbf{x}_i|_{y_i=\omega}) \cdot p^t_{y_i}(\omega) \tag{6}$$

where $\mathbf{x}_i$ is an assignment for all the variables in $\mathbf{x}^{f_i}$; we write $\mathbf{x}_i|_{y_i=\omega}$ to indicate that the random variable $y_i \in \mathbf{x}^{f_i}$ takes on the event $\omega \in \Omega_{y_i}$; $\mathbf{F}_Y = \{f_i \in \mathbf{F} \mid \mathbf{x}^{f_i} \cap \mathbf{Y} \neq \emptyset\}$ is the set of functions in $\mathbf{F}$ that involve random variables; $p^t_{y_i}(\omega)$ is the probability for the random variable y_i to assume value ω at time t, and defined as

$$p^t_{y_i}(\omega) = \sum_{\omega' \in \Omega_{y_i}} T_{y_i}(\omega', \omega) \cdot p^{t-1}_{y_i}(\omega') \tag{7}$$

Equation (3) considers the penalties due to the changes in the decision variables' values during the optimization process, where $\Delta : \Sigma \times \Sigma \rightarrow \mathbb{N}$ is a function counting the number of assignments to decision variables that differs from one time step to the next.

Equation (4) refers to the optimization over the last time step, which further accounts for discounted future rewards:

$$\tilde{\mathcal{F}}_x(\mathbf{x}) = \frac{\gamma^h}{1-\gamma} \mathcal{F}^h_x(\mathbf{x}) \tag{8}$$

$$\tilde{\mathcal{F}}_y(\mathbf{x}) = \sum_{f_i \in \mathbf{F}_Y} \sum_{\omega \in \Omega_{y_i}} \tilde{f}_i(\mathbf{x}_i|_{y_i=\omega}) \cdot p^h_{y_i}(\omega) \tag{9}$$

$$\tilde{f}_i(\mathbf{x}_i|_{y_i=\omega}) = \gamma^h \cdot f_i(\mathbf{x}_i|_{y_i=\omega}) + \gamma \sum_{\omega' \in \Omega_{y_i}} T_{y_i}(\omega, \omega') \cdot \tilde{f}_i(\mathbf{x}_i|_{y_i=\omega'}) \tag{10}$$

The goal of a PD-DCOP is to find an assignment of values to its decision variables that maximizes the sum of two terms. The first term maximizes the discounted net utility, that is, the discounted rewards for the functions that do not involve exogenous factors ($\mathcal{F}_x$) and the expected discounted random rewards ($\mathcal{F}_y$) minus the discounted penalties over the first h time steps. The second term maximizes the discounted future rewards for the problem.

While the PD-DCOP model can be used to capture the presence of exogenous factors in the dynamic aspect of the problem, 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 reward value for all decision variables; the addition of a constraint can be handled by defining a 0 reward constraint in the model from the start, and updating its reward when the constraint is added.

4.1 Modeling the Motivating Domain

PD-DCOPs can naturally handle the dynamic characteristic of the distributed dynamic meeting scheduling problem that we use as our motivating domain. Uncontrollable events, such as traffic conditions, affecting the meeting times of agents can be modeled via random variables. In particular, in our model, each pair of agent's variables x_{s_m} and x_{l_m}, describing the time and location of a meeting m, is associated to a random variable $y_m \in \mathbf{Y}$, describing the different traffic conditions that can affect agents' time and location preferences for the meeting. Traffic predictions are often available and they are modeled via the transition functions. Additionally, rescheduling meetings is inconvenient for the meeting participants and this inconvenience is modeled via switching costs for each attendee, imposed when a meeting is forced to be rescheduled. Finally, the PD-DCOP horizon captures the horizon (i.e., number of weeks) of the scheduling problem (e.g., the DCOP problem of each time step corresponds to the scheduling problem for a single week).

4.2 Theoretical Properties

We now describe some of the theoretical properties of PD-DCOPs.

THEOREM 1. *Solving a PD-DCOP is PSPACE-complete (-hard) if the horizon h is polynomial (exponential) in $|\mathbf{X}|$.*

PROOF: We first consider the case when 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 opened 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^0_y 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 reward function f_c, whose scope involves all and only the PD-DCOP-corresponding boolean variables appearing in c, and such that:

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

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 reward 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

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

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

We next consider the case when h is exponential in $\mathbf{X}$: In such case solving PD-DCOPs is PSPACE-hard as storing a solution requires space exponential in $|\mathbf{X}|$. □

Absolute Error Bound: Let U^{∞} denote the optimal solution quality with an infinite horizon and U^{h} denote the optimal solution quality with a finite horizon h. Thus, $\epsilon \geq U^{\infty} - U^{h}$ and we first describe this error bound. Let's define $F^{\Delta} = \max_{\mathbf{y}\in\Omega} \max_{\mathbf{x}\in\Sigma} (\mathbf{F}(\mathbf{x}^* \cup \mathbf{y}) - \mathbf{F}(\mathbf{x} \cup \mathbf{y}))$, where $\mathbf{x}$ and $\mathbf{y}$ are assignments of values to all decision and random variables, respectively; $\mathbf{F}$ is the overall reward function for a (regular) DCOP in the PD-DCOP; and $\mathbf{x}^*$ is an optimal value assignment of decision variables for that regular DCOP given value assignment $\mathbf{y}$.

THEOREM 2. *The absolute error bound ϵ equals $\frac{\gamma^h}{1-\gamma} F^{\Delta}$.*

PROOF: Let $\breve{\mathbf{x}}^* = \langle \breve{\mathbf{x}}_0^*, \ldots, \breve{\mathbf{x}}_h^*, \breve{\mathbf{x}}_{h+1}^*, \ldots \rangle$ be the vector of assignments that maximizes U^{∞}.

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

Ignoring switch costs after time h, we get U_{+}^{∞} and $U^{\infty} \leq U_{+}^{\infty}$, as:

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

Let $\mathbf{x}^* = \langle \mathbf{x}_0^*, \ldots, \mathbf{x}_h^* \rangle$ be the vector of assignments maximizing U^h,

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

For $\breve{\mathbf{x}}^*$, if we change the decision variable assignment after time step h to $\breve{\mathbf{x}}_h^*$, as $\langle \breve{\mathbf{x}}_0^*, \ldots, \breve{\mathbf{x}}_h^*, \breve{\mathbf{x}}_h^*, \ldots \rangle$, we get U_{-}^{∞}:

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

Since the value assignments of $\breve{\mathbf{x}}$ are identical for all time steps $t \geq h$, $U_{-}^{\infty} \leq U^{h}$. Therefore, we get $U_{-}^{\infty} \leq U^{h} \leq U^{\infty} \leq U_{+}^{\infty}$.

Next, we know that

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

Given a value assignment $\mathbf{y}$ for all random variables, the PD-DCOP becomes a regular DCOP, where one only needs to find value assignment of decision variables and maximize the total utility. For this DCOP, there is a utility difference between the best assignment and the worst assignment for all decision variables. Among all these utility differences, the maximum one is $F^{\Delta} = \max_{\mathbf{y}\in\Omega} \max_{\mathbf{x}\in\Sigma} (\mathbf{F}(\mathbf{x}^* \cup \mathbf{y}) - \mathbf{F}(\mathbf{x} \cup \mathbf{y}))$, where $\mathbf{x}$ and $\mathbf{y}$ are assignments of values to all decision and random variables, respectively; $\mathbf{F}$ is the overall reward function for a (regular) DCOP in the PD-DCOP; and $\mathbf{x}^*$ is an optimal value assignment of decision variables for that regular DCOP given value assignment $\mathbf{y}$. Thus, it is the maximum difference in overall reward over all combination of value assignments $\mathbf{x}$ and $\mathbf{y}$.

Notice that the quantity $\mathcal{F}_x^t(\breve{\mathbf{x}}_t^*) + \mathcal{F}_y^t(\breve{\mathbf{x}}_t^*) - \mathcal{F}_x^t(\breve{\mathbf{x}}_h^*) + \mathcal{F}_y^t(\breve{\mathbf{x}}_h^*)$ is the utility difference between the value assignment $\breve{\mathbf{x}}_t^*$ and $\breve{\mathbf{x}}_h^*$ for the regular DCOPs in time step t. So, we have:

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

which concludes the proof. □

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

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:

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

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

THEOREM 3. $\forall \mathbf{x} \in \Sigma^{h+1} : \ \mathcal{F}^h(\mathbf{x}) \leq \mathcal{F}^h(\mathbf{x}^*) \leq \hat{\mathcal{F}}^h(\hat{\mathbf{x}}^*)$.

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

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

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

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

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

$$\begin{aligned} \mathcal{F}_t^h(\mathbf{x}_t^*) &= \gamma^t \left[\mathcal{F}_x^t(\mathbf{x}_t^*) + \mathcal{F}_y^t(\mathbf{x}_t^*) - [c\Delta(\mathbf{x}_t, \mathbf{x}_{t+1})] \right] \\ &\leq \gamma^t \left[\mathcal{F}_x^t(\mathbf{x}_t^*) + \mathcal{F}_y^t(\mathbf{x}_t^*) \right] \\ &\leq \max_{\mathbf{x}\in\Sigma} \gamma^t \left[\mathcal{F}_x^t(\mathbf{x}) + \mathcal{F}_y^t(\mathbf{x}) \right] = \hat{\mathcal{F}}_t^h(\hat{\mathbf{x}}_t^*) \end{aligned}$$

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:

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

Thus, from the two inequalities above, it follows:

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

which concludes the proof. □

COROLLARY 2. *The approximation ratio ρ is $\frac{\hat{\mathcal{F}}^h(\hat{\mathbf{x}}^*)}{\mathcal{F}^h(\mathbf{x})}$ for any solution $\mathbf{x}$.*

5. PD-DCOP ALGORITHMS

We now introduce exact and approximation PD-DCOP algorithms.

5.1 Exact Approach

We first propose an exact approach, which 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 reward function F_i in this new DCOP captures the sum of rewards of the reward function $f_i \in \mathbf{F}$ across all time steps, and (2) the domain of each decision variable is the set of all possible combination of values of that decision variable across all time steps. However, this process needs to be done in a distributed manner.

We divide the reward functions into two types: (1) The functions $f_i \in \mathbf{F}$ whose scope $\mathbf{x}^{f_i} \cap \mathbf{Y} = \emptyset$ includes exclusively decision variables, and (2) the functions $f_i \in \mathbf{F}$ whose scope $\mathbf{x}^{f_i} \cap \mathbf{Y} \neq \emptyset$ includes one random variable. In both cases, let $\mathbf{x}_i = \langle \mathbf{x}_i^0, \ldots, \mathbf{x}_i^h \rangle$ denote the vector of value assignments to all *decision* variables in $\mathbf{x}^{f_i}$ for each time step.

Then, each function $f_i \in \mathbf{F}$ whose scope includes only decision variables can be replaced by a function F_i:

$$F_i(\mathbf{x}_i) = \sum_{t=0}^{h-1} F_i^t(\mathbf{x}_i^t) + F_i^h(\mathbf{x}_i^h) \tag{11}$$

$$= \left[\sum_{t=0}^{h-1} \gamma^t \cdot f_i(\mathbf{x}_i^t) \right] + \left[\frac{\gamma^h}{1-\gamma} f_i(\mathbf{x}_i^h) \right] \tag{12}$$

where the first term with the summation is the reward for the first h time steps and the second term is the reward for the remaining time steps. Each function $f_i \in \mathbf{F}$ whose scope includes random variables can be replaced by a *unary* function F_i:

$$F_i(\mathbf{x}_i) = \sum_{t=0}^{h-1} F_i^t(\mathbf{x}_i^t) + F_i^h(\mathbf{x}_i^h) \tag{13}$$

$$= \sum_{t=0}^{h-1} \gamma^t \sum_{\omega \in \Omega_{y_i}} f_i(\mathbf{x}_i^t|_{y_i=\omega}) \cdot p_{y_i}^t(\omega) \tag{14}$$

$$+ \sum_{\omega \in \Omega_{y_i}} \tilde{f}_i(\mathbf{x}_i^h|_{y_i=\omega}) \cdot p_{y_i}^h(\omega) \tag{15}$$

where the first term (Equation (14)) is the reward for the first h time steps and the second term (Equation (15)) is the reward for the remaining time steps. The function $\tilde{f}_i$ is recursively defined according to Equation (10). Additionally, each decision variable x_i will have a unary function C_i:

$$C_i(\mathbf{x}_i) = -\sum_{t=0}^{h-1} \gamma^t \left[c \cdot \Delta(\mathbf{x}_i^t, \mathbf{x}_i^{t+1}) \right] \tag{16}$$

which captures the cost of switching values across time steps. This collapsed DCOP can then be solved with any off-the-shelf DCOP algorithm. In our experiments, we use DPOP [22] to solve it.

5.2 Approximation Approach

Since optimally solving PD-DCOPs is P-SPACE-hard, the exact approach described earlier will fail to scale to large problems, as we show in our experimental results. Therefore, approximation approaches are necessary to solve the larger problems of interest. Our local search algorithm to solve PD-DCOPs is inspired by MGM [14], which has been shown to be robust in dynamically changing environments. Algorithm 1 shows its pseudocode, where each agent a_i maintains the following data structures:

Algorithm 1: LOCAL SEARCH( )

```
1  iter ← 1
2  ⟨v_i^{0*}, v_i^{1*}, ..., v_i^{h*}⟩ ← ⟨Null, Null, ..., Null⟩
3  ⟨v_i^0, v_i^1, ..., v_i^h⟩ ← INITIALASSIGNMENT()
4  context ← ⟨(x_j, t, Null | x_j ∈ N(a_i), 0 ≤ t ≤ h)⟩
5  Send VALUE(⟨v_i^0, v_i^1, ..., v_i^h⟩) to all neighbors
```

Procedure CalcGain()

```
6  ⟨u_i^0, u_i^1, ..., u_i^h⟩ ← CALCUTILS(⟨v_i^0, v_i^1, ..., v_i^h⟩)
7  u* ← −∞
8  foreach ⟨d_i^0, d_i^1, ..., d_i^h⟩ in ×_{i=0}^h D_{x_i} do
9      u ← CALCCUMULATIVEUTIL(⟨d_i^0, d_i^1, ..., d_i^h⟩)
10     if u > u* then
11         u* ← u
12         ⟨v_i^{0*}, v_i^{1*}, ..., v_i^{h*}⟩ ← ⟨d_i^0, d_i^1, ..., d_i^h⟩
13 if u* ≠ −∞ then
14     ⟨u_i^{0*}, u_i^{1*}, ..., u_i^{h*}⟩ ← CALCUTILS(⟨v_i^{0*}, v_i^{1*}, ..., v_i^{h*}⟩)
15     ⟨û_i^0, û_i^1, ..., û_i^h⟩ ← ⟨u_i^{0*}, u_i^{1*}, ..., u_i^{h*}⟩ − ⟨u_i^0, u_i^1, ..., u_i^h⟩
16 else
17     ⟨û_i^0, û_i^1, ..., û_i^h⟩ ← ⟨Null, Null, ..., Null⟩
18 Send GAIN(⟨û_i^0, û_i^1, ..., û_i^h⟩) to all neighbors
```

- $iter$ 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.
- $\langle v_i^0, v_i^1, \ldots, v_i^h \rangle$ is a vector of the agent's current value assignment for its variable x_i at each time step t.
- $\langle v_i^{0*}, v_i^{1*}, \ldots, v_i^{h*} \rangle$ is a vector of the agent's best value assignment for its variable x_i at each time step t.
- $\langle u_i^0, u_i^1, \ldots, u_i^h \rangle$ is a vector of the agent's utility (rewards from reward functions minus costs from switching costs) given its current value assignment at each time step t.
- $\langle u_i^{0*}, u_i^{1*}, \ldots, u_i^{h*} \rangle$ is a vector of the agent's best utility given its best value assignment at each time step t.
- $\langle \hat{u}_i^{0*}, \hat{u}_i^{1*}, \ldots, \hat{u}_i^{h*} \rangle$, which is a vector of the agent's best gain in utility at each time step t.

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 $0 \leq t \leq h$ and initializes its context $context$. (2) Each agent uses VALUE messages to ensure that it has the correct assumption on its neighboring agents' variables' values. (3) Each agent computes its current utilities given its current value assignments, its best utilities over all possible value assignments, and its best gain in utilities, and sends this gain in a GAIN message to all its neighbors. (4) Each agent changes the value of its variable for time step t if its gain for that time step is the largest over all its neighbors' gain for that time step, and repeats steps 2 through 4 until a termination condition is met. In more detail:

Step 1: Each agent initializes its vector of best values to a vector of *Nulls* (line 2) and calls INITIALASSIGNMENT to initializes its current values (line 3). The values can be initialized randomly or according to some heuristic function. We describe later one such heuristic. Finally, the agent initializes its context, where it assumes that the values for its neighbors is null for all time steps (line 4).

Step 2: The agent sends its current value assignment in a VALUE message to all neighbors (line 5). When it receives a VALUE message from its neighbor, it updates its context with the value assignments in that message (lines 19-21). When it has received VALUE

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

19 **foreach** t from 0 to h **do**
20 **if** $v_s^{t*} \neq$ *Null* **then**
21 Update $(x_s, t, v_s^t) \in context$ with (x_s, t, v_s^{t*})
22 **if** received VALUE messages from all neighbors in this iteration **then**
23 CALCGAIN()
24 $iter \leftarrow iter + 1$

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

25 **if** $\langle \hat{u}_s^0, \hat{u}_s^1, \ldots, \hat{u}_s^h \rangle \neq \langle$Null, Null, $\ldots$, Null$\rangle$ **then**
26 **foreach** t from 0 to h **do**
27 **if** $\hat{u}_i^t \leq 0 \vee \hat{u}_s^t > \hat{u}_i^t$ **then**
28 $v_i^{t*} \leftarrow$ Null
29 **if** received GAIN messages from all neighbors in this iteration **then**
30 **foreach** t from 0 to h **do**
31 **if** $v_i^{t*} \neq$ Null **then**
32 $v_i^t \leftarrow v_i^{t*}$
33 Send VALUE($\langle v_i^{1*}, v_i^{2*}, \ldots, v_i^{h*} \rangle$) to all neighbors

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 23).

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 6). The utility for a time step t is made out of two components (line 41). The first component is the sum of rewards over all reward 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 reward 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 Equations (11) to (12). 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 Equations (13) to (15). 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 35-40). The (net) utility is thus the reward from the reward functions minus the switching cost (line 41).

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 8-12). 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 13-15).

Step 4: The agent sends its gains in a GAIN message to all neighbors (line 18). 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 (line 27). 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 29-33).

Function CalcUtils($\langle v_i^0, v_i^1, \ldots, v_i^h \rangle$)

34 **foreach** t from 0 to h **do**
35 **if** $t = 0$ **then**
36 $c_i^t \leftarrow \gamma^0 \cdot \text{cost}(v_i^0, v_i^1)$
37 **else if** $t = h$ **then**
38 $c_i^t \leftarrow \gamma^{h-1} \cdot \text{cost}(v_i^{h-1}, v_i^h)$
39 **else**
40 $c_i^t \leftarrow \gamma^{t-1} \cdot \text{cost}(v_i^{t-1}, v_i^t) + \gamma^t \cdot \text{cost}(v_i^t, v_i^{t+1})$
41 $u_i^t \leftarrow \sum_{F_j^t | x_i \in \mathbf{x}^{F_j^t}} F_j^t - c_i^t$
42 **return** $\langle u_i^0, u_i^1, \ldots, u_i^h \rangle$

Function CalcCumulativeUtil($\langle v_i^0, v_i^1, \ldots, v_i^h \rangle$)

43 $u \leftarrow \sum_{t=0}^{h} \sum_{F_j^t | x_i \in \mathbf{x}^{F_j^t}} F_j^t$
44 $c \leftarrow 0$
45 **foreach** t from 0 to $h-1$ **do**
46 $c \leftarrow c + \gamma^t \cdot \text{cost}(v_i^t, v_i^{t+1})$
47 **return** $u - c$

Heuristics for INITIALASSIGNMENT: We simplify the PD-DCOP into 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 h DCOPs. We initially used DPOP to do this, but our preliminary experimental results show that this approach is computationally inefficient; the runtimes with this approach were larger than with the random assignment heuristic.

However, we observed that these 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 h DCOPs and the runtimes decreased marginally.

We further optimized this approach by designing a new pseudo-trees 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 [8], assigns a score to each agent a according to heuristic $h_1(a)$:

$$h_1(a) = (1 + I(a)) \cdot |N_y(a)| \tag{17}$$

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

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

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 introduced a hybrid heuristic $h_3(a) = w\, h_1(a) + (1 - w)\, h_2(a)$, which combines both heuristics and weigh them according to a *heuristic weight* w.

5.3 Theoretical Properties

In the following discussion, let $O(\mathcal{L})$ denote the agent's space requirement for the INITIALASSIGNMENT function in line 3.

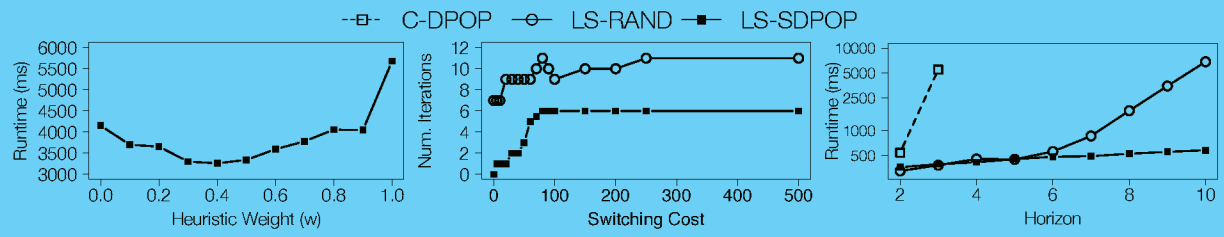

Figure 1: Experimental Results Varying Heuristic Weight (left), Switching Cost (middle), and Horizon (right)

THEOREM 4. *The agent's space required by the PD-DCOP approximation approach is* $O(\mathcal{L} + (h+1)|\mathbf{A}|)$.

PROOF: In our approximation algorithm, 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 3). Thus, the memory requirement of this step is $O((h+1) + \mathcal{L})$ at each agent. Next, each agent running our approximation algorithm performs a local search step (lines 4-5), 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 approximation algorithm is $O(\mathcal{L} + (h+1)|\mathbf{A}|)$ per agent. □

In Section 7, we present two approximation approaches: LS-DPOP, which uses DPOP as the INITIALASSIGNMENT function, and LS-RAND, which uses a random assignment function. Thus, the space complexity of LS-DPOP is $O(d^{w^*})$ – where d is the size of the largest domain and w^* is the induced width of the pseudo-tree – as DPOP dominates the space complexity in the initialization step. The space complexity of LS-RAND is instead dominated by the second step of the approximation PD-DCOP algorithm, and it is thus $O((h+1)|\mathbf{A}|)$.

6. RELATED WORK

Aside from the D-DCOPs described in Sections 1 and 2, 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 [29, 10]. The goal is typically to find a solution that is robust to possible changes. Other related models include *Mixed CSPs* [6], 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* [30, 27], 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 model D-DCOPs with state variables that are beyond the control of agents [20]. However, they assume that the state is observable to the agents, while PD-DCOPs assume otherwise. Additionally, MD-DCOP agents do not incur a cost for changing values in MD-DCOPs and only a reactive online learning approach to solving the problem has been proposed thus far.

Another related body of work is *Decentralized Markov Decision Processes* (*Dec-MDPs*) [2]. 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 too solves a sequential decision making problem. However, Dec-MDPs are typically solved in a centralized manner [2, 1, 4, 5] due to its high complexity – solving Dec-MDPs optimally is NEXP-hard even for the case with only two agents [2]. In contrast, PD-DCOPs are solved in a decentralized manner and its complexity is only PSPACE-hard. 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*) [2] 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-hard [2] and they are also typically solved in a centralized manner [9, 26, 25, 31, 5, 21] with some exceptions [18]. Researchers have also developed a hybrid model, called ND-POMDP [19], 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 expressiveness, DCOPs can solve single timestep problems while Dec-(PO)MDPs can solve sequential problems. However, DCOPs are NP-hard while Dec-(PO)MDPs are NEXP-hard. PD-DCOPs attempt to balance the trade off between expressiveness and complexity by searching for open-loop policies instead of closed-loop policies of Dec-(PO)MDPs. They are more expressive than DCOPs at the cost of a higher complexity, yet not as expressive as Dec-(PO)MDPs, but also without their prohibitive complexity.

7. EXPERIMENTAL RESULTS

We empirically evaluate *Collapsed DPOP* (*C-DPOP*), which collapses the PD-DCOP into a DCOP and solves it with DPOP; *Local Search (Random)* (*LS-RAND*), which runs the local search algorithm with random initial values; and *Local Search (S-DPOP)* (*LS-SDPOP*), which runs the algorithm with S-DPOP and the h_3 heuristic function to generate pseudo-trees. In contrast to many experiments in the literature, our experiments are performed in an *actual distributed system*, where each agent is an Intel i7 Quadcore 3.4GHz machine with 16GB of RAM, connected in a local area network. We thus report *actual* distributed runtimes. We impose a timeout of 30 minutes and a memory limit of 16 GB. Results are averaged over 30 runs. We use the following default configuration: Number of agents and decision variables $|\mathbf{A}| = |\mathbf{X} \setminus \mathbf{Y}| = 12$; number of random variables $|\mathbf{Y}| = 0.25 \cdot |\mathbf{X} \setminus \mathbf{Y}|$; domain size $|D_x| = |\Omega_y| = 3$; horizon $h = 3$; switching cost $c = 50$; constraint densities $p_1^a = p_1^b = p_1^c = 0.5$;[3] and constraint tightness $p_2 = 0.8$.

[3] p_1^a is the density of functions between two decision variables, p_1^b is the density of functions between a decision variable and a random variable, and p_1^c is the fraction of decision variables that are constrained with random

$\|\mathbf{A}\|$	C-DPOP		LS-SDPOP			LS-RAND	
	time (ms)	ρ	time (ms)		ρ	time (ms)	ρ
2	223	1.001	197	(207)	1.003	203	1.019
4	489	1.000	255	(307)	1.009	273	1.037
6	5547	1.000	382	(456)	1.011	385	1.045
8	—		739	(838)	1.001	556	1.034
12	—		4821	(7091)	1.003	1092	1.031
16	—		264897	(595245)	1.033	2203	1.015

Table 1: Experimental Results Varying Number of Agents

$\|\mathbf{A}\|$	C-DPOP		LS-SDPOP		LS-RAND	
	time (ms)	% SAT	time (ms)	% SAT	time (ms)	% SAT
2	509	100	262	100	271	100
4	4786	100	367	100	399	100
6	—		2651	96	718	93
8	—		71726	96	3249	86
10	—		—		9723	86
12	—		—		15370	86

Table 2: Results for Dynamic Distributed Meeting Scheduling

Random Networks. We first vary the weight w of the pseudo-tree construction heuristic h_3 to identify the best weight for LS-SDPOP. Figure 1(left) shows the runtimes of LS-SDPOP. 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. The runtimes are 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 large depths, which in turn result in large runtimes. The best weight is found at $w = 0.4$, thus we use this value for the remaining experiments.

We then vary the switching cost c of the problem from 0 to 500 to investigate its impact on the algorithms' performance. Figure 1(center) shows the number of iterations it takes for the local search algorithms to converge from the initial solution. When $c = 0$, the initial solution found by LS-SDPOP is an optimal solution since the initial solution already optimizes the utilities of the problem over all time steps ignoring switching costs. Thus, it requires 0 iterations to converge. For sufficiently large costs ($c \geq 100$), the optimal solution is one where the values for each agent is the same across all time steps since the cost of changing values is larger than the gain in utility. Thus, the number of iterations they take to converge is the same for all large switching costs. At intermediate cost values ($0 < c < 100$), they require an increasing number of iterations to converge. Finally, LS-RAND requires more iterations to converge than LS-SDPOP since it starts with poorer initial solutions.

We also vary the horizon h of the problem from 2 to 10 to evaluate the scalability of the algorithms.[4] Figure 1(right) shows the runtimes of all three algorithms. As expected, the runtimes increase when the horizon increases. When the horizon $h \geq 6$ is sufficiently large, LS-SDPOP is faster than LS-RAND indicating that the overhead of finding good initial solutions with S-DPOP is worth the savings in runtime to converge to the final solution.

Finally, we vary the number of agents $|\mathbf{A}|$ (and thus the number of the decision variables) of the problem from 2 to 16. Table 1 tabulates the runtimes and the approximation ration ρ for all three algorithms. The runtimes of LS-SDPOP without reusing information are shown in parentheses. C-DPOP times out after $|\mathbf{A}| \geq 8$. In general, the runtimes of C-DPOP is largest, followed the by the runtimes of LS-SDPOP and the runtimes of LS-RAND. The difference in runtimes increases with increasing number of agents, indicating that the overhead to find good initial solutions with S-DPOP is not worth the savings in convergence runtime. As expected, the approximation ratio ρ with C-DPOP the is the smallest, since it finds optimal solutions, whilst the ratios of the local search algorithms are of similar values, indicating that they converge to solutions with similar qualities. Therefore, LS-SDPOP is preferred in problems with few agents but large horizons and LS-RAND is preferred in problems with many agents but small horizons.

However, another factor to consider when choosing which algorithm to run is their memory requirement. LS-SDPOP suffer from the same exponential memory requirement (in the induced width of the pseudo-tree) of DPOP [22]. In contrast, LS-RAND's memory requirement is only linear in the number of agents and the horizon. Thus, LS-RAND is preferred in problems where agents have a limited amount of memory (e.g., sensor networks).

Dynamic Distributed Meeting Scheduling. We also evaluate our PD-DCOP algorithms on the dynamic distributed meeting scheduling problem introduced in Section 3, where we use the following parameters: We allow each meeting to be scheduled in 4 different starting time and 2 locations. We generate the underlying graph topology randomly, using the same settings described in the previous experiments. Thus, the number of meetings and the number of meetings participants are not bounded by any fixed value. In order to ensure that each agent controls exactly one decision variable, we use the pseudo-agent decomposition technique [35]. Inequality constraints between the meeting start time and locations ensure that an agent can attend at most one meeting at a given time, and that no two meetings are held in the same location at the same time. In addition, start times and locations of each meeting's participants are enforced to be equal, so as to produce feasible schedules. Finally, agents' preferences on time and meeting locations are modeled through unary costs functions. We use the same heuristic weight, switching costs, and horizon settings from the previous experiment on random networks.

Table 2 illustrates the average runtimes (in ms) and the percentage of feasible solutions over 30 instances, returned by the algorithms at varying the number of agents $|\mathbf{A}|$ from 2 to 12. Similar to the results analyzed for random networks, both approximation approaches (LS-SDPOP and LS-RAND) can produce solutions faster than the exact approach. However, the number of satisfiable instances decreases with increasing number of agents. In particular, the quality of solutions found by LS-SDPOP degrades slower than the quality of solutions found by LS-RAND. The reason is likely because LS-SDPOP starts with a better initial solution than LS-RAND. As expected, these results reveal that for our PD-DCOP approximation approach, the initial solution is crucial to ensure convergence to solutions of high quality within a bounded runtime.

8. CONCLUSIONS

In real-world applications, agents often act in dynamic environments. Thus, the Dynamic DCOP formulation is attractive to model such problems. Current 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, we (*i*) introduce *Proactive Dynamic DCOPs* (*PD-DCOPs*), which model the dynamism in Dynamic DCOPs; (*ii*) provide theoretical results on the complexity class of PD-DCOPs; and (*iii*) develop an exact PD-DCOP algorithm that solves the problem *proactively* as well as an approximation algorithm with quality guarantees that can scale to larger and more complex problems. Finally, in contrast to many experiments in the literature, we evaluate our algorithms on an actual distributed system, which will ease the transition to real-world applications.

variables.

[4]In this experiment, we set the number of decision variables to 6 in order for the algorithms to scale to larger horizons.

REFERENCES

[1] R. Becker, S. Zilberstein, V. Lesser, and C. Goldman. Solving transition independent decentralized Markov decision processes. *Journal of Artificial Intelligence Research*, 22:423–455, 2004.

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

[3] F. Delle Fave, A. Rogers, Z. Xu, S. Sukkarieh, and N. Jennings. Deploying the max-sum algorithm for decentralised coordination and task allocation of unmanned aerial vehicles for live aerial imagery collection. In *Proceedings of ICRA*, pages 469–476, 2012.

[4] J. S. Dibangoye, C. Amato, and A. Doniec. Scaling up decentralized MDPs through heuristic search. In *Proceedings of UAI*, pages 217–226, 2012.

[5] J. S. Dibangoye, C. Amato, A. Doniec, and F. Charpillet. Producing efficient error-bounded solutions for transition independent decentralized MDPs. In *Proceedings of AAMAS*, pages 539–546, 2013.

[6] H. Fargier, J. Lang, and T. Schiex. Mixed constraint satisfaction: A framework for decision problems under incomplete knowledge. In *Proceedings of AAAI*, pages 175–180, 1996.

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

[8] Y. Hamadi, C. Bessière, and J. Quinqueton. Distributed intelligent backtracking. In *Proceedings of ECAI*, pages 219–223, 1998.

[9] E. A. Hansen, D. S. Bernstein, and S. Zilberstein. Dynamic programming for partially observable stochastic games. In *Proceedings of AAAI*, pages 709–715, 2004.

[10] A. Holland and B. O'Sullivan. Weighted super solutions for constraint programs. In *Proceedings of AAAI*, pages 378–383, 2005.

[11] A. Kumar, B. Faltings, and A. Petcu. Distributed constraint optimization with structured resource constraints. In *Proceedings of AAMAS*, pages 923–930, 2009.

[12] R. Lass, E. Sultanik, and W. Regli. Dynamic distributed constraint reasoning. In *Proceedings of AAAI*, pages 1466–1469, 2008.

[13] T. Léauté and B. Faltings. Coordinating logistics operations with privacy guarantees. In *Proceedings of IJCAI*, pages 2482–2487, 2011.

[14] R. Maheswaran, J. Pearce, and M. Tambe. Distributed algorithms for DCOP: A graphical game-based approach. In *Proceedings of PDCS*, pages 432–439, 2004.

[15] R. Maheswaran, M. Tambe, E. Bowring, J. Pearce, and P. Varakantham. Taking DCOP to the real world: Efficient complete solutions for distributed event scheduling. In *Proceedings of AAMAS*, pages 310–317, 2004.

[16] S. Miller, S. Ramchurn, and A. Rogers. Optimal decentralised dispatch of embedded generation in the smart grid. In *Proceedings of AAMAS*, pages 281–288, 2012.

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

[18] R. Nair, M. Tambe, M. Yokoo, D. Pynadath, and S. Marsella. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *Proceedings of IJCAI*, pages 705–711, 2003.

[19] R. Nair, P. Varakantham, M. Tambe, and M. Yokoo. Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs. In *Proceedings of AAAI*, pages 133–139, 2005.

[20] D. T. Nguyen, W. Yeoh, H. C. Lau, S. Zilberstein, and C. Zhang. Decentralized multi-agent reinforcement learning in average-reward dynamic DCOPs. In *Proceedings of AAAI*, pages 1447–1455, 2014.

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

[22] A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *Proceedings of IJCAI*, pages 1413–1420, 2005.

[23] A. Petcu and B. Faltings. Superstabilizing, fault-containing multiagent combinatorial optimization. In *Proceedings of AAAI*, pages 449–454, 2005.

[24] A. Petcu and B. Faltings. Optimal solution stability in dynamic, distributed constraint optimization. In *Proceedings of IAT*, pages 321–327, 2007.

[25] S. Seuken and S. Zilberstein. Memory-bounded dynamic programming for DEC-POMDPs. In *Proceedings of IJCAI*, pages 2009–2015, 2007.

[26] D. Szer, F. Charpillet, and S. Zilberstein. MAA*: A heuristic search algorithm for solving decentralized POMDPs. In *Proceedings of UAI*, pages 576–590, 2005.

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

[28] S. Ueda, A. Iwasaki, and M. Yokoo. Coalition structure generation based on distributed constraint optimization. In *Proceedings of AAAI*, pages 197–203, 2010.

[29] R. Wallace and E. Freuder. Stable solutions for dynamic constraint satisfaction problems. In *Proceedings of CP*, pages 447–461, 1998.

[30] T. Walsh. Stochastic constraint programming. In *Proceedings of ECAI*, pages 111–115, 2002.

[31] S. Witwicki and E. Durfee. Towards a unifying characterization for quantifying weak coupling in Dec-POMDPs. In *Proceedings of AAMAS*, pages 29–36, 2011.

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

[33] W. Yeoh, P. Varakantham, X. Sun, and S. Koenig. Incremental DCOP search algorithms for solving dynamic DCOPs. In *Proceedings of IAT*, pages 257–264, 2015.

[34] W. Yeoh and M. Yokoo. Distributed problem solving. *AI Magazine*, 33(3):53–65, 2012.

[35] M. Yokoo, editor. *Distributed Constraint Satisfaction: Foundation of Cooperation in Multi-agent Systems*. Springer, 2001.

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