Pseudo-tree Construction Heuristics for DCOPs and Evaluations on the *ns-2* Network Simulator

Atena M. Tabakhi*, Reza Tourani†, Francisco Natividad†, William Yeoh*, Satyajayant Misra†
*Department of Computer Science and Engineering, Washington University in St. Louis
{amtabakhi, wyeoh}@wustl.edu
†Department of Computer Science, New Mexico State University
{rtourani, fnativid, misra}@cs.nmsu.edu

***Abstract*—Distributed Constraint Optimization Problems (DCOPs) are commonly used to model multi-agent coordination problems. However, empirical evaluations of DCOP algorithms are typically done in simulation under the assumption that the communication times between all pairs of agents are identical, which is unrealistic in many real-world applications. In this paper, we investigate the impact of empirically evaluating a DCOP algorithm under the assumption that communication times between pairs of agents can vary and propose the use of *ns-2*, a de-facto simulator used by the computer networking community, to simulate the communication times. Additionally, we introduce heuristics that exploit the non-uniform communication times to speed up DCOP algorithms that operate on pseudo-trees.**

I. 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 [1, 2]. They are well-suited for modeling multi-agent coordination problems where the primary interactions are between local subsets of agents [3, 4, 5, 6, 7]. Unfortunately, empirical evaluations of DCOP algorithms are typically done in simulation under the assumption that the communication time between any pair of agents is identical for all pairs. In many coordination problems, this assumption is unrealistic. For example, in a sensor network problem, communication between pairs of sensors may depend on factors such as the distance between the sensors and the topology of the environment. Therefore, in this paper, we extend the DCOP model to include communication-related information, specifically, the communication times for each constraint. This model then allows us to investigate the impact of empirically evaluating a DCOP algorithm under different assumptions for the communication times. We perform this evaluation with DPOP [8], one of the more popular complete DCOP algorithms that has been widely extended by the DCOP community.

Another common practice in the DCOP community is to use the simulated runtime metric [9] as a proxy to measure distributed runtime. The community thus implicitly assumes that there is a high correlation between simulated runtimes and actual distributed runtimes. In this paper, we empirically verify this assumption, where we specify the communication time for each message, defined as the time it takes for the source node to send a message until the sink node receives it completely, within the simulated runtime metric. However, it is often difficult to estimate communication times accurately as it depends on a combination of factors: communication protocols, communication medium, interference model, transmission delay, propagation delay, queuing delay, and processing delay [10]. Thus, we use *Network Simulator 2* (*ns-2*) [11], a de-facto simulator used by the computer networking community [12, 13, 14], to simulate transmission of messages and measure the communication time of those transmissions. Using these communication times and actual computation times, we compute "actual" runtimes, and experimentally verify the implicit assumption made by DCOP researchers. To the best of our knowledge, such an experiment has not been done with realistic networking simulators before.

Finally, we also propose heuristics that can be used to exploit the non-uniform communication times in a DCOP to speed up algorithms that operate on pseudo-trees. Complete and correct DCOP algorithms [1, 8, 15] typically require some form of ordering for the variables in the DCOP and use pseudo-trees for this purpose. Additionally, some approximate DCOP algorithms [16, 17] also use pseudo-trees to order the DCOP variables. As such, our heuristics can improve a large class of existing DCOP algorithms. We evaluate our heuristics in random graphs and power network topologies with communication times that depend on physical distances that are sampled from two distributions (uniform and Gaussian). Our experimental results show that (*i*) the runtime of DPOP is positively correlated with the depth of its pseudo-tree and (*ii*) our heuristics find pseudo-trees with smaller depths than the existing max-degree heuristic by up to 20%.

II. 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*. 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 *constraints*, each defined over a mixed set of decision variables: $f_i : \bigtimes_{x \in \mathbf{x}^{f_i}} D_x \rightarrow \mathbb{R}^+ \cup \{\perp\}$, where $\mathbf{x}^{f_i} \subseteq \mathbf{X}$ is the *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.

- $\alpha : \mathbf{X} \rightarrow \mathbf{A}$ is a function that associates each decision variable to one agent.

Following common conventions, we restrict our attention to binary constraints and assume that α is a bijection: each agent controls exactly one variable. Thus, we will use the terms "variable" and "agent" interchangeably and assume that $\alpha(x_i) = a_i$. However, our approach can be easily generalized to the unrestricted version, as we demonstrate with our Customer-Driven Microgrid (CDMG) application domain in our experiments.

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 $\mathbf{F}(\sigma) = \sum_{f \in \mathbf{F}, \mathbf{x}^f \subseteq \mathbf{x}_\sigma} f(\sigma)$[1] is the sum of the utilities across all the applicable constraints in σ. A solution σ is *complete* if $\mathbf{x}_\sigma = \mathbf{X}$. The goal is to find an optimal complete solution $\mathbf{x}^* = \text{argmax}_{\mathbf{x}} \mathbf{F}(\mathbf{x})$.

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. Edges of G that are in E_T are called *tree edges* and edges that are not are called *backedges*. Tree edges connect a node with its parent and its children, while backedges connect a node with its *pseudo-parents* and its *pseudo-children*. The *separator* of agent a_i (denoted by $sep(a_i)$) is the set of ancestors of a_i that are directly connected (via tree edges or backedges) with a_i or with descendants of a_i. 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; and $P(a_i)$, $C(a_i)$, $PP(a_i)$, and $PC(a_i)$ to denote the parent, the set of children, the set of pseudo-parents, and the set of pseudo-children of agent a_i in the pseudo-tree.

Distributed DFS: Since constructing optimal pseudo-trees is NP-hard, one typically uses greedy approaches like the Distributed DFS algorithm [18] to construct pseudo-trees. This algorithm is commonly used in many implementations of complete DCOP algorithms including those within the DCOPolis [9] and FRODO [19] repositories. Both of these repositories include a large number of common DCOP algorithms and are frequently used by DCOP researchers.

The Distributed DFS algorithm operates as follows: It assigns a score to each variable according to some heuristic function. Then, it selects a variable with the largest score as the root of the pseudo-tree. Once the root is selected, it initiates a DFS-traversal of the constraint graph, greedily adding the neighboring variable with the largest score as the child of the current variable. This process repeats until all variables in the constraint graph are added to the pseudo-tree.

The variables' scores can be chosen arbitrarily. A commonly used heuristic is the max-degree heuristic $h(x_i)$:

$$h(x_i) = |N(x_i)| \tag{1}$$

[1]With a slight abuse of notation, we use $\mathcal{F}$ to denote the set of constraints as well as the overall utility of the DCOP.

which sets a variable's score to its number of neighbors. In situations where multiple variables have the same maximal score, the algorithm breaks ties according to a different heuristic, such as the variable-ID heuristic, which assigns to each variable a score that is equal to its unique ID. In our experiments, we use the max-degree heuristic and break ties with the variable-ID heuristic as the benchmark heuristic.

DPOP: The *Distributed Pseudo-tree Optimization Procedure* (*DPOP*) [8] is a complete *inference algorithm* composed of three phases:

- *Pseudo-tree Construction Phase:* Agents coordinate to build a pseudo-tree using Distributed DFS.
- *UTIL Propagation Phase:* 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, and sends the optimal utilities up to its parent in UTIL messages. The agent computes the optimal sum of utilities 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 Phase:* Each agent, starting from the pseudo-tree root, determines the optimal value for its variables and sends the optimal values down to its children in VALUE messages. The agent determines the optimal value of its variables based on its UTIL computations and the value of variables in the VALUE messages.

Network Simulator 2: Building a test-bed for performance analysis is sometimes not feasible. In addition, the number of agents in real-world applications often increases with the complexity of the problems; each with various configuration for performance analysis. For these reasons, researchers have created a simulation model of existing network topologies to study the behavior of agents. *Network Simulator 2* (*ns-2*) is a discrete event-driven network simulator that resembles actual network behavior with the ability to support a variety of communication protocols [11].

In our experiments, we use *ns-2* version 2.35. The communication between nodes occurs through a wireless communication channel with omni-directional antennas and uses the Two Ray Ground radio propagation model with a 11Mb/s communication bandwidth. The simulator uses the Medium Access Control (MAC) IEEE 802.11 protocol in the data link layer; a static and fixed routing protocol in the network layer, where the routing tables of all the nodes involved in the communication are loaded with the shortest paths to the destination; the Transmission Control Protocol (TCP) in the transport layer; and the Constant Bit Rate (CBR) application with a bit rate of 0.1Mb/s and packet size of 500 bytes in the application layer.

III. Motivating Application

We use the comprehensive *Customer-Driven Microgrid* (CDMG) optimization problem described by Gupta *et al.* [20] to motivate our work. In this problem, there is a neighborhood

of homes, each capable of generating energy, consuming energy, and transmitting energy to its neighboring homes. Each home is represented by an agent and its generation, consumption, and transmission capabilities are variables controlled by that agent. The domain of these variables are thus the range of energy that can be generated, consumed, and transmitted. Two neighboring agents are constrained with one another if they can transmit energy to each other. Furthermore, there are also constraints that enforce Kirchhoff's law, that the sum of energy flowing into an agent and the energy produced by that agent must equal the sum of energy consumed by that agent and the amount of energy flowing out of that agent.

Like in most DCOP literature, Gupta *et al.* [20] also assume that the communication time is uniform across all homes. However, in practice, the communication time may not be uniform and depends largely upon the underlying communication topologies and communication technologies used by the agents. For example, most homes today are equipped with smart meters that are used to measure the amount of energy flowing into and out of the homes. These smart meters communicate over a wireless network to an aggregator, which then transmits the information to the energy provider potentially through other aggregators.

Researchers have proposed a hierarchical architecture, where homes are connected to aggregators in a star topology and aggregators are connected to each other in a mesh topology [21]. It is argued that this configuration is necessary for the microgrid to meet reliability, self-configuring, and self-healing requirements of smart grid applications. Thus, we also assume this communication model, and the communication between any two agents must go through at least one aggregator and the communication times depend on the distance between the agents and the aggregator.

IV. Variable Communication Times

We extend the DCOP model to include communication-related information, specifically, the communication times for each constraint. Therefore, this new DCOP is defined by a tuple $\langle \mathbf{A}, \mathbf{X}, \mathbf{D}, \mathbf{F}, \mathbf{C}, \alpha \rangle$, where $\mathbf{A}$, $\mathbf{X}$, $\mathbf{D}$, $\mathbf{F}$, and α are as described for regular DCOPs; and $\mathbf{C} = \{c_i\}_{i=1}^{m}$ is the set of communication times, where $c_i \in \mathbf{C}$ specifies the communication time for agents in the scope $\mathbf{x}^{f_i}$ of constraint $f_i \in \mathbf{F}$ to communicate with one another.

Definition *1 (Communication Time):* The communication time for a constraint is the time it takes for the source node to send a message until the sink node receives it completely, where both the source and sink nodes are in the scope of the constraint.

In other words, we assume that agents can only communicate with neighboring agents that they share constraints with, and the time of those communication is specific to each constraint. Each communication time c_i can either be a constant, indicating that there is no uncertainty in the communication time, or a probability distribution, indicating that the actual communication time is sampled from that distribution. In our experiments, we investigate two distributions – uniform and Gaussian. The objective is still identical to that of DCOPs, to find an optimal complete solution that maximizes the sum of utilities over all constraints.

Aside from the common assumptions that messages sent are never lost and they are received in the same order that they were sent, all off-the-shelf DCOP algorithms do not make any additional assumption on the communication times. Therefore, they can be used to solve our problem with non-uniform communication times. However, the new communication time specifications may impact the efficiency of DCOP algorithms in several ways compared to when they run on problems with uniform communication times.

Complete DCOP algorithms typically require that the variables in the problem be ordered according to some complete ordering, in which case the variables are ordered into a chain, or some partial ordering, in which case the variables are ordered into a pseudo-tree; a large number of complete algorithms, including DPOP [8], ADOPT [1], and their many variants, operate on pseudo-trees. Additionally, some incomplete algorithms such as Distributed UCT [16] and Distributed Gibbs [17] also operate on pseudo-trees. As such, the properties of these algorithms are highly dependent on the properties of the underlying pseudo-tree. For example, DPOP's memory requirement and message size complexity is $O(\exp(w^*))$, where w^* is the induced width of the pseudo-tree, and its required number of message is $O(d^*)$, where d^* is the depth of the pseudo-tree. On the other hand, ADOPT's memory requirement and message size complexity is $O(d^*)$, but its required number of messages is $O(\exp(d^*))$. Since communication times are no longer uniform across all edges of the pseudo-tree, we generalize the definition of depth of pseudo-trees to a *generalized depth* definition:

Definition *2 (Generalized Depth):* The generalized depth of a pseudo-tree is the largest sum of communication times $c_i \in \mathbf{C}$ across all constraints over all branches of the pseudo-tree. More specifically, the generalized depth $\hat{d}^*$ is defined recursively by:

$$\hat{d}^* = \hat{d}_{root} \tag{2}$$

$$\hat{d}_{x_i} = \max_{f_k \in \mathbf{F}: \{x_i, x_j\} \in \mathbf{x}^{f_k} \wedge x_j \in C(x_i) \cup PC(x_i)} c_k + \hat{d}_{x_j} \tag{3}$$

where f_k is the constraint between x_i and its child or pseudo-child x_j, c_k is the communication time associated with that function, and $\hat{d}_{x_j}$ is the generalized depth of the sub-tree rooted at x_j.

It is straightforward to see that this generalized depth definition subsumes the previous depth definition for pseudo-trees with uniform communication times of 1.

A. Impact on DPOP

Using this definition, we investigate the relationship between the runtimes of algorithms and the generalized depth of their underlying pseudo-trees. We use DPOP as a case study in this paper. Specifically, we investigate two types

Constraint Density p_1	0.2	0.3	0.4	0.5	0.6	0.7
Maximum Separator Size	3.5	5.8	7.3	8.5	9.6	10.2
Largest Message Size (Bytes)	8,165.8	120,056.5	1,068,614.1	5,127,311.2	14,953,595.6	14,953,599.8
Largest Number of Packets	16.6	240.6	2,137.7	10,255.4	29,907.8	29,907.7

TABLE I: Maximum Separator Size, Largest Message Size, and Largest Number of Packets

of runtimes: (1) *Simulated Runtimes* [9], a commonly used runtime metric within the DCOP community, which assume that the communication times between all pairs of agents are identical; and (2) *"Actual" Runtimes*, which are computed via *Network Simulator 2.0* (*ns-2*) [11], a de-facto simulator used by the computer networking community [12, 13, 14].

We generate problem instances with random graph topologies [22]. We vary the number of variables $|\mathbf{X}| = \{10, 20\}$[2] and the constraint density $p_1 = \{0.2, \ldots, 0.7\}$,[3] and set the domain size of all variables $|D_i| = 3$. For all these instances, we assume that the communication time of each constraint $f_i \in \mathbf{F}$ to be the product

$$c_i = C \cdot d_i \tag{4}$$

where C is a constant and d_i is the physical distance between the two variables that are in the scope $\mathbf{x}^{f_i}$ of the constraint. We sample the x- and y-coordinates of each variable from two possible truncated distributions – uniform and Gaussian $\mathcal{N}(50, 25)$ – from the range $[1, 100]$. In other words, the variables are randomly distributed over a 100×100 square meter area. We generate 20 instances for each configuration, resulting in 160 instances in total, for this experiment.

We solve these problems using DPOP implemented on the FRODO simulation framework [19], and store the computation time of each agent in the UTIL and VALUE propagation phases as well as the size of messages that the agents transmitted to each other. Using the assumed communication times computed using Equation 4, we measure the simulated runtimes of DPOP in these runs.[4]

We then represent these instances in *ns-2*, where the two variables in the scope $\mathbf{x}^{f_i}$ of the same function f_i are physically separated *exactly* by the distance d_i. In order to simulate the *exact* same trace of execution of the agents in *ns-2*, we need to set two parameters in *ns-2*, namely the computation time of each agent and the size of messages sent by each agent. Both of these parameters are stored from the runs on FRODO earlier. To mimic an agent's computation time in *ns-2*, we apply an equal delay at the agent before it starts its communication. For example, assume that the computation time of an agent is 10ms during the UTIL propagation phase. Then, in *ns-2*, after it receives all messages from its children, we enforce that it waits for 10ms before it sends its UTIL message to its parent. We also set the number of packets an agent needs to send in its communications according to the size of its messages as determined by FRODO. As we set the TCP packet size to 500 bytes, the number of packets sent for each message is $\lceil \frac{\text{message size}}{500} \rceil$. While packets may be dropped due to a variety of factors such as congestion, the TCP protocol ensures that a dropped packet will be resent through the use of acknowledgement (ACK) messages. When a variable correctly receives a packet, it sends an ACK to the sender. The sender considers the packet to be lost if it does not receive an ACK before its timeout. In this case, the sender resends the lost packet.

Definition *3 ("Actual" Runtime):* The "actual" runtime of DPOP is the duration between the time the *ns-2* simulation starts UTIL propagation until the time the last agent finishes processing its message in VALUE propagation.

In summary, this measured runtime is similar to the measured runtime in FRODO except that the communication time between two agents is now determined by the *ns-2* simulator instead of the assumed communication times c_i. The communication time determined by *ns-2* is dependent on the message size, the distance between the agents, the congestion in the network, and the protocols in the various networking layers. For example, an agent with many children in the pseudo-tree may receive messages from all of them simultaneously, resulting in congestion and packet drops. *ns-2* is able to simulate the message delays automatically.

B. Theoretical Results

We now describe the theoretical runtime complexities measured using the two runtime metrics mentioned above. Let b denote the set of agents along a branch of a pseudo-tree and $\mathbf{B}$ denote the set of all branches in the pseudo-tree.

Theorem *1:* The simulated runtime of DPOP is $\max_{b \in \mathbf{B}} \sum_{a_i \in b} O(\exp(|sep(a_i)|))$.

The runtime of DPOP is dominated by its runtime in the UTIL phase, where the agents along each branch of the pseudo-tree sequentially compute their UTIL tables and sends them up to their respective parents. Therefore, along each branch $b \in \mathbf{B}$ of the pseudo-tree, the runtime along that branch is

$$\sum_{a_i \in b} O(\exp(|sep(a_i)|)) + O(d_i) \tag{5}$$

where the first term is the time to compute the UTIL table and the second term is the time to send the table to the parent.[5] This sum is then simplified to $\sum_{a_i \in b} O(\exp(|sep(a_i)|))$.

[2]We did not generate larger instances as DPOP cannot solve large problems due to memory limitations.

[3]p_1 is defined as the ratio between the number of binary constraints in the problem and the maximum possible number of binary constraints in the problem.

[4]We set $C = 1$ millisecond per meter for all the experiments in this paper.

[5]We use d_i to refer to the physical distance between a_i and its parent in the pseudo-tree, which is used to calculate the communication time using Equation 4.

Constraint Density p_1	0.2	0.3	0.4	0.5	0.6	0.7
Correlation of "Actual" and Simulated Runtimes	0.74	0.84	0.97	0.91	0.94	0.95
Correlation of Depth and Simulated Runtime	0.99	0.97	0.82	0.66	0.69	0.67
Correlation of Depth and "Actual" Runtime	0.74	0.74	0.74	0.62	0.63	0.59

TABLE II: Correlations

Theorem *2:* The "actual" runtime of DPOP is $\max_{b\in\mathbf{B}} \sum_{a_i\in b} O(\exp(|sep(a_i)|) \cdot d_i)$.

Using the similar argument as above for the simulated runtime case, the runtime of the UTIL phase along a branch $b \in \mathbf{B}$ is

$$\sum_{a_i\in b} O(\exp(|sep(a_i)|)) + O(\exp(|sep(a_i)|) \cdot d_i) \qquad (6)$$

which simplifies to $\sum_{a_i\in b} O(\exp(|sep(a_i)|) \cdot d_i)$. The communication time is exponential in the separator size according to a preliminary experiment, where we observe that the largest message size and number of packets grow exponentially with the maximum separator size (Table I).

C. Experimental Results

We empirically evaluate the correlation between "actual" runtimes, simulated runtimes, and pseudo-tree depths on the instances described in Section IV-A. Table II tabulates the correlation factors according to the Pearson correlation metric.

Correlation between "Actual" and Simulated Runtimes: There is a positive correlation between the two runtimes, and the correlation increases as p_1 increases. The reason is that the separator size increases with p_1 and, thus, the simulated computation runtime increasingly dominates the simulated communication runtime. Therefore, the total simulated runtime becomes increasingly proportional to $\max_{b\in\mathbf{B}} \sum_{a_i\in b} O(\exp(|sep(a_i)|))$ and increasingly correlated with the "actual" runtime, which is also proportional to $\max_{b\in\mathbf{B}} \sum_{a_i\in b} O(\exp(|sep(a_i)|))$.

Correlation of Both Runtimes and Pseudo-tree Depth: Both runtimes are positively correlated with the depth, and decreases as p_1 increases. The simulated runtimes also have a higher correlation than the "actual" runtimes.

The positive correlations are due to the communication runtimes. For example, the simulated communication runtime is $\sum_{a_i\in b} O(d_i)$ for the branch b with the longest simulated (computation and communication) time, which is proportional to $\sum_{a_i\in b} O(c_i)$ (Equation 4) and is a close approximation of the pseudo-tree depth (Definition 2). As p_1 increases, the simulated communication runtimes become increasingly dominated by the simulated computation runtimes, thereby decreasing the correlation with the depth.

On the other hand, the "actual" communication runtime is $\sum_{a_i\in b} O(\exp(|sep(a_i)|) \cdot d_i)$, which is also positively correlated with the depth for the same reason as above. However, it is a weaker correlation due to the exponential factor in the communication runtimes. Due to the positive correlations, we use pseudo-tree depths as the proxy for DPOP runtimes in the remainder of the paper.

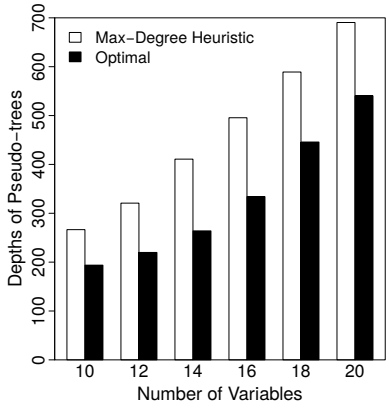

Fig. 1: Optimal and Default Pseudo-tree Depths

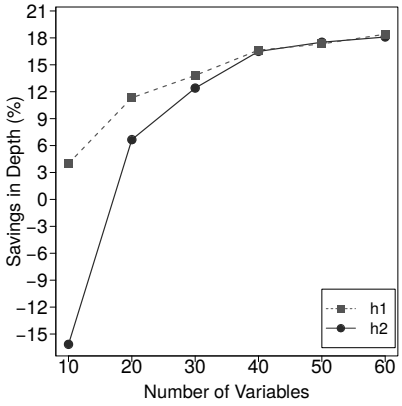

Fig. 2: Random Graphs (Uniform Distribution)

V. Pseudo-tree Construction Heuristics

In this section, we seek to develop heuristics that exploit the non-uniform communication times to construct pseudo-trees with small depths. We first evaluate the potential improvement to existing pseudo-trees constructed using the default max-degree heuristic (Equation 1). Figure 1 shows a depth comparison between the default pseudo-trees and the optimal pseudo-trees. These pseudo-trees are for problems, where we vary the number of variables $|\mathbf{X}|$ from 10 to 20, set the constraint density p_1 to 0.3, and choose distances with a truncated Gaussian $\mathcal{N}(50, 25)$ distribution from the range $[1, 100]$ and define the communication time c_i with these distances (Equation 4). Due to memory and time constraints, we could not find optimal pseudo-trees for larger problems. One can observe that the difference between the depth of the default pseudo-tree and the optimal depth increases as the number of variables increases, thereby indicating that there is a larger room for improvement in larger problems.

We thus introduce several heuristics that can be used by Distributed DFS to create pseudo-trees with small depths:

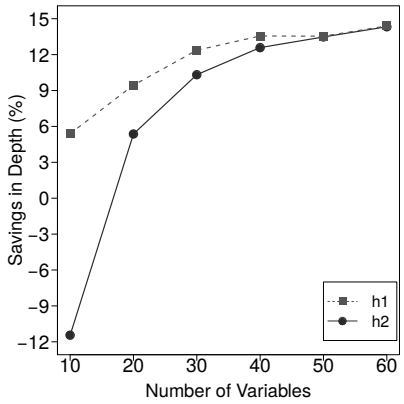

Fig. 3: Random Graphs (Gaussian Distribution)

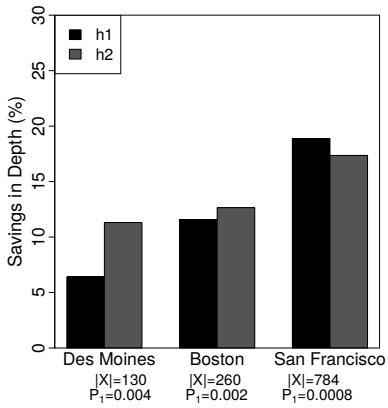

Fig. 4: CDMGs

- The *max-weighted-sum* (mws) heuristic h_{mws}:

$$h_{mws}(x_i) = \sum_{f_k \in \mathbf{F}: \{x_i, x_j\} \in \mathbf{x}^{f_k} \wedge x_j \in N(x_i) \setminus [\{P(x_i)\} \cup PP(x_i)]} c_k \quad (7)$$

It sums the communication times between variable x_i and all its neighbors x_j that are not yet part of the pseudo-tree. We do not consider neighbors that are already part of the pseudo-tree for the following reason: From the depth definition in Equation 3, the depth of a variable x_p is the largest sum of the depth of a (pseudo) child x_q and the communication time with that (pseudo) child over all children and pseudo-children. Therefore, once the variable x_p is already chosen to be part of the pseudo-tree, it is desirable for its neighbor x_q that has a large communication time with x_p to be a pseudo-child instead of a regular child. The reason is that the farther a variable is from the root, generally, the smaller the depth of that variable. We thus ignore neighbors that are already part of the pseudo-tree in this heuristic as well as the two heuristics below.

- The *max-weighted-average* (mwa) heuristic h_{mwa}:

$$h_{mwa}(x_i) = \frac{h_{mws}(x_i)}{|N(x_i) \setminus [\{P(x_i)\} \cup PP(x_i)]|} \quad (8)$$

It is identical to the previous h_{mws} heuristic except that it averages the values over the number of neighboring variables that are not yet part of the pseudo-tree.

		Root Variable		
		h_{mws}	h_{mwa}	h_{mus}
Non-Root Variables	h_{mws}	h_1	h_2	h_3
	h_{mwa}	h_4	h_5	h_6
	h_{mus}	h_7	h_8	h_9

TABLE III: Pseudo-tree Construction Heuristics

- The *max-unweighted-sum* (mus) heuristic h_{mus}:

$$h_{mus}(x_i) = |N(x_i) \setminus [\{P(x_i)\} \cup PP(x_i)]| \quad (9)$$

It is identical to the default max-degree heuristic except that it considers only neighboring variables that are not yet part of the pseudo-tree.

As the heuristic used to select the root of the pseudo-tree can differ from the heuristic used to select non-root variables, we use all nine combinations of the three heuristics above. Table III tabulates these heuristics.

A. Experimental Results

We empirically evaluate our nine heuristics against the default max-degree heuristic (Equation 1) on (a) random graphs [22] and (b) graphs motivated by our motivating CDMG application. We measure the depths of pseudo-trees constructed by the heuristics and use them as the proxy for the runtimes of DPOP. Results are averaged over 500 instances, except for CDMG results, which are averaged over 50 instances only due to time constraints. All experiments were run on a machine with an Intel Core i7-3770 CPU at 3.40GHz and 16GB of RAM.

Random Graphs: We vary the number of variables $|\mathbf{X}| = \{10, 20, 30, 40, 50, 60\}$ and set the constraint density $p_1 = 0.3$. For each configuration, we sample the physical distances d_i of the constraints from two possible truncated distributions – uniform and Gaussian $\mathcal{N}(50, 25)$ – from the range [1,100] and define the communication time c_i with these distances (Equation 4). Figures 2 and 3 show the results, where we plot the two best heuristics. Generally, the savings increase as the number of variables increases. The reason is because the number of possible pseudo-tree configurations increases as the number of variables increases. Thus, there is more room for improvement. The h_1 and h_2 heuristics converge to savings (≈18% with uniform and ≈15% with Gaussian distributions) indicating that heuristics that take communication times into account perform better than those that do not. Further, Table IV tabulates the depths of the pseudo-trees and both the "actual" and simulated runtimes of DPOP with those pseudo-trees constructed by our two best heuristics compared against the default max-degree heuristic.[6] These results show that the savings in pseudo-tree depths of the two heuristics translate to savings in both runtimes as well.

Customer-driven Microgrids (CDMGs): We sample neighborhoods in three cities in the United States (Des Moines, IA; Boston, MA; and San Francisco, CA) and estimate the

[6]These results are averaged over $|\mathbf{X}| = \{10, 20\}$ only as DPOP failed to scale for larger problems.

	Default max-degree heuristic	h_1	h_2
Pseudo-tree Depth	519.45	472.85	481
Simulated Runtime of DPOP (ms)	1427.57	1317.20	1320.46
"Actual" Runtime of DPOP (ms)	3585.15	3328.94	2995.24

TABLE IV: Random Graphs

density of houses in each city. The average density (in houses per square kilometers) is 718 in Des Moines, 1357 in Boston, and 3766 in San Francisco. For each city, we created a 200m$\times$200m grid, where the distance between intersections is 20m, and randomly placed houses in this grid until the density is the same as the sampled density. Each house is constrained with its immediate neighbors in the four cardinal directions of the grid. If the resulting graphs are disjoint, then for each pair of disjoint graphs, we find a pair of house that has the smallest distance between them and constrain them. Finally, we greedily placed aggregators, with a communication radius of 100m, in this grid until all houses are within the radius of at least one aggregator. Aggregators can then communicate with all homes and aggregators within its communication radius. Figure 4 shows the results, where the trends are similar to those in random graphs.

VI. Related Work

There is a very limited amount of work on the study of communication times in the context of DCOPs. Cruz et al. [23] recently investigated the importance of communication times in evaluation of DCOP algorithms by conducting experiments where agents are located physically apart in different machines connected by LAN. They observed that communication times are orders of magnitude larger than what is typically assumed in the DCOP community, thereby issuing a call of action to better investigate this area. Our research, in a large part, is in response to this call. The main differences between their work and ours is that they limit the number of agents in their experiments to six and they limit the constraint density p_1 to 0.5. In contrast, our experiments are with a larger number of agents and over a larger combination of configurations (e.g., different communication times, constraint densities, and graph topologies). Furthermore, we use wireless communication instead of wired communication, which is more common in applications such as our motivating CDMG application.

In terms of distributed constraint satisfaction problems (DCSPs), Zivan and Meisels [24] investigated the impact of message delays in those problems. The main difference with our work is that they introduce an Asynchronous Message Delay simulator (AMD) that measures the logical time of the algorithm run and does not capture a real transmission and communication protocol. In contrast, we investigate the effect of non-uniform message delays on DCOPs. The simulator we use provides a more accurate estimate of the message delays through the simulation of TCP, routing, and multicast protocols over wireless network scenarios. Fernàndez et al. [25] have also studied the impact of communication delays on DCSP algorithms and found that random delays can actually improve the performance and robustness of AWC. Wahbi and Brown [26] decoupled the communication graph with the underlying constraint graph of the problem and studied the effect of different communication graph topologies on ABT and AFC-ng. In their work, the communication load is measured by the number of transmission messages during the algorithm execution *(#transmission)* and the computation effort that takes the message delay into account which is measured by the average of the equivalent non-concurrent constraint checks *(#ncccs)* [27]. The main difference with our work is that they studied the effect of uniform message delays in terms of the number of messages transmitted and the number of constraint checks. In contrast, we study the effect of non-uniform delays, that is simulated with a more realistic communication protocols, in terms of the pseudo-tree depth. The pseudo-tree depth serves as a proxy for simulated runtimes, hence, in this paper, we propose methods to construct shorter pseudo-trees with the aim of speeding up the performance of large class of DCOP solvers. Unlike others, we use *ns-2* simulator to measure realistic communication latency. The simulator in this paper, models packet loss, re-transmissions due to packet drops, and network congestion that have been disregarded in recent studies.

Finally, Okimoto et al. [28] have also studied the effect of different variable-ordering heuristics for ABT in scale-free networks. We also propose different variable-ordering heuristics and study the effect of the proposed heuristics, specifically for the DPOP algorithm. The difference is that our proposed heuristics exploit the non-uniform communication times in a DCOP to speed up algorithms that operate on pseudo-trees.

VII. Conclusions

The existing DCOP model does not explicitly model communication times between agents. As a result, DCOP algorithms have typically assumed that communication times are identical for all pairs of agents, which can be unrealistic in many real-world applications. In this paper, we extend the DCOP model to include communication times for each constraint and incorporate these communication times within the simulated runtime metric. We also measure communication times through *ns-2* simulations, use it to compute "actual" runtimes, and show that these runtimes are positively correlated with simulated runtimes. Finally, we propose pseudo-tree construction heuristics that exploit the non-uniform communication times to find pseudo-trees that are up to 20% shorter than those constructed by the max-degree heuristic. These heuristics can thus be used to speed up a large class of DCOP algorithms that operate on pseudo-trees. We leave the study of the effect of the proposed heuristic on different algorithms as

well as integration with our efforts in smart homes in CDMGs [29, 30] to the future.

Acknowledgments

This research is partially supported by NSF grants 1345232 and 1550662. 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.

References

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

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

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

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

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

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

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

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

[9] E. Sultanik, R. Lass, and W. Regli. DCOPolis: a framework for simulating and deploying distributed constraint reasoning algorithms. In *Proceedings of the Distributed Constraint Reasoning Workshop*, 2007.

[10] J. Kurose and K. Ross. *Computer Networking: A Top Down Approach*. Pearson, 2012.

[11] S. McCanne and S. Floyd. ns–network simulator. http://nsnam.sourceforge.net/wiki/.

[12] J. Sommers and A. Moore. Scaling the practical education experience. In *Proceedings of the ACM SIGCOMM Education Workshop*, 2011.

[13] N. Rozhnova and S. Fdida. An effective hop-by-hop interest shaping mechanism for ccn communications. In *Proceedings of IEEE INFOCOM Workshops*, pages 322–327, 2012.

[14] N. E. Majd, S. Misra, and R. Tourani. Split-cache: A holistic caching framework for improved network performance in wireless ad hoc networks. In *Proceedings of IEEE GLOBECOM*, pages 137–142, 2014.

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

[16] B. Ottens, C. Dimitrakakis, and B. Faltings. DUCT: An upper confidence bound approach to distributed constraint optimization problems. In *Proceedings of AAAI*, pages 528–534, 2012.

[17] D. Thien Nguyen, W. Yeoh, and H. C. Lau. Distributed Gibbs: A memory-bounded sampling-based DCOP algorithm. In *Proceedings of AAMAS*, pages 167–174, 2013.

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

[19] T. Léauté, B. Ottens, and R. Szymanek. FRODO 2.0: An open-source framework for distributed constraint optimization. In *Proceedings of the Distributed Constraint Reasoning Workshop*, pages 160–164, 2009.

[20] S. Gupta, P. Jain, W. Yeoh, S. Ranade, and E. Pontelli. Solving customer-driven microgrid optimization problems as DCOPs. In *Proceedings of the Distributed Constraint Reasoning Workshop*, pages 45–59, 2013.

[21] W. Meng, R. Ma, and H. Chen. Smart grid neighborhood area networks: A survey. *IEEE Network*, 28(1):24–32, 2014.

[22] P. Erdös and A. Rényi. On Random Graphs I. *Publicationes Mathematicae Debrecen*, 6:290, 1959.

[23] F. Cruz, P. Gutierrez, and P. Meseguer. Simulation vs real execution in DCOP solving. In *Proceedings of the Distributed Constraint Reasoning Workshop*, 2014.

[24] R. Zivan and A. Meisels. Message delay and DisCSP search algorithms. *Annals of Mathematics and Artificial Intelligence*, 46(4):415–439, 2006.

[25] C. Fernàndez, R. Béjar, B. Krishnamachari, and C. Gomes. Communication and computation in distributed CSP algorithms. In *Proceedings of CP*, pages 664–679, 2002.

[26] M. Wahbi and K. N. Brown. The impact of wireless communication on distributed constraint satisfaction. In *Proceedings of CP*, pages 738–754, 2014.

[27] A. Chechetka and K. Sycara. No-commitment branch and bound search for distributed constraint optimization. In *Proceedings of AAMAS*, pages 1427–1429, 2006.

[28] T. Okimoto, A. Iwasaki, and M. Yokoo. Effect of DisCSP variable-ordering heuristics in scale-free networks. In *Proceedings Of PRIMA*, pages 166–180, 2012.

[29] F. Fioretto, W. Yeoh, and E. Pontelli. A multiagent system approach to scheduling devices in smart homes. In *Proceedings of AAMAS*, pages 981–989, 2017.

[30] A. M. Tabakhi, T. Le, F. Fioretto, and W. Yeoh. Preference elicitation for DCOPs. In *Proceedings of CP*, pages 278–296, 2017.