Large Neighborhood Search with Quality Guarantees for Distributed Constraint Optimization Problems*

(Extended Abstract)

Ferdinando Fioretto†, Federico Campeotto†, Agostino Dovier‡,
Enrico Pontelli†, William Yeoh†
†Department of Computer Science, New Mexico State University, Las Cruces, NM 88003, USA
‡Department of Mathematics and Computer Science, University of Udine, 208 33100 Udine, IT
{ffiorett,fcampeot,epontell,wyeoh}@cs.nmsu.edu ‡agostino.dovier@uniud.it

ABSTRACT

This paper proposes *Distributed Large Neighborhood Search (D-LNS)*, an incomplete DCOP algorithm that builds on the strengths of centralized LNS. D-LNS: *(i)* is anytime; *(ii)* provides guarantees on solution quality (upper and lower bounds); and *(iii)* can learn online the best neighborhood to explore. Experimental results show that D-LNS outperforms other incomplete DCOP algorithms in random and scale-free network instances.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed AI

General Terms

Algorithms; Experimentation

Keywords

DCOP; LNS; Local Search; Distributed Q-Learning

1. INTRODUCTION

In *distributed constraint optimization problems (DCOPs)*, agents coordinate their value assignments to maximize the sum of resulting constraint utilities. DCOPs have been used to model various multi-agent coordination and resource allocation problems. Optimally solving DCOPs is NP-hard, thus incomplete algorithms are often necessary to solve large problems of interest. Good quality assessment are essential for sub-optimal DCOP solutions. However, current incomplete DCOP approaches can provide arbitrarily poor quality assessments. In this paper, we introduce the *Distributed LNS (D-LNS)* framework, which builds on the strengths of *Large Neighboring Search (LNS)* to solve DCOPs. LNS is a *centralized* local search algorithm that is very effective in solving large optimization problems. It alternates between 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*).

*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.

Appears in: *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015), Bordini, Elkind, Weiss, Yolum (eds.), May 4–8, 2015, Istanbul, Turkey.*

2. BACKGROUND

A *DCOP* is a tuple $\langle \mathcal{A}, \mathcal{D}, \mathcal{F} \rangle$, where $\mathcal{A} = \{x_i\}_1^n$ is a set of *agents*, $\mathcal{D} = \{D_i\}_1^n$ is a set of finite *domains* (i.e, $x_i \in D_i$), $\mathcal{F} = \{f_i\}_1^e$ is the set of binary *utility functions*, where $f_i : D_{i_1} \times D_{i_2} \to \mathbb{N} \cup \{-\infty\}$, A *solution* $\mathbf{x}$ is a value assignment for all agents. Its utility $\mathbf{F}(\mathbf{x})$ is the sum of the utilities across all the applicable utility functions in $\mathbf{x}$. Given a DCOP P, $G_P = (\mathcal{A}, E_{\mathcal{F}})$ is the *constraint graph* of P, where $\{x_{i_1}, x_{i_2}\} \in E_{\mathcal{F}}$ iff $\exists f_i \in \mathcal{F}$. A *DFS pseudo-tree* of G_P is a spanning tree $T = \langle \mathcal{A}, E_T \rangle$ of G_P s.t. if $f_i \in \mathcal{F}$, then x_{i_1}, x_{i_2} 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*).

3. DISTRIBUTED LNS

We introduce D-LNS, a distributed LNS framework to solve DCOPs. Each agent executing D-LNS iterates through the destroy and repair phases as in LNS until a termination condition occurs.

Destroy: This phase aims at *destroying* part of the current solution, by *freezing* some agents (i.e., maintain their values from the previous iteration), and enabling the optimization over the non-frozen agents (large neighborhood). D-LNS uses *Distributed Q-Learning* to select a large neighborhood that enable the repair phase to find good solutions quickly. *Distributed Q-Learning* is used to find an optimal action-choice policy in multi-agent *Markov Decision Processes* (MDPs). It works by learning an action-value function $Q : \mathbf{S} \times \mathbf{A} \to \mathbb{R}$, where $\mathbf{S}$ is a finite set of states and $\mathbf{A}$ is a finite set of executable actions. An optimal policy $\pi^* : \mathbf{S} \to \mathbf{A}$ can be decomposed into n component-policies $\pi_i^* : \mathbf{S} \to \mathbf{A}_i$ with $\mathbf{A}_i$ being the set of actions executable by agent x_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,

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

where $Q_i^1(s, z_i) = 0$, $R_i : \mathbf{S} \times \mathbf{A}_i \to \mathbb{R}$ is a *reward function* of agent x_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 $\mathbf{T}_i : \mathbf{S} \times \mathbf{A}_i \to \mathbf{S}$. We 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 x_i, and iteration k, $\mathbf{A}_i$ is

$\|\mathcal{A}\|$	D-BLNS			BMS			KOPT2			KOPT3			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
10	11	**1.03**	**1.01**	211	1.05	1.29	44	4.33	1.06	57	3.50	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	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	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	1630	1.64	582	2.12
250	**1405**	**1.63**	**1.00**	18372	2.34	1.43	86675	42.2	1.68		–		23091	2.97	6533	3.22
500	**12938**	**1.65**	**1.00**	242236	2.75	1.63		–			–		135366	1.67	54188	2.22
1000	**133047**	**1.59**	**1.00**		–			–			–		278300	1.60	–	

Table 1: Experimental Results on Distributed RLFA Problems

the set $\{\text{fr}, \neg\text{fr}\}$, as in *freeze* or *not freeze*; the reward computed at iteration k is $R_i(s, z_i) = \mathbf{F}(\mathbf{x}^k)$, 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 decision of the agent to freeze its variable or not based on the complete solution in the previous iteration.

Repair: The goal of this phase is to find an improved solution by searching over a large neighborhood. To do so, we introduce the *Distributed Bounded LNS* (D-BLNS) algorithm, which attempts to iteratively refine the DCOP solution bounds. In each iteration k, it executes the following phases.

RELAXATION PHASE: Given a DCOP P, this phase computes two relaxations[1] 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. Let LN^k be the set of large neighborhood agents 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 $E_k = \{\{x, y\} \mid \{x, y\} \in E_{\mathcal{F}}; x, y \in LN^k\}$. In the relaxed DCOP $\check{P}^k$, we wish to find a solution $\check{\mathbf{x}}^k$ using[2]

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

where $\check{\mathbf{x}}_j^{k-1}$ is the value assigned to agent 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 agent and a frozen one. 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:

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

where $\Gamma_f^k = \left\{k' \mid f \in E_{T^{k'}} \wedge 1 \leq k' \leq k\right\}$ is the set of past iteration indices for which the function f was a tree edge in the pseudo-tree. 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 agents 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. 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 the equations above. At a high-level, D-BLNS is an inference-based algorithm, similar to DPOP [5]. Each agent starting from the leaves of the pseudo-tree, projects itself out and sends its projected utilities to its parent. These utilities are propagated up the pseudo-tree until they reach the root. 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.

BOUNDING PHASE: Finally, D-BLNS computes the lower and upper bounds based on the solutions $\check{\mathbf{x}}^k$ and $\hat{\mathbf{x}}^k$. A guaranteed approximation ratio for P is $\rho = \frac{\min_k \hat{F}^k(\hat{\mathbf{x}}^k)}{\max_k \mathbf{F}(\check{\mathbf{x}}^k)}$

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

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

4. 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) [6], k-optimal algorithms (KOPT) [4], Distributed Breakout Algorithm (DBA) [2], and Distributed Gibbs (D-Gibbs) [3]. We conduct our experiments on distributed Radio Link Frequency Assignment (RLFA) problems [1] and report the simulated runtime [7] averaged over 20 DCOP instances. We model the constraint graphs as scale-free graphs, and vary $|\mathcal{A}|$ from 10 to 1000, and set $|D_i| = 10$. Table 1 reports, for each algorithm, 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. The results show D-BLNS can scale better to large problems than algorithms with quality guarantees (i.e., BMS, KOPT2, and KOPT3). Additionally, it also finds better solutions (i.e., better approximation ratios ρ and better quality ratios ϵ) than those of the competing algorithms.

REFERENCES

[1] F. Fioretto, T. Le, W. Yeoh, E. Pontelli, and T. C. Son. Improving DPOP with branch consistency for solving distributed constraint optimization problems. In *CP*, pages 307–323, 2014.

[2] K. Hirayama and M. Yokoo. The distributed breakout algorithms. *Artificial Intelligence*, 161(1–2):89–115, 2005.

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

[4] J. Pearce and M. Tambe. Quality guarantees on k-optimal solutions for distributed constraint optimization problems. In *IJCAI*, pages 1446–1451, 2007.

[5] A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *IJCAI*, pages 1413–1420, 2005.

[6] 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.

[7] E. Sultanik, P. J. Modi, and W. C. Regli. On modeling multiagent task scheduling as a distributed constraint optimization problem. In *IJCAI*, pages 1531–1536, 2007.