A Large Neighboring Search Schema for Multi-Agent Optimization

Khoi D. Hoang[1], Ferdinando Fioretto[2], William Yeoh[1], Enrico Pontelli[3], and Roie Zivan[4]

[1] Washington University in St. Louis, USA
{khoi.hoang,wyeoh}@wustl.edu
[2] University of Michigan, USA
fioretto@umich.edu
[3] New Mexico State University, USA
epontell@cs.nmsu.edu
[4] Ben Gurion University of the Negev, Israel
zivanr@cs.bgu.ac.il

Abstract. The Distributed Constraint Optimization Problem (DCOP) is an elegant paradigm for modeling and solving multi-agent problems which are distributed in nature, and where agents cooperate to optimize a global objective within the confines of localized communication. Since solving DCOPs optimally is NP-hard, recent effort in the development of DCOP algorithms has focused on incomplete methods. Unfortunately, many of such proposals do not provide quality guarantees or provide a loose quality assessment. Thus, this paper proposes the *Distributed Large Neighborhood Search (DLNS)*, a novel iterative local search framework to solve DCOPs, which provides guarantees on solution quality refining lower and upper bounds in an iterative process. Our experimental analysis of DCOP benchmarks on several important classes of graphs illustrates the effectiveness of DLNS in finding good solutions and tight upper bounds in both problems with and without hard constraints.

Keywords: Multiagent Systems; Distributed Constraint Optimization; Large Neighborhood Search

1 Introduction

In a cooperative *Multi-Agent System (MAS)*, multiple autonomous agents interact to pursue personal interests and to achieve common objectives. *Distributed Constraint Optimization Problems (DCOPs)* [24, 30, 8] have emerged as a prominent agent model to govern the agents' behavior in cooperative MAS. In this context, agents control variables of a weighted constrained problem and coordinate their value assignments to maximize the overall sum of resulting constraint utilities. DCOPs are suitable to model problems that are distributed in nature and where a collection of agents attempts to optimize a global objective within the confines of localized communication. They have been employed to model distributed versions of meeting scheduling problems [22, 39], allocation of targets to

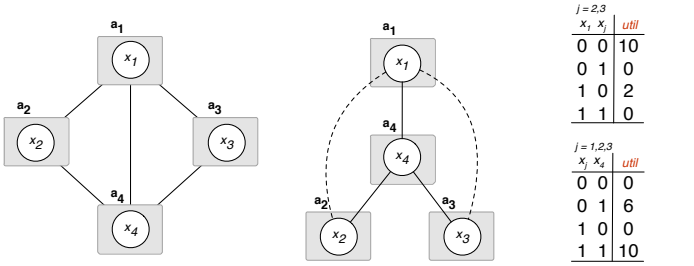
sensors in a network [5], channel selection in wireless networks [41], coordination of multi-robot teams [43], optimization in smart grids [19, 23, 13], generation of coalition structures [37], and device scheduling in smart homes [34, 12, 18].

DCOP algorithms are classified as either *complete* or *incomplete*. Complete DCOP algorithms find optimal solutions at the cost of large runtimes, while incomplete approaches trade optimality for faster runtimes. Since finding optimal DCOP solutions is NP-hard [24], incomplete algorithms are often necessary to solve larger problems' instances. Unfortunately, several local search algorithms (e.g., DSA [42] and MGM [21]) and local inference algorithms (e.g., Max-Sum [5]) do not provide guarantees on the quality of the solutions found. More recent developments, such as region-optimal algorithms [28, 17, 38], sampling-based algorithms [27, 25, 10] and (Improved) Bounded Max-Sum [32, 33] alleviate this limitation. Region-optimal algorithms allow the specification of regions with a maximum size of k agents or t hops from each agent, and they optimally solve the subproblem within each region. Solution quality bounds are provided as a function of k [28], t [17], or a combination of both [38]. Sampling-based algorithms such as DUCT [27] and D-Gibbs [25, 10] extend the centralized UCT [1] and Gibbs [14] sampling algorithms, respectively. They are able to bound the quality of solutions found as a function of the number of samples used by the algorithms. Bounded Max-Sum [32] extends Max-Sum by solving an acyclic version of the DCOP graph and bounding its solution quality as a function of the edges removed from the graph. Improved Bounded Max-Sum [33] further provides tighter upper bounds. Although good quality assessments are essential for sub-optimal solutions, many incomplete DCOP approaches provide poor quality assessments and are unable to exploit domain-dependent knowledge and/or hard constraints present in problems.

We address these limitations by introducing the *Distributed Large Neighborhood Search (DLNS)* framework.[5] DLNS solves DCOPs by building on the strengths of LNS [35], a *centralized* meta-heuristic algorithm that iteratively explores complex neighborhoods of the search space to find better candidate solutions. LNS has been shown to be very effective in solving a number of optimization problems [15]. While typical LNS approaches focus on iteratively refining lower bounds of a solution, we propose a method that refines both lower and upper bounds, imposing no restriction on the objective and constraints.

Contributions: This paper makes the following contributions: **(1)** We provide a novel distributed local search framework for DCOPs, which provides quality guarantees by refining both lower and upper bounds of the solution found during the iterative process; **(2)** We introduce a novel distributed search algorithm called *Tree-based DLNS (T-DLNS)*, which is built within the DLNS framework and characterized by the ability to exploit the problem structure—T-DLNS provides also a low computational complexity per agent; and **(3)** Evaluations against state-of-the-art incomplete DCOP algorithms that also return bounded solutions show that T-DLNS converges to better solutions providing tighter quality bounds.

[5] An extended abstract of this work [6] appeared at AAMAS 2015.



(a) Constraint Graph (b) Pseudo-tree (c) Constraints

Fig. 1: Example DCOP.

2 Background

DCOP: A *Distributed Constraint Optimization Problem (DCOP)* is a tuple $\langle \mathcal{X}, \mathcal{D}, \mathcal{F}, \mathcal{A}, \alpha \rangle$, where: $\mathcal{X} = \{x_1, \ldots, x_n\}$ is a set of *variables*; $\mathcal{D} = \{D_1, \ldots, D_n\}$ is a set of finite *domains* (i.e., $x_i \in D_i$); $\mathcal{F} = \{f_1, \ldots, f_e\}$ is a set of *utility functions* (also called *constraints*), where $f_i : \prod_{x_j \in \mathbf{x}^{f_i}} D_i \rightarrow \mathbb{R}_+ \cup \{-\infty\}$ and $\mathbf{x}^{f_i} \subseteq \mathcal{X}$ is the set of the variables (also called the *scope*) relevant to f_i; $\mathcal{A} = \{a_1, \ldots, a_p\}$ is a set of *agents*; and $\alpha : \mathcal{X} \rightarrow \mathcal{A}$ is a function that maps each variable to one agent. f_i specifies the utility of each combination of values assigned to the variables in $\mathbf{x}^{f_i}$. To ease readability, in the following, we assume all constraints are binary, and all agents control exactly one variable. Thus, we will use the terms "variable" and "agent" interchangeably and assume that $\alpha(x_i) = a_i$. The extensions to the n-ary constraint and multi-variable agents are straightforward [11].

A *partial assignment* σ is a value assignment to a set of variables $X_\sigma \subseteq \mathcal{X}$ that is consistent with the variables' domains. The utility $\mathcal{F}(\sigma) = \sum_{f \in \mathcal{F}, \mathbf{x}^f \subseteq X_\sigma} f(\sigma)$ is the sum of the utilities of all the applicable utility functions in σ. A *solution* is a partial assignment σ for all the variables of the problem, i.e., with $X_\sigma = \mathcal{X}$. We will denote with $\mathbf{x}$ a solution, while $\mathbf{x}_i$ is the value of x_i in $\mathbf{x}$. The goal is to find an optimal solution $\mathbf{x}^* = \text{argmax}_{\mathbf{x}} \mathcal{F}(\mathbf{x})$.

Given a DCOP P, $G = (\mathcal{X}, E)$ is the ***constraint graph*** of P, where $(x, y) \in E$ iff $\exists f_i \in \mathcal{F}$ s.t. $\{x, y\} = \mathbf{x}^{f_i}$. A ***DFS pseudo-tree*** arrangement for G is a *spanning tree* $T = \langle \mathcal{X}, E_T \rangle$ of G s.t. if $f_i \in \mathcal{F}$ 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* (resp. *out* of) E_T are called *tree edges* (resp. *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 the agent a_i. We denote with $G^k = \langle X^k, E^k \rangle$, the subgraph of G used in the execution of our iterative algorithms, where $X^k \subseteq \mathcal{X}$ and $E^k \subseteq E$.

Figure 1 depicts: (a) the constraint graph of a DCOP with agents $a_1, \ldots, a_4$, each controlling a variable with domain {0,1}, (b) a pseudo-tree (solid lines identify tree edges, dotted lines refer to backedges), and (c) the DCOP constraints.

Algorithm 1: DLNS

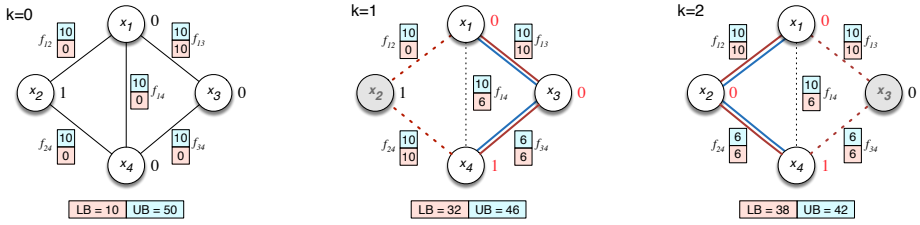
1 $k \leftarrow 0$;
2 $\langle \mathbf{x}_i^0, LB_i^0, UB_i^0 \rangle \leftarrow$ Value-Initialization();
3 **while** *termination condition is not met* **do**
4 $\quad k \leftarrow k + 1$;
5 $\quad z_i^k \leftarrow$ Destroy-Algorithm();
6 $\quad$ **if** $z_i^k = \circ$ **then** $\mathbf{x}_i^k \leftarrow NULL$; **else** $\mathbf{x}_i^k \leftarrow \mathbf{x}_i^{k-1}$;
7 $\quad \mathbf{x}_i^k \leftarrow$ Repair-Algorithm(z_i^k);
8 $\quad \langle LB_i^k, UB_i^k \rangle \leftarrow$ Bound-Algorithm($\mathbf{x}_i^k$);
9 $\quad$ **if** $LB_i^k = -\infty$ **then** $\mathbf{x}_i^k \leftarrow \mathbf{x}_i^{k-1}$;

LNS: In (*centralized*) *Large Neighborhood Search (LNS)* [35], an initial solution is iteratively improved by being repeatedly *destroyed* and *repaired*. Destroying a solution means selecting a subset of variables whose current values will be discarded. The set of such variables is the ***large neighborhood (LN)***. Repairing a solution means finding a new value assignment for the LN variables, given that the non-destroyed variables maintain their values from the previous iteration. The peculiarity of LNS, compared to other local search techniques, is the (larger) size of the neighborhood to explore at each step. It relies on the intuition that searching over a larger neighborhood allows the process to escape local optima and find better candidate solutions.

3 The DLNS Framework

In this section, we introduce DLNS, a general distributed LNS framework to solve DCOPs. It takes into account the restriction that each agent is only aware of its local subproblem (i.e., its neighbors and constraints) which makes centralized LNS techniques unsuitable and infeasible for solving DCOPs.

Algorithm 1 shows the general structure of DLNS, as executed by each agent $a_i \in \mathcal{A}$. After initializing its iteration counter k (line 1), its current value assignment $\mathbf{x}_i^0$ (as a random choice, solving a relaxed problem, or by exploiting domain knowledge, when available), and its current lower and upper bounds LB_i^0 and UB_i^0 of the optimal utility (line 2), the agent, like in LNS, iterates through the destroy and repair phases (lines 3–7). Next, the agent executes a *bound* phase (line 8) which updates the current lower and upper bounds. If the solution is not satisfiable (i.e., if it has a negative infinite lower bound utility), the agent restores its value assignment to that of the previous iteration (line 9). The process repeats until a termination condition occurs (line 3). Possible termination conditions include reaching a maximum value of k, a timeout limit, or a confidence threshold on the error of the reported best solution.



(a) (b) (c)

Fig. 2: DLNS with T-DBR example trace.

3.1 Destroy Phase

The result of this phase is the generation of a LN, which we refer to as LN^k as the subset of variables in $\mathcal{X}$ that will need to be *repaired* in each iteration k. This step is executed in a distributed fashion, having each agent a_i calling a DESTROY-ALGORITHM to determine if its local variable x_i should be ***destroyed*** ($\circ$) or ***preserved*** ($\star$), as indicated by the flag z_i^k (line 5). We say that destroyed (resp. preserved) variables are (resp. are not) in LN^k. In a destroy process, such decisions can be either random or made by exploiting domain knowledge. DLNS allows the agents to use any destroy schema to achieve the desired outcome. Once the destroyed variables are determined, the agents reset their values and keep the values of the preserved variables from the previous iteration (line 6).

Example 1. Figure 2 illustrates the execution of the DLNS algorithm (whose details will be discussed later) over the first 3 iterations. The value of each variable is shown on its right. Gray shaded nodes denote variables that have been preserved, while white nodes denote those that have been destroyed, and thus are in the LN of that iteration. The values for the variable x_2 in iteration 1 and x_3 in iteration 2 are preserved to their values in iterations 0 and 1, respectively.

3.2 Repair Phase

In the repair phase, the DLNS agents seek to find a new value assignment for the destroyed variables by calling the REPAIR-ALGORITHM function (line 7). This process is carried out exclusively by the destroyed agents with the goal of finding an improved solution by searching over the large neighborhood. The DLNS framework imposes no restriction on the choice of algorithms by agents. In each iteration k, the agents coordinate the resolution of two problems: $\check{P}^k$ and $\hat{P}^k$, which we call *relaxations* of the original DCOP problem P. They are used to compute, respectively, a lower and an upper bound on the optimal utility for P, and are defined as follows. Let $E^k_{LN} = \{(x,y) \mid (x,y) \in E; x,y \in LN^k\}$ be the set of constraints involving exclusively destroyed variables (i.e., those in LN^k),

- $\hat{G}^k = \langle LN^k, \hat{E}^k \rangle$ is the ***relaxation graph*** of $\hat{P}$ in iteration k, where $\hat{E}^k \subseteq E^k_{LN}$, is any subset of E^k_{LN}. The decision which edges to include in $\hat{E}^k$ characterizes the subproblem to solve in iteration k.

- $\check{G}^k = \langle LN^k, \check{E}^k \rangle$ is the ***relaxation graph*** of $\check{P}$, where $\check{E}^k = \hat{E}^k \cup \{(x, y) \mid (x, y) \in E; x \in LN^k, y \notin LN^k\}$. It is the union of $\hat{E}^k$ and the set of constraints whose scope has at least one destroyed variable.

Selecting $\hat{E}^k$ and $\check{E}^k$ is algorithmically dependent, and it is the factor that affects, in general, the algorithm's complexity—we show later a simple choice for $\hat{E}^k$ which allows our agents to solve each iteration in polynomial time.

In the problem $\check{P}^k$, we wish to find a partial assignment:

$$\underline{\check{\mathbf{x}}}^k = \underset{\mathbf{x}}{\operatorname{argmax}} \Big[\sum_{f \in \hat{E}^k} f(\mathbf{x}_i, \mathbf{x}_j) \quad + \sum_{\substack{f \in \mathcal{F},\ \mathbf{x}^f = \{x_i, x_j\} \\ x_i \in LN^k,\ x_j \notin LN^k}} f(\mathbf{x}_i, \check{\mathbf{x}}_j^{k-1}) \Big]$$

where $\check{\mathbf{x}}_j^{k-1}$ is the value assigned to the preserved variable x_j for problem $\check{P}^{k-1}$ in the previous iteration. The first summation is over all functions in $\hat{E}^k$, while the second is over all functions between a destroyed and a preserved variable. Thus, solving $\check{P}^k$ means optimizing over all the destroyed variables given that the preserved ones take on their previous value, and ignoring the set of edges $E \setminus \check{E}^k$ that are not part of the relaxation graph. This partial assignment is used to compute lower bounds during the *bounding phase*.

In the problem $\hat{P}^k$, we wish to find a partial assignment:

$$\underline{\hat{\mathbf{x}}}^k = \underset{\mathbf{x}}{\operatorname{argmax}} \sum_{f \in \hat{E}^k} f(\mathbf{x}_i, \mathbf{x}_j)$$

Thus, solving $\hat{P}^k$ means optimizing over all the destroyed variables considering exclusively the set of edges $\hat{E}^k$ that are part of the relaxation graph. This partial assignment is used to compute upper bounds during the *bounding phase*. Note that the partial assignments returned while solving these two relaxed problems involve exclusively the variables in LN^k.

Example 2. Consider again the example of Figure 2. The relaxation graphs $\check{G}^1$ (in red), $\hat{G}^1$ (in blue) and $\check{G}^2$, $\hat{G}^2$ are illustrated in subfigures (b) and (c), respectively, with the nodes colored white and edges represented by bold solid lines. All other constraints (of the original problem P) are represented by black dotted lines. In more detail, $LN^1 = \{x_1, x_3, x_4\}$, $\hat{E}^1 = \{f_{13}, f_{34}\}$, $\check{E}^1 = \{f_{13}, f_{34}, f_{12}, f_{24}\}$, and $LN^2 = \{x_1, x_2, x_4\}$, $\hat{E}^2 = \{f_{12}, f_{24}\}$, $\check{E}^2 = \{f_{12}, f_{24}, f_{13}, f_{34}\}$. At each step, the resolution of the relaxed problems involves the functions represented by bold lines—$\hat{P}$ is solved optimizing over the blue colored functions, and $\check{P}$ over the red ones. Recall that while solving $\hat{P}$ focuses solely on the functions in G^k, solving $\check{P}$ further accounts for the functions that involve a destroyed and a preserved variable.

3.3 Bounding Phase

Once the relaxed problems are solved, *all* agents start the bounding phase, which results in computing the lower and upper bounds based on the partial assignments $\underline{\check{\mathbf{x}}}^k$ and $\underline{\hat{\mathbf{x}}}^k$. To do so, both solutions to the problems $\check{P}^k$ and

$\hat{P}^k$ are extended to a solution $\check{\mathbf{x}}^k$ and $\hat{\mathbf{x}}^k$, respectively, for P, where the preserved variables $x_j \notin LN^k$ are assigned the values $\check{\mathbf{x}}_j^{k-1}$ from the previous iteration. The lower bound is computed by evaluating $\mathcal{F}(\check{\mathbf{x}}^k)$. The upper bound is computed by *combining* the optimal solution costs of two relaxed problems $\hat{P}^k$ and $\hat{P}^\ell$, solved in two different iterations. We focus on the case where $\ell < k$ is the iteration with the smallest upper bound found so far. Notice that $\sum_{f\in\hat{E}^k} f(\mathbf{x}_i,\mathbf{x}_j) \geq \sum_{f\in\hat{E}^k} f(\mathbf{x}_i^*,\mathbf{x}_j^*)$, where $\mathbf{x}^*$ is the optimal solution of P; therefore, reporting the optimal solutions found in two iterations will result in a larger utility, which is guaranteed to be an upper bound, albeit a conservative estimate. Thus, if a constraint is optimized in both iterations, we sum up the two solution qualities and subtract the minimum utility of that constraint. That will make the upper bound $\hat{F}^k(\hat{\mathbf{x}}^k) = \sum_{f\in\mathcal{F}} \hat{f}^k(\hat{\mathbf{x}}_i^k,\hat{\mathbf{x}}_j^k)$ smaller while preserving the correctness of the bound where:

$$\hat{f}^k(\mathbf{x}_i,\mathbf{x}_j) = \begin{cases} \hat{f}^k(\hat{\mathbf{x}}_i^k,\hat{\mathbf{x}}_j^k), & \text{if } f \in \hat{E}^k \setminus \hat{E}^\ell \\ \hat{f}^\ell(\hat{\mathbf{x}}_i^\ell,\hat{\mathbf{x}}_j^\ell), & \text{if } f \in \hat{E}^\ell \setminus \hat{E}^k \\ \hat{f}^k(\hat{\mathbf{x}}_i^k,\hat{\mathbf{x}}_j^k) + \hat{f}^\ell(\hat{\mathbf{x}}_i^\ell,\hat{\mathbf{x}}_j^\ell) - \min\limits_{d_i\in D_i, d_j\in D_j} f(d_i,d_j), & \text{if } f \in \hat{E}^k \cap \hat{E}^\ell \\ \max\limits_{d_i\in D_i, d_j\in D_j} f(d_i,d_j), & \text{otherwise.} \end{cases}$$

In other words, the utility of $\hat{F}^k(\hat{\mathbf{x}}^k)$ is composed of four parts. The first part involves the constraints considered while solving $\hat{P}^k$ at iteration k, excluding those involved at iteration ℓ. The second part includes the constraints considered at iteration ℓ, excluding those involved at current iteration k. The third part involves the constraints adopted in both problems' iterations ℓ and k, and the fourth part involves the remaining constraints which where excluded when constructing both problems $\hat{P}^\ell$ and $\hat{P}^k$ at iterations ℓ and k, respectively. We illustrate this process with the following example.

Example 3. Consider our example in Figure 2. When $k = 0$, in subfigure (a), each agent randomly assigns a value to its variable, which results in a solution with utility $\mathcal{F}(\check{\mathbf{x}}^0) = f(\check{\mathbf{x}}_1^0,\check{\mathbf{x}}_2^0) + f(\check{\mathbf{x}}_1^0,\check{\mathbf{x}}_3^0) + f(\check{\mathbf{x}}_1^0,\check{\mathbf{x}}_4^0) + f(\check{\mathbf{x}}_2^0,\check{\mathbf{x}}_4^0) + f(\check{\mathbf{x}}_3^0,\check{\mathbf{x}}_4^0) = 0{+}10{+}0{+}0{+}0{=}10$ to get the lower bound. Moreover, $\hat{P}^0$ chooses the maximum utility of every constraint at iteration 0 and yields an upper bound as $\hat{F}^0(\hat{\mathbf{x}}^0) = \max \hat{f}^0(\hat{\mathbf{x}}_1^0,\hat{\mathbf{x}}_2^0){+}\max \hat{f}^0(\hat{\mathbf{x}}_1^0,\hat{\mathbf{x}}_3^0){+}\max \hat{f}^0(\hat{\mathbf{x}}_1^0,\hat{\mathbf{x}}_4^0){+}\max \hat{f}^0(\hat{\mathbf{x}}_2^0,\hat{\mathbf{x}}_4^0){+}\max \hat{f}^0(\hat{\mathbf{x}}_3^0,\hat{\mathbf{x}}_4^0) = 10{+}10{+}10{+}10{+}10{=}50$.

In the first iteration ($k{=}1$), the destroy phase preserves x_2, thus $\check{\mathbf{x}}_2^1 {=} \check{\mathbf{x}}_2^0 {=} 1$. In this example, the chosen algorithm builds the spanning tree with the remaining variables choosing f_{13} and f_{34} as tree edges, so $E^1 = \{f_{13}, f_{34}\} \subset E_{LN}^1 = \{f_{13}, f_{34}, f_{14}\}$. Thus the relaxation graph for $\check{P}^1$ involves the edges $\{f_{13}, f_{34}, f_{12}, f_{24}\}$ (in red), and the relaxation graph for $\hat{P}^1$ involves the edges $\{f_{13}, f_{34}\}$ (in blue). Solving $\check{P}^1$ yields a partial assignment $\underline{\check{\mathbf{x}}}^1$ with utility $\check{F}^1(\underline{\check{\mathbf{x}}}^1) {=} f(\check{\mathbf{x}}_1^1,\check{\mathbf{x}}_3^1){+}f(\check{\mathbf{x}}_3^1,\check{\mathbf{x}}_4^1){+}f(\check{\mathbf{x}}_1^1,\check{\mathbf{x}}_2^1){+}f(\check{\mathbf{x}}_2^1,\check{\mathbf{x}}_4^1) {=} 10+6+0+10 {=} 26$, which results in a lower bound $\mathcal{F}(\check{\mathbf{x}}^1) = \check{F}^1(\underline{\check{\mathbf{x}}}^1) + f(\check{\mathbf{x}}_1^1,\check{\mathbf{x}}_4^1) = 26{+}6 = 32$. Solving $\hat{P}^1$ yields a solution $\hat{\mathbf{x}}^1$ with utility $\hat{F}^1(\hat{\mathbf{x}}^1) {=} \hat{f}^1(\hat{\mathbf{x}}_1^1,\hat{\mathbf{x}}_3^1){+}\hat{f}^1(\hat{\mathbf{x}}_3^1,\hat{\mathbf{x}}_4^1){+}\max \hat{f}^1(\hat{\mathbf{x}}_1^1,\hat{\mathbf{x}}_2^1){+}$

$\max \hat{f}^1(\hat{\mathbf{x}}_1^1, \hat{\mathbf{x}}_4^1) + \max \hat{f}^1(\hat{\mathbf{x}}_2^1, \hat{\mathbf{x}}_4^1) = 10+6+10+10+10 = 46$, which is the current upper bound. After the first iteration, we have $\ell = 1$ as $\hat{F}^1(\hat{\mathbf{x}}^1) < \hat{F}^0(\hat{\mathbf{x}}^0)$.

Finally, in the second iteration ($k=2$), the destroy phase retains x_3's value in the previous iteration $\check{\mathbf{x}}_3^2 = \check{\mathbf{x}}_3^1 = 0$, and the repair phase builds the new spanning tree with the remaining variables choosing f_{12} and f_{24} as tree edges with $E^2 = \{f_{12}, f_{24}\}$. Thus the relaxation graph for $\check{P}^2$ involves the edges $\{f_{12}, f_{24}, f_{13}, f_{34}\}$, and the relaxation graph for $\hat{P}^2$ involves the edges $\{f_{12}, f_{24}\}$. Solving $\check{P}^2$ and $\hat{P}^2$ yields partial assignments $\check{\underline{\mathbf{x}}}^2$ and $\hat{\underline{\mathbf{x}}}^2$, respectively, with utilities $\check{F}^2(\check{\underline{\mathbf{x}}}^2) = 10+6+10+6 = 32$, which results in a lower bound $\mathcal{F}(\check{\mathbf{x}}^2) = 32+6 = 38$, and an upper bound $\hat{F}^2(\hat{\mathbf{x}}^2) = \hat{f}^2(\hat{\mathbf{x}}_1^2, \hat{\mathbf{x}}_2^2) + \hat{f}^1(\hat{\mathbf{x}}_2^2, \hat{\mathbf{x}}_4^2) + \hat{f}^1(\hat{\mathbf{x}}_1^1, \hat{\mathbf{x}}_3^1) + \hat{f}^1(\hat{\mathbf{x}}_3^1, \hat{\mathbf{x}}_4^1) + \max \hat{f}^1(\hat{\mathbf{x}}_1^1, \hat{\mathbf{x}}_4^1) = 10+6+10+6+10 = 42$. After this iteration, $\ell = 2$.

Crucially, this framework enables DLNS to iteratively refine both lower and upper bounds of the solution, without imposing any restrictions on the form of the objective function and of the constraints adopted.[6]

4 Tree-based DLNS (T-DLNS)

Having discussed the general DLNS framework, we now introduce an efficient (polynomial-time) DLNS algorithm by specifying the construction of the problem relaxation graphs.

Tree-based DLNS (T-DLNS) defines the relaxed DCOPs $\check{P}^k$ and $\hat{P}^k$ using a spanning tree $T^k = \langle LN^k, E_{T^k} \rangle$, computed from G and LN^k and ignoring back edges. Solving the problem $\check{P}^k$ means optimizing over T^k and considering edges connecting destroyed and preserved variables. Thus, $\check{G}^k = \langle LN^k, \check{E}^k \rangle$ where $\check{E}^k = E_{T^k} \cup \{(x,y) \mid (x,y) \in E; x \in LN^k, y \notin LN^k\}$. Solving the problem $\hat{P}^k$ means optimizing over the spanning tree $\hat{G}^k = T^k$.

T-DLNS uses a complete inference-based algorithm composed of two phases operating on a tree-structured network [30]. This algorithm is complete on tree networks. Thus, while it will solve optimally and efficiently our relaxations, it will not guarantee to find an optimal solution for the original DCOP problem:

- In the *utility propagation* phase, each agent, starting from the leaves of the pseudo-tree, projects out its variable and sends its projected utilities to its parent. These utilities are propagated up the tree induced from E_{T^k} until they reach the root. The hard constraints of the problem are handled in this phase by pruning all inconsistent values before sending a message to its parent.
- Once the root agent receives the utilities from all its children, it starts the *value propagation* phase by selecting the value that maximizes its utility and sends it to its children, which repeat the same process. The problem is solved as soon as the values reach the leaves.

The solving schema of T-DLNS is similar to that of DPOP [30] in that it uses utility and value propagation phases; however, the different underlying relaxation graph adopted imposes several important differences. Algorithm 2

[6] However, it does not imply that the lower and upper bounds will converge to the same value.

Algorithm 2: T-DLNS(z_i^k)

10 $\mathbf{T}_i^k \leftarrow$ RELAXATION(z_i^k)
11 UTIL-PROPAGATION($\mathbf{T}_i^k$)
12 $\langle \check{\chi}_i^k, \hat{\chi}_i^k \rangle \leftarrow$ VALUE-PROPAGATION($\mathbf{T}_i^k$)
13 $\langle LB_i^k, UB_i^k \rangle \leftarrow$ BOUND-PROPAGATION($\check{\chi}_i^k, \hat{\chi}_i^k$)
14 **return** $\langle \check{\mathbf{x}}_i^k, LB_i^k, UB_i^k \rangle$

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

15 **receive** $\text{UTIL}_{a_c}(\check{U}_c, \hat{U}_c)$ from each $a_c \in C_i^k$
16 **forall** *values* $\mathbf{x}_i, \mathbf{x}_{P_i^k}$ **do**
17 $\quad \check{U}_i(\mathbf{x}_i, \mathbf{x}_{P_i^k}) \leftarrow f(\mathbf{x}_i, \mathbf{x}_{P_i^k}) + \sum_{a_c \in C_i^k} \check{U}_c(\mathbf{x}_i) + \sum_{x_j \notin LN^k} f(\mathbf{x}_i, \check{\mathbf{x}}_j^{k-1})$
18 $\quad \hat{U}_i(\mathbf{x}_i, \mathbf{x}_{P_i^k}) \leftarrow f(\mathbf{x}_i, \mathbf{x}_{P_i^k}) + \sum_{a_c \in C_i^k} \hat{U}_c(\mathbf{x}_i)$
19 **forall** *values* $\mathbf{x}_{P_i^k}$ **do**
20 $\quad \langle \check{U}_i'(\mathbf{x}_{P_i^k}), \hat{U}_i'(\mathbf{x}_{P_i^k}) \rangle \leftarrow \langle \max_{\mathbf{x}_i} \check{U}_i(\mathbf{x}_i, \mathbf{x}_{P_i^k}), \max_{\mathbf{x}_i} \hat{U}_i(\mathbf{x}_i, \mathbf{x}_{P_i^k}) \rangle$
21 **send** $\text{UTIL}_{a_i}(\check{U}_i', \hat{U}_i')$ msg to P_i^k

Function *VALUE-Propagation*($\mathbf{T}_i^k$)

22 **if** $P_i^k = NULL$ **then**
23 $\quad \langle \check{\mathbf{x}}_i^k, \hat{\mathbf{x}}_i^k \rangle \leftarrow \langle \text{argmax}_{\mathbf{x}_i} \check{U}_i(\mathbf{x}_i), \text{argmax}_{\mathbf{x}_i} \hat{U}_i(\mathbf{x}_i) \rangle$
24 $\quad$ **send** $\text{VALUE}_{a_i}(\check{\mathbf{x}}_i^k, \hat{\mathbf{x}}_i^k)$ msg to each $a_j \in N(a_i)$
25 $\quad$ **forall** $a_j \in N(a_i)$ **do**
26 $\quad\quad$ **receive** $\text{VALUE}_{a_j}(\check{\mathbf{x}}_j^k, \hat{\mathbf{x}}_j^k)$ msg from a_j
27 $\quad\quad$ Update x_j in $\langle \check{\chi}_i^k, \hat{\chi}_i^k \rangle$ with $\langle \check{\mathbf{x}}_j^k, \hat{\mathbf{x}}_j^k \rangle$
28 **else**
29 $\quad$ **forall** $a_j \in N(a_i)$ **do**
30 $\quad\quad$ **receive** $\text{VALUE}_{a_j}(\check{\mathbf{x}}_j^k, \hat{\mathbf{x}}_j^k)$ msg from a_j
31 $\quad\quad$ Update x_j in $\langle \check{\chi}_i^k, \hat{\chi}_i^k \rangle$ with $\langle \check{\mathbf{x}}_j^k, \hat{\mathbf{x}}_j^k \rangle$
32 $\quad\quad$ **if** $a_j = P_i^k$ **then**
33 $\quad\quad\quad \langle \check{\mathbf{x}}_i^k, \hat{\mathbf{x}}_i^k \rangle \leftarrow \langle \text{argmax}_{\mathbf{x}_i} \check{U}_i(\mathbf{x}_i), \text{argmax}_{\mathbf{x}_i} \hat{U}_i(\mathbf{x}_i) \rangle$
34 $\quad\quad\quad$ **send** $\text{VALUE}_{a_i}(\check{\mathbf{x}}_i^k, \hat{\mathbf{x}}_i^k)$ msg to each $a_j \in N(a_i)$
35 **return** $\langle \check{\chi}_i^k, \hat{\chi}_i^k \rangle$

shows the pseudocode of T-DLNS. We use the following notations: P_i^k, C_i^k, PP_i^k denote the parent, the set of children, and pseudo-parents of the agent a_i, at iteration k. The set of these items is referred to as $\mathbf{T}_i^k$, which is a_i's *local view* of the pseudo-tree T^k. $\check{\chi}_i$ and $\hat{\chi}_i$ denote a_i's *context* (i.e., the values for each $x_j \in N(a_i)$) with respect to problems $\check{P}$ and $\hat{P}$, respectively. We assume that by the end of the destroy phase (line 6) each agent knows its current context as well as which of its neighboring agents has been destroyed or preserved.

In each iteration k, T-DLNS executes these phases:

Repair Phase. It constructs a pseudo-tree T^k (line 10), which ignores, from G, the preserved variables as well as the functions involving these variables in their scopes. The construction prioritizes tree-edges that have not been chosen in

Procedure *BOUND-Propagation*($\check{\chi}_i^k, \hat{\chi}_i^k$)

36 **receive** BOUNDS$_{a_c}$(LB_c^k, UB_c^k) msg from each $a_c \in C_i$
37 $LB_i^k \leftarrow f(\check{\mathbf{x}}_i^k, \check{\mathbf{x}}_{P_i}^k) + \sum_{a_j \in PP_i} f(\check{\mathbf{x}}_i^k, \check{\mathbf{x}}_j^k) + \sum_{a_c \in C_i} LB_c^k$
38 $UB_i^k \leftarrow \hat{f}^k(\hat{\mathbf{x}}_i, \hat{\mathbf{x}}_{P_i}) + \sum_{a_j \in PP_i} \hat{f}^k(\hat{\mathbf{x}}_i, \hat{\mathbf{x}}_j) + \sum_{a_c \in C_i} UB_c^k$
39 **send** BOUNDS$_{a_i}$(LB_i^k, UB_i^k) msg to P_i

previous pseudo-trees over the others. The T-DLNS solving phase is composed of two phases operating on the relaxed pseudo-tree T^k, and executed synchronously:

1. *Utility Propagation:* After the pseudo-tree T^k is constructed (line 11), each leaf agent computes the optimal sum of utilities in its subtree considering exclusively tree edges (i.e., edges in E_{T^k}) and edges with destroyed variables. Each leaf agent computes the utilities $\check{U}_i(\mathbf{x}_i, \mathbf{x}_{P_i^k})$ and $\hat{U}_i(\mathbf{x}_i, \mathbf{x}_{P_i^k})$ for each pair of values of its variable $\mathbf{x}_i$ and its parent's variable $\mathbf{x}_{P_i^k}$ (lines 16-18), in preparation for retrieving the solutions of $\check{P}$ and $\hat{P}$, used during the bounding phase. The agent projects itself out (lines 19-20) and sends the projected utilities to its parent in a UTIL message (line 21). Each agent, upon receiving the UTIL message from each child, performs the same operations. Thus, these utilities will propagate up the pseudo-tree until they reach the root agent.
2. *Value Propagation:* Once the utility propagation is compleated (line 12) the root agent computes its optimal values $\check{\mathbf{x}}_i^k$ and $\hat{\mathbf{x}}_i^k$ for the relaxed DCOPs $\check{P}$ and $\hat{P}$, respectively (line 23). Then, it sends its values to all its neighbors in a VALUE message (line 24). When any of its children receive this message, they also compute their optimal values and sends them to all their neighbors (lines 32-34). Thus, these values propagate down the pseudo-tree until they reach the leaves, at which point every agent has chosen its respective values. In this phase, in preparation for the bounding phase, when each agent receives a VALUE message from its neighbor, it will also update the corresponding value in its contexts $\check{\chi}_i^k$ and $\hat{\chi}_i^k$ (lines 25-27 and 30-31).

Bounding Phase. Once the relaxed DCOPs $\check{P}$ and $\hat{P}$ have been solved, the algorithm starts the bound propagation phase (line 13). Each leaf agent of the pseudo-tree T computes the lower and upper bounds LB_i^k and UB_i^k (lines 37-38). These bounds are sent to the agent's parent in T (line 39). When its parent receives this message from all its children (line 36), it performs the same operations. The lower and upper bounds of the whole problem are determined when the bounds reach the root agent.

5 Theoretical Properties

Theorem 1. *For each LN^k, $\mathcal{F}(\check{\mathbf{x}}^k) \leq \mathcal{F}(\mathbf{x}^*) \leq \hat{F}^k(\hat{\mathbf{x}}^k)$.*

Proof. The result $\mathcal{F}(\check{\mathbf{x}}^k) \leq \mathcal{F}(\mathbf{x}^*)$ follows from that $\check{\mathbf{x}}^k$ is an optimal solution of the relaxed problem $\check{P}$ whose functions are a subset of $\mathcal{F}$.

By definition of $\hat{F}^k(\mathbf{x})$, it follows that:

$$
\begin{aligned}
\hat{F}^k(\hat{\mathbf{x}}^k) &= \sum_{f\in\mathcal{F}} \hat{f}^k(\hat{\mathbf{x}}^k_i, \hat{\mathbf{x}}^k_j) \\
&= \sum_{f\in\hat{E}^k\setminus\hat{E}^\ell} \hat{f}^k(\hat{\mathbf{x}}^k_i, \hat{\mathbf{x}}^k_j) + \sum_{f\in\hat{E}^\ell\setminus\hat{E}^k} \hat{f}^\ell(\hat{\mathbf{x}}^\ell_i, \hat{\mathbf{x}}^\ell_j) + \sum_{f\notin\hat{E}^k\cup\hat{E}^\ell} \max_{d_i\in D_i, d_j\in D_j} f(d_i, d_j) \\
&\quad + \sum_{f\in\hat{E}^k\cap\hat{E}^\ell} \left(\hat{f}^k(\hat{\mathbf{x}}^k_i, \hat{\mathbf{x}}^k_j) + \hat{f}^\ell(\hat{\mathbf{x}}^\ell_i, \hat{\mathbf{x}}^\ell_j) - \min_{d_i\in D_i, d_j\in D_j} f(d_i, d_j) \right) \\
&= \sum_{f\in\hat{E}^k} \hat{f}^k(\hat{\mathbf{x}}^k_i, \hat{\mathbf{x}}^k_j) + \sum_{f\in\hat{E}^\ell} \hat{f}^\ell(\hat{\mathbf{x}}^\ell_i, \hat{\mathbf{x}}^\ell_j) \\
&\quad + \sum_{f\notin\hat{E}^k\cup\hat{E}^\ell} \max_{d_i\in D_i, d_j\in D_j} f(d_i, d_j) - \sum_{f\in\hat{E}^k\cap\hat{E}^\ell} \min_{d_i\in D_i, d_j\in D_j} f(d_i, d_j) \\
&\geq \sum_{f\in\hat{E}^k} f(\hat{\mathbf{x}}^*_i, \hat{\mathbf{x}}^*_j) + \sum_{f\in\hat{E}^\ell} f(\hat{\mathbf{x}}^*_i, \hat{\mathbf{x}}^*_j) + \sum_{f\notin\hat{E}^k\cup\hat{E}^\ell} f(\hat{\mathbf{x}}^*_i, \hat{\mathbf{x}}^*_j) - \sum_{f\in\hat{E}^k\cap\hat{E}^\ell} f(\hat{\mathbf{x}}^*_i, \hat{\mathbf{x}}^*_j) \\
&\geq \sum_{f\in\hat{E}^k\cup\hat{E}^\ell} f(\hat{\mathbf{x}}^*_i, \hat{\mathbf{x}}^*_j) + \sum_{f\notin\hat{E}^k\cup\hat{E}^\ell} f(\hat{\mathbf{x}}^*_i, \hat{\mathbf{x}}^*_j) \\
&\geq \sum_{f\in\mathcal{F}} f(\hat{\mathbf{x}}^*_i, \hat{\mathbf{x}}^*_j) \\
&\geq \mathcal{F}(\mathbf{x}^*)
\end{aligned}
$$

and, thus, $\mathcal{F}(\check{\mathbf{x}}^k) \leq \mathcal{F}(\mathbf{x}^*) \leq \hat{F}^k(\hat{\mathbf{x}}^k)$ for each LN^k. ∎

Corollary 1. *An approximation ratio for the problem is*

$$\rho = \frac{\min_k \hat{F}^k(\hat{\mathbf{x}}^k)}{\max_k \mathcal{F}(\check{\mathbf{x}}^k)} \geq \frac{\mathcal{F}(\mathbf{x}^*)}{\max_k \mathcal{F}(\check{\mathbf{x}}^k)}.$$

Theorem 2. *In each iteration, T-DLNS requires $O(|\mathcal{F}|)$ number of messages of size in $O(d)$, where $d = \max_{a_i\in\mathcal{A}} |D_i|$.*

Proof. In the *Value Propagation Phase* of Algorithm 2, each agent sends a message to its neighbors (lines 24 and 34). Thus, the overall amount of messages sent in this phase by the agents is $2\|\mathcal{F}\|$. All other phases use up to $|\mathcal{A}|$ messages (which are reticulated from the leaves to the root of the pseudo-tree and vice-versa). Therefore, T-DLNS requires $O(|\mathcal{F}|)$ messages in each iteration. The largest messages are sent during the *Utility Propagation Phase*, where each agent (excluding the root agent) sends a message containing a value for each element of its domain (line 21). Thus, the size of the DLNS messages is in $O(d)$. ∎

Theorem 3. *In each iteration, the number of constraint checks of each T-DLNS agent is in $O(d^2)$, where $d = \max_{a_i\in\mathcal{A}} |D_i|$.*

Proof. The largest amount of constraint checks per iteration is performed during the Util-Propagation Phase. In this phase, each agent (except the root agent) computes the lower and upper bound utilities for each value of its variable $\mathbf{x}_i$ and its parent's variable $\mathbf{x}_{P^k_i}$ (lines 17–18). Therefore, the number of constraint checks per iteration of each agent is in $O(d^2)$. ∎

6 Related Work

In addition to the algorithms described in the introduction, several extensions to complete search-based algorithms that trade solution quality for faster runtimes have been proposed [24, 40]. Another line of work has investigated non-iterative versions of inference-based incomplete DCOP algorithms, such as ADPOP [29] and p-OPT [26]. These algorithms operate on relaxations of the original DCOP. ADPOP is an incomplete version of DPOP that bounds the maximal message size transmitted over the network, trading off message size for better runtimes. p-OPT ignores some edges of the induced chordal graph of the DCOP and solves exactly the problem over such subgraphs to generate an approximate solution. Both algorithms are different from DLNS in that they operate in a single iteration only and, thus, do not refine the solution found. The algorithm that shares most similarities with DLNS is LS-DPOP [31]. LS-DPOP runs several local searches on pseudo-trees. However, unlike DLNS, it operates in a single iteration, does not change its neighborhood, and does not provide quality guarantees.

7 Experimental Results

We evaluate the DLNS framework against representative state-of-the-art incomplete DCOP algorithms, with and without quality guarantees. We choose T-DLNS as a representative algorithm of the DLNS framework. We select DSA as a representative incomplete *search*-based DCOP algorithm; Max-Sum (MS) and Bounded MS (BMS) as representative *inference*-based DCOP algorithms; and k-optimal algorithms (KOPT2 and KOPT3) as representative *region optimal*-based DCOP methods. All algorithms are selected based on their performance and popularity. We use the FRODO framework [20] to run MS and DSA,[7] the authors' code of BMS [32], and the DALO framework [17] for KOPT. We also force T-DLNS first large neighboring exploration to use the same tree as that used by BMS. We experimentally observed that using this option improves the effectiveness of T-DLNS in finding high quality solutions.

Random DCOPs. First, we evaluate the algorithms on random DCOPs over *random*, *grid*, and *scale-free* topologies. The instances for each topology are generated as follows: For random networks, we create an n-node network, whose density p_1 produces $\lfloor n\,(n-1)\,p_1 \rfloor$ edges in total. We do not bound the tree-width, which is based on the underlying graph. For grid networks, we create an n-node network arranged in a rectangular grid, where internal nodes are connected to four neighboring nodes and nodes on the edges (resp. corners) are connected to two (resp. three) neighbors. Finally, for scale-free networks, we create an n-node network based on the Barabasi-Albert model [2]. Starting from a connected 2-node network, we repeatedly add a new node, randomly connecting it to two existing nodes. In turn, these two nodes are selected with probabilities that are

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

$\|\mathcal{A}\|$	HC	T-DLNS				BMS				KOPT2				KOPT3				MS		DSA	
		ρ	LB	UB	t	ρ	LB	UB	t	ρ	LB	UB	t	ρ	LB	UB	t	LB	t	LB	t
25		**1.36**	**0.95**	**0.97**	377	1.75	0.78	0.92	310	9.14	0.88	0.15	293	7.10	0.89	0.20	1204	0.88	253	0.93	**84**
	✓	**1.86**	**0.94**	**0.97**	341	2.70	0.65	**0.97**	321	9.14	0.82	0.22	298	7.10	0.85	0.28	1319	0.84	235	0.91	**81**
64		**1.57**	0.94	**0.97**	2989	1.91	0.82	0.92	2704	21.88	0.87	0.07	3051	16.66	0.76	0.11	4935	0.86	2471	**0.96**	**241**
	✓	**2.46**	0.92	**0.97**	1703	3.27	0.69	**0.97**	2936	21.88	0.77	0.13	3124	16.66	0.64	0.20	4943	0.78	2188	**0.95**	**235**
100		**1.64**	0.94	**0.97**	7580	1.95	0.84	0.92	6312	33.64	0.82	0.05	5222	25.48	0.80	0.07	8193	0.86	6368	**0.97**	**432**
	✓	**2.70**	0.92	**0.97**	5089	3.49	0.71	**0.97**	7324	33.64	0.68	0.10	5166	25.48	0.64	0.14	7884	0.76	6852	**0.96**	**540**
144		**1.70**	0.94	**0.97**	26156	1.97	0.86	0.92	19372	48.02	0.82	0.03	14223	36.26	0.83	0.05	17239	0.88	6578	**0.97**	**722**
	✓	**2.88**	0.91	**0.97**	16562	3.58	0.73	**0.97**	20313	48.02	0.68	0.07	15421	36.26	0.68	0.10	16342	0.78	7161	**0.96**	**1517**

Table 1: Experimental results on *random* networks. Times are in ms.

$\|\mathcal{A}\|$	HC	T-DLNS				BMS				KOPT2				KOPT3				MS		DSA	
		ρ	LB	UB	t	ρ	LB	UB	t	ρ	LB	UB	t	ρ	LB	UB	t	LB	t	LB	t
25		**1.06**	**0.96**	**0.94**	270	1.26	0.83	0.92	**17**	9.14	0.91	0.11	85	7.10	0.93	0.14	108	0.93	31	0.90	38
	✓	**1.16**	**0.94**	**0.92**	279	1.42	0.75	0.90	**19**	9.14	0.85	0.12	82	7.10	0.88	0.15	107	0.87	31	0.82	35
64		**1.08**	**0.96**	**0.97**	759	1.29	0.83	0.94	79	21.88	0.91	0.05	199	16.66	0.92	0.06	266	0.82	**51**	0.91	52
	✓	**1.21**	**0.95**	**0.93**	716	1.49	0.77	0.92	64	21.88	0.87	0.05	200	16.66	0.88	0.07	264	0.75	55	0.85	**53**
100		**1.09**	**0.96**	**0.97**	1422	1.30	0.83	0.94	173	33.64	0.91	0.03	319	25.48	0.92	0.04	415	0.84	**58**	0.91	64
	✓	**1.23**	**0.96**	**0.96**	1253	1.51	0.77	0.94	151	33.64	0.87	0.03	319	25.48	0.89	0.04	422	0.75	**58**	0.87	69
144		**1.10**	**0.96**	**0.97**	2900	1.31	0.83	0.94	249	48.02	0.92	0.02	833	36.26	0.93	0.03	1082	0.83	**69**	0.92	185
	✓	**1.24**	**0.95**	**0.96**	2489	1.52	0.76	0.95	227	48.02	0.86	0.02	460	36.24	0.89	0.03	655	0.69	**44**	0.87	165

Table 2: Experimental results on *grid* networks. Times are in ms.

$\|\mathcal{A}\|$	HC	T-DLNS				BMS				KOPT2				KOPT3				MS		DSA	
		ρ	LB	UB	t	ρ	LB	UB	t	ρ	LB	UB	t	ρ	LB	UB	t	LB	t	LB	t
25		**1.22**	**0.95**	**0.94**	523	1.58	0.78	0.90	137	9.14	0.89	0.13	594	7.10	0.89	0.17	1498	0.85	166	0.91	**111**
	✓	**1.54**	**0.93**	**0.83**	299	2.17	0.65	0.82	142	9.14	0.82	0.15	160	7.10	0.81	0.20	535	0.83	162	0.85	**60**
25		**1.28**	**0.95**	**0.96**	1099	1.62	0.79	0.91	393	21.88	0.90	0.05	641	16.66	0.84	0.08	1901	0.85	414	0.93	**150**
	✓	**1.64**	**0.94**	**0.96**	880	2.23	0.69	0.95	423	21.88	0.84	0.08	544	16.66	0.75	0.11	1256	0.78	414	0.89	**106**
25		**1.30**	**0.96**	**0.97**	1922	1.61	0.80	0.93	712	33.64	0.91	0.03	935	25.48	0.84	0.05	3199	0.85	532	0.94	**180**
	✓	**1.70**	**0.95**	**0.95**	1540	2.26	0.71	0.94	832	33.64	0.86	0.05	947	25.48	0.75	0.08	1914	0.75	584	0.93	**293**
25		**1.31**	**0.96**	**0.97**	4322	1.61	0.82	0.93	1585	48.02	0.92	0.02	1651	36.26	0.84	0.04	3614	0.84	626	0.95	**447**
	✓	**1.74**	**0.94**	**0.97**	3294	2.27	0.71	0.96	1681	48.02	0.87	0.03	1714	36.26	0.74	0.05	3211	0.72	647	0.93	**447**

Table 3: Experimental results on *scale-free* networks. Times are in ms.

proportional to the numbers of their connected edges. The total number of edges is $2\,(n-2)+1$.

We generate 50 instances for each topology, ensuring that the underlying graph is connected. The utility functions are generated using random utilities in $[0, 100]$. We set as default parameters, $|D_i| = 10$ for all variables, $p_1 = 0.5$, and for instances with hard constraints, $p_2 = 0.5$. We use a random destroy strategy for the T-DLNS algorithms, in which each agent destroies a variable with probability $p = 0.5$. The runtime of all the algorithms is measured using the *simulated runtime* metric [36], and averaged over all instances. The experiments are performed on an Intel i7 Quadcore 3.4GHz machine with 16GB of RAM.

Tables 1– 3 report the approximation ratio ρ, simulated runtime t, the normalized lower bound (LB), and the normalized upper bound (UB) values. The LB (UB) value of each algorithm is normalized over the LBs (UBs) reported by all algorithms. A normalized lower (upper) bound of 0 means that it is the worst lower (upper) bound among all lower (upper) bounds. Similarly, a normalized lower (upper) bound of 1 means that it is the best lower (upper) bound. The best

approximation ratios, normalized lower (upper) bounds, and runtimes are shown in bold. All tables report results for problem with and without hard constraints (HC).

Table 1 tabulates the results for random networks. Among all algorithms that do provide upper bounds (i.e., T-DLNS, BMS, KOPT2, and KOPT3), T-DLNS find the highest values. Additionally, T-DLNS also provides the best approximation ratios among all such algorithms. However, this comes at a cost of increased runtimes compared to the other algorithms. The quality of the solutions reported by DSA exceeds, albeit slightly, that of T-DLNS. However, DSA provides no quality guarantees. In general, all algorithms that do provide upper bounds have larger runtimes than those that do not provide these bounds. This behavior is not surprising since these algorithms require additional computation to compute and provide these bounds.

Table 2 tabulates the results for grid networks. These results are similar to the ones on random networks. Additionally, T-DLNS also outperforms DSA in finding solutions of high qualities (i.e., large LBs). However, this dominance comes at a price: T-DLNS has the largest runtime, on average, among all algorithms. Also, whereas DSA is the fastest algorithm for solving random networks, MS is the fastest for solving grid networks. This is due to that the computational time for MS is exponential in the arity of each variable and in grid networks each variable has significantly fewer neighbors than in random networks.

Finally, Table 3 tabulates the results for scale-free networks. The trend in this topology is similar to those in random and grid networks: T-DLNS provides better lower and upper bounds and, consequently, better approximation ratios compared to all other algorithms.

While T-DLNS does have larger runtimes than its competitors, it consistently outperforms state-of-the-art incomplete DCOP algorithms that do provide error bounds: It finds both higher solutions' qualities and tighter upper bounds. In the majority of the cases, it also outperforms state-of-the-art incomplete DCOP algorithms that do not provide error bounds.

Distributed Meeting Scheduling. Next, we evaluate the ability of T-DLNS to exploit the domain knowledge over *distributed meeting scheduling problems*. In such problems, one wishes to schedule a set of events within a time range. We use the *event as variable* formulation [21], where events are modeled as decision variables. Meeting participants can attend different meetings and have time preferences that are taken into account in the problem formulation. Each variable can take on a value from the interval $[0, 100]$. The problem requires that no meetings sharing some participants can overlap. We generate the underlying constraint network using the random network model described earlier. Our analysis focuses on compare T-DLNS using a *random* (RN) destroy and a *domain-knowledge* (DK) destroy strategies. The former randomly selects a set of variables to destroy, while the latter destroys the set of variables that are in overlapping meetings. Table 4 reports the percentage of satisfied instances reported (% SAT) and the time needed to find the first satisfiable solution (TF), averaged over 50 runs. The *domain-knowledge* destroy has a clear advantage over

Meetings	20		50		100	
	% SAT	t (ms)	% SAT	t (ms)	% SAT	t (ms)
DK destroy	**80.05**	**78**	**54.11**	**342**	**31.20**	**718**
RN destroy	12.45	648	1.00	52207	0.00	–
KOPT3	4.30	110367	0.00	–	–	–

Table 4: Experimental results on *meeting scheduling*.

the *random* one, being able to effectively exploit domain knowledge. All other local search algorithms tested failed to report satisfiable solutions for any of the problems—only KOPT3 was able to find some solutions for problem instances with 20 meetings.

Thus, our experiments suggest that DLNS can bring a decisive advantage on both general and domain-specific problems, where exploiting structure can be done explicitly within the destroy phase.

8 Conclusions

In this paper, we proposed a *Distributed Large Neighborhood Search* (DLNS) framework to find quality-bounded solutions in DCOPs. DLNS is composed of a destroy phase, which selects a neighborhood to search, and a repair phase, which performs the search over such neighborhood. Within DLNS, we proposed a novel distributed algorithm, T-DLNS, characterized by low network usage and low computational complexity per agent. Our experimental results showed that T-DLNS finds better solutions compared to representative search-, inference-, and region-optimal-based incomplete DCOP algorithms. The proposed results are significant—the anytime property and the ability to refine online quality guarantees makes DLNS-based algorithms good candidates to solve a wide class of DCOP problems. We strongly believe that this framework has the potential to solve very large distributed constraint optimization problems, with thousands of agents, variables, and constraints. In the future, we plan to investigate other schemes to incorporate into the repair phase of DLNS, including constraint propagation techniques [3, 7, 16] to better prune the search space, techniques that actively exploit the bounds reported during the iterative procedure, as well as the use of general purpose graphics processing units to parallelize the search for better speedups [4, 9], especially when agents have large local subproblems.

Acknowledgments

The research at the Washington University in St. Louis was supported by the National Science Foundation (NSF) under grant numbers 1550662 and 1540168. The research at New Mexico State University was supported by the NSF under grant numbers 1458595 and 1345232. The views and conclusions contained in this document are those of the authors only.

References

1. Auer, P.: Using confidence bounds for exploitation-exploration trade-offs. Journal of Machine Learning Research **3**(Nov), 397–422 (2002)
2. Barabási, A.L., Albert, R.: Emergence of scaling in random networks. Science **286**(5439), 509–512 (1999)
3. Bessiere, C., Gutierrez, P., Meseguer, P.: Including Soft Global Constraints in DCOPs. In: Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP). pp. 175–190 (2012)
4. Campeotto, F., Dovier, A., Fioretto, F., Pontelli, E.: A GPU implementation of large neighborhood search for solving constraint optimization problems. In: Proceedings of the European Conference on Artificial Intelligence (ECAI). pp. 189–194 (2014)
5. Farinelli, A., Rogers, A., Petcu, A., Jennings, N.: Decentralised coordination of low-power embedded devices using the Max-Sum algorithm. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS). pp. 639–646 (2008)
6. Fioretto, F., Campeotto, F., Dovier, A., Pontelli, E., Yeoh, W.: Large Neighborhood Search with Quality Guarantees for Distributed Constraint Optimization Problems. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS). pp. 1835–1836 (2015)
7. Fioretto, F., Le, T., Yeoh, W., Pontelli, E., Son, T.C.: Improving DPOP with branch consistency for solving distributed constraint optimization problems. In: Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP). pp. 307–323 (2014)
8. Fioretto, F., Pontelli, E., Yeoh, W.: Distributed constraint optimization problems and applications: A survey. Journal of Artificial Intelligence Research **61**, 623–698 (2018)
9. Fioretto, F., Pontelli, E., Yeoh, W., Dechter, R.: Accelerating exact and approximate inference for (distributed) discrete optimization with GPUs. Constraints **23**(1), 1–43 (2018)
10. Fioretto, F., Yeoh, W., Pontelli, E.: A dynamic programming-based MCMC framework for solving DCOPs with GPUs. In: Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP). pp. 813–831 (2016)
11. Fioretto, F., Yeoh, W., Pontelli, E.: Multi-variable agents decomposition for DCOPs. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI). pp. 2480–2486 (2016)
12. Fioretto, F., Yeoh, W., Pontelli, E.: A multiagent system approach to scheduling devices in smart homes. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS). pp. 981–989 (2017)
13. Fioretto, F., Yeoh, W., Pontelli, E., Ma, Y., Ranade, S.: A DCOP approach to the economic dispatch with demand response. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS). pp. 999–1007 (2017)
14. Geman, S., Geman, D.: Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. IEEE Transactions on Pattern Analysis and Machine Intelligence **6**(6), 721–741 (1984)
15. Godard, D., Laborie, P., Nuijten, W.: Randomized large neighborhood search for cumulative scheduling. In: Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS). vol. 5, pp. 81–89 (2005)

16. Gutierrez, P., Lee, J.H.M., Lei, K.M., Mak, T.W.K., Meseguer, P.: Maintaining Soft Arc Consistencies in BnB-ADOPT$^+$ during Search. In: Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP). pp. 365–380 (2013)
17. Kiekintveld, C., Yin, Z., Kumar, A., Tambe, M.: Asynchronous algorithms for approximate distributed constraint optimization with quality bounds. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS). pp. 133–140 (2010)
18. Kluegel, W., Iqbal, M.A., Fioretto, F., Yeoh, W., Pontelli, E.: A realistic dataset for the smart home device scheduling problem for DCOPs. In: Sukthankar, G., Rodriguez-Aguilar, J.A. (eds.) AAMAS 2017 Workshops (Visionary Papers), pp. 125–142. Springer International Publishing (2017)
19. Kumar, A., Faltings, B., Petcu, A.: Distributed constraint optimization with structured resource constraints. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS). pp. 923–930 (2009)
20. Léauté, T., Ottens, B., Szymanek, R.: Frodo 2.0: An open-source framework for distributed constraint optimization. In: International Workshop on Distributed Constraint Reasoning (DCR). pp. 160–164 (2009)
21. Maheswaran, R., Pearce, J., Tambe, M.: Distributed algorithms for DCOP: A graphical game-based approach. In: Proceedings of the Conference on Parallel and Distributed Computing Systems (PDCS). pp. 432–439 (2004)
22. Maheswaran, R., Tambe, M., Bowring, E., Pearce, J., Varakantham, P.: Taking DCOP to the real world: Efficient complete solutions for distributed event scheduling. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS). pp. 310–317 (2004)
23. Miller, S., Ramchurn, S., Rogers, A.: Optimal decentralised dispatch of embedded generation in the smart grid. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS). pp. 281–288 (2012)
24. Modi, P., Shen, W.M., Tambe, M., Yokoo, M.: ADOPT: Asynchronous distributed constraint optimization with quality guarantees. Artificial Intelligence **161**(1–2), 149–180 (2005)
25. Nguyen, D.T., Yeoh, W., Lau, H.C.: Distributed Gibbs: A memory-bounded sampling-based DCOP algorithm. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS). pp. 167–174 (2013)
26. Okimoto, T., Joe, Y., Iwasaki, A., Yokoo, M., Faltings, B.: Pseudo-tree-based incomplete algorithm for distributed constraint optimization with quality bounds. In: Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP), pp. 660–674 (2011)
27. Ottens, B., Dimitrakakis, C., Faltings, B.: DUCT: An upper confidence bound approach to distributed constraint optimization problems. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI). pp. 528–534 (2012)
28. Pearce, J., Tambe, M.: Quality guarantees on k-optimal solutions for distributed constraint optimization problems. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI). pp. 1446–1451 (2007)
29. Petcu, A., Faltings, B.: Approximations in distributed optimization. In: Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP), pp. 802–806 (2005)
30. Petcu, A., Faltings, B.: A scalable method for multiagent constraint optimization. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI). pp. 1413–1420 (2005)

31. Petcu, A., Faltings, B.: A hybrid of inference and local search for distributed combinatorial optimization. In: Proceedings of the International Conference on Intelligent Agent Technology (IAT). pp. 342–348 (2007)
32. Rogers, A., Farinelli, A., Stranders, R., Jennings, N.: Bounded approximate decentralised coordination via the max-sum algorithm. Artificial Intelligence **175**(2), 730–759 (2011)
33. Rollon, E., Larrosa, J.: Improved bounded max-sum for distributed constraint optimization. In: Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP). pp. 624–632. Springer (2012)
34. Rust, P., Picard, G., Ramparany, F.: Using message-passing dcop algorithms to solve energy-efficient smart environment configuration problems. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI). pp. 468–474 (2016)
35. Shaw, P.: Using constraint programming and local search methods to solve vehicle routing problems. In: Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP), pp. 417–431 (1998)
36. Sultanik, E., Modi, P.J., Regli, W.C.: On modeling multiagent task scheduling as a distributed constraint optimization problem. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI). pp. 1531–1536 (2007)
37. Ueda, S., Iwasaki, A., Yokoo, M.: Coalition structure generation based on distributed constraint optimization. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI). pp. 197–203 (2010)
38. Vinyals, M., Shieh, E., Cerquides, J., Rodriguez-Aguilar, J., Yin, Z., Tambe, M., Bowring, E.: Quality guarantees for region optimal DCOP algorithms. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS). pp. 133–140 (2011)
39. Yeoh, W., Felner, A., Koenig, S.: BnB-ADOPT: An asynchronous branch-and-bound DCOP algorithm. Journal of Artificial Intelligence Research **38**, 85–133 (2010)
40. Yeoh, W., Sun, X., Koenig, S.: Trading off solution quality for faster computation in DCOP search algorithms. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI). pp. 354–360 (2009)
41. Yeoh, W., Yokoo, M.: Distributed problem solving. AI Magazine **33**(3), 53–65 (2012)
42. Zhang, W., Wang, G., Xing, Z., Wittenberg, L.: Distributed stochastic search and distributed breakout: Properties, comparison and applications to constraint optimization problems in sensor networks. Artificial Intelligence **161**(1–2), 55–87 (2005)
43. Zivan, R., Yedidsion, H., Okamoto, S., Glinton, R., Sycara, K.: Distributed constraint optimization for teams of mobile sensing agents. Journal of Autonomous Agents and Multi-Agent Systems **29**(3), 495–536 (2015)