International Journal on Artificial Intelligence Tools
Vol. 27, No. 7 (2018) 1860008 (24 pages)

DOI: 10.1142/S0218213018600084

World Scientific
www.worldscientific.com

Communication-Sensitive Pseudo-Tree Heuristics for DCOP Algorithms

Atena M. Tabakhi and William Yeoh

Department of Computer Science and Engineering, Washington University in St. Louis
Saint Louis, Missouri, USA
{*amtabakhi, wyeoh*}*@wustl.edu*

Reza Tourani*, Francisco Natividad†,‡ and Satyajayant Misra†,§

**Department of Computer Science, Saint Louis University, Saint Louis, Missouri, USA*
†*Department of Computer Science, New Mexico State University*
Las Cruces, New Mexico, USA
**reza.tourani@slu.edu*
‡*fnativid@cs.nmsu.edu*
§*misra@cs.nmsu.edu*

Received 1 March 2018
Accepted 12 June 2018
Published 14 November 2018

Distributed Constraint Optimization Problem (DCOP) is a powerful paradigm to model multi-agent systems through enabling multiple agents to coordinate with each other to solve a problem. These agents are often assumed to be cooperative, that is, they communicate with other agents in order to optimize a global objective. However, the communication times between all pairs of agents are assumed to be identical in the evaluation of most DCOP algorithms. This assumption is impractical in almost all real-world applications. In this paper, we study the impact of empirically evaluating a DCOP algorithm under the assumption that communication times between pairs of agents can vary. In addition, we evaluate a DCOP algorithm using *ns-2*, a discrete-event simulator that is widely used in the computer networking community, to simulate the communication times, as opposed to the standard DCOP simulators that are used to evaluate DCOP algorithms in the AI community. Furthermore, we propose heuristics that exploit the non-uniform communication times to speed up DCOP algorithms that operate on pseudo-trees. Our empirical results demonstrate that the proposed heuristics improve the runtime of those algorithms up to 20%. These heuristics are evaluated on different benchmarks such as scale-free graphs, random graphs, and an instance of the smart grid, Customer-Driven Microgrid (CDMG) application.

Keywords: Distributed constraint optimization problems; smart grid; customer-driven microgrid; network simulators; communication times; multi-agent systems.

1. Introduction

Distributed Constraint Optimization Problems (DCOPs) are widely used in cooperative Multi-Agent Systems (MAS), where each agent holds one or more variables and assigns them an optimal value from a domain of possible assignments. Agents communicate to

decide on an assignment for all variables such that the sum of resulting constraint utilities is maximized.[1,2] DCOPs have a wide range of multi-agent system applications such as mobile sensor network and meeting scheduling problems, where subset of agents interact to solve a problem.[3–7] Considering the NP-hardness of such problems, within the past decades, an increasing number of algorithms have been proposed to solve DCOPs. These algorithms are often classified as complete or incomplete depending on whether they can guarantee finding the optimal solution or a non-optimal solution for a shorter running time. Typically, complete DCOP algorithms[1,8,9] use pseudo-trees to provide some forms of ordering of the variables in the DCOP. Similarly, some of the incomplete DCOP algorithms[10,11] also utilize pseudo-trees for ordering DCOP variables.

Although the algorithms to solve DCOPs have matured significantly since the field emergence, the number of DCOP realistic benchmarks, which are used to evaluate the performance of DCOP algorithms, is not significantly increasing. Due to unrealistic assumptions of typical DCOP algorithms, researchers evaluate such algorithms on synthetic random problems. One of the DCOP's unrealistic assumptions is that the empirical evaluations of most DCOP algorithms are done in simulation in which the interaction between all pairs of agents incur identical communication costs. Unfortunately, this assumption is not true in many multi-agent problems. For example, in a mobile sensor network, the communication between pairs of sensors depends on several factors, such as the distance between the sensors, the environment, and communication protocols. Thus, we extend the DCOP framework to include communication-related information; more specifically, the communication time of each constraint.

This extended framework allows us to empirically study and evaluate the impact of variable communication times on DCOP algorithms. We conduct an investigation with DPOP,[8] one of the most popular complete DCOP algorithms which has been widely extended by DCOP community. In order to evaluate the performance of DCOP algorithms, different efficiency metrics have been proposed such as *NCCCs*,[12] *number of messages*, and *simulated runtime*.[13] The first metric counts the number of non-concurrent constraint checks, the second metric counts the number of messages exchanged among agents, and last but not least is a common metric among the DCOP community which is used as a proxy to measure distributed runtime. In this work, we empirically verify the assumption of a high correlation between simulated runtimes and actual runtimes by specifying the communication time for each message as the time it takes for the source node to send a message until the sink node receives it. We define the communication time within the simulated runtime. Accurate estimation of communication times is often difficult due to its dependency to a combination of factors: communication protocols, communication medium, interference model, transmission delay, propagation delay, queuing delay, and processing delay.[14] Thus, we use *Network Simulator 2* (*ns-2*),[15] a discrete-event simulator used by the computer networking community,[16–18] 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.

In this work, we develop heuristics that can be used to exploit the non-uniform communication times in a DCOP to speed up the runtime of algorithms that operate on pseudo-trees. We study the impact of communication times on the performance of the DPOP algorithm that is a complete and inference DCOP algorithm. Complete and correct DCOP algorithms[1,8,9] typically require some forms of ordering for the variables in the DCOP. The majority of these algorithms use pseudo-trees to achieve the required ordering. Similarly, some approximate DCOP algorithms[10,11] also use pseudo-trees for ordering of the DCOP variables. As such, our heuristics can improve a large class of existing DCOP algorithms. We evaluate our heuristics in random graphs, scale-free graphs, and power network topologies with communication times that depend on physical distances, which are sampled from two — uniform and Gaussian — distributions. 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%.

2. Related Work

A *Distributed Constraint Optimization Problem* (DCOP) is a fundamental problem that can formalize popular cooperative multi-agent applications including meeting scheduling[3] to scheduling smart devices in smart homes.[19,20] Even though the development of various sophisticated algorithms for solving DCOPs has matured the DCOP field over the past decades, there is still a very limited contribution on the effect of communication times in the context of DCOPs.

Cruz *et al.*[21] investigated the importance of message communication times in evaluation of DCOP algorithms by conducting experiments in which agents are physically located apart in different machines connected by a 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 for better investigation of this area. Our research, in a large part, is in response to this call. However, there are differences between their work and ours; in their experiments they limited the number of agents to six and the constraint density p_1 to 0.5. In the communication protocol between agents, a master machine broadcast messages to client computers in the network. Then the master machine detects the active computers in the network and sends direct messages to clients. The master computer is responsible for assigning problem variables to the clients and itself. Once agents have been assigned with the variables, each agent runs the BnB-ADOPT algorithm to solve the distributed problem. The elapsed time is measured from the time that the DCOP resolution starts until the latest agent terminates. To assess the elapsed time, the authors measured the communication times between agents. Their experimental results showed the importance of communication effort that is not fairly reflected in evaluation metrics such as NCCCs and simulated runtime. In contrast, our experiments includes a larger number of agents over a larger and more realistic combination of configurations (e.g., different communication times, constraint densities, communication protocols, and graph topologies). Furthermore, we use wireless communication medium instead of wired communication, which is more common in applications such as our motivating CDMG application. We

also precisely measure the communication times through the network simulator *ns-2* that captures several factors, such as network congestion, packet drops, and possible delays that affect the communication effort in real-world environments.

In the context of distributed constraint satisfaction problems (DCSPs), Fernandez *et al.*[22] have also studied the effect of communication network data load on DCSP algorithms and found that including random delays in message delivery times can in fact improve the performance and robustness of AWC. Wahbi *et al.*[23] studied the effect of different communication costs on the performance of DCSP algorithms. They first decoupled the communication graph from the underlying constraint graph of the problem that models different communication layers and showed that different communication layers significantly affect the message costs and changing communication graph topologies affect the performance of the ABT and AFC-ng algorithms. In their work, the authors measured the communication load by the number of transmitted 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*).[12] The main difference with our work is that the authors 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 more realistic communication protocols, in terms of the pseudo-tree depth. The pseudo-tree depth serves as a proxy for simulated runtimes. Hence, we propose methods to construct shorter pseudo-trees with the aim of speeding up the performance of large class of DCOP solvers. Different from Fernandez and Wahbi's work, we use *ns-2* simulator to measure a more realistic communication latency. The *ns-2* simulator that we use in this paper, models packet loss, re-transmissions due to packet drops, and network congestion that have been disregarded in recent studies.

Zivan *et al.*[24] investigated the impact of message delays in DCSP problems. The main difference with our work is that the authors introduced an Asynchronous Message Delay simulator (AMD), which 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 a transport, routing, and multicast protocols over wireless network scenarios.

Okimoto *et al.*[25] have also studied the effect of different variable-ordering heuristics for ABT in scale-free graphs. Similarly, we propose different variable-ordering heuristics and study the effect of the proposed heuristics, specifically for the DPOP algorithm. However, different from Okimoto's work, our proposed heuristics exploit the non-uniform communication times in a DCOP to speed up algorithms that operate on pseudo-trees. While Okimoto *et al.* do not consider communication efforts in ABT algorithm. Similar to their work, we have evaluated our heuristics on scale-free graphs and showed the effect of communication times in such graphs.

Finally, Ali *et al.*[26] have proposed preprocessing techniques that are based on dynamic programming to accelerate the ADOPT algorithm. Similar to their work, we propose several heuristics that construct shorter pseudo-trees as the preprocessing phase of DPOP to

accelerate its runtime. Unlike Ali *et al.*, our heuristics in the preprocessing phase of DPOP take non-uniform communication times into account while communication effort is totally neglected in their work. This article is a revised and extended version of our preliminary effort[27] which is augmented with more details on the network simulator *ns-2* and concrete experimental results on different benchmarks.

3. Preliminaries

We will describe the *Distributed Constraint Optimization Problem* (DCOP) paradigm and the pseudo-tree structure used in the DPOP algorithm discussed in this paper.

3.1. *Distributed constraint optimization problems*

DCOPs are problems from the multi-agent systems filed in which agents can send messages to one another to cooperatively decide on their variable assignments to find a solution to a global maximization reward function. DCOP is formally defined as a tuple $\langle \mathbf{A}, \mathbf{X}, \mathbf{D}, \mathbf{F}, \alpha \rangle$, where:

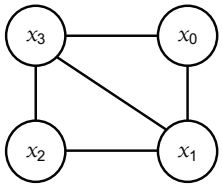
- $\mathbf{A} = \{a_i\}_{i=1}^{p}$ is a set of *agents*.
- $\mathbf{X} = \{x_i\}_{i=1}^{n}$ is a set of decision *variables*.
- $\mathbf{D} = \{D_x\}_{x \in \mathbf{X}}$ is a set of finite *domains*. Each variable $x \in \mathbf{X}$ takes values from the set $D_x \in \mathbf{D}$.
- $\mathbf{F} = \{f_i\}_{i=1}^{m}$ is a set of *constraints*, each defined over a mixed set of decision variables: $f_i : \bigtimes_{x \in \mathbf{x}^{f_i}} D_x \rightarrow \mathbb{R}^+ \cup \{\bot\}$, where $\mathbf{x}^{f_i} \subseteq \mathbf{X}$ is the *scope* of f_i and $\bot$ is a special element used to denote that a given combination of values for the variables in $\mathbf{x}^{f_i}$ is not allowed.
- $\alpha : \mathbf{X} \rightarrow \mathbf{A}$ is a function that associates each decision variable to one agent.

A *solution* σ is a value assignment for a set $\mathbf{x}_\sigma \subseteq \mathbf{X}$ of variables that is consistent with their respective domains. The utility $\mathbf{F}(\sigma) = \sum_{f \in \mathbf{F}, \mathbf{x}^f \subseteq \mathbf{x}_\sigma} f(\sigma)$[a] 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 $\mathbf{x}^* = \operatorname{argmax}_{\mathbf{x}} \mathbf{F}(\mathbf{x})$.

For simplicity we draw 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 $\alpha(x_i) = a_i$. Even though this is a common assumption in the DCOP literature as there exist pre-processing techniques that transform a general DCOP into this more restrictive DCOP,[28,29] our approach can be easily generalized to the unrestricted version, as we demonstrate with our Customer-Driven Microgrid (CDMG) application domain in our experiments.

Figure 1 illustrates a typical DCOP problem where each of four agents tries to determine an optimal value assignment for their variables independently. Note, that an agent in a DCOP has limited knowledge of the entire problem. Hence, factors such as how agents

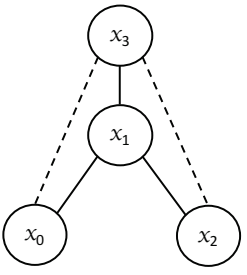
[a]With a slight abuse of notation, we use $\mathbf{F}$ to denote the set of constraints as well as the overall utility of the DCOP.



(a) Constraint Graph

x_i	x_j	utility
0	0	5
0	1	10
1	0	4
1	1	15

(b) Constraint Table



(c) Pseudo-Tree

Fig. 1. Example of a DCOP.

communicate with each other and what message is exchanged are crucial in DCOP algorithms.

3.2. *Distributed DFS pseudo-tree*

Given a DCOP P, $G = (\mathbf{X}, E)$ is the *constraint graph* of P, shown in Fig. 1(a), where $\{x_i, x_j\} \in E$ iff $\exists f_i \in \mathbf{F}$ such that $\{x_i, x_j\} = \mathbf{x}^{f_i}$. A *DFS pseudo-tree* arrangement shown in Fig. 1(c), for G is a *spanning tree* $T = \langle \mathbf{X}, E_T \rangle$ of G such that if $f_i \in \mathbf{F}$ and $\{x_i, x_j\} \subseteq \mathbf{x}^{f_i}$, then x_i and x_j appear in the same branch of T. Edges of G that are in E_T are called *tree edges*, shown with solid lines, and other edges, shown with dashed lines, 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*. 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. A pseudo-tree is often evaluated based on its maximum depth or separator.

Definition 3.1. Separator. In a pseudo-tree the *separator* of agent a_i denoted by $sep(a_i)$, is the set of all ancestors of a_i denoted by $ans(a_i)$, that are also pseudo-parents of a_i or its descendants denoted by $des(a_i)$.

$$sep(a_i) = ans(a_i) \cap \left(PP(a_i) \cup \left(\bigcup_{a_j \in des(a_i)} PP(a_j) \right) \right).$$

Figure 1(c) shows a pseudo-tree obtained from the constraint graph in 1(a), with x_3 as the root, the separators of this example are $sep(x_3) = \emptyset$, $sep(x_1) = \{x_3\}$, $sep(x_2) = \{x_1, x_3\}$, and $sep(x_0) = \{x_1, x_3\}$. As mentioned before, "variable" and "agent" are used interchangeably.

Distributed DFS. Such pseudo-tree structures can be obtained using a depth-first-traversal of the constraint graph. Given a constraint graph, constructing an optimal pseudo-tree is an NP-hard problem, the Distributed DFS algorithm has been proposed by Hamadi *et al.*[30]

to construct pseudo-trees. This greedy-based algorithm is commonly used in many implementations of complete DCOP algorithms including those within the DCOPolis[13] and FRODO[31] 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** $h(x_i)$:

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

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.

3.3. *DPOP algorithm*

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 bottom of the pseudo-tree leaf nodes, 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. Let us denote the UTIL message sent from agent a_i to agent a_j by $UTIL_{i\to j}$, which is a multi-dimensional matrix consist of a dimension of each variable in $sep(a_i)$. The dimension of $UTIL_{i\to j}$ denoted by $dim(UTIL_{i\to j})$ is the set of variables considered by the message. For an assignment $\sigma' \in dim(UTIL_{i\to j})$,

$$UTIL_{i\to j}(\sigma') = \bigcup_{\forall \sigma'', \sigma'' = dim(UTIL_{i\to j})\setminus\sigma'} UTIL_{i\to j}(\sigma' \cup \sigma'').$$

  It simply shows that each 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. Each agent determines the optimal value of its variables based on its UTIL computations and the value of variables in the VALUE messages.

4. Customer-Driven Microgrid Application

In this section, we motivate our work using the *Customer-Driven Microgrid* (CDMG) optimization problem proposed by Gupta *et al.*[32] The customer-driven microgrid has been introduced[32] as one of the smart grid representatives, where the energy consumption, generation, and transmission are distributed among multiple agents in the system. Our CDMG instantiation consists of a neighborhood of homes each of which capable of generating energy, consuming energy and transmitting energy to its neighboring homes. We model this problem as a DCOP and represent each home by an agent, the energy consumption, generation, and transmission of each home are the 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. Moreover, there are constraints that must be satisfied when the amount of energy generated minus the amount of power consumed must be equal to the amount of energy transmitted out of a home to its neighboring homes. Finally the objective of the DCOP model is to minimize the difference between the largest amount of energy consumed and the largest amount of energy generated in the neighborhood.

Like in most DCOP literature, Gupta *et al.*[32] also assume that the communication time of transmitting energy is uniform across all homes. However, in practice, the communication time may not be uniform and depends largely on the underlying network 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. The home and power plants or energy providers are connected to each other through a communication network (e.g., wireless network). Thus these smart meters can 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.[33] 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, as we adopt this communication model, in order to maintain privacy and reliability, we assumed that 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. Figure 2 illustrates this application with 10 homes and 3 aggregators.

4.1. *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.[15]

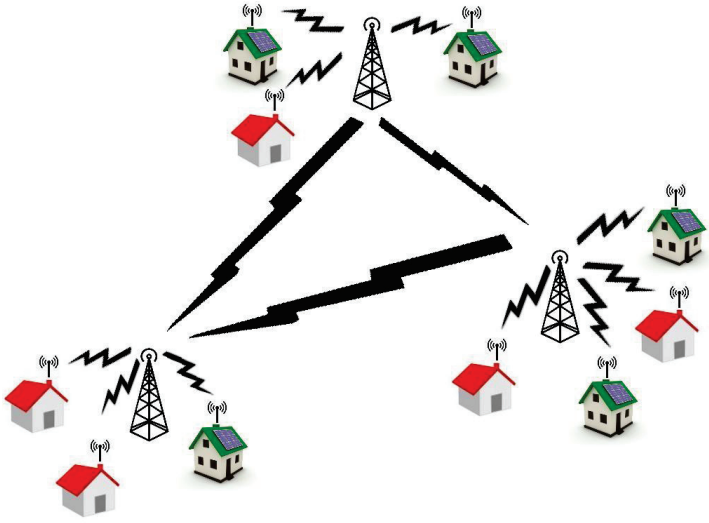


Fig. 2. CDMG illustration.

Before discussing our simulation configuration, we briefly explain *ns-2*'s simulation procedure as depicted in Fig. 3. To simulate a network for agents' behavior study, one needs to first generate a scenario, which includes information such as the network topology, the protocols in the nodes' network stacks, and the simulation duration. The topology

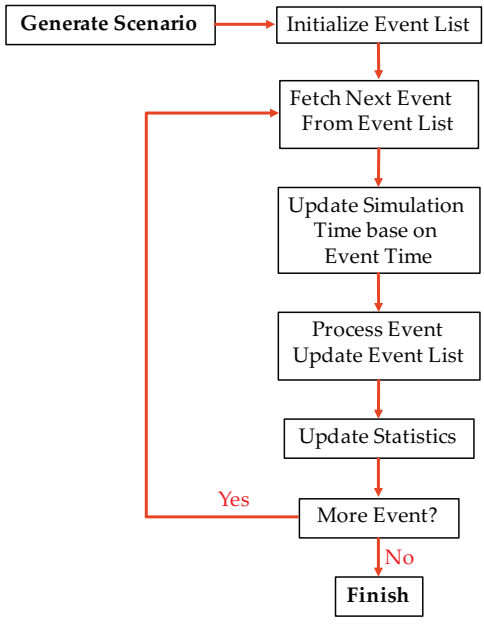


Fig. 3. Network simulator flow.

is defined in form of a graph in which vertices are the nodes (agents) and the edges indicate the connection between the nodes (agents relationships). The network stack of a node defines various protocols, from the application to the physical layer, that a node uses for communication.

Given this scenario file, *ns-2* generates an initial list of events including packet generation, forwarding, reception, etc. As the simulation progresses, *ns-2* fetches an event from the list in the chronological order. Since each event is associates with a time stamp, *ns-2* updates the simulation time according to the selected event and processes the event. Processing an event may result in generation of a new event(s). For instance, a successful packet delivery using TCP protocol causes an acknowledgment packet generation by the receiving node. Thus, after processing the selected event, *ns-2* updates the event list by adding the corresponding time-based event(s) to the list. Then, *ns-2* updates the statistical information that corresponds to the completed event. At this time, *ns-2* checks the event list for further events. In case that the list is non-empty, *ns-2* selects the next event in chronological order and follows the aforementioned steps. Otherwise, the simulation is finished.

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 11 Mb/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 0.1 Mb/s and packet size of 500 bytes in the application layer.

5. Non-uniform 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 5.1. 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 a complete message, where both the source and the sink nodes are in the scope of the constraint.

One of the DCOP assumptions is 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[10] and Distributed Gibbs[11] 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 5.2. 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.

5.1. *Communication times impact*

Given the above generalized depth definition, we investigate the impact of non-uniform communication times on the class of DCOP algorithms that operate on pseudo-trees. As a case study, we exploit DPOP algorithm in this work and study the relationship between

the runtimes of DCOP algorithms and the generalized depth of their underlying pseudo-trees. It is worth mentioning that the commonly used runtime performance metric by the DCOP community is the *Simulated Runtime*,[13] which assumes that the communication times between all pairs of agents are uniform and identical. However, we empirically show that the simulated runtime of our algorithm no longer assumes identical communication times between all pairs of agents and define the "*Actual*" *Runtime*, which is computed via *Network Simulator 2.0* (*ns-2*),[15] a de-facto simulator used by the computer networking community,[16–18] to precisely measure the communication times.

Definition 5.3. "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.

To measure the simulated runtime the algorithm with non-uniform communication times, we first generate problem instances with random graph topologies[34] and vary the number of variables $|\mathbf{X}| = \{10, 20\}$[b] and the constraint density $p_1 = \{0.2, \ldots, 0.7\}$,[c] 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 240 instances in total, for this experiment.

We solve these problems using DPOP implemented on the FRODO simulation framework,[31] 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 Eq. (4).[d]

For each of these instances we create an *ns-2* scenario for the network simulator, such that variables are connected in a mesh topology using a routing protocol that defines the constraints between variables. The routing table follows the order of UTIL and VALUE messages to define the corresponding source and sink nodes of the constraints. Then for every two variables in the scope $\mathbf{x}^{f_i}$ of the same function f_i we use d_i to create an exact physical distance between them. To simulate the *exact* same trace of execution of the agents we need to set two parameters in *ns-2*, namely the computation time of each agent and the

[b]As one of the limitation of DPOP algorithm is the exponential growth of memory with the maximum number of separators, we did not generate larger problem instances.

[c]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.

[d]We set $C = 1$ millisecond per meter for all the experiments in this paper, note that the physical distances between the two variables are sampled from distributions.

size of messages sent by each agent. We use the stored computation time and message size of each problem instance from the runs on FRODO earlier. Then in the simulator scenario, in order to mimic an agent's computation time we set 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. The TCP protocol ensures a reliable communication for each agent to send complete message to other agents. In order to mimic agent's communication we 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 acknowledgment (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.

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. We believe that the network latency, captured in message communication time, has an important impact in the performance of DCOP algorithms, therefore it must be taken into account to accurately approximate the "actual" time of the solving process.

5.2. *Theoretical results*

In this section we describe the theoretical runtime complexities measured using the "actual" runtimes and simulated runtimes metrics mentioned above and then empirically evaluate the correlation between the two runtime metrics. 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 5.1. 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.[e] This sum is then simplified to $\sum_{a_i \in b} O(\exp(|sep(a_i)|))$.

Theorem 5.2. 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

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

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 1).

Table 1. Maximum separator size, largest message size, and largest number of packets.

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)	8165.8	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	2137.7	10 255.4	29 907.8	29 907.7

Table 2. Correlations.

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

Now, we empirically evaluate the correlation between "actual" runtimes, simulated runtimes, and pseudo-tree depths on the instances described in Section 5.1. Table 2 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 $\max_{b \in \mathbf{B}} \sum_{a_i \in b} O(\exp(|sep(a_i)|))$.

[e]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 Eq. (4).

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 $\sum_{a_i \in b} O(c_i)$ (Eq. (4)) and is a close approximation of the pseudo-tree depth (Definition 5.2). 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.

6. Pseudo-Tree Construction Heuristics

Recall that a large class of DCOP algorithms including ADOPT and DPOP order the variables in the problem using pseudo-trees. The runtimes of these algorithms thus typically depend on the depth of the pseudo-tree as information needs to propagate either downwards from the root of the pseudo-tree to all the leafs of the pseudo-tree or upwards from the leafs to the root. Therefore, in this section, we introduce several heuristics methods that exploit the non-uniform communication times to construct pseudo-trees with small depths. These small depth pseudo-trees aim at accelerating the runtime of those algorithms that operate on pseudo-trees.

We first evaluate the potential improvement to existing pseudo-trees constructed using the default max-degree heuristic (Eq. (1)). In order to find optimal pseudo-trees with smallest depth, we run an exhaustive search and construct all possible pseudo-trees of every DCOP and return the tree with minimum depth as the optimal pseudo-tree. Figure 4 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 (Eq. (4)). Since DPOP algorithm does not scale well to larger number of variables due its memory constraint, we could not find optimal pseudo-trees for larger problems. But one can easily 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 propose various heuristics that can be used by Distributed DFS to construct pseudo-trees with small depths:

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

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

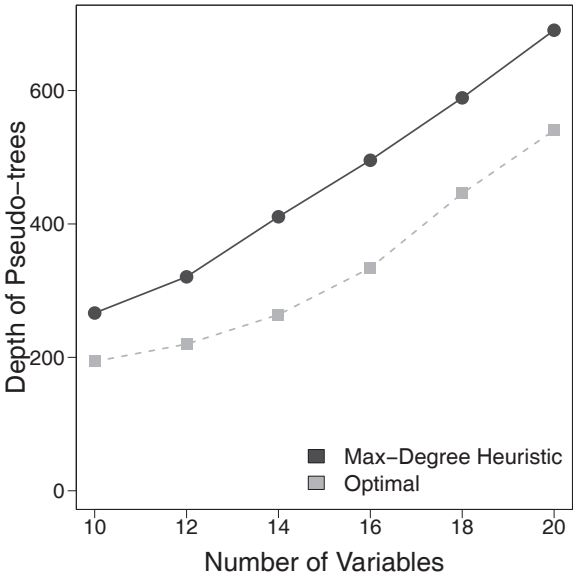


Fig. 4. Comparing the optimal and default pseudo-tree depths.

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 Eq. (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}:

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

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.

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

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

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.

It is common to use two different heuristic functions for choosing the root of the pseudo-trees and the non-root variables of the pseudo-tree. We do the same here and the

Table 3. Pseudo-tree construction heuristics.

		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

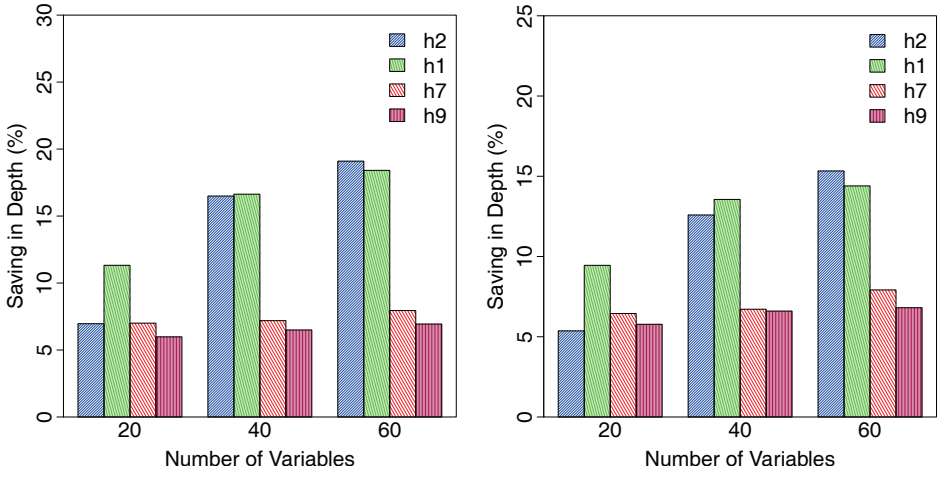
combination of all the three heuristics above generates nine alternatives that are tabulated in Table 3.

6.1. *Experimental results*

We empirically evaluate our nine heuristics against the default max-degree heuristic (Eq. (1)) on (a) random graphs,[34] (b) scale-free graphs,[35] and (c) 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. We averaged our results over 500 instances, except for CDMG results, which are averaged over 50 instances. We ran all the experiments on a machine with an Intel Core i7-3770 CPU at 3.40 GHz and 16 GB of RAM.

Random Graphs: We generate 500 DCOP instances following the *Erdös-Rényi* graph model to create random graphs. Such random graphs have low heterogeneity and edges have equal probability of connecting to any vertex in a graph. As described earlier, when we compared the depth between the default pseudo-trees and the optimal pseudo-trees, we observed significant differences as the number of variables increased. Therefore, we vary the number of variables as $|\mathbf{X}| = \{20, 40, 60\}$ with increment of 20, fix the constraint density $p_1 = 0.3$, $|D_i| = 3$ and all constraints are binary. 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 (Eq. (4)). Figure 5 illustrates the percentage of savings in depth of pseudo-trees constructed by the four best heuristics. The results state that the savings increase with the number of variables. As we increase the number of variables, the number of possible pseudo-parents and pseudo-children each variable can have, increase as well. A larger set of pseudo-parents and pseudo-children lead to a larger number of possible pseudo-tree configurations. Then our heuristics construct the tree with the possible smaller depth.

The h_1 and h_2 heuristics converge to savings $\approx 19\%$ with uniform and $\approx 16\%$ with Gaussian distributions; indicating that heuristics that take communication times into account perform better than those that do not. The h_7 and h_9 heuristics converge to smaller savings ($\approx 7\%$ with uniform and $\approx 8\%$ with Gaussian distributions). Hence, for random DCOPs, using h_1 and h_2 heuristics to construct pseudo-trees results in approximately 19% improvement of DPOP algorithm runtime. Further, Table 4 tabulates the depths



(a) Uniform Distribution (b) Gaussian Distribution

Fig. 5. Saving in pseudo-tree depths, when the distances between agents are sampled from uniform and Gaussian distribution in **Random Graphs**.

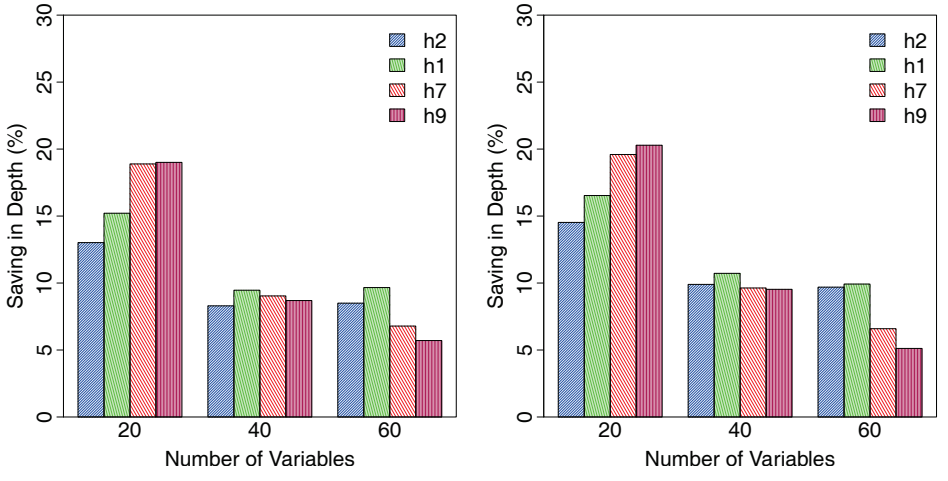
Table 4. DPOP runtimes comparison in random graphs.

	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

of the pseudo-trees and both the "actual" and simulated runtimes of DPOP with those pseudo-trees constructed by h_1 and h_2 heuristics compared against the default max-degree heuristic.[f] The "actual" runtime is precisely measured by *ns-2*. These results show that the savings in pseudo-tree depths of the two heuristics translate to savings in both runtimes as well.

Scale-Free Graphs: We generate 500 DCOP instances following the *Barabási-Albert* graph model to create scale-free graphs. The structure of such graphs combines heterogeneity and randomness. When a new vertex is added to the graph, it has a higher probability to connect to vertices with larger degrees. We vary the number of variables $|\mathbf{X}| = \{20, 40, 60\}$ with increment of 20 and set the maximum degree of all variables to 4. We also set $|D_i| = 3$ and all constraints are binary. Similar to random graphs, we sample the physical distances using the same two truncated distributions and use them to define communication times. Unlike random graphs, the constraint density p_1 is not constant and decreases as the number of variables increases.

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



(a) Uniform Distribution

(b) Gaussian Distribution

Fig. 6. Saving in pseudo-tree depths, when the distances between agents are sampled from uniform and Gaussian distribution in **Scale-free graphs**.

Figure 6 shows the percentage of savings in pseudo-tree depths, which decreases by increasing the number of variables, for uniform and Gaussian distributions. We observe a larger saving in all four heuristics when $|\mathbf{X}| = 20$ because of its larger constraint density ($p_1 \approx 0.2$) while in graphs with 40 and 60 variables, the constraint density drops significantly. The smaller the constraint density the fewer the number of pseudo-parents and pseudo-children of a variable that leads to a lower number of pseudo-tree configurations. Therefore, graphs with smaller constraint density has a higher probability of constructing a chain rather than a pseudo-tree. Moreover, the h_7 and h_9 heuristics converge to savings ($\approx$19% with uniform and $\approx$20% with Gaussian distributions) when the constraint density is at its largest. However, heuristics h_1 and h_2 still converge to savings $\approx$15% with uniform and $\approx$17% with Gaussian distributions. The results show that h_1 and h_2 perform better than h_7 and h_9 on larger number of variables even if the constraint density is very small and the pseudo-trees look like chains.

We perform an additional experiment on random graphs with $|\mathbf{X}| = 20$ in which we vary p_1 from 0.1 to 0.8 (refer to Fig. 7 for results). For all four heuristics, at small constraint densities ($p_1 \leq 0.4$), the savings decrease with decreasing p_1, thereby explaining their behavior in scale-free graphs. Moreover, unlike the savings of h_1 and h_2, the savings of h_7 and h_9 do not increase by increasing the constraint density as these heuristics do not take communication times into account. We thus choose h_1 and h_2 as our best heuristics to construct small depth pseudo-trees in our CDMG application.

Customer-Driven Microgrids (CDMGs): Here, we explains how we generate graphs of our motivating CDMGs. We first sample neighborhoods in three cities in the United States (*Des Moines, IA*; *Boston, MA*; and *San Francisco, CA*) and estimate the density of houses

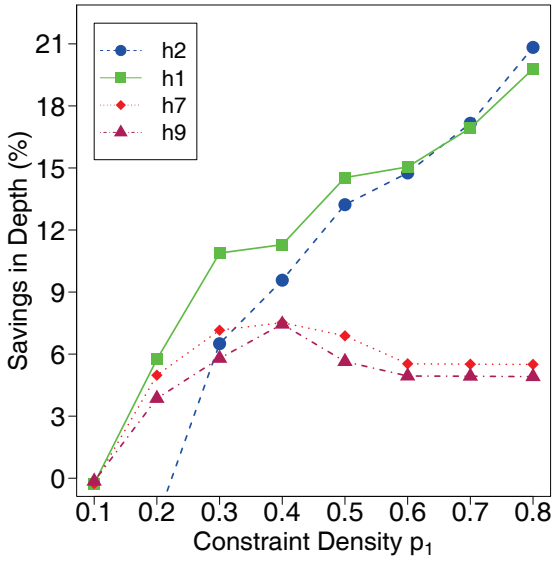


Fig. 7. Saving in pseudo-tree depths,varying the constraints density p_1 of random graphs with 20 variables.

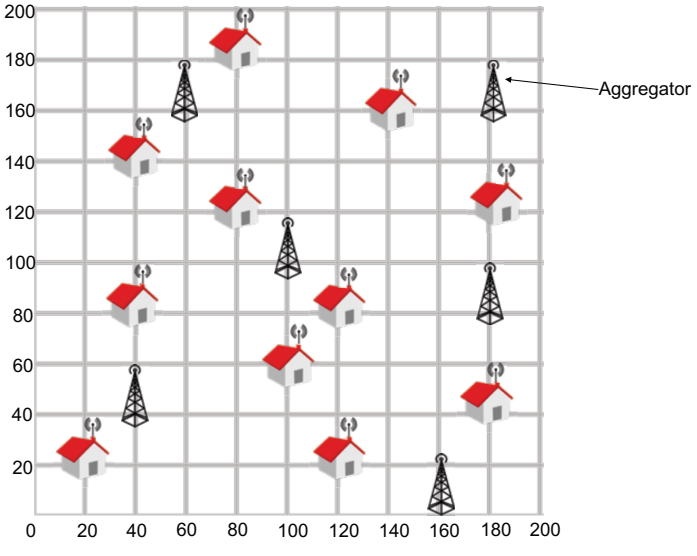


Fig. 8. Example of a smart neighborhood communication in CDMG.

in each city. The average density (in houses per square kilometers) is 718 in Des Moines, 1357 in Boston, and 3766 in San Francisco.

As Fig. 8 shows, for each city, we generated a $200\ \mathrm{m} \times 200\ \mathrm{m}$ grid, where the distance between intersections is 20 m, and randomly placed houses in this grid until the density is the same as the sampled density. As explained in Section 4, houses can transfer energy to

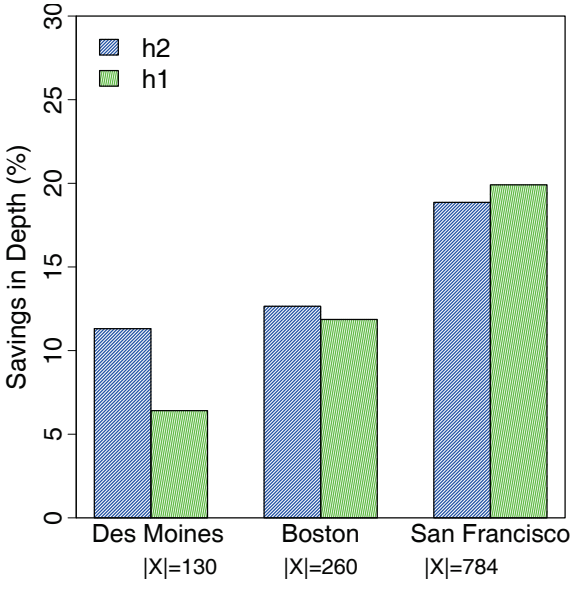


Fig. 9. Saving in pseudo-tree depths in **CDMG graphs**.

their neighboring houses while communicating through aggregators. Therefore, we enforce each house to be 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 houses that has the smallest distance between them and constrain them, using the Dijkstra's algorithm. Finally, we greedily placed aggregators, with a communication radius of 100 m, in the grid until all houses are within the radius of at least one aggregator. Aggregators can then communicate with all houses and other aggregators that are within their communication radius.

Unlike random and scale-free graphs, in the resulting CDMG graphs, each agent controls multiple variables. It is worth mentioning that constraints among variables are no longer binary, as a house energy consumption, generation, and transmission are in constrained with each other (explained in Section 4). The number of variables for resulting CDMG graphs is set to 130 in Des Moines, 260 in Boston and 784 in San Francisco ($|\mathbf{X}| = \{130\,260\,784\}$ and $|D_i| = \{2, 5\}$). Figure 9 illustrates the savings in pseudo-tree depths of our CDMG graphs. As we expected, our best heuristics (h_1 and h_2) that take the communication times into account demonstrate increasing savings ($\approx 20\%$) with the number of variables. It implies that by constructing a shorter pseudo-tree we improved the runtime of DPOP algorithm by approximately 20%, even with non-uniform communication times among agents.

7. Conclusions

The communication efforts among agents have been neglected by the DCOP community. Consequently, the existing 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 extended framework, in the DCOP model we 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. We empirically show the impact of communication times on the performance of a DCOP algorithm as a case study. Finally, we propose communication-sensitive pseudo-tree construction heuristics to generate pseudo-trees that have smaller depths than the pseudo-trees that are generated by the max-degree heuristic. These heuristic methods exploit the non-uniform communication times and find pseudo-trees that are up to 20% shorter than those constructed by the max-degree heuristic. As a large class of DCOP algorithms operate on pseudo-trees in their variable-ordering phase, the proposed heuristics can speed up the runtime of these algorithms. All these heuristics are evaluated on three problem domains such as random graphs, scale-free graphs, and CDMG graphs. We use the pseudo-tree depth as the proxy of the algorithm runtime, and show the behavior of our heuristics in different problem domains in terms savings in pseudo-tree depths.

Acknowledgments

This research is partially supported by NSF grants 1345232, 1550662, and 1812619. 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) (2005) 149–180.
2. W. Yeoh and M. Yokoo, Distributed problem solving, *AI Magazine* **33**(3) (2012) 53–65.
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 *Proc. of AAMAS* (2004), pp. 310–317.
4. A. Farinelli, A. Rogers, A. Petcu and N. Jennings, Decentralised coordination of low-power embedded devices using the Max-Sum algorithm, in *Proc. of AAMAS* (2008), pp. 639–646.
5. S. Miller, S. Ramchurn and A. Rogers, Optimal decentralised dispatch of embedded generation in the smart grid, in *Proc. of AAMAS* (2012), pp. 281–288.
6. D. T. Nguyen, W. Yeoh, H. C. Lau, S. Zilberstein and C. Zhang, Decentralized multi-agent reinforcement learning in average-reward dynamic DCOPs, in *Proc. of AAAI* (2014), pp. 1447–1455.
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) (2015) 495–536.
8. A. Petcu and B. Faltings, A scalable method for multiagent constraint optimization, in *Proc. of IJCAI* (2005), pp. 1413–1420.
9. W. Yeoh, A. Felner and S. Koenig, BnB-ADOPT: An asynchronous branch-and-bound DCOP algorithm, *Journal of Artificial Intelligence Research* **38** (2010) 85–133.

10. B. Ottens, C. Dimitrakakis and B. Faltings, DUCT: An upper confidence bound approach to distributed constraint optimization problems, in *Proc. of AAAI* (2012), pp. 528–534.
11. D. T. Nguyen, W. Yeoh and H. C. Lau, Distributed Gibbs: A memory-bounded sampling-based DCOP algorithm, in *Proc. of AAMAS* (2013), pp. 167–174.
12. A. Chechetka and K. Sycara, No-commitment branch and bound search for distributed constraint optimization, in *Proc. of AAMAS* (2006), pp. 1427–1429.
13. E. Sultanik, R. Lass and W. Regli, DCOPolis: A framework for simulating and deploying distributed constraint reasoning algorithms, in *Proc. of the Distributed Constraint Reasoning Workshop* (2007).
14. J. Kurose and K. Ross, *Computer Networking: A Top Down Approach* (Pearson, 2012).
15. S. McCanne and S. Floyd, ns–network simulator, http://nsnam.sourceforge.net/wiki/.
16. J. Sommers and A. Moore, Scaling the practical education experience, in *Proc. of the ACM SIGCOMM Education Workshop* (2011).
17. N. Rozhnova and S. Fdida, An effective hop-by-hop interest shaping mechanism for CCN communications, in *Proc. of IEEE INFOCOM Workshops* (2012), pp. 322–327.
18. N. E. Majd, S. Misra and R. Tourani, Split-cache: A holistic caching framework for improved network performance in wireless ad hoc networks, in *Proc. of IEEE GLOBECOM* (2014), pp. 137–142.
19. F. Fioretto, W. Yeoh and E. Pontelli, A multiagent system approach to scheduling devices in smart homes, in *Proc. of the 16th Conf. on Autonomous Agents and MultiAgent Systems* (2017), pp. 981–989.
20. A. M. Tabakhi, T. Le, F. Fioretto and W. Yeoh, Preference elicitation for DCOPs, in *Proc. of CP* (2017), pp. 278–296.
21. F. Cruz, P. Gutierrez and P. Meseguer, Simulation vs real execution in DCOP solving, in *Proc. of the Distributed Constraint Reasoning Workshop* (2014).
22. C. Fernàndez, R. Béjar, B. Krishnamachari and C. Gomes, Communication and computation in distributed CSP algorithms, in *Proc. of CP* (2002), pp. 664–679.
23. M. Wahbi and K. N. Brown, The impact of wireless communication on distributed constraint satisfaction, in *Proc. of CP* (2014), pp. 738–754.
24. R. Zivan and A. Meisels, Message delay and DisCSP search algorithms, *Annals of Mathematics and Artificial Intelligence* **46**(4) (2006) 415–439.
25. T. Okimoto, A. Iwasaki and M. Yokoo, Effect of DisCSP variable-ordering heuristics in scale-free networks, in *Proc. of PRIMA* (2012), pp. 166–180.
26. S. Ali, S. Koenig and M. Tambe, Preprocessing techniques for accelerating the DCOP algorithm ADOPT, in *Proc. of AAMAS* (2005), pp. 1041–1048.
27. A. M. Tabakhi, R. Tourani, F. Natividad, W. Yeoh and S. Misra, Pseudo-tree construction heuristics for DCOPs and evaluations on the ns-2 network simulator.
28. D. Burke and K. Brown, Efficiently handling complex local problems in distributed constraint optimisation, in *Proc. of ECAI* (2006), pp. 701–702.
29. M. Yokoo (Ed.), *Distributed Constraint Satisfaction: Foundation of Cooperation in Multi-Agent Systems* (Springer, 2001).
30. Y. Hamadi, C. Bessière and J. Quinqueton, Distributed intelligent backtracking, in *Proc. of ECAI* (1998), pp. 219–223.
31. T. Léauté, B. Ottens and R. Szymanek, FRODO 2.0: An open-source framework for distributed constraint optimization, in *Proc. of the Distributed Constraint Reasoning Workshop* (2009), pp. 160–164.
32. S. Gupta, P. Jain, W. Yeoh, S. Ranade and E. Pontelli, Solving customer-driven microgrid optimization problems as DCOPs, in *Proc. of the Distributed Constraint Reasoning Workshop* (2013), pp. 45–59.

33. W. Meng, R. Ma and H. Chen, Smart grid neighborhood area networks: A survey, *IEEE Network* **28**(1) (2014) 24–32.
34. P. Erdös and A. Rényi, On Random Graphs I, *Publicationes Mathematicae Debrecen* **6** (1959) 290.
35. A.-L. Barabási and R. Albert, Emergence of scaling in random networks, *Science* **286**(5439) (1999) 509–512.