Proactive Dynamic DCOPs

Khoi Hoang[†], Ferdinando Fioretto[†], Ping Hou[†], Makoto Yokoo[‡], William Yeoh[†], and Roie Zivan[⋆]
[†]Department of Computer Science, New Mexico State University
[‡]Department of Informatics, Kyushu University
[⋆]Department of Industrial Engineering and Management, Ben Gurion University of the Negev

Abstract

The current approaches to model dynamism in DCOPs solve a sequence of static problems, *reacting* to the changes in the environment as the agents observe them. Such approaches, thus, ignore possible predictions on the environment evolution. To overcome such limitations, we introduce the *Proactive Dynamic DCOP* (PD-DCOP) model, 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.

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 (Modi et al. 2005; Yeoh and Yokoo 2012). They are well-suited for modeling multi-agent coordination problems where the primary interactions are between local subsets of agents (Maheswaran et al. 2004; Farinelli et al. 2008; Ueda, Iwasaki, and Yokoo 2010). Unfortunately, DCOPs only model static problems or, in other words, problems that do not evolve over time. However, within a real-world application, agents often act in dynamic environments. For instance, in a disaster management scenario, 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.

Consequently, researchers have introduced the *Dynamic DCOP* (D-DCOP) model (Petcu and Faltings 2005b; 2007; Lass, Sultanik, and Regli 2008), where utility functions can change during the problem solving process. All of these models make the common assumption that information on how the problem might change is unavailable. As such, existing approaches *reacts* to the change in the problem and solve the current problem at hand. However, in many applications, information on how the problem might change is indeed available, within some degree of uncertainty.

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

Background

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 mixed set of decision variables: $f_i : \times_{x \in \mathbf{x}^{f_i}} D_x \to \mathbb{R}^+ \cup \{\perp\}$, where $\mathbf{x}^{f_i} \subseteq \mathbf{X}$ is *scope* of f_i and $\perp$ 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.

Dynamic DCOPs: A *Dynamic DCOP* (*D-DCOP*) is defined as a sequence of DCOP 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 a *reactive* approach since it does not consider future changes. The advantage of this approach is that solving a D-DCOP is no harder than solving h DCOPs, where h is the horizon of the problem. Petcu and Faltings (2005b) have used this approach to solve D-DCOPs, where they introduce a super-stabilizing DPOP algorithm that is able to reuse information from previous DCOPs to speed up the search for the solution for the current DCOP. 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 (Petcu and Faltings 2007), a model where agents can have imperfect knowledge about their environment (Lass, Sultanik, and Regli 2008), and a model where changes in the constraint graph depends on the value assignments of agents (Zivan et al. 2015).

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

- *Pseudo-tree Generation:* Agents coordinate to build a pseudo-tree (Hamadi, Bessière, and Quinqueton 1998).
- *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*) (Petcu and Faltings 2005b) is a self-stabilizing extension of DPOP, where the agents restart the DPOP phases when they detect changes in the problem. S-DPOP makes use of information that is not affected by the changes in the problem.

Motivating Domain

We now introduce a *Dynamic Distributed Electric Vehicle (EV) Charging Scheduling Problem* (D-DEVCSP), which we use as a representative domain to motivate the introduction of PD-DCOPs. We consider an n days scheduling problem, with hourly time intervals. In our scenario, we consider a set of houses $\mathcal{H}$, each of which constitutes a charging station for an EV $v \in \mathcal{H}$. Neighboring houses are connected to each other via transmission lines, and power is provided by a electricity provider. The set of houses and transmission lines can be visualized as a undirected graph $G = (\mathcal{H}, \mathcal{L})$, whose vertices represent houses and edges represent transmission lines connecting pairs of neighboring houses. EVs can be charged exclusively when connected to their own station (i.e., when parked at home), and each EV v needs to be charged for a given minimum amount of energy p_v, according to the user usage and is influenced by the amount of time the EV is driven when not connected to the charging station. In a realistic scenario, such information is inevitably stochastic, as factors such as user users leaving time schedules and driving distances are outside the control of the decision process.

In addition, each house $v \in \mathcal{H}$ has a background load b_v^t and a maximal energy usage limit q_v^t for each time step t. The latter is a common factor in several European countries. Furthermore, a transmission line $l \in \mathcal{L}$ is associated to a maximal *thermal capacity* c_l, which limits the maximal amount of energy that can travel on l at a given time. Users can express their preferences on the charging schedule, which may, for instance, reflect the dynamic price schema imposed by the retailer. The goal is that of finding a feasible charging schedule for all the EVs that satisfies the capacity constraints of the underlying electricity network while maximizing the utilities associated to the user preferences.

The D-DEVCSP can be formalized as a dynamic DCOP, where each EV v is modeled by an agent a_v; start times for the charging schedules are modeled via decision variables x_v. Other decision variables involve the power flow on the transmission lines l_{uv}, for pair of neighboring houses u, v. All the capacity constraints, relative to houses and transmission lines, can be captured via hard constraints. Further user preferences on scheduling times can be captured via soft constraints.

Notice that the D-DCOP formulation can only partially capture the complexity of the D-DEVCSP. Indeed such formulation has some shortcoming, as they: (a) fail to capture the presence of exogenous characters in the dynamic aspect of the problem, and (b) do not take account the inconvenience of the users to change their schedule, when the preference of some agents are updated. The PD-DCOP model solves these shortcomings by acting proactively in the problem resolution, and allowing us to make a step forward toward more refined dynamic solutions.

PD-DCOP Model

A *Proactive Dynamic DCOP* (PD-DCOP) is a tuple $\langle \mathbf{A}, \mathbf{X}, \mathbf{D}, \mathbf{F}, \mathbf{T}, \gamma, h, c, p^0_{\mathbf{Y}}, \alpha \rangle$, 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., 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 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: $f_i : \times_{x \in \mathbf{x}^{f_i}} D_x \to \mathbb{R}^+ \cup \{\perp\}$, where $\mathbf{x}^{f_i} \subseteq \mathbf{X}$ is *scope* of f_i and $\perp$ 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* $T_y : \Omega_y \times \Omega_y \to [0, 1] \subseteq \mathbb{R}$ 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 $t > 0$,

[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.

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*, defined as the cost associated to the action of changing the values of a decision variable from one time step to the next.
- $\gamma \in [0, 1)$, is a *discount factor*, which represents the difference in importance between future rewards/costs and present rewards/costs.
- $p^0_{\mathbf{Y}} = \{p^0_y\}_{y \in \mathbf{Y}}$ is a set of initial *probability distributions* for the random variables $y \in \mathbf{Y}$.
- $\alpha : \mathbf{X} \setminus \mathbf{Y} \to \mathbf{A}$ 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 $\omega \in \Omega_{y_i}$; $\mathbf{F}_Y = \{f_i \in \mathbf{F} | \mathbf{x}^{f_i} \cap \mathbf{Y} \neq \emptyset\}$ is the set of functions in $\mathbf{F}$ which 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

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

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

Equation (3) takes into account the penalties due to the changes in the decision variables' values during the optimization process, where $\Delta : \Sigma \times \Sigma \to \mathbb{N}$ is a function that counts 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:

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

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

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

In summary, 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.

D-DEVCSP as a P-DCOP: PD-DCOPs can naturally handle the dynamic character of the D-DEVCSP, presented in the previous Section. Uncontrollable events, such as user leaving times and travel distance affecting agents charging times decisions can be modeled via random variables. In particular, in our model each agent's variable is associated to two random variables $y_{v_l}, y_{v_d} \in \mathbf{Y}$, describing the different user leaving times and travel distances per day, which can affect preferences the agents' preferences. Hourly user profiles on both leaving times and travel distance can be modeled via transition functions $T_y(t, t')$, for every time of the day t, t'. EV charging rescheduling over time, is inconvenient for the EV drivers, thus it is important to find stable schedules past the given horizon. This character is modeled via switching costs for each EV schedule, occurring when a charing plan is forced to be rescheduled. Finally the PD-DCOP horizon captures the dynamic character of the domain.

PD-DCOP Algorithms

We introduce two approaches: An exact approach, which transforms a PD-DCOP into an equivalent DCOP and solves it using any off-the-shelf DCOP algorithm, and an approximation approach, which uses local search.

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,

Algorithm 1: LOCAL SEARCH( )

$iter \leftarrow 1$
$\langle v_i^{0*}, v_i^{1*}, \dots, v_i^{h*} \rangle \leftarrow \langle$Null, Null, …, Null$\rangle$
$\langle v_i^0, v_i^1, \dots, v_i^h \rangle \leftarrow$ INITIALASSIGNMENT()
$context \leftarrow \langle (x_j, t,$ Null $\mid x_j \in N(a_i), 0 \leq t \leq h) \rangle$
Send VALUE($\langle v_i^0, v_i^1, \dots, v_i^h \rangle$) to all neighbors

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 some random variable. In both cases, let $\mathbf{x}_i = \langle \mathbf{x}_i^0, \dots, \mathbf{x}_i^h \rangle$ 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:

$$F_i(\mathbf{x}_i) = \left[\sum_{t=0}^{h-1} \gamma^t \cdot f_i(\mathbf{x}_i^t) \right] + \left[\frac{\gamma^h}{1-\gamma} f_i(\mathbf{x}_i^h) \right] \tag{11}$$

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:

$$F_i(\mathbf{x}_i) = \sum_{t=0}^{h-1} \gamma^t \sum_{\omega \in \Omega_{y_i}} f_i(\mathbf{x}_i^t|_{y_i=\omega}) \cdot p_{y_i}^t(\omega) \tag{12}$$

$$+ \sum_{\omega \in \Omega_{y_i}} \tilde{f}_i(\mathbf{x}_i^h|_{y_i=\omega}) \cdot p_{y_i}^h(\omega) \tag{13}$$

where the first term (Equation (12)) is the reward for the first h time steps and the second term (Equation (13)) 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:

$$C_i(\mathbf{x}_i) = -\sum_{t=0}^{h-1} \gamma^t \left[c \cdot \Delta(\mathbf{x}_i^t, \mathbf{x}_i^{t+1}) \right] \tag{14}$$

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

Theorem 1 *This collapsed DCOP is equivalent to the original PD-DCOP.*

Approximation Approach

Since solving PD-DCOP is P-SPACE-hard, incomplete approaches are necessary to solve interesting problems. Our

Procedure CalcGain()

$\langle u_i^0, u_i^1, \dots, u_i^h \rangle \leftarrow$ CALCUTILS($\langle v_i^0, v_i^1, \dots, v_i^h \rangle$)
$u^* \leftarrow -\infty$
foreach $\langle d_i^0, d_i^1, \dots, d_i^h \rangle$ in $\times_{i=0}^h D_{x_i}$ **do**
  $u \leftarrow$ CALCCUMULATIVEUTIL($\langle d_i^0, d_i^1, \dots, d_i^h \rangle$)
  if $u > u^*$ **then**
    $u^* \leftarrow u$
    $\langle v_i^{0*}, v_i^{1*}, \dots, v_i^{h*} \rangle \leftarrow \langle d_i^0, d_i^1, \dots, d_i^h \rangle$
if $u^* \neq -\infty$ **then**
  $\langle u_i^{0*}, u_i^{1*}, \dots, u_i^{h*} \rangle \leftarrow$ CALCUTILS($\langle v_i^{0*}, v_i^{1*}, \dots, v_i^{h*} \rangle$)
  $\langle \hat{u}_i^0, \hat{u}_i^1, \dots, \hat{u}_i^h \rangle \leftarrow \langle u_i^{0*}, u_i^{1*}, \dots, u_i^{h*} \rangle - \langle u_i^0, u_i^1, \dots, u_i^h \rangle$
else
  $\langle \hat{u}_i^0, \hat{u}_i^1, \dots, \hat{u}_i^h \rangle \leftarrow \langle$Null, Null, …, Null$\rangle$
Send GAIN($\langle \hat{u}_i^0, \hat{u}_i^1, \dots, \hat{u}_i^h \rangle$) to all neighbors

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

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

local search algorithm to solve PD-DCOPs, is inspired by MGM (Maheswaran, Pearce, and Tambe 2006), 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:

- $iter$ 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 the variable x_j is assigned value v_j^t at time step t.
- $\langle v_i^0, v_i^1, \dots, v_i^h \rangle$ is a vector of the agent's current value assignment for its variable x_i at each time step t.
- $\langle v_i^{0*}, v_i^{1*}, \dots, v_i^{h*} \rangle$ is a vector of the agent's best value assignment for its variable x_i at each time step t.
- $\langle u_i^0, u_i^1, \dots, u_i^h \rangle$ 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.
- $\langle u_i^{0*}, u_i^{1*}, \dots, u_i^{h*} \rangle$ is a vector of the agent's best utility given its best value assignment at each time step t.
- $\langle \hat{u}_i^{0*}, \hat{u}_i^{1*}, \dots, \hat{u}_i^{h*} \rangle$, 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,

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, ..., 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

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         $c_i^t \leftarrow \text{cost}(v_i^0, v_i^1)$
37   **else if** $t = h$ **then**
38         $c_i^t \leftarrow \text{cost}(v_i^{h-1}, v_i^h)$
39   **else**
40         $c_i^t \leftarrow \text{cost}(v_i^{t-1}, v_i^t) + \text{cost}(v_i^t, v_i^{t+1})$
41     $u_i^t \leftarrow \sum_{f_j \,|\, x_i \in \mathbf{x}^{f_j}} f_j(v_i^t, v_j^t \,|\, x_j \in \mathbf{x}^{f_j}, (x_j, t, v_j^t) \in context) - c_i^t$
42 **return** $\langle u_i^0, u_i^1, \ldots, u_i^h \rangle$

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 details:

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 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. The first component is the sum of rewards over all reward functions $f_j \,|\, x_i \in \mathbf{x}^{f_j}$ that involve the agent, under the assumption

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

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

that the agent takes on its current value and its neighbors take on their values according to its context. 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 at each time step (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).

Related Work

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

Researchers have also introduced *Markovian D-DCOPs* (MD-DCOPs), which models D-DCOPs with state variables that are beyond the control of agents (Nguyen et al. 2014). However, they assume that the state is observable to the agents, while PD-DCOPs assume otherwise. 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) and *Decentralized Partially Observable MDPs* (Dec-POMDPs) (Bernstein et al. 2002). In a Dec-MDP, agents can also observe its local state (the global state is the combination of all local states). In a Dec-POMDP, agents may not accurately observe their local states and, thus, maintain a belief of their local states. The goal is to find a policy that maps each local state (in a Dec-MDP) or each belief (in a Dec-POMDP) to the action for each agent. Thus, like PD-DCOPs, they too solve a sequential decision making problem. However, Dec-MDPs and Dec-POMDPs are typically solved in a centralized manner (Bernstein et al. 2002; Becker et al. 2004; Dibangoye et al. 2013; Hansen, Bernstein, and Zilberstein 2004; Szer, Charpillet, and Zilberstein 2005; Oliehoek et al. 2013) due to its high complexity – solving Dec-(PO)MDPs optimally is NEXP-hard even for the case with only two agents (Bernstein et al. 2002). In contrast, PD-DCOPs are solved in a decentralized manner and its complexity is only PSPACE-hard. The reason for the lower complexity is because the solution of PD-DCOPs are *open-loop* policies, which are policies that are independent on state observations.

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 only 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 thus 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.

Experimental Results

We empirically evaluate our three PD-DCOP algorithms: *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. 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

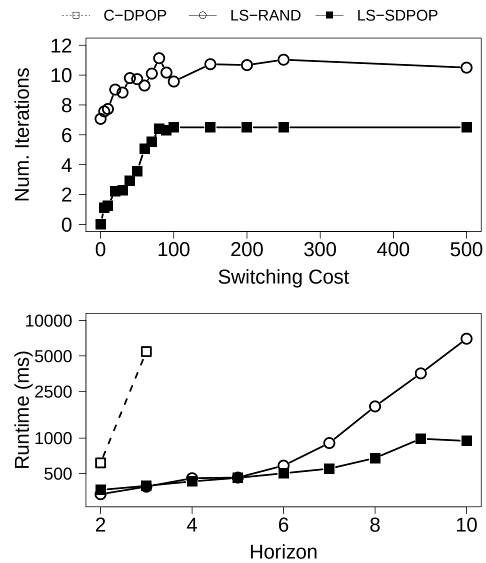

Figure 1: Experimental Results Varying Switching Cost (top) and Horizon (bottom)

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 $p_2 = 0.8$.

We then vary the switching cost c of the problem from 0 to 500. 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.[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

[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 variables.

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

$\|\mathbf{A}\|$	C-DPOP		LS-SDPOP			LS-RAND	
	time (ms)	ρ	time (ms)		ρ	time (ms)	ρ
2	223	1.001	197.5	(207.7)	1.003	203.7	1.019
4	489	1.000	255.7	(307.3)	1.009	273.4	1.037
6	5547	1.000	382.3	(456.3)	1.011	385.9	1.045
8	—		739.2	(838.1)	1.001	556.0	1.034
12	—		4821.6	(7091.1)	1.003	1092.9	1.031
16	—		264897	(595245)	1.033	2203.0	1.015

Table 1: Experimental Results Varying Number of Agents

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.

Conclusions

Within real-world multi-agent applications, agents often act in dynamic environments. Thus, the Dynamic DCOP formulation is attractive to model such problems. Unfortunately, current research has focused at solving such problems *reactively*, thus discarding the information on the environment evolution (i.e., possible future changes), which is often available in many applications. To cope with this limitation, we (*i*) introduce a *Proactive Dynamic DCOP* (PD-DCOP) model, which models the dynamism in Dynamic DCOPs; and (*ii*) develop an exact PD-DCOP algorithm that solves the problem *proactively* as well as an approximate 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.

In the future, we will perform an extensive evaluation to compare the proposed PD-DCOP model against a reactive DCOP approach, and to evaluate our algorithm within the D-DEVCSP domain. Finally, we will enrich the current D-DEVCSP model considering factors such as changes in the price of electricity at different times (based on the distribution of energy consumption in the area), or limitation on total in house energy consumption at different times. These factors can be elegantly modeled as random variables in our PD-DCOP model.

Acknowledgments

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.

References

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

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

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

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

Farinelli, A.; Rogers, A.; Petcu, A.; and Jennings, N. 2008. Decentralised coordination of low-power embedded devices using the Max-Sum algorithm. In *Proceedings of AAMAS*, 639–646.

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

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

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

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

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

Maheswaran, R. T.; Pearce, J. P.; and Tambe, M. 2006. A family of graphical-game-based algorithms for distributed constraint optimization problems. In *Coordination of Large-Scale Multiagent Systems*. Springer. 127–146.

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

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

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

Petcu, A., and Faltings, B. 2005a. A scalable method for multi-agent constraint optimization. In *Proceedings of IJCAI*, 1413–1420.

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

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

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

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

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

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

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

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

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