Large Neighborhood Search with Quality Guarantees for Distributed Constraint Optimization Problems

Ferdinando Fioretto[1,2], Federico Campeotto[2],
Agostino Dovier[2], Enrico Pontelli[1], and William Yeoh[1]

[1] Department of Computer Science, New Mexico State University
[2] Department of Mathematics & Computer Science, University of Udine
{ffiorett,wyeoh,epontell}@cs.nmsu.edu,
{federico.campeotto,agostino.dovier}@uniud.it

Abstract. The field of Distributed Constraint Optimization has gained momentum in recent years, thanks to its ability to address various applications related to multi-agent cooperation. Nevertheless, solving Distributed Constraint Optimization Problems (DCOPs) optimally is NP-hard. Therefore, in large-scale applications, incomplete DCOP algorithms are desirable. Current incomplete search techniques have subsets of the following limitations: (a) they find local minima without quality guarantees; (b) they provide loose quality assessment; or (c) they cannot exploit certain problem structures such as hard constraints. Therefore, capitalizing on strategies from the centralized constraint reasoning community, we propose to adapt the *Large Neighborhood Search* (LNS) strategy to solve DCOPs, resulting in the general *Distributed LNS* (D-LNS) framework. The characteristics of this framework are as follows: *(i)* it is anytime; *(ii)* it provides quality guarantees by refining online upper and lower bounds on its solution quality; and *(iii)* it can learn online the best neighborhood to explore. Experimental results show that D-LNS outperforms other incomplete DCOP algorithms for both random and scale-free network instances.

1 Introduction

In a *distributed constraint optimization problem (DCOP)*, agents coordinate their value assignments to maximize the sum of resulting constraint utilities [17, 29]. Researchers have used them to model various multi-agent coordination and resource allocation problems [14, 30].

Complete DCOP algorithms [17, 21] find optimal DCOP solutions at the cost of a large runtime or network load, while incomplete approaches trade off optimality for lower usage of resources. Since finding optimal DCOP solutions is NP-hard, incomplete algorithms are often necessary to solve large problems of interest. Unfortunately, several local search algorithms (e.g., DBA [12], MGM [15]) and local inference algorithms (e.g., Max-Sum [6]) do not provide any guarantees on the quality of the solutions found. More recent developments, such as region-optimal algorithms [20, 27], Bounded Max-Sum [23], and Divide-and-Conquer (DaC) algorithms [26, 11] alleviate this limitation. Region-optimal algorithms allow one to specify regions with a maximum size of k agents or t hops from each agent, and they optimally solve the subproblem within each

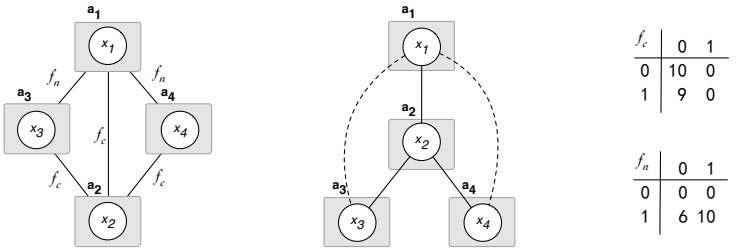
region. Solution quality bounds are provided as a function of k or t. Bounded Max-Sum is an extension of Max-Sum, which solves optimally an acyclic version of the DCOP graph, bounding its solution quality as a function of the edges removed from the cyclic graph. Finally, DaC-based algorithms use Lagrangian decomposition techniques to solve agent subproblems sub-optimally and to provide quality bounded solutions. Researchers have also developed extensions to complete algorithms that trade off solution qualities and runtimes. For example, complete search algorithms have mechanisms that allow users to specify absolute or relative error bounds [28, 22]. Another class of DCOP algorithms that provide quality guarantees are sampling-based algorithms [18], which provide probabilistic bounds as a function of the number of samples. Researchers have also used distributed learning algorithms to solve Markovian Dynamic DCOPs (MD-DCOPs), a hybrid MDP and DCOP model, where the agents use distributed Q-learning to choose actions that maximize their cumulative average reward [19]

Large Neighborhood Search (LNS) [25] is a *centralized* local search algorithm that is very effective in solving large centralized optimization problems. It alternates two phases: a *destroy phase*, which selects a subset of variables to *freeze* and assigns them values from the previous iteration, and a *repair phase*, in which it determines an improved solution by searching over the values of non-frozen variables (i.e., a large *neighborhood*).

Good quality assessment is essential for sub-optimal DCOP algorithms. However, current incomplete DCOP approaches can provide arbitrarily poor quality assessments (as illustrated in our experimental results). In addition, they are unable to exploit the problem structure provided by the presence of hard constraints. In this paper, we address these limitations by introducing the *Distributed LNS (D-LNS)* framework, which builds on the strengths of LNS to solve DCOPs. This work advances the state of the art in DCOP resolution: **(1)** We provide a novel distributed local search framework for DCOPs, based on LNS, which learns online the neighborhood to search at each iteration; **(2)** We introduce two novel distributed search algorithms, D-BLNS and D-RLNS, characterized by low network usage and low computational complexity per agent; and **(3)** Given our D-LNS framework, we show that, compared to existing incomplete DCOP algorithms, D-BLNS converges to better solutions, converges to them faster, provides tighter bounds, and scales better to large problems.

2 Background: DCOP

A *DCOP* is defined by 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 $f_i : \bigtimes_{x_j \in \mathsf{scope}(f_i)} D_j \longrightarrow \mathbb{N} \cup \{-\infty\}$ and $\mathsf{scope}(f_i) \subseteq \mathcal{X}$ is the set of the variables 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. Following common conventions, we restrict our attention to binary utility functions and assume that α is a bijection: each agent controls exactly one variable. We will use the terms "variable" and "agent" interchangeably and assume that $\alpha(x_i) = a_i$. Each f_j specifies the utility of each combination of values to the variables in $\mathsf{scope}(f_j)$.



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

Fig. 1: Example DCOP

A *solution* σ is a value assignment for a set $X_\sigma \subseteq \mathcal{X}$ of variables that is consistent with their respective domains. The utility $\mathcal{F}(\sigma) = \sum_{f \in \mathcal{F}, \text{scope}(f) \subseteq X_\sigma} f(\sigma)$[3] is the sum of the utilities across all the applicable utility functions in σ. A solution σ is *complete* if $X_\sigma = \mathcal{X}$. We will denote by $\mathbf{x}$ a complete solution, while $\mathbf{x}_i$ is the value of x_i in $\mathbf{x}$. The goal is to find an optimal complete solution $\mathbf{x}^* = \text{argmax}_{\mathbf{x}} \mathcal{F}(\mathbf{x})$.

Given a DCOP P, $G_P = (\mathcal{X}, E_\mathcal{F})$ is the *constraint graph* of P, where $\{x, y\} \in E_\mathcal{F}$ iff $\exists f_i \in \mathcal{F}$ s.t. $\{x, y\} \subseteq \text{scope}(f_i)$. A *DFS pseudo-tree* arrangement for G_P is a spanning tree $T = \langle \mathcal{X}, E_T \rangle$ of G_P s.t. if $f_i \in \mathcal{F}$ and $\{x, y\} \subseteq \text{scope}(f_i)$, then x, y appear in the same branch of T. Edges of G_P that are *in* (resp. *out* of) E_T are called *tree edges* (resp. *backedges*). The tree edges connect parent-child nodes, while backedges connect a node with its *pseudo-parents* and its *pseudo-children*. We use $N(a_i) = \{a_j \in \mathcal{A} \mid \{x_i, x_j\} \in E_\mathcal{F}\}$ to denote the neighbors of the agent a_i.

Figure 1(a) shows the constraint graph of a DCOP with four agents $a_1, \ldots, a_4$, each controlling one variable with domain {0,1}. Figure 1(b) shows one possible pseudo-tree (solid lines are tree edges and dotted lines are backedges). Each of the five edges is associated to one utility function, labeled as f_c or f_n. Figure 1(c) shows their utilities.

When pseudo-trees are referred in iterative algorithms execution, T^1 denotes the initial pseudo-tree, and T^k denotes the pseudo-tree at the k^{th} iteration.

3 D-LNS Framework

In this section, we introduce D-LNS, a general distributed LNS framework to solve DCOPs that allows (1) different strategies to choose the neighborhood as well as (2) different strategies to search over that neighborhood. We propose two learning-based approaches to choose neighborhoods and two search strategies to search over the neighborhoods. Our D-LNS solutions take into account factors such as network loads (i.e., number and size of messages exchanged by agents) as well as the restriction that each agent is only aware of its local subproblem (i.e., its neighbors and the constraints whose scope includes its variables).

Algorithm 1 describes the general structure of the D-LNS framework, which is executed by each agent $a_i \in \mathcal{A}$. Agent a_i first initializes its counter k (line 1) and calls

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

Algorithm 1: D-LNS

1 $k \leftarrow 1$;
2 $\langle \mathbf{x}_i^1, LB_i^1, UB_i^1, Q_i^1, Q_i^2 \rangle \leftarrow$ VALUE-INITIALIZATION();
3 **while** *termination condition is not met* **do**
4 $\quad k \leftarrow k + 1$;
5 $\quad Q_i^k \leftarrow$UPDATE-RL(Q_i^{k-1});
6 $\quad z_i^k \leftarrow$ DESTROY-ALGORITHM(Q_i^k);
7 $\quad$ **if** $z_i^k = \text{fr}$ **then** $\mathbf{x}_i^k \leftarrow \mathbf{x}_i^{k-1}$;
8 $\quad$ **else** $\mathbf{x}_i^k \leftarrow$ *NULL*;
9 $\quad \langle \mathbf{x}_i^k, LB_i^k, UB_i^k \rangle \leftarrow$ REPAIR-ALGORITHM();
10 $\quad$ **if** $\neg$ Accept ($\mathbf{x}_i^k, \mathbf{x}_i^{k-1}$) **then** $\mathbf{x}_i^k \leftarrow \mathbf{x}_i^{k-1}$;

the VALUE-INITIALIZATION function to initialize: its current local value assignment $\mathbf{x}_i^1$, its current lower and upper bounds LB_i^1 and UB_i^1 of the optimal utility, and its Q-values Q_i^1 and Q_i^2, which are used to learn good neighborhoods (see Sections 3.1 and 3.2). In the rest of the algorithm (lines 3-10), we iterate through the destroy and repair phases as in LNS, until a termination condition occurs (line 3), such as reaching a maximum value of k, a timeout limit, or a threshold on the confidence error of the reported best solution.

- DESTROY PHASE: This phase aims at *destroying* part of the current solution, by *freezing* some variables (i.e., maintain their values from the previous iteration), and enabling the optimization over the non-frozen variables (large neighborhood). Locally, the agent a_i calls the function DESTROY-ALGORITHM to determine if its local variable x_i should be frozen or not, as indicated by the flag $z_i^k \in \{\text{fr}, \neg\text{fr}\}$ (line 6). Such decision is made through Q-values that are updated according to the reinforcement learning function UPDATE-RL (line 5). D-LNS allows the agent to use any learning algorithm—we discuss two distributed Q-learning processes in Sect. 3.1. Once the frozen variables are selected, the agent assigns them the values from the previous iteration and resets the value of the other variables (lines 7-8).
- REPAIR PHASE: In this phase, the agent seeks a new value assignment for the non-frozen variables using the REPAIR-ALGORITHM function (line 9). The generality of D-LNS allows the agent to use any algorithm to solve this problem; we introduce two distributed algorithms to do so in Sect. 3.2. Once the agent finds and evaluates a new solution, it either accepts it or rejects it (line 10).

While most of the current local search DCOP algorithms fail to guarantee the consistency of the solution returned with respect to the hard constraints of the problem [20, 12, 18], D-LNS naturally accommodates the consistency check during the repair phase. In our two distributed algorithms, the agent accepts the solution if it does not violate any hard constraints, that is, its utility is not $-\infty$.

3.1 Destroy Phase

The destroy phase determines a set of variables to include in the large neighborhood, such that the repair algorithm can find good solutions quickly. This process can be challenging without taking into account the structural properties of the problem at hand. We

propose to use a model-free distributed reinforcement learning-based approach, called *Distributed Q-Learning*. Q-Learning does not make any assumption on the model, its convergence to an optimal policy is guaranteed for an arbitrary exploration strategy and, crucially, the optimal policy can be learned in distributed, deterministic environments. Let us provide some background on Distributed Q-Learning [16].

Distributed Q-Learning is used to find an optimal action-choice policy in multi-agent *Markov Decision Processes* (MDPs) by analyzing the history of the action-reward pairs obtained by the agent. It works by learning an action-value function $Q : \mathbf{S} \times \mathbf{A} \rightarrow \mathbb{R}$, where $\mathbf{S}$ is a finite set of states and $\mathbf{A}$ is a finite set of executable actions (e.g., possible choices of large neighborhoods). In such a problem, an optimal policy $\pi^* : \mathbf{S} \rightarrow \mathbf{A}$ can be decomposed into n component-policies $\pi_i^* : \mathbf{S} \rightarrow \mathbf{A}_i$ with $\mathbf{A}_i$ being the set of actions executable by agent a_i and $\pi^*(s) = (\pi_1^*(s), \ldots, \pi_n^*(s))$. The distributed Q-learning for multi-agent deterministic MDPs is described by the following iterative rule,

$$Q_i^{k+1}(s, z_i) = \max \begin{Bmatrix} Q_i^k(s, z_i), \\ R_i(s, z_i) + \lambda \max\limits_{z_i' \in \mathbf{A}_i} Q_i^k(s', z_i') \end{Bmatrix} \tag{1}$$

where $Q_i^1(s, z_i) = 0$, $R_i : \mathbf{S} \times \mathbf{A}_i \rightarrow \mathbb{R}$ is a *reward function* of agent a_i that determines the reward of performing action z_i in state s, and s' is the resulting state after executing action z_i' in state s—described by the transition function $\mathbf{T}_i : \mathbf{S} \times \mathbf{A}_i \rightarrow \mathbf{S}$. The idea is that the results of the selected transitions are accumulated into Q_i using successive maximizations.

One can map the problem of selecting the large neighborhood to a deterministic multi-agent MDP, where: $\mathbf{S}$ is the set of all possible complete solutions; for a state s, action z_i of agent a_i, and iteration k, $\mathbf{A}_i$ is the set $\{\text{fr}, \neg\text{fr}\}$; the reward computed at iteration k is $R_i(s, z_i) = \mathcal{F}(\mathbf{x}^k)$, with $\mathbf{x}^k$ the complete solution in such iteration; and $\mathbf{T}_i(s, z_i)$ is the next solution found by the repair algorithm. The goal is to find a mapping $\pi_i^* : \mathbf{S} \rightarrow \mathbf{A}_i$ that determines the action for each agent at each state, which corresponds to the decision of the agent to freeze its variable or not based on the complete solution in the previous iteration.

To update their Q-values based on Equation (1), each agent needs to know the complete solutions of states s and s'. Thus, we allow the agents to broadcast their local assignments to the other agents so that all agents can construct the complete solution. While allowing agents to broadcast is not uncommon in DCOP algorithms [8], it may be a shortcoming for applications where privacy is a concern.

We also introduce a second learning approach, called *Distributed Local Q-Learning*, that preserves privacy by considering only local states. In this approach, $\mathbf{S} = \bigtimes_{a_i \in \mathcal{A}} \mathbf{S}_i$, where the set of local states $\mathbf{S}_i = \bigtimes_{a_j \in N(a_i)} D_j \times D_i$ is the set of local solutions. For a local state s_i, action z_i of agent a_i, and iteration k, $R_i(s_i, z_i) = \sum_{\{f \in \mathcal{F} \mid x_i \in \mathsf{scope}(f)\}} f(\mathbf{x}^k)$ is the utility of the local solution in iteration k; and $\mathbf{T}_i(s_i, z_i)$ is the next local solution found by the repair algorithm. Although convergence guarantees are not available, our experimental results in Sect. 5 show that this approach works as well as the (broadcasting) Distributed Q-Learning approach.

3.2 Repair Phase

The goal of the repair phase is to find a new improved solution by searching over a large neighborhood. We developed two distributed algorithms, *Distributed Bounded LNS* and *Distributed Random LNS*, where the former provides an online error bound and the latter does not.

Distributed Bounded LNS The *Distributed Bounded LNS* (D-BLNS) algorithm attempts to iteratively refine the lower and upper bounds of the DCOP solution. In each iteration k, it executes the following phases.

- RELAXATION PHASE: Given a DCOP P, this phase computes two relaxations[4] of P, $\check{P}^k$ and $\hat{P}^k$, which are used to compute, respectively, a lower and an upper bound on the optimal utility for P at each iteration k of the algorithm. Let LN^k be the set of large neighborhood variables (i.e., non-frozen variables) in iteration k. Both problem relaxations are solved using the same underlying pseudo-tree structure $T^k = \langle LN^k, E_{T^k} \rangle$, computed from the subgraph of G_P restricted to the nodes in LN^k, i.e., $\langle LN^k, E_k \rangle$ where $E_k = \{\{x, y\} \mid \{x, y\} \in E_{\mathcal{F}}; x, y \in LN^k\}$. Note that this process may create a forest, which does not affect the correctness of the algorithm.[5] In particular, the heuristic used for building T^k prefers the inclusion of edges of G_P that have not been used in $T^1, \ldots, T^{k-1}$. In the relaxed DCOP $\check{P}^k$, we wish to find a solution $\check{\mathbf{x}}^k$ using[6]

$$\begin{aligned} \check{\mathbf{x}}^k &= \operatorname*{argmax}_{\mathbf{x}} \check{F}^k(\mathbf{x}) \\ &= \operatorname*{argmax}_{\mathbf{x}} \sum_{f \in E_{T^k}} f(\mathbf{x}_i, \mathbf{x}_j) + \sum_{x_i \in LN^k, x_j \notin LN^k} f(\mathbf{x}_i, \check{\mathbf{x}}_j^{k-1}) \end{aligned} \tag{2}$$

  where $\check{\mathbf{x}}_j^{k-1}$ is the value assigned to variable x_j for problem $\check{P}^{k-1}$ in the previous iteration. The first summation is over all functions listed as tree edges in T^k, while the second is over all functions between a non-frozen variable and a frozen variable. Thus, solving problem $\check{P}^k$ means optimizing over all the non-frozen variables under the assumption that the frozen variables take on their previous value and ignoring backedges. This solution is used to compute lower bounds, during the *bounding phase*. In the problem $\hat{P}^k$, we wish to find a solution $\hat{\mathbf{x}}^k$ using

$$\hat{\mathbf{x}}^k = \operatorname*{argmax}_{\mathbf{x}} \hat{F}^k(\mathbf{x}) = \operatorname*{argmax}_{\mathbf{x}} \sum_{f \in \mathcal{F}} \hat{f}^k(\mathbf{x}_i, \mathbf{x}_j) \tag{3}$$

  where $\hat{f}^k(\mathbf{x}_i, \mathbf{x}_j)$ is defined as:

$$\hat{f}^k(\mathbf{x}_i, \mathbf{x}_j) = \begin{cases} \max\limits_{d_i \in D_i, d_j \in D_j} f(d_i, d_j) & \text{if } \Gamma_f^k = \emptyset \\ \max\left\{ f(\mathbf{x}_i, \mathbf{x}_j), \max\limits_{k' \in \Gamma_f^{k-1}} f(\hat{\mathbf{x}}_i^{k'}, \hat{\mathbf{x}}_j^{k'}) \right\} & \text{otherwise} \end{cases}$$

[4] I.e., defined on a relaxed version of the constraint graph G_P.

[5] One can execute the algorithm in each tree of the forest.

[6] We identify each edge $\{x, y\}$ of the tree with the constraint $f(x, y)$ that generated it.

where Γ_f^k is the set of past iteration indices for which the function f was a tree edge in the pseudo-tree. Specifically,

$$\Gamma_f^k = \{k' \mid f \in E_{T^{k'}} \wedge 1 \leq k' \leq k\} \tag{4}$$

Therefore, the utility of $\hat{F}^k(\hat{\mathbf{x}}^k)$ is composed of two parts. The first part involves all functions that have never been part of a pseudo-tree up to the current iteration, and the second part involves all the remaining functions. The utility of each function in the first part is the maximal utility over all possible pairs of value combinations of variables in the scope of that function, and the utility of each function in the second part is the largest utility over all pairs of value combinations in previous solutions. In summary, solving $\hat{P}^k$ means finding the solution $\hat{\mathbf{x}}^k$ that maximizes $\hat{F}^k(\hat{\mathbf{x}}^k)$. This solution is used to compute upper bounds (during the *bounding phase*).

- SOLVING PHASE: Next, D-BLNS solves the relaxed DCOPs $\hat{P}^k$ and $\check{P}^k$ using Equations (2) and (3). At a high-level, D-BLNS is an inference-based algorithm, similar to DPOP [21], where each agent, starting from the leaves of the pseudo-tree, projects out its own variable and sends its projected utilities to its parent. These utilities are propagated up the pseudo-tree until they reach the root. The hard constraints of the problem are naturally handled in this phase, where the agent prunes all inconsistent values before sending a message to its parent. Once the root receives utilities from each of its children, it selects the value that maximizes its utility and sends it down to its children, which repeat the same process. These values are propagated down the pseudo-tree until they reach the leaves, at which point the problem is solved. D-BLNS solves the two relaxed DCOPS in parallel. In the utility propagation phase, each agent computes two sets of utilities, one for each relaxed problem, and sends them to its parent. In the value propagation phase, each agent selects two values, one for each relaxed problem, and sends them to its children.
- BOUNDING PHASE: Once the relaxed problems are solved, D-BLNS computes the lower and upper bounds based on the solutions $\check{\mathbf{x}}^k$ and $\hat{\mathbf{x}}^k$. As we show in Sect. 4, $\mathcal{F}(\check{\mathbf{x}}^k) \leq \mathcal{F}(\mathbf{x}^*) \leq \hat{F}^k(\hat{\mathbf{x}}_k)$ (Theorems 1 and 2). Therefore $\rho = \dfrac{\min_k \hat{F}^k(\hat{\mathbf{x}}^k)}{\max_k \mathcal{F}(\check{\mathbf{x}}^k)}$ is a guaranteed approximation ratio for P.

Example Trace: Figure 2 presents a running example illustrating the behavior of the D-BLNS during the first three LNS iterations. The example uses the DCOP of Figure 1. The pseudo-trees T^1, T^2, and T^3 are represented by bold lines (solid for tree edges and dotted for backedges). The frozen variables in each LN^k are shaded gray, and the constraints they participate in are represented by thin dotted lines. Nodes representing non-frozen variables are labeled with red values representing $\check{\mathbf{x}}_i^k$; nodes representing frozen variables are labeled with black values representing $\check{\mathbf{x}}_i^{k-1}$. Each tree edge is labeled with a black bold value representing the utility $f(\check{\mathbf{x}}_i^k, \check{\mathbf{x}}_j^k)$ of the function corresponding to that edge. Each backedge is labeled with a pair of values representing the utilities $\hat{f}^k(\check{\mathbf{x}}_i^k, \check{\mathbf{x}}_j^k)$ (top, in parenthesis) and $f(\check{\mathbf{x}}_i^k, \check{\mathbf{x}}_j^k)$ (bottom) of the corresponding functions. The lower and upper bounds of each iteration are shown below the pseudo-tree.

In the first iteration ($k=1$), the algorithm solves $\check{P}^1$ finding solution $\check{\mathbf{x}}^1$ with utility $\check{F}^1(\check{\mathbf{x}}^1) = f(\check{\mathbf{x}}_1^1, \check{\mathbf{x}}_2^1) + f(\check{\mathbf{x}}_2^1, \check{\mathbf{x}}_3^1) + f(\check{\mathbf{x}}_2^1, \check{\mathbf{x}}_4^1) = 10 + 10 + 10 = 30$. It evaluates this

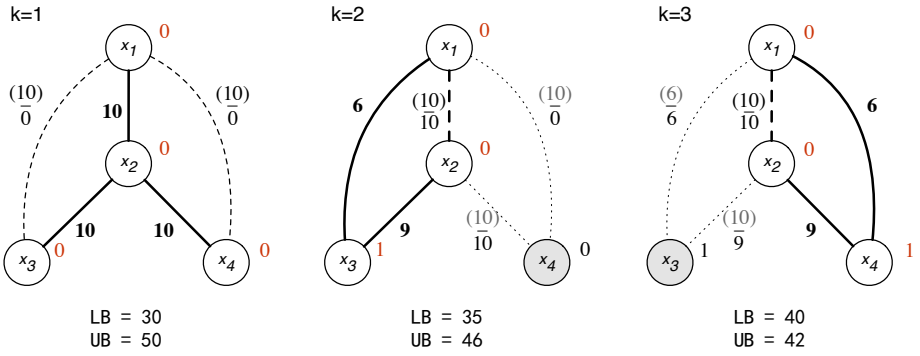


Fig. 2: Distributed Bounded LNS Example Trace

solution with the true utility function $\mathcal{F}(\check{\mathbf{x}}^1) = \check{F}^1(\check{\mathbf{x}}^1) + f(\check{\mathbf{x}}^1_1, \check{\mathbf{x}}^1_3) + f(\check{\mathbf{x}}^1_1, \check{\mathbf{x}}^1_4) = 30 + 0 + 0 = 30$ to get the lower bound. The algorithm also solves $\hat{P}^1$ and finds a solution $\hat{\mathbf{x}}^1$ with utility $\hat{F}^1(\hat{\mathbf{x}}^1) = \hat{f}^1(\hat{\mathbf{x}}^1_1, \hat{\mathbf{x}}^1_2) + \hat{f}^1(\hat{\mathbf{x}}^1_2, \hat{\mathbf{x}}^1_3) + \hat{f}^1(\hat{\mathbf{x}}^1_2, \hat{\mathbf{x}}^1_4) + \hat{f}^1(\hat{\mathbf{x}}^1_1, \hat{\mathbf{x}}^1_3) + \hat{f}^1(\hat{\mathbf{x}}^1_1, \hat{\mathbf{x}}^1_4) = 10 + 10 + 10 + 10 + 10 = 50$, which is the upper bound.

In the second iteration ($k = 2$), the algorithm freezes x_4 and assigns it its value in the previous iteration $\check{\mathbf{x}}^2_4 = \check{\mathbf{x}}^1_4 = 0$. Then, it builds the pseudo-tree with the remaining variables and chooses $f(x_1, x_3)$ as a tree edge, as it has never been chosen as a tree edge before. Solving $\check{P}^2$ yields solution $\check{\mathbf{x}}^2$ with utility $\check{F}^2(\check{\mathbf{x}}^2) = f(\check{\mathbf{x}}^2_1, \check{\mathbf{x}}^2_3) + f(\check{\mathbf{x}}^2_2, \check{\mathbf{x}}^2_3) + f(\check{\mathbf{x}}^2_1, \check{\mathbf{x}}^2_4) + f(\check{\mathbf{x}}^2_2, \check{\mathbf{x}}^2_4) = 6 + 9 + 0 + 10 = 25$, which results in a lower bound $\mathcal{F}(\check{\mathbf{x}}^2) = \check{F}^2(\check{\mathbf{x}}^2) + f(\check{\mathbf{x}}^2_1, \check{\mathbf{x}}^2_2) = 25 + 10 = 35$. Solving $\hat{P}^2$ yields solution $\hat{\mathbf{x}}^2$ with utility $\hat{F}^2(\hat{\mathbf{x}}^2) = \hat{f}^2(\hat{\mathbf{x}}^2_1, \hat{\mathbf{x}}^2_2) + \hat{f}^2(\hat{\mathbf{x}}^2_2, \hat{\mathbf{x}}^2_3) + \hat{f}^2(\hat{\mathbf{x}}^2_2, \hat{\mathbf{x}}^2_4) + \hat{f}^2(\hat{\mathbf{x}}^2_1, \hat{\mathbf{x}}^2_3) + \hat{f}^2(\hat{\mathbf{x}}^2_1, \hat{\mathbf{x}}^2_4) = 10 + 10 + 10 + 6 + 10 = 46$, which is the current upper bound.

Finally, in the third iteration ($k = 3$), the algorithm freezes x_3 and assigns it its value in the previous iteration $\check{\mathbf{x}}^3_3 = \check{\mathbf{x}}^2_3 = 1$ and builds the new pseudo-tree with the remaining variables. Solving $\check{P}^3$ and $\hat{P}^3$ yields solutions $\check{\mathbf{x}}^3$ and $\hat{\mathbf{x}}^3$, respectively, with utilities $\check{F}^3(\check{\mathbf{x}}^3) = 6 + 9 + 9 + 6 = 30$, which results in a lower bound $\mathcal{F}(\check{\mathbf{x}}^3) = 30 + 10 = 40$, and an upper bound $\hat{F}^2(\hat{\mathbf{x}}^2) = 10 + 10 + 10 + 6 + 6 = 42$.

Distributed Random LNS In complex application domains with a large number of variables and/or large domain sizes, D-BLNS may be computationally inefficient. To cope with such problem, we propose *Distributed Random LNS* (D-RLNS), an alternative approach relying on the use of random value selections to reduce computation and communication costs. This algorithm is similar to D-BLNS except for the solving and bounding phases.

During the solving phase, D-RLNS starts by having each leaf agent randomly sample S values from its domain, and send them to its parent. Each agent, upon receiving the values from all of its children, performs the same operations, as well as aggregating the sum of the utilities in its subtree considering only tree edges and edges with frozen variables. Thus, these utilities are propagated up to the pseudo-tree until they reach the root. Once the root agent has received utilities from each of its children, it chooses the value (from among the S samples) that maximizes its utility, and sends this value down

to its children, which, in turn repeat the same process. Finally, at each iteration, the D-RLNS bounding phase computes exclusively the lower bound of the current solution.

4 Theoretical Properties

Theorem 1. *For each large neighborhood LN^k,*

$$\mathcal{F}(\check{\mathbf{x}}^k) \leq \mathcal{F}(\mathbf{x}^*).$$

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

Lemma 1. *For each iteration k,*

$$\sum_{f \in T^k} f(\hat{\mathbf{x}}_i^k, \hat{\mathbf{x}}_j^k) \geq \sum_{f \in T^k} f(\mathbf{x}_i^*, \mathbf{x}_j^*)$$

where $\hat{\mathbf{x}}_i^k$ is the value assignment to variable x_i when solving the relaxed DCOP $\hat{P}$ and $\mathbf{x}_i^*$ is the value assignment to variable x_i when solving the original DCOP P.

PROOF (SKETCH). This lemma follows from the fact that, in each iteration k, the functions associated with the tree edges in T^k are solved optimally. □

Lemma 2. *For each iteration k,*

$$\sum_{f \in \Theta^k} \hat{f}^k(\hat{\mathbf{x}}_i^k, \hat{\mathbf{x}}_j^k) \geq \sum_{f \in \Theta^k} f(\mathbf{x}_i^*, \mathbf{x}_j^*).$$

where $\Theta^k = \{f \in \mathcal{F} \mid \Gamma_f^k \neq \emptyset\}$ is the set of functions that have been chosen as tree edges of a pseudo-tree in a previous iteration.

PROOF (SKETCH). We prove by induction on the iteration k. For $k = 0$, then $\Theta^0 = \emptyset$ and, thus, the statement vacuously holds. Assume the claim holds up to iteration $k-1$. For iteration k it follows that,

$$\begin{aligned}
&\sum_{f \in \Theta^k} \hat{f}^k(\hat{\mathbf{x}}_i^k, \hat{\mathbf{x}}_j^k) \\
&\quad = \sum_{f \in \Theta^k \setminus \Theta^{k-1}} \hat{f}^k(\hat{\mathbf{x}}_i^k, \hat{\mathbf{x}}_j^k) + \sum_{f \in \Theta^{k-1}} \hat{f}^k(\hat{\mathbf{x}}_i^k, \hat{\mathbf{x}}_j^k) \\
&\quad = \sum_{f \in \Theta^k \setminus \Theta^{k-1}} \hat{f}^k(\hat{\mathbf{x}}_i^k, \hat{\mathbf{x}}_j^k) + \sum_{f \in \Theta^{k-1} \cap E_{T^k}} \hat{f}^k(\hat{\mathbf{x}}_i^k, \hat{\mathbf{x}}_j^k) + \sum_{f \in \Theta^{k-1} \setminus E_{T^k}} \hat{f}^k(\hat{\mathbf{x}}_i^k, \hat{\mathbf{x}}_j^k) \\
&\quad = \sum_{f \in \Theta^k \setminus \Theta^{k-1}} \hat{f}^k(\hat{\mathbf{x}}_i^k, \hat{\mathbf{x}}_j^k) + \sum_{f \in \Theta^{k-1} \cap E_{T^k}} \max\left\{f(\hat{\mathbf{x}}_i^k, \hat{\mathbf{x}}_j^k), \hat{f}^{k-1}(\hat{\mathbf{x}}_i^{k-1}, \hat{\mathbf{x}}_j^{k-1})\right\} \\
&\qquad + \sum_{f \in \Theta^{k-1} \setminus E_{T^{k+1}}} \hat{f}^k(\hat{\mathbf{x}}_i^k, \hat{\mathbf{x}}_j^k) \qquad \text{(by definition of } \hat{f}^k\text{)} \\
&\quad \geq \sum_{f \in E_{T^k}} \hat{f}^k(\hat{\mathbf{x}}_i^k, \hat{\mathbf{x}}_j^k) + \sum_{f \in \Theta^{k-1}} \hat{f}^{k-1}(\hat{\mathbf{x}}_i^{k-1}, \hat{\mathbf{x}}_j^{k-1})
\end{aligned}$$

$$
\begin{aligned}
&\quad - \sum_{f \in E_{T^k} \cap \Theta^{k-1}} \max\left\{ f(\hat{\mathbf{x}}_i^k, \hat{\mathbf{x}}_j^k), \hat{f}^{k-1}(\hat{\mathbf{x}}_i^{k-1}, \hat{\mathbf{x}}_j^{k-1}), f(\mathbf{x}_i^*, \mathbf{x}_j^*) \right\} \\
&\geq \sum_{f \in E_{T^k}} f(\mathbf{x}_i^*, \mathbf{x}_j^*) + \sum_{f \in \Theta^{k-1}} f(\mathbf{x}_i^*, \mathbf{x}_j^*) \\
&\quad - \sum_{f \in E_{T^k} \cap \Theta^{k-1}} \max\left\{ f(\hat{\mathbf{x}}_i^k, \hat{\mathbf{x}}_j^k), \hat{f}^{k-1}(\hat{\mathbf{x}}_i^{k-1}, \hat{\mathbf{x}}_j^{k-1}), f(\mathbf{x}_i^*, \mathbf{x}_j^*) \right\} \\
&\qquad\qquad \text{(by Lemma 1 and induction assumption)} \\
&\geq \sum_{f \in \Theta^k \setminus \Theta^{k-1}} f(\mathbf{x}_i^*, \mathbf{x}_j^*) + \sum_{f \in \Theta^{k-1}} f(\mathbf{x}_i^*, \mathbf{x}_j^*) \qquad \text{(since } \Theta^k \setminus \Theta^{k-1} = T^k \setminus (T^k \cap \Theta^{k-1})\text{)} \\
&= \sum_{f \in \Theta^k} f(\mathbf{x}_i^*, \mathbf{x}_j^*)
\end{aligned}
$$

which concludes the proof. □

Theorem 2. *For each large neighborhood LN^k,*

$$\hat{F}^k(\hat{\mathbf{x}}_k) \geq \mathcal{F}(\mathbf{x}^*).$$

PROOF (SKETCH).

$$
\begin{aligned}
\hat{F}^k(\mathbf{x}) &= \sum_{f \in \mathcal{F}} \hat{f}^k(\hat{\mathbf{x}}_i^k, \hat{\mathbf{x}}_j^k) \\
&= \sum_{f \in \Theta^k} \hat{f}^k(\hat{\mathbf{x}}_i^k, \hat{\mathbf{x}}_j^k) + \sum_{f \notin \Theta^k} \hat{f}^k(\hat{\mathbf{x}}_i^k, \hat{\mathbf{x}}_j^k) \\
&= \sum_{f \in \Theta^k} \hat{f}^k(\hat{\mathbf{x}}_i^k, \hat{\mathbf{x}}_j^k) + \sum_{f \notin \Theta^k} \max_{d_i, d_j} f(d_i, d_j) && \text{(by definition of } \hat{f}^k\text{)} \\
&\geq \sum_{f \in \Theta^k} f(x_i^*, x_j^*) + \sum_{f \notin \Theta^k} f(x_i^*, x_j^*) && \text{(by Lemma 2)} \\
&= \mathcal{F}(\mathbf{x}^*)
\end{aligned}
$$

which concludes the proof. □

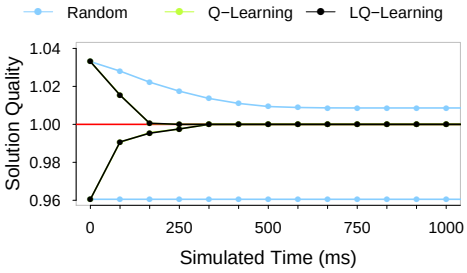
Corollary 1. *An approximation ratio for the problem is*

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

PROOF (SKETCH). This result follows from $\max_k \mathcal{F}(\check{\mathbf{x}}^k) \leq \mathcal{F}(\mathbf{x}^*)$ (Theorem 1) and $\min_k \hat{F}^k(\hat{\mathbf{x}}^k) \geq \mathcal{F}(\mathbf{x}^*)$ (Theorem 2). □

Property 1. In each iteration, D-BLNS requires $O(|\mathcal{A}|)$ number of messages of size $O(d)$, where $d = \max_{a_i \in \mathcal{A}} |D_i|$.

Property 2. In each iteration, the computation complexity of each D-BLNS agent is $O(d^2)$, where $d = \max_{a_i \in \mathcal{A}} |D_i|$.



	$\|D_i\|=10$		$\|D_i\|=25$		$\|D_i\|=50$		$\|D_i\|=100$	
	time	ρ	time	ρ	time	ρ	time	ρ
D-RLNS	**60**	2.07	**75**	2.17	**120**	2.31	**261**	2.48
D-BLNS	364	**2.00**	1507	**2.07**	5389	**2.17**	19792	**2.23**

Fig. 3: D-LNS Destroy (left) and Repair (right) Methods

5 Experimental Results

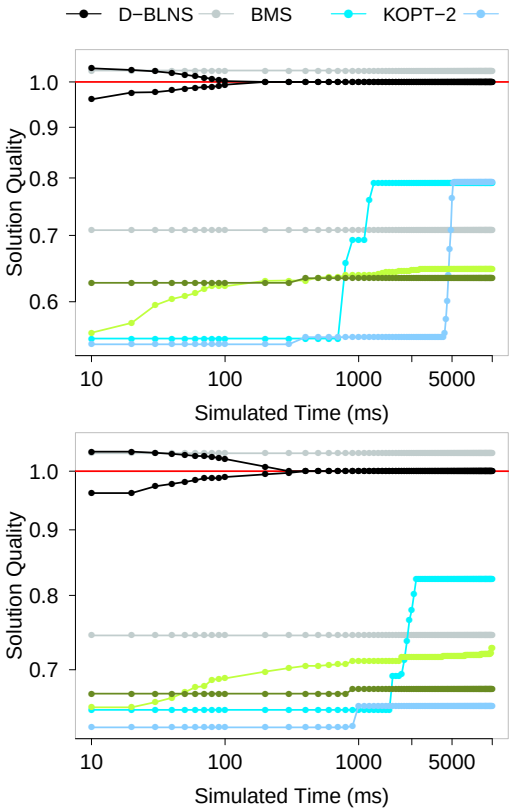
We evaluate our D-LNS framework against state-of-the-art incomplete DCOP algorithms with and without quality guarantees: Bounded Max-Sum (BMS) [23], k-optimal algorithms (KOPT) [20], t-optimal algorithms (TOPT) [13], Distributed Breakout Algorithm (DBA) [12], and Distributed Gibbs (D-Gibbs) [18]. We use publicly-available implementations of BMS, KOPT, and TOPT and use our own implementations of DBA and D-Gibbs. All experiments are performed on an Intel i7 Quadcore 3.3GHz machine with 8GB of RAM with a 300-second timeout. If an algorithm fails to solve a problem, it is due to either memory limitations or timeout. We conduct our experiments on random graphs [5], where we systematically vary the constraint density p_1[7] and distributed Radio Link Frequency Assignment (RLFA) problems [3, 7], where we vary the number of agents $|\mathcal{A}|$ in the problem. We generated 20 instances for each experimental setting, and we report the average runtime, measured using the simulated runtime metric [24]. All experiments are statistically significant with p-values $< 1.6 \times 10^{-6}$.

Random Graphs: We set $|\mathcal{A}| = 50$, $|D_i| = 20$ for all variables, and $p_1 = 0.5$. We did not impose restrictions on the tree-width, which is determined based on the underlying randomly generated graph.

We first evaluate the two learning algorithms, Distributed Q-Learning (labeled *Q-Learning*) and Distributed Local Q-Learning (labeled *LQ-Learning*), against a random neighborhood selection (labeled *Random*), on learning good neighborhoods. In our implementation, the action value function Q is constructed incrementally in order to keep memory usage as small as possible. Figure 3 (left) shows the quality of solutions found with the three algorithms. They all run D-BLNS in the repair phase. For each algorithm, we report the lower (upper) bound in the bottom (top) part of the plot found at each time interval. These measures are normalized with respect to the best lower (upper) bound found by the three algorithms, which is illustrated through a red bold line. The closer an algorithm to the red line, the better. Both learning algorithms find solutions with almost identical qualities and they both outperform random selections, as expected.

Next, we evaluate the two repair algorithms, D-BLNS and D-RLNS. They run LQ-Learning in the destroy phase. Figure 3 (right) reports the runtimes at varying domain

[7] 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.



Approximation ratio ρ

p_1	D-BLNS	BMS	KOPT3	KOPT2
0.1	**1.10**	1.47	13.5	17.6
0.2	**1.16**	1.63	13.5	17.6
0.3	**1.16**	1.90	13.5	17.6
0.4	**1.17**	1.96	13.5	17.6
0.5	**2.00**	3.14	13.5	17.6
0.6	**2.07**	3.37	13.5	17.6
0.7	**2.34**	3.53	13.5	17.6
0.8	**2.39**	3.50	13.5	17.6
0.9	**2.45**	3.57	13.5	17.6
1.0	**2.50**	3.60	13.5	17.6

Fig. 4: Experimental Results on Random Graphs

sizes $|D_i|$. For D-RLNS, we set $S = 5$. D-RLNS is always faster than D-BLNS but it finds worse solutions.

We now evaluate D-BLNS with LQ-Learning against state-of-the-art incomplete algorithms. We ran KOPT for $k = 2$ (KOPT2) and $k = 3$ (KOPT3), and ran TOPT for $t = 1$ (TOPT1). Figure 4 shows the convergence results (normalized lower and upper bounds) at $p_1 = 0.4$ (top-left) and $p_1 = 0.8$ (bottom-left) as well as the approximation ratio ρ of each algorithm with quality guarantees (right). We omitted TOPT1 as it solved problems with $p_1 = 0.1$ only (reporting $\rho = 26.0$). The best approximation ratio is shown in bold. As the upper bounds of the k-optimal algorithms were very loose, we omitted them from the figures so that the lower bounds are easier to distinguish.

These results clearly show that D-BLNS not only converges to better solutions and converges to them faster, but also provides tighter upper bounds. Consequently, it finds better approximation ratios compared to the other algorithms.

Distributed RLFA Problem: Each link, consisting of a transmitter and a receiver, must be assigned a frequency from a given set such that the total interference at the receivers falls within an acceptable level. We model each transmitter as a variable (and an agent), whose domain consists of the frequencies that can be assigned to the corresponding

$\|\mathcal{A}\|$	D-BLNS			BMS			KOPT2			KOPT3			D-RLNS		D-Gibbs		DBA	
	sim. time	ρ	ϵ	sim. time	ρ	ϵ	sim. time	ρ	ϵ	sim. time	ρ	ϵ	sim. time	ϵ	sim. time	ϵ	sim. time	ϵ
10	11	**1.03**	**1.01**	211	1.05	1.29	44	4.33	1.06	57	3.50	1.03	10	1.03	13	1.17	**7**	1.51
25	14	**1.42**	**1.00**	595	1.83	1.28	231	9.33	1.25	438	7.25	1.18	**11**	1.07	87	1.43	29	3.85
50	29	**1.59**	**1.00**	861	2.20	1.34	709	17.6	1.44	1264	13.5	1.51	**19**	1.13	375	1.59	108	3.48
100	112	**1.66**	**1.00**	2074	2.21	1.28	3208	34.3	1.72	9113	26.0	1.92	**57**	1.17	1630	1.64	582	2.12
250	1405	**1.63**	**1.00**	18372	2.34	1.43	86675	42.2	1.68	–			**312**	1.21	23091	2.97	6533	3.22
500	12938	**1.65**	**1.00**	242236	2.75	1.63	–			–			**1622**	1.28	135366	1.67	54188	2.22
1000	133047	**1.59**	**1.00**	–			–			–			**24728**	1.35	278300	1.60	–	

Table 1: Experimental Results on Distributed RLFA Problems

transmitter. The interference between transmitters are modeled as constraints of the form $|x_i - x_j| > s$, where $x_i, x_j \in \mathcal{X}$ and $s \geq 0$ is a randomly generated required frequency separation.

We model the constraint graphs as scale-free graphs [1], where the fraction $P(k)$ of nodes in the network having k connections is modeled by $P(k) \sim k^{-\gamma}$. We set $\gamma = 2.5$, vary $|\mathcal{A}|$ from 10 to 1000, set $|D_i| = 10$, and randomly choose $s \in \left[1, {}^{|\mathcal{A}|}/_{2}\right]$.

Table 1 shows the results for the different algorithms, using the same algorithm settings as in the previous experiments. We report the time needed to find the best solution, its approximation ratio ρ, and the ratio of the best quality found versus its quality ϵ. Best runtimes, approximation ratios, and quality ratios are shown in bold. TOPT1 failed to solve any of the problems, while KOPT2 and KOPT3 run out of memory for instances with more than 500 and 250 agents, respectively. The results show D-BLNS can scale better to large problems than algorithms with quality guarantees (i.e., BMS, KOPT2, and KOPT3). Additionally, similar to the trends in random graphs, it also finds better solutions (i.e., better approximation ratios ρ and better quality ratios ϵ) than those of the competing algorithms. The main reason why the D-LNS algorithms are faster than BMS, is that, on average, about half of the agents are frozen at each iteration, and the search space is thus significantly smaller. Lastly, like in random graphs, D-RLNS converges to its solution fastest but its solutions are not as good as those found by D-BLNS.

6 Conclusions

In this paper, we proposed a Distributed Large Neighborhood Search (D-LNS) framework, that can be used to find quality-bounded solutions in DCOPs. Like centralized LNS, D-LNS is composed of a destroy phase, which selects the large neighborhood to search, and a repair phase, which performs the search over the selected neighborhood. We introduced Distributed Q-learning and Distributed Local Q-learning to select the neighborhood in the destroy phase. Distributed Q-learning is guaranteed to converge to the optimal neighborhood, but both learning algorithms perform equally well empirically. We also introduced D-BLNS and D-RLNS to perform the search in the repair phase, characterized by low network load demand and low computational complexity per agent. D-RLNS finds solutions faster, but D-BLNS finds better solutions. Empirical results also show that, compared to existing incomplete DCOP algorithms, D-BLNS with Distributed Local Q-learning converges to better solutions, converges to them faster, provides tighter upper bounds, and scales better to large problems, thereby clearly highlighting the strengths of this approach.

In the future, we plan to investigate other schemes that can be incorporated into the repair phase of D-LNS, such us propagation techniques to better prune the search space, some of which have been applied to DCOP algorithms [2, 10, 9, 7], as well as the use of graphics processing units (GPUs) to parallelize the search for better speedups [4].

Acknowledgment

We would like to thank the anonymous reviewers for their useful comments. This research is partially supported by INdAM-GNCS 2015 and NSF grant HRD-1345232. 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. A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.
2. C. Bessiere, P. Gutierrez, and P. Meseguer. Including Soft Global Constraints in DCOPs. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP)*, pages 175–190, 2012.
3. B. Cabon, S. De Givry, L. Lobjois, T. Schiex, and J. P. Warners. Radio Link Frequency Assignment. *Constraints*, 4(1):79–89, 1999.
4. F. Campeotto, A. Dovier, F. Fioretto, and E. Pontelli. A GPU implementation of large neighborhood search for solving constraint optimization problems. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pages 189–194, 2014.
5. P. Erdös and A. Rényi. On Random Graphs I. *Publicationes Mathematicae Debrecen*, 6:290, 1959.
6. A. Farinelli, A. Rogers, A. Petcu, and N. Jennings. 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)*, pages 639–646, 2008.
7. F. Fioretto, T. Le, W. Yeoh, E. Pontelli, and T. C. Son. 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)*, pages 307–323, 2014.
8. A. Gershman, A. Meisels, and R. Zivan. Asynchronous Forward-Bounding for distributed COPs. *Journal of Artificial Intelligence Research*, 34:61–88, 2009.
9. P. Gutierrez, J. H.-M. Lee, K. M. Lei, T. W. K. Mak, and P. Meseguer. Maintaining Soft Arc Consistencies in BnB-ADOPT + during Search. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP)*, pages 365–380, 2013.
10. P. Gutierrez and P. Meseguer. Improving BnB-ADOPT$^+$-AC. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 273–280, 2012.
11. D. Hatano and K. Hirayama. Deqed: An efficient divide-and-coordinate algorithm for dcop. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 566–572, 2013.
12. K. Hirayama and M. Yokoo. The distributed breakout algorithms. *Artificial Intelligence*, 161(1–2):89–115, 2005.

13. C. Kiekintveld, Z. Yin, A. Kumar, and M. Tambe. Asynchronous algorithms for approximate distributed constraint optimization with quality bounds. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 133–140, 2010.
14. R. N. Lass, W. C. Regli, A. Kaplan, M. Mitkus, and J. J. Sim. Facilitating communication for first responders using dynamic distributed constraint optimization. In *Proceedings of the Symposium on Technologies for Homeland Security (IEEE HST)*, pages 316–320, 2008.
15. R. Maheswaran, J. Pearce, and M. Tambe. Distributed algorithms for DCOP: A graphical game-based approach. In *PDCS*, pages 432–439, 2004.
16. L. Martin and R. Martin. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *ICML*, pages 535–542, 2000.
17. P. Modi, W.-M. Shen, M. Tambe, and M. Yokoo. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161(1–2):149–180, 2005.
18. D. T. Nguyen, W. Yeoh, and H. C. Lau. Distributed Gibbs: A memory-bounded sampling-based DCOP algorithm. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 167–174, 2013.
19. D. T. Nguyen, W. Yeoh, H. C. Lau, S. Zilberstein, and C. Zhang. Decentralized multi-agent reinforcement learning in average-reward dynamic DCOPs. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 1447–1455, 2014.
20. J. Pearce and M. Tambe. Quality guarantees on k-optimal solutions for distributed constraint optimization problems. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1446–1451, 2007.
21. A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1413–1420, 2005.
22. A. Petcu and B. Faltings. A hybrid of inference and local search for distributed combinatorial optimization. In *Proceedings of the International Conference on Intelligent Agent Technology (IAT)*, pages 342–348, 2007.
23. A. Rogers, A. Farinelli, R. Stranders, and N. Jennings. Bounded approximate decentralised coordination via the max-sum algorithm. *Artificial Intelligence*, 175(2):730–759, 2011.
24. E. Sultanik, P. J. Modi, and W. C. Regli. On modeling multiagent task scheduling as a distributed constraint optimization problem. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1531–1536, 2007.
25. P. Van Hentenryck and L. Michel. *Constraint-based local search*. The MIT Press, 2009.
26. M. Vinyals, M. Pujol, J. A. Rodriguez-Aguilar, and J. Cerquides. Divide-and-coordinate: Dcops by agreement. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 149–156, 2010.
27. M. Vinyals, E. Shieh, J. Cerquides, J. Rodriguez-Aguilar, Z. Yin, M. Tambe, and E. Bowring. Quality guarantees for region optimal DCOP algorithms. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 133–140, 2011.
28. W. Yeoh, X. Sun, and S. Koenig. Trading off solution quality for faster computation in DCOP search algorithms. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 354–360, 2009.
29. W. Yeoh and M. Yokoo. Distributed problem solving. *AI Magazine*, 33(3):53–65, 2012.
30. R. Zivan, S. Okamoto, and H. Peled. Explorative anytime local search for distributed constraint optimization. *Artificial Intelligence*, 212:1–26, 2014.