New Algorithms for Continuous Distributed Constraint Optimization Problems

Khoi D. Hoang
Washington University in St. Louis, USA
khoi.hoang@wustl.edu

William Yeoh
Washington University in St. Louis, USA
wyeoh@wustl.edu

Makoto Yokoo
Kyushu University, Japan
yokoo@inf.kyushu-u.ac.jp

Zinovi Rabinovich
Nanyang Technological University, Singapore
zinovi@ntu.edu.sg

ABSTRACT

Distributed Constraint Optimization Problems (DCOPs) are a powerful tool to model multi-agent coordination problems that are distributed by nature. The formulation is suitable for problems where variables are discrete and constraint utilities are represented in tabular form. However, many real-world applications have variables that are continuous and tabular forms thus cannot accurately represent constraint utilities. To overcome this limitation, researchers have proposed the *Continuous DCOP* (C-DCOP) model, which are DCOPs with continuous variables. But existing approaches usually come with some restrictions on the form of constraint utilities and are without quality guarantees. Therefore, in this paper, we (*i*) propose an exact algorithm to solve a specific subclass of C-DCOPs; (*ii*) propose an approximation method with quality guarantees to solve general C-DCOPs; (*iii*) propose additional C-DCOP algorithms that are more scalable; and (*iv*) empirically show that our algorithms outperform existing state-of-the-art C-DCOP algorithms when given the same communication limitations.

KEYWORDS

Distributed Problem Solving; Distributed Constraint Optimization

ACM Reference Format:
Khoi D. Hoang, William Yeoh, Makoto Yokoo, and Zinovi Rabinovich. 2020. New Algorithms for Continuous Distributed Constraint Optimization Problems. In *Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), Auckland, New Zealand, May 9–13, 2020,* IFAAMAS, 9 pages.

1 INTRODUCTION

The *Distributed Constraint Optimization Problem* (DCOP) [9, 28, 31] formulation is a powerful tool to model cooperative multi-agent problems. DCOPs are well-suited to model many problems that are distributed by nature and where agents need to coordinate their value assignments to maximize the aggregate constraint utilities. DCOPs are widely employed to model distributed problems such as meeting scheduling [17, 26], sensor and wireless network coordination [8, 39], multi-robot coordination [44], smart grid optimization [12, 23, 27], smart home automation [11, 32], and cloud computing applications [20]. Recent advances improve the state of the art [1, 2, 4, 5, 10, 18, 21, 22, 24, 29, 30, 40, 42]; solve extensions like Asymmetric DCOPs [6, 7, 37, 43] and Dynamic DCOPs [17, 19]; and improve key metrics like privacy [14, 15, 35, 36].

Typically, DCOPs assume that the variables are discrete and the constraint utilities are represented in tabular form (i.e., a utility is defined for every combination of discrete values of variables). While these assumptions are reasonable in some applications where values of variables correspond to a set of *discrete* possibilities (e.g., the set of tasks of robots in multi-robot coordination), they make less sense in applications where values of variables correspond to a *continuous* range of possibilities (e.g., the range of sensor orientations in sensor networks or the range of frequencies in wireless networks).

These limiting assumptions have prompted Stranders et al. [33] to propose *Continuous DCOPs* (C-DCOPs), which extend DCOPs to allow for continuous variables. As variables can now take values from a continuous range, constraint utilities are also extended from tabular forms to functional forms. Approaches to solve C-DCOPs include *Continuous MS* (CMS) [33], *Hybrid CMS* (HCMS) [38], *Particle Swarm Based Functional DCOP* (PFD) [3], and *Bayesian DPOP* (B-DPOP) [13]. Both CMS and HCMS extend the discrete *Max-Sum* (MS) algorithm [8], where CMS approximate utility functions with piecewise linear functions and HCMS combines the discrete MS algorithm with continuous non-linear optimization methods. On the other hand, PFD uses *particle swarm optimization* techniques and B-DPOP combines the *Bayesian optimization* framework with *Distributed Pseudo-tree Optimization Procedure* (DPOP) [31] to solve C-DCOPs. A key limitation of the first three algorithms is that they do not provide quality guarantees on the solutions found. B-DPOP does guarantee that it will eventually converge to the global optimum for Lipschitz-continuous objective functions, but do not provide guarantees on intermediate solutions prior to convergence.

To overcome this limitation, we extend the inference-based DPOP algorithm to three extensions – *Exact Continuous DPOP* (EC-DPOP); *Approximate Continuous DPOP* (AC-DPOP); and *Clustered AC-DPOP* (CAC-DPOP). We also extend the search-based *Distributed Stochastic Algorithm* (DSA) [41] to *Continuous DSA* (C-DSA). While EC-DPOP provides an exact approach to solve C-DCOPs with linear or quadratic utility functions and are defined over tree-structured graphs, AC-DPOP, CAC-DPOP, and C-DSA solve C-DCOPs approximately with any smooth, differentiable utility functions and without restriction on graph structure. [1] We also

Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), B. An, N. Yorke-Smith, A. El Fallah Seghrouchni, G. Sukthankar (eds.), May 9–13, 2020, Auckland, New Zealand.

[1] As we consider both convex and non-convex functions, optimization methods such as sub-gradient, interior-point and ellipsoid methods are not applicable. Even in the case that we deal with binary quadratic functions, we assume they can be either concave or convex.

provide theoretical properties on the error bounds of AC-DPOP and communication complexities of AC-DPOP, CAC-DPOP, and C-DSA. Finally, we show that these algorithms outperform HCMS in randomly generated instances when given the same communication limitations.

2 BACKGROUND

We now provide background on DCOPs as well as DPOP and DSA, which we extend later to solve Continuous DCOPs.

DCOPs: A *Distributed Constraint Optimization Problem* (*DCOP*) [9, 28, 31] 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 *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.
- $\mathbf{F} = \{f_i\}_{i=1}^{m}$ is a set of *utility functions*, each defined over a set of variables: $f_i : \prod_{x \in \mathbf{x}^{f_i}} D_x \rightarrow \mathbb{R} \cup \{-\infty\}$, where infeasible configurations have $-\infty$ utilities and $\mathbf{x}^{f_i} \subseteq \mathbf{X}$ is the *scope* of f_i.
- $\alpha : \mathbf{X} \rightarrow \mathbf{A}$ is a *mapping function* that associates each variable to one agent.

In this paper, we assume that each agent controls exactly one variable and thus use the terms "agent" and "variable" interchangeably. We also assume that all utility functions are binary functions between two variables.

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 $\mathbf{x}^* = \text{argmax}_{\mathbf{x}} \mathbf{F}(\mathbf{x})$.

A *constraint graph* visualizes a DCOP, where nodes in the graph correspond to variables in the DCOP and edges connect pairs of variables appearing in the same utility function. A *pseudo-tree* arrangement has the same nodes and edges as the constraint graph and satisfies that (*i*) there is a subset of edges, called *tree edges*, that form a rooted tree and (*ii*) two variables in a utility function appear in the same branch of that tree. The other edges are called *backedges*. Tree edges connect parent-child nodes, while backedges connect a node with its *pseudo-parents* and its *pseudo-children*.

DPOP: *Distributed Pseudo-tree Optimization Procedure (DPOP)* [31] is a complete *inference algorithm* that is composed of three phases:

- *Pseudo-tree Generation:* In this phase, all agents start building a pseudo-tree [16] (line 1).
- *UTIL Propagation:* Each agent, starting from the leaves of the pseudo-tree, computes the optimal sum of utilities in its subtree for all variables in its separator.[2] 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 (line 5). The agent then *projects* out its variable (line 7) and sends the projected function in a UTIL message to its parent (line 8).
- *VALUE Propagation:* Each agent, starting from the root of the pseudo-tree, determines the optimal value for its variable and then sends the optimal value to its children and pseudo-children. The root agent does so by choosing the values of its variables from its UTIL computations, and send them as VALUE messages.

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

Algorithm 1: DPOP()

1 $\mathbf{T}_i \leftarrow$ PseudotreeGeneration()
2 Util-Propagation($\mathbf{T}_i$)
3 Value-Propagation($\mathbf{T}_i$)

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

4 **receive** $\text{UTIL}_c(f_c)$ from each $a_c \in Children_i$
5 $f_{\text{agent\_view}} \leftarrow \text{ADD}\left(\underset{a_s \in Separator_i}{f_i^s}, \underset{a_c \in Children_i}{f_c} \right)$
6 **if** isRoot() is False **then**
7 $\quad f_{p_i} \leftarrow \text{PROJECT}(f_{\text{agent\_view}}, x_i)$
8 $\quad$ **send** $\text{UTIL}_i(f_{p_i})$ msg to $Parent_i$

Procedure VALUE-Propagation($\mathbf{T}_i$)

9 **if** isRoot() **then**
10 $\quad v_i \leftarrow \text{argmax}_{x_i} f_i(x_i)$
11 $\quad$ **send** $\text{VALUE}_i(v_i)$ msg to $a_c \in Children_i$
12 **else**
13 $\quad$ **receive** $\text{VALUE}_j(\mathbf{v}_j)$ msg from $Parent_i$
14 $\quad v_i \leftarrow \text{argmax}_{x_i} f_{\text{agent\_view}}(x_i, x_{j1} = v_{j1}, \ldots, x_{jn} = v_{jn})$
15 $\quad$ **send** $\text{VALUE}_i(\mathbf{v}_i)$ msg to to all $a_c \in Children_i$

DSA: *Distributed Stochastic Algorithm (DSA)* [41] is an incomplete, synchronous *search algorithm*. In DSA, each agent, after initially choosing a random value, loops over a sequence of steps until the termination condition is met. In each loop, the agent exchanges the information about its latest values with all neighboring agents. Then, the agent will choose the value with the largest gain in its local utility with neighboring agents, and decide stochastically to change its assignment to the new value or keep the current value. The process repeats until the termination condition is met such as timeout or the solution quality doesn't improve.

3 CONTINUOUS DCOP MODEL

The *Continuous DCOP* (C-DCOP) model generalizes the regular discrete DCOP model by modeling the variables as continuous variables [33]. It is defined by a tuple $\langle \mathbf{A}, \mathbf{X}, \mathbf{D}, \mathbf{F}, \alpha \rangle$, where $\mathbf{A}$, $\mathbf{F}$, and α are exactly as defined in DCOPs. The key differences are:

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

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

4 C-DCOP ALGORITHMS

We now introduce four C-DCOP algorithms: *Exact Continuous DPOP* (EC-DPOP), *Approximate Continuous DPOP* (AC-DPOP), and *Clustered AC-DPOP* (CAC-DPOP), which are based on DPOP; and *Continuous DSA* (C-DSA), which is based on DSA. These algorithms

Function ADD-Functions(f_{pw}, g_{pw})

16 Initialize a piecewise function h_{pw}
17 $\langle \mathbf{x}, \mathbf{d} \rangle \leftarrow$ GetCommonVariablesAndRanges(f_{pw}, g_{pw})
18 **foreach** domain $d \in \mathbf{d}$ **do**
19 **foreach** $f \in f_{pw}$ with domain d_f **do**
20 **foreach** $g \in g_{pw}$ with domain d_g **do**
21 **if** d is a sub-domain of f and g **then**
22 $h \leftarrow f + g$
23 $d_h = d \cup d_f \cup d_g$
24 Add h with domain d_h to h_{pw}
25 **return** h_{pw}

Function PROJECT-Function(f_{pw}, x_i)

26 Initialize a piecewise function h_{pw}
27 **foreach** $f \in f_{pw}$ **do**
28 Solve $\frac{\partial f}{\partial x_i} = 0$ for closed-form solutions $\bar{x}_i = g^*(\mathbf{x})$
29 Compute $\bar{g}(\mathbf{x}) = f(x_i = \bar{x}, \mathbf{x})$
30 Compute $g(\check{\mathbf{x}}) = f(x_i = LB_x, \mathbf{x})$
31 Compute $g(\hat{\mathbf{x}}) = f(x_i = UB_x, \mathbf{x})$
32 Solve $\bar{g}$, $\check{g}$, and $\hat{g}$ pairwise for intersection range set $\mathbf{r}$
33 **foreach** $r \in \mathbf{r}$ **do**
34 Detemine either $\bar{g}$, $\check{g}$, or $\hat{g}$ is the largest on range r
35 Add the function with range r to h_{pw}
36 **return** h_{pw}

extend the capability of their original algorithms such that they can solve C-DCOPs with continuous variables and utility functions.

4.1 Exact Continuous DPOP

In this section, we propose *Exact Continuous DPOP* (EC-DPOP), which is an exact algorithm. EC-DPOP solves C-DCOPs that are defined over tree-structured graphs with linear or quadratic utility functions. The algorithm extends the two primary operations of DPOP in the UTIL propagation phase – Add and Project. Those modification are modified such that they can be applied to C-DCOPs in the context of continuous variables and real-valued functions.

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

ADD-Functions: In EC-DPOP, each UTIL message contains a piecewise function that is derived from the Project-Function operation (described below). The addition of two piecewise functions is done by adding their sub-functions, which may have different domains. We will use the following two functions for illustration:

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

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

When adding two piecewise functions, we first identify the common variables between the two functions and create a new set of atomic ranges for the variables (line 17). For example, when adding the functions f_{12} and f_{23} above, the only common variable is x_2, and the atomic ranges for x_2 are $[0, 3]$, $[3, 6]$, and $[6, 10]$. The ranges of the other variables remain unchanged from their original functions. We then take the Cartesian product of the range sets of all common variables and associate the appropriate function to that range. For example, adding f_{12} and f_{23} will result in f_{123} (line 18-24):

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

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

$$g(x_{i_1}, \dots, x_{i_k}) = \max_{x_i} f(x_i, x_{i_1}, \dots, x_{i_k}) \tag{1}$$

First, we solve the following for closed-form solutions (line 28):

$$\frac{\partial f(x_i, x_{i_1}, \dots, x_{i_k})}{\partial x_i} = 0 \tag{2}$$

Let $\bar{x}_i = g^*(x_{i_1}, \dots, x_{i_k})$ be the solution to the above equation, one candidate function for g is (line 29):

$$\bar{g}(x_{i_1}, \dots, x_{i_k}) = f(x_i = \bar{x}_i, x_{i_1}, \dots, x_{i_k}) \tag{3}$$

We then compute other two candidate functions (line 30-31):

$$\check{g} = f(x_i = LB_{x_i}, x_{i_1}, \dots, x_{i_k}) \tag{4}$$

$$\hat{g} = f(x_i = UB_{x_i}, x_{i_1}, \dots, x_{i_k}) \tag{5}$$

Next, we need to find the intervals where each of the functions $\bar{g}, \check{g}$ and $\hat{g}$ is the largest (line 32-35). Those intervals are the intersections between the three functions and, thus, we solve each of the equations below to find them:

$$\check{g}(x_{i_1}, \dots, x_{i_k}) = \hat{g}(x_{i_1}, \dots, x_{i_k}) \tag{6}$$

$$\check{g}(x_{i_1}, \dots, x_{i_k}) = \bar{g}(x_{i_1}, \dots, x_{i_k}) \tag{7}$$

$$\hat{g}(x_{i_1}, \dots, x_{i_k}) = \bar{g}(x_{i_1}, \dots, x_{i_k}) \tag{8}$$

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

Unfortunately, it is not always possible to find closed-form solutions to the partial derivative in Equation (2). We discuss below two types of functions – binary linear and quadratic functions –

where it is possible to find closed-form solutions. We assume that all coefficients are non-zero.

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

$$g(x_{i_1}) = \begin{cases} f(x_i = LB_{x_i}, x_{i_1}) & \text{if} \quad a > 0 \\ f(x_i = UB_{x_i}, x_{i_1}) & \text{otherwise} \end{cases} \tag{9}$$

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

$$\frac{\partial f(x_i, x_{i1})}{\partial x_i} = 0 \tag{10}$$

$$\bar{x}_i = \frac{-b - ex_{i_1}}{2a} \tag{11}$$

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

$$LB_{x_i} \leq \frac{-b - ex_{i_1}}{2a} \leq UB_{xi} \tag{12}$$

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

$$f(x_1, x_2) = -2x_1^2 + 4x_1 + 2x_2^2 + x_2 + 7x_1x_2 - 10$$

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

$$\check{g}(x_2) = f(x_1 = -5, x_2) = 2x_2^2 - 34x_2 - 80 \qquad x_2 \in [-10, 10]$$
$$\hat{g}(x_2) = f(x_1 = 5, x_2) = 2x_2^2 + 36x_2 - 40 \qquad x_2 \in [-10, 10]$$

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

$$\frac{\partial f(x_1, x_2)}{\partial x_1} = 0$$
$$-4x_1 + 4 + 7x_2 = 0$$
$$x_1 = \frac{7x_2 + 4}{4}$$

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

$$-5 \leq x_1 \leq 5$$
$$-5 \leq \frac{7x_2 + 4}{4} \leq 5$$
$$\frac{-24}{7} \leq x_2 \leq \frac{16}{7}$$

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

$$\bar{g}(x_2) = f(x_1 = \frac{7x_2 + 4}{4}, x_2)$$
$$= \frac{65}{8}x_2^2 + 8x_2 - 8$$

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

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

$$\check{g}(x_2) = \hat{g}(x_2)$$
$$2x_2^2 - 34x_2 - 80 = 2x_2^2 + 36x_2 - 40$$
$$x_2 = -\frac{4}{7}$$

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

$$\check{g}(x_2) = \bar{g}(x_2)$$
$$2x_2^2 - 34x_2 - 80 = \frac{65}{8}x_2^2 + 8x_2 - 8$$
$$x_2 = -\frac{24}{7}$$

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

$$\hat{g}(x_2) = \bar{g}(x_2)$$
$$2x_2^2 + 36x_2 - 40 = \frac{65}{8}x_2^2 + 8x_2 - 8$$
$$x_2 = \frac{16}{7}$$

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

$$g(x_2) = \max_{x_1} f(x_1, x_2)$$
$$= \max_{x_1} \left(-2x_1^2 + 4x_1 + 2x_2^2 + x_2 + 7x_1x_2 - 10\right)$$
$$= \begin{cases} 2x_2^2 - 34x_2 - 80, & x_2 \in [-10, -\frac{24}{7}] \\ \frac{65}{8}x_2^2 + 8x_2 - 8, & x_2 \in [-\frac{24}{7}, -\frac{4}{7}] \\ \frac{65}{8}x_2^2 + 8x_2 - 8, & x_2 \in [-\frac{4}{7}, \frac{16}{7}] \\ 2x_2^2 + 36x_2 - 40, & x_2 \in [\frac{16}{7}, 10] \end{cases}$$

4.2 Approximate Continuous DPOP

In general C-DCOPs, it is not always possible to find a closed-form solution to Eq. (2) (e.g., it is a multivariate equation). Therefore, an approximation approach is desired for C-DCOPs.

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

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

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

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

37 **if** isLeaf() **then**
38 $V \leftarrow$ DiscretizePPDomain()
39 $V' \leftarrow$ LeafMoveValues(V)
40 $T_{p_i} \leftarrow$ CreateUtilTable(V')
41 **else**
42 **receive** $\text{UTIL}_c(T_c)$ from each $a_c \in Children_i$
43 Add additional tuples and interpolate utilities for all T_c
44 $UTIL_i \leftarrow \text{Add}\left(f_i^s \underset{a_s \in Separator_i}{,} T_c \underset{a_c \in Children_i}{}\right)$
45 $V' \leftarrow$ NonLeafMovePPValues($UTIL_i$)
46 $T_{pi} \leftarrow$ Interpolate(V', $UTIL_i$)
47 **send** $\text{UTIL}_i(T_{pi})$ to $Parent_i$

UTIL Propagation: In this phase, each leaf agent first discretizes the domains of agents in its separator (i.e., its parent and pseudo-parents) and then stores the Cartesian product of these discrete values in set V (line 38). Therefore, each element $v \in V$ is a tuple $\langle v_{i_1}, \ldots, v_{i_k} \rangle$, where v_{i_j} is the value of separator agent a_{i_j}.

Then, for each tuple $v \in V$, the agent "moves" each value v_{i_j} in the tuple along the gradient of each function that is relevant to agent a_{i_j} (line 39). Specifically, the agent updates value v_{i_j} for each separator agent x_{i_j} of the leaf agent x_i:

$$v_{i_j} = v_{i_j} + \alpha \left. \frac{\partial f_{i_j}(x_i, x_{i_j})}{\partial x_{i_j}} \right|_{\text{argmax}_{x_{i_j}} f_{i_j}(x_i, x_{i_j} = v_{i_j})}^{v_{i_j}} \tag{13}$$

where $f_{i_j}(x_i, x_{i_j})$ is the utility function between the leaf agent x_i and the separator agent x_{i_j} and α is the *learning rate* of the algorithm. The agent "moves" the values until they have either converged or a maximum number of iterations is reached. Then, the updated values in V' and their corresponding utilities define the UTIL table that is sent to the parent agent in a UTIL message (line 40).

As in DPOP, each non-leaf agent will first wait for the UTIL messages from each of its children. When all the UTIL messages are received, the agent processes the UTIL tables in the UTIL message from each child. Note that in regular DPOP, the Cartesian product of the values of agents are consistent across the UTIL tables of all children (i.e., if the values of an agent a exists in the Cartesian products of two children, then those values are identical). The reason is because all agents agree on the discretization of the domain of agent a and do not update the value of that agent (such as through Eq. (13)). Therefore, each agent can easily add up the utilities in the UTIL tables received together with the utilities of constraints between the agent and its separator.

In contrast, since the values of agents are updated according to Eq. (13) in AC-DPOP, these values may no longer be consistent across different UTIL tables received. To remedy this issue, each agent first adds additional tuples to each UTIL table received such that the Cartesian product of the values of agents are consistent across all the UTIL tables. Then, it approximates the utilities of the newly added tuples by interpolating between the utilities of the existing tuples. Finally, since the UTIL tables are now all consistent, the agent adds up the utilities in the UTIL tables of children together with the utilities of constraints between the agent and its separator in the same way as DPOP. However, if some variables in the separator are missing, the agent will discretize and add their domains to the UTIL table (line 43-44).

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

$$v_{i_j} = v_{i_j} + \alpha \left. \frac{\partial f_{i_j}(x_i, x_{i_j})}{\partial x_{i_j}} \right|_{\text{argmax}_{x_{i_j}} UTIL_i(x_i, v_{i_1}, \ldots, v_{i_k})}^{v_{i_j}} \tag{14}$$

where $UTIL_i$ is the utility table that is constructed from the summation of the children's utilities and the utilities of constraints between the agent x_i with its separator set (line 45). The key difference between this Eq. (14) and the Eq. (13) used by leaf agents is that the substitution of $f_{i_j}(x_{i_j} = v_{i_j})$ with $UTIL_i(v_{i_1}, \ldots, v_{i_k})$. The reason for this substitution is that the utilities in the UTIL tables of leaf agents are only a function of constraints with their separator agents and the functional form of those constraints are known. Therefore, leaf agents can optimize exactly those functions to get accurate gradients. In contrast, utilities in the UTIL tables of non-leaf agents are also a function of the constraints between its descendant agents and its separator agent, and the functional form of those constraints are not known. They are only represented by samples within the UTIL tables received and are now integrated into the UTIL table of the non-leaf agent. Therefore, in Eq. (14), the agent approximates its maximum value x_i by choosing the best value of under the assumption that the values of the other separator agents are exactly the same as in the tuple $\langle v_{i_1}, \ldots, v_{i_k} \rangle$ that is being updated.

After these values are all updated, the agent approximates their corresponding utilities by interpolating between known utilities and sends these utilities up to its parent in a UTIL message (line 46). These UTIL messages propagate up to the root agent, which then starts the VALUE phase.

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

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

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

$$f_{13}(x_1, x_3) = 16x_1^2 + 13x_1 + 12x_3^2 + 18x_3 + 9x_1x_3 - 13$$
$$f_{34}(x_3, x_4) = -3x_3^2 + 18x_3 - 8x_4^2 + 8x_4 + 2x_3x_4 + 12$$

Agent x_3 discretizes its domain $[-100, 100]$ into the set of values $V = \{-100, -50, 0, 50, 100\}$, and computes the Cartesian product of x_1 and x_4's values $V \times V =$

$\{\langle -100, -100\rangle, \langle -100, -50\rangle, \ldots \langle 0, 0\rangle, \ldots\}$. To move values of x_1 and x_4 in the tuple $\langle 0, 0\rangle$, the agent follows the Eq. (13):

$$\begin{aligned} v_{x_1} &= v_{x_1} + \alpha \left.\frac{\partial f(x_1, x_3)}{\partial x_1}\right|_{\text{argmax}_{x_3} f_{13}(x_1 = v_{x_1}, x_3)}^{v_{x_1}} \\ &= 0 + 0.001\ (32x_1 + 9x_3 + 13)\big|_{x_3=100}^{x_1=0} \\ &= 0.913 \end{aligned}$$

Similarly, x_3 moves value of its parent x_4:

$$\begin{aligned} v_{x_4} &= v_{x_4} + \alpha \left.\frac{\partial f(x_3, x_4)}{\partial x_4}\right|_{\text{argmax}_{x_3} f_{34}(x_3, x_4 = v_{x_4})}^{v_{x_4}} \\ &= 0 + 0.001\,(-16x_4 + 2x_3 + 8)\big|_{x_3=3}^{x_4=0} \\ &= 0.014 \end{aligned}$$

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

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

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

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

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

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

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

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

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

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

(c) UTIL_{x_4}

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

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

$$\begin{aligned} v_{x_1} &= v_{x_1} + \alpha \left.\frac{\partial f(x_1, x_4)}{\partial x_1}\right|_{\text{argmax}_{x_4} UTIL_{x_4}(x_1 = v_{x_1}, x_4)}^{v_{x_1}} \\ &= 2.2 + 0.001\ (2x_1 + 16x_4 + 19)\big|_{x_4=1.75}^{x_1=2.2} \\ &= 2.25 \end{aligned}$$

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

4.3 Clustered Approximate Continuous DPOP

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

With this motivation in mind, we extend AC-DPOP to *Clustered AC-DPOP* (CAC-DPOP), which bounds the number of tuples sent in UTIL messages to limit the message size. CAC-DPOP is identical to AC-DPOP in every way except that agents choose k representative tuples and their corresponding utilities to be sent up to their parents in UTIL messages. To choose these k representative tuples, we use the k-means clustering algorithm [25] to cluster the tuples and then approximate the utilities of those tuples through interpolation. This approach assumes that tuples that are close to each other will have similar values.

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

4.4 Continuous DSA

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

5 THEORETICAL PROPERTIES

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

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

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

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

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

Theorem 5.1. *The error bound of* discrete DPOP *is* $|\mathbf{F}|m\delta$.

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

point where:

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

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

$$\begin{aligned}|\nabla f(v')| &= |\frac{\partial f}{\partial x_i}(v_i)| + |\frac{\partial f}{\partial x_{i_1}}(v_{i_1})| + \ldots + |\frac{\partial f}{\partial x_{i_k}}(v_{i_k})| \\ &\geq |\frac{\partial f}{\partial x_{i_1}}(v_{i_1})| + \ldots + |\frac{\partial f}{\partial x_{i_k}}(v_{i_k})| \\ &= |\nabla g(v'_{-i})| \\ &> \delta\end{aligned}$$

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

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

Theorem 5.2. *The error bound of* AC-DPOP *is* $|\mathbf{F}|(m + |\mathbf{A}|k\alpha\delta)\delta$, *where* k *is the number of times each agent "moves" values of its separator by calling Eqs.* (13) *or* (14).

Proof. After each "move" by either Eqs. (13) or (14), the maximum size of the hypercubes increases by $\alpha\delta$, where α is the learning rate. Since each agent performs this update only k times, the largest increase in the size of the hypercube is $k\alpha\delta$. Finally, since the value of an agent can be updated by any of its children or pseudo-children, the total increase in the size of the hypercube is thus $|\mathbf{A}|k\alpha\delta$, where $|\mathbf{A}|$ is the number of agents in the problem. Therefore, this combined with the proof of the bound for discrete DPOP, the error bound is thus $|\mathbf{F}|(m + |\mathbf{A}|k\alpha\delta)\delta$. □

Theorem 5.3. *In a binary constraint graph* $G = (\mathbf{X}, E)$, *the number of messages of* HCMS *and* C-DSA *with* k *iterations is* $4k|E|$ *and* $2k|E|$, *respectively. The number of messages of* discrete DPOP, AC-DPOP, *and* CAC-DPOP *is* $2|\mathbf{X}|$.

Proof. HCMS has the same number of messages as that of the Max-Sum algorithm [8]. Every edge of the constraint graph has two variable nodes and one function node and, thus, it takes 4 messages per edge in one iteration. The total number of messages in HCMS is thus $4k|E|$. On the other hand, C-DSA requires 2 messages per edge in one iteration and, thus, requiring $2k|E|$ messages in total.

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

Theorem 5.4. *The message size complexity of* discrete DPOP, AC-DPOP *and* CAC-DPOP *is* $O(d^w)$, $O((d|\mathbf{X}|)^w)$, *and* $\max\{|\mathbf{A}|, k\}$, *respectively, where* d *is the number of points used by each agent to discretize the domain of its separator agents,* w *is the induced width of the pseudo-tree, and* k *is the number of clusters used by CAC-DPOP.*

Proof. For DPOP, the message size complexity is $O(d^w)$ [31]. For AC-DPOP, as the values of an agent are "moved" by their children and pseudo-children, in the worst case, all the values are unique and the maximum number of such values is $O(d|\mathbf{X}|)$. The message sizes are then similar to discrete DPOP with $O(d|\mathbf{X}|)$ values per agent. Therefore, its message size complexity is $O((d|\mathbf{X}|)^w)$. For CAC-DPOP, the message size complexity of UTIL messages is $O(k)$ since only the utilities of the centroids of k clusters are sent. And the message size complexity of VALUE messages is $O(|\mathbf{A}|)$, such as in a fully-connected graph where an agent sends the values of every agent from the root of the pseudo-tree down to itself in a VALUE message to its child. Therefore, the message complexity of the algorithm is the $O(\max\{|\mathbf{A}, k\})$. □

6 EXPERIMENT RESULTS

We empirically evaluate *EC-DPOP*, *AC-DPOP*, *CAC-DPOP*, and *C-DSA*[3] against (discrete) *DPOP* and *HCMS* on both random trees and random graphs. We adapt the (discrete) DPOP algorithm to solve C-DCOPs by discretizing the continuous domain into discrete representative points.

We measure the quality of solutions, simulated runtimes [34] as well as the number of messages taken by the algorithm. Since HCMS and C-DSA are iterative algorithms that may take a long time and a large number of messages before converging, in order for fair comparisons, we initially planned to terminate the two algorithms after it sends as many messages as the DPOP-variants. However, in a single iteration, HCMS requires more messages than the DPOP-variants, and C-DSA requires the exact number of messages as the DPOP-variants. We thus let HCMS and C-DSA terminate after one iteration. We did not report the actual number of messages since they could be trivially computed via Theorem 5.3.

Tables 1 and 2 show the reported solution qualities in a unit of 1,000 and simulated runtimes in milliseconds and seconds (ending with *s*) on random trees and graphs, respectively, where we vary the number of agents $|\mathbf{A}|$ and every algorithm discretizes the domains of variables into three points. We also vary the number of times AC-DPOP and CAC-DPOP agents "move" a point (by calling Eqs. (13) or (14)) from 5 to 20. Tables 3(a) and 3(b) show the results on random trees and graphs, respectively, where we set the number of agents $|\mathbf{A}|$ to 20 and vary the number of discrete points from 1 to 9. In all our experiments, we set the domain of each agent to be in the range $[-100, 100]$. We generate utility functions that are binary quadratic functions, where the signs and coefficients are randomly chosen. Our experiments were performed on a 2.10GHz machine with 8GB of RAM. Results are averaged over 20 runs, each with a timeout of 30 minutes.

Random Trees: We omit the results of CAC-DPOP from Table 1 since it finds identical solutions to AC-DPOP on trees – there is no need to perform any clustering on trees since an agent does not receive utilities for value combinations of its parent from its children since there are no backedges in the pseudo-tree.

Not surprisingly, EC-DPOP finds the best solution since it is an exact algorithm. However, it could only solve the smallest of instances – due to memory limitations, the agents could not store the necessary number of piecewise functions to accurately represent the utility functions after adding functions and projecting out

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

Table 1: Varying the Number of Agents on Random Trees with Three Initial Discrete Points

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

Table 2: Varying the Number of Agents on Random Graphs with $p_1 = 0.2$ and Three Initial Discrete Points

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

Table 3: Varying the Number of Discretized Points

(a) Random Trees with $|\mathbf{A}| = 20$

Points	HCMS (time)	DPOP (time)	AC-DPOP (time)
1	0 (174)	0 (238)	254 (572)
3	306 (190)	541 (270)	870 (1.4s)
9	554 (291)	990 (369)	1133 (1.1s)

(b) Random Graphs with $|\mathbf{A}| = 20$ and $p_1 = 0.2$

HCMS (time)	DPOP (time)	AC-DPOP (time)	CAC-DPOP (time)
0 (263)	15 (220)	428 (613)	431 (1.5s)
345 (308)	865 (6.3s)	1285 (17.6s)	1017 (2.5s)
706 (552)	—	—	1272 (1185.9s)

variables. In general, AC-DPOP finds better solutions than DPOP, C-DSA and HCMS but at the cost of higher runtimes. AC-DPOP finds better solutions than DPOP because AC-DPOP spends more time on updating the value of representative points before propagating up the pseudo-tree. In contrast, the values chosen by DPOP is fixed from the start. HCMS performs poorly because a single iteration is insufficient for it to converge to a good solution. Interestingly, a single iteration is sufficient for C-DSA to find solutions that are comparable in quality to those found by DPOP. Additionally, as expected, the quality of solutions found by AC-DPOP improves with increasing number of times points are "moved" by the algorithm.

We omit the results of EC-DPOP from Table 3(a) as it failed to solve these instances and we omit the results of CAC-DPOP because it finds identical solutions to AC-DPOP on trees. We do not include C-DSA in the experiment because it does not discretize the domains. Not surprisingly, the quality of solutions found by all the three algorithms and their runtimes increase with increasing number of points. The reason is that the agents can more accurately represent the utility function with more points.

Random Networks: The trends in Table 2 are similar to those in random trees, except that CAC-DPOP finds solutions with qualities between that of AC-DPOP and DPOP. The reason is that CAC-DPOP clusters the points into k clusters and only propagate a representative point from each cluster. Therefore, the k points represent the utility functions less accurately than the full number of unclustered points that AC-DPOP uses. However, this reduced number of points propagated also improves the scalability of CAC-DPOP, where it is able to solve problems larger problems than AC-DPOP and DPOP.

The trends in Table 3(b) are again similar to that in random trees, except that both AC-DPOP and DPOP ran out of memory with 9 points. Interestingly, CAC-DPOP also finds better solutions than AC-DPOP when they use only 1 point.

7 CONCLUSIONS

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

ACKNOWLEDGMENTS

Hoang and Yeoh are partially supported by NSF grants 1812619 and 1838364, Yokoo is partially supported by JSPS KAKENHI grant JP17H00761, and Rabinovich is supported by Singapore MoE AcRF Tier-1 RG24/18 (S).

REFERENCES

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

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

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

[4] Liel Cohen, Rotem Galiki, and Roie Zivan. 2020. Governing Convergence of Max-Sum on DCOPs through Damping and Splitting. *Artificial Intelligence* 279 (2020).

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

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

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

[8] Alessandro Farinelli, Alex Rogers, Adrian Petcu, and Nicholas Jennings. 2008. Decentralised Coordination of Low-Power Embedded Devices Using the Max-Sum Algorithm. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 639–646.

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

[10] Ferdinando Fioretto, Enrico Pontelli, William Yeoh, and Rina Dechter. 2018. Accelerating Exact and Approximate Inference for (Distributed) Discrete Optimization with GPUs. *Constraints* 23, 1 (2018), 1–43.

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

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

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

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

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

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

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

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

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

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

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

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

[23] Akshat Kumar, Boi Faltings, and Adrian Petcu. 2009. Distributed constraint optimization with structured resource constraints. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 923–930.

[24] Tiep Le, Tran Cao Son, Enrico Pontelli, and William Yeoh. 2017. Solving Distributed Constraint Optimization Problems with Logic Programming. *Theory and Practice of Logic Programming* 17, 4 (2017), 634–683.

[25] James MacQueen. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Berkeley Symposium on Mathematical Statistics and Probability*. 281–297.

[26] Rajiv Maheswaran, Milind Tambe, Emma Bowring, Jonathan Pearce, and Pradeep Varakantham. 2004. Taking DCOP to the Real World: Efficient Complete Solutions for Distributed Event Scheduling. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 310–317.

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

[28] Pragnesh Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. 2005. ADOPT: Asynchronous Distributed Constraint Optimization with Quality Guarantees. *Artificial Intelligence* 161, 1–2 (2005), 149–180.

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

[30] Brammert Ottens, Christos Dimitrakakis, and Boi Faltings. 2017. DUCT: An Upper Confidence Bound Approach to Distributed Constraint Optimization Problems. *ACM Transactions on Intelligent Systems and Technology* 8, 5 (2017), 69:1–69:27.

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

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

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

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

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

[36] Tamir Tassa, Tal Grinshpoun, and Roie Zivan. 2017. Privacy Preserving Implementation of the Max-Sum Algorithm and its Variants. *Journal of Artificial Intelligence Research* 59 (2017), 311–349.

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

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

[39] William Yeoh and Makoto Yokoo. 2012. Distributed Problem Solving. *AI Magazine* 33, 3 (2012), 53–65.

[40] Zhepeng Yu, Ziyu Chen, Jingyuan He, and Yancheng Deng. 2017. A Partial Decision Scheme for Local Search Algorithms for Distributed Constraint Optimization Problems. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 187–194.

[41] Weixiong Zhang, Guandong Wang, Zhao Xing, and Lars Wittenberg. 2005. Distributed Stochastic Search and Distributed Breakout: Properties, Comparison and Applications to Constraint Optimization Problems in Sensor Networks. *Artificial Intelligence* 161, 1–2 (2005), 55–87.

[42] Roie Zivan, Tomer Parash, Liel Cohen, Hilla Peled, and Steven Okamoto. 2017. Balancing Exploration and Exploitation in Incomplete Min/Max-sum Inference for Distributed Constraint Optimization. *Journal of Autonomous Agents and Multi-Agent Systems* 31, 5 (2017), 1165–1207.

[43] Roie Zivan, Tomer Parash, Liel Cohen-Lavi, and Yarden Naveh. 2020. Applying Max-Sum to Asymmetric Distributed Constraint Optimization Problems. *Journal of Autonomous Agents and Multi-Agent Systems* 34, 1 (2020), 1–29.

[44] Roie Zivan, Harel Yedidsion, Steven Okamoto, Robin Glinton, and Katia Sycara. 2015. Distributed Constraint Optimization for Teams of Mobile Sensing Agents. *Journal of Autonomous Agents and Multi-Agent Systems* 29, 3 (2015), 495–536.