ER-DCOPs: A Framework for Distributed Constraint Optimization with Uncertainty in Constraint Utilities

Tiep Le, Ferdinando Fioretto, William Yeoh, Tran Cao Son, and Enrico Pontelli
New Mexico State University
Las Cruces, New Mexico, USA
{tile,ffiorett,wyeoh,tson,epontell}@cs.nmsu.edu

ABSTRACT

Distributed Constraint Optimization Problems (DCOPs) have been used to model a number of multi-agent coordination problems. In DCOPs, agents are assumed to have complete information about the utility of their possible actions. However, in many real-world applications, such utilities are stochastic due to the presence of exogenous events that are beyond the direct control of the agents. This paper addresses this issue by extending the standard DCOP model to *Expected Regret DCOP* (ER-DCOP) for DCOP applications with uncertainty in constraint utilities. Different from other approaches, ER-DCOPs aim at minimizing the overall expected regret of the problem. The paper proposes the *ER-DPOP* algorithm for solving ER-DCOPs, which is complete and requires a linear number of messages with respect to the number of agents in the problem. We further present two implementations of ER-DPOP—GPU- and ASP-based implementations—that orthogonally exploit the problem structure and present their evaluations on random networks and power network problems.

Keywords

DCOP; Expected Regret; GPU; ASP

1. INTRODUCTION

Distributed Constraint Optimization Problems (DCOPs) are problems where agents need to cooperatively determine their variables' value assignment to maximize the sum of resulting constraint utilities [22, 25, 19, 31]. Researchers have used them to model various multi-agent coordination and resource allocation problems [18, 7, 28, 30, 32].

A DCOP is typically specified by a finite set of *decision variables* (throughout the paper, whenever we mention variables, we mean decision variables) and a finite set of *constraints* among these variables. Each constraint indicates *a utility* for each value assignment of the variables involved in it. One limitation of DCOPs is that the constraint utilities are assumed to be *known* and *deterministic*. Thus, the DCOP model is not suitable for modeling problems where constraint utilities are stochastic (e.g., the utility of a constraint might depend on exogenous factors that are beyond the *direct* control of the agents, such as weather conditions, properties of the surrounding environment, etc.).

To address the above limitation, several DCOP extensions that handle stochasticity in the constraint utilities have been developed. These extensions require that the constraint utilities **(i)** are sampled from a predefined function that is either known [1] or concave [23]; or **(ii)** depend on additional random variables (not controlled by agents) whose probability distributions are known [16, 14]. Unfortunately, such requirements are not met in several real-world applications (e.g., accurate weather models are often unavailable).

Uncertain Reward DCOP (UR-DCOP) [29] is another extension of the DCOP model that uses an additional *random variable* for each constraint, in order to handle stochastic utilities. Different from other approaches, it can model problems where the probability distribution (referred to as *belief*) of random variables is not known, but sampled from a *known probability distribution space* (referred to as the *belief space*). UR-DCOP *(i)* assumes that the belief of a random variable is *independent* from the values of the decision variables; and *(ii)* uses the *minimax regret* strategy, which minimizes the worst-case loss (the *regret*) over the known belief space. This strategy may be overly pessimistic in many situations.

In this paper, we develop an alternative approach for DCOPs with stochastic constraint utilities that relaxes some of the assumptions made by previously developed models. Specifically, we **(1)** propose the *Expected Regret DCOP* (ER-DCOP) model, which removes the independence assumption in UR-DCOPs and optimizes the *minimum expected regret* in response to the constraint utilities' uncertainty; **(2)** develop the *ER-DPOP* algorithm to solve ER-DCOPs, proposing two implementations: one that exploits GPUs to speed up the process of solving ER-DCOPs, and another one that uses ASP and exploits inference rules to prune the solution search space; **(3)** evaluate the performance of the ER-DPOP implementations on random graphs and power network problems; and **(4)** experimentally show that ER-DCOP solutions outperform corresponding UR-DCOP solutions in terms of the actual regret.

The rest of the paper is organized as follows. We start with a motivating example and review some background in Sections 2 and 3, respectively. The ER-DCOP model is introduced in Section 4, the ER-DPOP algorithm to solve this model is proposed in Section 5, followed by some theoretical analysis in Section 6. Next, we introduce the GPU-based and the ASP-based ER-DPOP implementations in Section 7. Section 8 provides the experimental results, and we conclude the paper with some discussions and related works.

2. MOTIVATING EXAMPLE

In this section, we introduce a motivating example that is used to illustrate the proposed model and algorithm.

EXAMPLE 1. *Consider a Mars Rover problem consisting of two* robotic workers x_1 *and* x_2 *that will collect soil samples on Mars (i.e.,* $x_1, x_2 \in \{0\}$[1]*), and an* assistant robot x_3 *(*$x_3 \in \{0, 1\}$*)*

[1]We use 0 to denote this single task for simplicity.

Appears in: *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016), J. Thangarajah, K. Tuyls, C. Jonker, S. Marsella (eds.), May 9–13, 2016, Singapore.*

x_1	x_3	r_1	U_1	P^e	P^u
0	0	0	0	10%	30%
		1	50	90%	70%
0	1	0	0	30%	50%
		1	30	70%	50%

(a)

x_2	x_3	r_2	U_2	P^e	P^u
0	0	0	0	50%	90%
		1	40	50%	10%
0	1	0	0	20%	50%
		1	50	80%	50%

(b)

Table 1: Constraints and their Probabilistic Models

that will be able to support one worker, either x_1 ($x_3 = 0$) or x_2 ($x_3 = 1$). The random variables r_1 and r_2 are introduced as follows: with $i \in \{1, 2\}$, $r_i = 1$ (resp. $r_i = 0$) represents the fact that the worker x_i succeeds (resp. fails) in collecting soil samples. The first four columns of the Table 1(a) (resp. 1(b)) describe the constraint utilities U_1 (resp. U_2) for the joint work between the robots x_1 and x_3 (resp. x_2 and x_3). For example, in Table 1(a), with robot x_3 supporting (i.e., $x_3 = 0$), if worker x_1 succeeds, it collects 50 samples (i.e., $U_1 = 50$); if it fails, it collects 0 sample (i.e., $U_1 = 0$). Moreover, we assume that the chance for workers to succeed or fail depends only on the terrain condition (i.e., even (e) or uneven (u)). The terrain condition is 88% uneven and 12% even. The columns P^e (resp. P^u) in Table 1(a and b) define the probability distributions of the respective random variable for even (resp. uneven) terrain. For example, the first two rows of the column P^e in the Table 1(a) show that, given that the terrain is even and $x_3 = 0$, the probability of the worker x_1 to fail (i.e., $r_1 = 0$) or succeed (i.e., $r_1 = 1$) is 10% or 90%, respectively.

3. BACKGROUND

In this section, we present an overview of DCOPs; describe DPOP, a popular, complete algorithm to solve DCOPs, whose resolution process is central in our ER-DPOP algorithm; and review the formal definition of UR-DCOPs. We further provide some fundamental definitions of GPU and ASP.

3.1 DCOP

A *Distributed Constraint Optimization Problem* (DCOP) is a tuple $\mathcal{M} = \langle \mathcal{X}, \mathcal{D}, \mathcal{F}, \mathcal{A}, \alpha \rangle$ where $\mathcal{X} = \{x_1, \ldots, x_n\}$ is a finite set of (decision) *variables*; $\mathcal{D} = \{D_1, \ldots, D_n\}$ is a set of finite *domains*, where each D_i is the domain of variable $x_i \in \mathcal{X}$; $\mathcal{F} = \{f_1, \ldots, f_m\}$ is a finite set of *constraints*, where each k_j-ary constraint $f_j : D_{j_1} \times D_{j_2} \times \ldots \times D_{j_{k_j}} \mapsto \mathbb{R} \cup \{-\infty\}$ specifies the utility of each combination of values of the variables in its *scope*; the scope is denoted by $scp(f_j) = \{x_{j_1}, \ldots, x_{j_{k_j}}\}$; $\mathcal{A} = \{a_1, \ldots, a_p\}$ is a finite set of *agents*; and $\alpha : \mathcal{X} \mapsto \mathcal{A}$ maps each variable to an agent. Each constraint in $\mathcal{F}$ can be either *hard*, indicating that some value combinations result in a utility of $-\infty$ and must be avoided, or *soft*, indicating that all value combinations result in a finite utility and need not be avoided. Given a constraint f_j and a *complete value assignment* x for all decision variables, we denote with x_{f_j} a *partial value assignment* from x for all variables in $scp(f_j)$. A *solution* of a DCOP is a complete value assignment $\mathbf{x}$ for all variables such that $\mathbf{x} = \text{argmax}_x \sum_{j=1}^{m} f_j(x_{f_j})$.

A DCOP can be described by its *constraint graph*, i.e., a graph whose nodes correspond to the variables in $\mathcal{X}$ and whose edges connect pairs of variables in the scope of the same constraint. A simplifying assumption, which is common practice in the DCOP literature, is that each agent owns exactly one variable. Thus, nodes of a constraint graph can be seen as representing agents without causing any confusion. A *pseudo-tree* arrangement of a DCOP has the same nodes as the constraint graph and a subset of its edges such that *(i)* the included edges (called *tree edges*) form a rooted tree, and *(ii)* two variables in the scope of the same constraint appear in the same branch of the tree. The remaining edges of the constraint graph not included in the pseudo-tree are called *back edges*. Tree edges connect a node with its parent and its children while back edges connect a node with its pseudo-parents and pseudo-children (i.e., higher nodes are parents or pseudo-parents, and lower nodes are children or pseudo-children). In a pseudo-tree, the *separator* of a node x_i (denoted by sep_i) is the set of ancestors of x_i that are directly connected (via tree edges or back edges) with x_i or with descendants of x_i.

3.2 DPOP

The *Distributed Pseudo-tree Optimization Procedure* (DPOP) [25] is a complete distributed algorithm to solve DCOPs. DPOP consists of three phases: Pseudo-tree generation, UTIL propagation, and VALUE propagation.

- **Pseudo-tree Generation:** DPOP calls existing distributed DFS algorithms [12] to build a pseudo-tree.
- **UTIL Propagation:** Starting from the leafs of the pseudo-tree, each agent computes the optimal sum of utilities in its subtree for each value combination of variables in its separator and sends these utilities in a UTIL message to its parent.
- **VALUE Propagation:** Each agent, starting from the root of the pseudo-tree, determines its variable's optimal value and sends this value, together with its ancestors' optimal values, in VALUE messages to its children. The root agent does so by choosing the value of its variable from its UTIL computations.

3.3 UR-DCOP

Uncertain Reward DCOP (UR-DCOP) [29] is an extension of the original DCOP model with two additional components:

- $\mathcal{E} = \{s_1, \ldots, s_m\}$ is a set of random variables, each random variable s_j is for each constraint $f_j \in \mathcal{F}$.
- $\mathcal{S} = \{S_1, \ldots, S_m\}$ is a set of finite domains where S_j is the domain of random variable s_j.

The constraints in $\mathcal{F}$ are augmented to consider both decision variables and random variables—i.e., each k_j-ary constraint has the form $f_j : S_j \times D_{j_1} \times D_{j_2} \times \ldots \times D_{j_{k_j}} \mapsto \mathbb{R} \cup \{-\infty\}$.

Given a random variable s_j in UR-DCOPs, let $b_j \in \triangle(S_j)$ be the probability distribution over domain S_j, and $\mathbf{b} = \langle b_1, \ldots, b_m \rangle$. When $\mathbf{b}$ is known, solving a UR-DCOP involves finding an assignment of all decision variables x that minimizes the sum of expected values $V(\mathbf{b}, x) = \sum_{j=1}^{m} \sum_{s_j \in S_j} b_j(s_j) \cdot f_j(s_j, x_{f_j})$.

However, since $\mathbf{b}$ is unknown, given the space $\mathcal{B}$ of $\mathbf{b}$, the objective of UR-DCOPs is the *minimax regret*:

$$V_{regret} = \min_{x'} \max_{\mathbf{b} \in \mathcal{B}} \max_{x^*} [V(\mathbf{b}, x^*) - V(\mathbf{b}, x')]$$

where x^* is the optimal solution given $\mathbf{b}$.

3.4 GPUs

Graphics Processing Units (GPUs) are multiprocessor devices, offering hundreds of computing cores and a rich memory hierarchy. A parallel computation is described by a collection of *kernels* (i.e., procedures) executed by several *threads*. Threads are in turn organized hierarchically into *blocks* and *grids of blocks*, and have access to several memory levels, each with different properties in terms of speed and capacity. Each thread can store its private variables in very fast, but few in number, registers. Threads within a block can communicate by reading and writing a common *shared memory* area. Communication between blocks and between blocks and the host (i.e., the CPU) is done through a large but slow *global memory*. Memory levels have significantly different sizes (e.g., registers are in the order of dozens per thread, while shared mem-

ory is in the order of a few kilobytes per block) and access times (e.g., global memory access typically requires 400-600 clock cycles, shared memory 20-40 cycles). The conceptual model of parallelism supported by CUDA [26] is *Single-Instruction Multiple-Thread* (SIMT), where the same instruction is executed by different threads that run on identical cores, while data and operands may differ from thread to thread.

3.5 ASP

We now review some background on *Answer Set Programming* (ASP). An *answer set program* [3] Π is a set of *rules* of the form

$$c \leftarrow a_1, \ldots, a_s, not\ b_1, \ldots, not\ b_t$$

where $s \geq 0$, $t \geq 0$, and the a_i's and b_i's are ground literals of a first order language $\mathcal{L}$. The formula $not\ b_i$ is called a *negation-as-failure* literal (or naf-literal). c can be a ground literal or omitted. A program is *definite* if it does not contain naf-literals. If $s = t = 0$, the rule is called a *fact*.

A set of ground literals X is *consistent* if there is no atom a such that $\{a, \neg a\} \subseteq X$. A literal ℓ is true (false) in a set of literals X if $\ell \in X$ ($\ell \notin X$). A set of ground literals X satisfies a ground rule if any of the following cases is true: *(i)* $c \in X$; *(ii)* some a_i is false in X; or *(iii)* some b_i is true in X. A solution of a program, called an *answer set* [10], is a consistent set S of ground literals such that:

- If Π is a definite program, then S is a subset-minimal set of ground literals satisfying all rules in Π.
- If Π contains some naf-literals, then S is an answer set of Π^S that is obtained from Π by deleting *(i)* each rule that has a naf-literal *not* b in its body with $b \in S$, and *(ii)* all naf-literals in the bodies of the remaining rules.

A rule with variables is viewed as a shorthand of the collection of all of its ground instances. Similarly, a non-ground program is a shorthand for all ground instances of its rules. The ASP language includes also language-level extensions to facilitate the encoding of aggregates (*min*, *max*, *sum*, etc.). ASP solves a problem by encoding it as an ASP program whose answer sets correspond one-to-one to the solutions of the problem [20, 24]. Solutions can be computed using ASP solvers like CLASP [9] and DLV [5].

4. THE ER-DCOP MODEL

In this section, we introduce the *Expected Regret DCOP* (ER-DCOP) model—a new representation for DCOPs with uncertainty in constraint utilities. An ER-DCOP model is always defined over a so-called *belief space*. So, we first propose the ER-DCOP model and then introduce the definition of a belief space.

An ER-DCOP is an extension of DCOP to model stochastic constraint utilities. Formally, it is a tuple $\mathcal{E} = \langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \alpha, \mathcal{R}, \mathcal{S}, \mathcal{F} \rangle$, where:

- $\mathcal{A}, \mathcal{X}, \mathcal{D}$, and α are the same as in a DCOP.
- $\mathcal{R} = \{r_1, \ldots, r_m\}$ is a set of random variables modeling external factors beyond the direct control of agents (e.g., the factor of whether robots succeed or not in Example 1). We note that an agent cannot change the value of the random variable directly.
- $\mathcal{S} = \{S_1, \ldots, S_m\}$ is a set of finite domains, where S_j is the domain of the random variable $r_j \in \mathcal{R}$ (e.g., success or failure in Example 1).
- $\mathcal{F} = \{f_1, \ldots, f_m\}$ is a finite set of constraints, where each k_j-ary constraint $f_j : r_j \times D_{j_1} \times D_{j_2} \times \ldots \times D_{j_{k_j}} \mapsto \mathbb{R} \cup \{-\infty\}$, different from that in DCOPs, is augmented to consider both a random variable and decision variables in specifying the corresponding utilities. The scope of a constraint f_j is denoted by $scp(f_j) = \{x_{j_1}, \ldots, x_{j_{k_j}}\}$, which is similar as the scope of constraints in standard DCOP.

We assume that each constraint f_j is associated to one random variable r_j. If a constraint is associated with multiple random variables (i.e., multiple external factors affect the utilities of a constraint), we can merge all such random variables into a single random variable. Furthermore, we assume that random variables are *affected* by the decision variables (i.e., random variables are beyond the direct control of the agents but are influenced by the agent's decisions). Moreover, their values are drawn from *unknown probability distributions* but can be sampled from *known probability distribution spaces*. For example, in Table 1(b), the probability distributions of r_2 varies in: *(i)* different terrain conditions (see columns P^e and P^u) and *(ii)* different value assignments of x_2 and x_3. For example, given $x_2 = x_3 = 0$, the probabilities of $r_2 = 1$ in the even and in the uneven terrain conditions are 50% and 10%, respectively. Additionally, with the even terrain condition, the probabilities of $r_2 = 1$ where $x_2 = x_3 = 0$ and where $x_2 = 0, x_3 = 1$ are 50% and 80%, respectively.

Given a constraint f_j, let X_{f_j} be the set of all possible value assignments for the decision variables in $scp(f_j)$. In ER-DCOPs, the conditional probability distribution of a random variable r_j over its domain S_j given a specific value assignment of the variables in $scp(f_j)$, denoted by b_j, is called a *belief* of the random variable r_j. Thus, the probability that $r_j = s_j \in S_j$ given a specific value assignment $x_{f_j} \in X_{f_j}$ is $b_j(s_j|x_{f_j})$, and we have $\sum_{s_j \in S_j} b_j(s_j|x_{f_j}) = 1$. For example, in Table 1(a), the column P^e denotes the belief b_1^e—the belief of the random variable r_1 in the even terrain. Let x_{f_1} be a partial value assignment in which $x_1 = x_3 = 0$. We have, $b_1^e(0|x_{f_1}) = 0.1$, and $b_1^e(1|x_{f_1}) = 0.9$.

We denote with $B = \langle b_1, \ldots, b_m \rangle$ a *joint belief* of all random variables, where each b_j is a belief of $r_j \in \mathcal{R}$. For example, the columns P^e (resp. P^u) in Table 1(a and b) together detail the joint belief B^e (resp. B^u)—the joint belief for even (resp. uneven) terrain.

When the joint belief $B = \langle b_1, \ldots, b_m \rangle$ is known, solving ER-DCOPs *with respect to* the joint belief B is straightforward; it involves finding a complete value assignment $\mathbf{x}$ that maximizes the sum of the *expected utility* of the constraints:

$$E(B, \mathbf{x}) = \sum_{j=1}^{m} \underbrace{\sum_{s_j \in S_j} b_j(s_j|\mathbf{x}_{f_j}) \cdot f_j(s_j, \mathbf{x}_{f_j})}_{E(b_j, \mathbf{x}_{f_j})} \quad (1)$$

In Equation (1), $E(b_j, \mathbf{x}_{f_j})$ is the expected utility of the constraint f_j corresponding to the partial value assignment $\mathbf{x}_{f_j}$ w.r.t. b_j.

In Example 1 and throughout the rest of the paper, let x_a (resp. x_b) be a value assignment where $x_1 = x_2 = x_3 = 0$ (resp. $x_1 = x_2 = 0, x_3 = 1$). Since $E(B^e, x_a) = 65 > E(B^e, x_b) = 61$, we have that x_a is the solution for the ER-DCOP with respect to B^e. Moreover, we also have $E(B^u, x_a) = 39 < E(B^u, x_b) = 40$, so x_b is the solution for the ER-DCOP with respect to B^u.

The key challenge here in solving ER-DCOPs is that the joint belief is typically unknown—i.e., an ER-DCOP may have multiple joint beliefs that have to be considered. Formally, an ER-DCOP is defined over a finite nonempty set of joint beliefs, $\mathcal{B} = \{B^1, \ldots, B^v\}$, referred to as the *belief space* of the ER-DCOP. The uncertainty of joint beliefs in the belief space $\mathcal{B}$ is drawn under a probabilistic model $\mathcal{P}_\mathcal{B}$ where $\sum_{q=1}^{v} \mathcal{P}_\mathcal{B}(B^q) = 1$. The goal of an ER-DCOP, given the belief space $\mathcal{B}$ whose probability distribution is $\mathcal{P}_\mathcal{B}$, is to find a complete value assignment $\mathbf{x}'$ that *minimizes the*

expected regret:

$$ER(\mathcal{B}, \mathbf{x}') = \sum_{B^q \in \mathcal{B}} \mathcal{P}_{\mathcal{B}}(B^q) \cdot \max_{\mathbf{x}^{\mathbf{B^q}}} \underbrace{[E(B^q, \mathbf{x}^{\mathbf{B^q}}) - E(B^q, \mathbf{x}')]}_{R(B^q, \mathbf{x}')} \quad (2)$$

The assignment $\mathbf{x}^{\mathbf{B^q}}$ selected by the maximization operator is the solution of the ER-DCOP with respect to B^q—i.e., $E(B^q, \mathbf{x}^{\mathbf{B^q}})$ is the maximal expected utility achieved with respect to B^q and its associated $\mathbf{x}^{\mathbf{B^q}}$. $R(B^q, \mathbf{x}')$ is the *regret* (or loss) for a complete value assignment $\mathbf{x}'$ relative to B^q—i.e., the difference of the aggregate expected utilities achieved by $\mathbf{x}'$ and $\mathbf{x}^{\mathbf{B^q}}$. Note that the smallest value of the expected regret $ER(\mathcal{B}, \mathbf{x}')$ minimizes the expected loss over all possible joint beliefs $B^q \in \mathcal{B}$.

In Example 1, $\mathcal{B} = \langle B^e, B^u \rangle$, $\mathcal{P}_{\mathcal{B}}(B^e) = 12\%$, and $\mathcal{P}_{\mathcal{B}}(B^u) = 88\%$. We have $ER(\mathcal{B}, x_a) = 12\% \cdot (65-65) + 88\% \cdot (40-39) = 0.88$ and $ER(\mathcal{B}, x_b) = 12\% \cdot (65-61) + 88\% \cdot (40-40) = 0.48$. Thus, since $ER(\mathcal{B}, x_a) = 0.88 > ER(\mathcal{B}, x_b) = 0.48$, the value assignment x_b is the solution of the original ER-DCOP as it yields the minimal expected regret of 0.48.

5. ER-DPOP ALGORITHM

We now describe a distributed algorithm, called *Expected Regret-DPOP* (ER-DPOP), to solve ER-DCOPs. It is composed of three phases: **(1)** *Generation of the pseudo-tree*; **(2)** *Resolution of subproblems,* where ER-DPOP computes an optimal solution $\mathbf{x}^{\mathbf{B^q}}$ for each joint belief $B^q \in \mathcal{B}$; and **(3)** *Resolution of the main problem,* where ER-DPOP finds the solution $\mathbf{x}'$ of the original ER-DCOP that minimizes $ER(\mathcal{B}, \mathbf{x}')$. ER-DPOP makes use of repeated calls to the DPOP algorithm [25]. We choose DPOP to develop ER-DPOP because DPOP is one of the most popular complete DCOP algorithms. Moreover, since DPOP is synchronous, a modification of DPOP will allow to solve optimal solutions for every joint belief in Phase 2 simultaneously.

Phase 1 [*Generation of the pseudo-tree*]: In this phase, agents coordinate to build a pseudo-tree as in DPOP.

Phase 2 [*Resolution of subproblems*]: This phase consists of two steps. In the first step, the original ER-DCOP is transformed into a standard DCOP for each joint belief $B^q \in \mathcal{B}$, where the constraint utilities are updated with the respective expected utilities. More precisely, for each joint belief $B^q \in \mathcal{B}$, a new standard DCOP $M^{B^q} = \langle \mathcal{X}, \mathcal{D}, \mathcal{F}^{B^q}, \mathcal{A}, \alpha \rangle$ is generated where $\mathcal{F}^{B^q} = \{f_1^{B^q}, \ldots, f_m^{B^q}\}$ and $f_j^{B^q}$ is a constraint—whose scope is identical to that of $f_j \in \mathcal{F}$, and whose constraint utilities are given by $f_j^{B^q}(x_{f_j}) = E(b_j^q, x_{f_j})$ for each value assignment x_{f_j} of the variables in $scp(f_j)$, where $b_j^q \in B^q$.

In the second step, the $|\mathcal{B}|$ DCOPs built in the first step are solved, following the general model of DPOP. Rather than solving each DCOP M^{B^q} sequentially, ER-DPOP adapts the DPOP algorithm to solve them *simultaneously*. Intuitively, the UTIL messages produced in this phase (referred to as UTIL2 messages) transmit vectors of aggregate utilities—within such a vector, each of aggregate utilities is with respect to a joint belief $B^q \in \mathcal{B}$. The VALUE messages produced in this phase (referred to as VALUE2 messages) transmit vectors of optimal value assignments—within such a vector, each of optimal value assignments is with respect to a DCOP M^{B^q}. In detail, each agent $a_i \in \mathcal{A}$:

- Constructs its UTIL2 message that includes a vector $\vec{u}_{x_{sep_i}}$ (referred to as a utility vector) $\vec{u}_{x_{sep_i}} = (u^{B^1}, \ldots, u^{B^{|\mathcal{B}|}})$ for *each value assignment* x_{sep_i} of the variables in sep_i, where each u^{B^q} is the optimal utility for its subtree in the DCOP M^{B^q} given the value assignment x_{sep_i}.
- Constructs its VALUE2 message that includes a vector of optimal value assignments $x = (x^{B^1}, \ldots, x^{B^{|\mathcal{B}|}})$ for each variable $x \in sep_i \cup \{x_i\}$, where each x^{B^q} is the value of x in the solution of the DCOP M^{B^q}.

The actions of aggregating the utilities contained in each UTIL2 message and of determining the value assignment for their variables (given the VALUE2 message from the parent agent) are performed as in DPOP. Each operation is performed $|\mathcal{B}|$ times by each agent, one for each joint belief in the belief space of the original ER-DCOP. At the end of these two steps, ER-DPOP has an optimal solution $\mathbf{x}^{\mathbf{B^q}}$ for each $B^q \in \mathcal{B}$.

In Example 1, consider a pseudo-tree where $sep_1 = sep_2 = \{x_3\}$ and $\mathcal{B} = \langle B^e, B^u \rangle$. The UTIL2 message of agent a_1 is $\vec{u}_{x_3=0} = (45, 35)$ and $\vec{u}_{x_3=1} = (21, 15)$; the UTIL2 message of agent a_2 is $\vec{u}_{x_3=0} = (20, 4)$ and $\vec{u}_{x_3=1} = (40, 25)$. Agent a_3 joins the UTIL2 messages that are received from its children a_1 and a_2 as: $\vec{u}_{x_3=0} = (65, 39)$ and $\vec{u}_{x_3=1} = (61, 40)$, and selects its assignment solution $x_3 = 0$ (resp. $x_3 = 1$) as solutions for the joint belief B^e (resp. B^u).

Phase 3 [*Resolution of the main problem*]: Once Phase 2 terminates, ER-DPOP agents can compute $ER(\mathcal{B}, \mathbf{x}')$ for every possible complete value assignment $\mathbf{x}'$. To do so, ER-DPOP transforms all M^{B^q} DCOPs generated in the previous phase into a standard DCOP $M^{\mathcal{B}} = \langle \mathcal{X}, \mathcal{D}, \mathcal{F}^{\mathcal{B}}, \mathcal{A}, \alpha \rangle$, where $F^{\mathcal{B}} = \{f_1^{\mathcal{B}}, \ldots, f_m^{\mathcal{B}}\}$ and each $f_j^{\mathcal{B}}$ has the same scope as $f_j \in \mathcal{F}$ and its constraint utility values are given by:

$$f_j^{\mathcal{B}}(x_{f_j}) = \sum_{B^q \in \mathcal{B}} \mathcal{P}_{\mathcal{B}}(B^q) \cdot [f_j^{B^q}(\mathbf{x}_{f_j}^{\mathbf{B^q}}) - f_j^{B^q}(x_{f_j})] \quad (3)$$

for each value assignment x_{f_j} of the variables in $scp(f_j)$. In Equation (3), $\mathbf{x}_{f_j}^{\mathbf{B^q}}$ is a value assignment of the variables in $scp(f_j)$ from the optimal solution $\mathbf{x}^{\mathbf{B^q}}$, computed in the previous phase of ER-DCOP with respect to B^q. Each ER-DPOP agent computes the constraint $f_j^{\mathcal{B}}$ shared among its variables and its ancestor variables without incurring any privacy loss. In Example 1, agents a_1 and a_2 update their constraints $f_1^{\mathcal{B}}$ and $f_2^{\mathcal{B}}$, respectively, where: $f_1^{\mathcal{B}}(x_1 = 0, x_3 = 0) = 12\% \cdot (45-45) + 88\% \cdot (15-35) = -17.6$; $f_1^{\mathcal{B}}(x_1 = 0, x_3 = 1) = 2.88$; $f_2^{\mathcal{B}}(x_2 = 0, x_3 = 0) = 18.48$; and $f_2^{\mathcal{B}}(x_2 = 0, x_3 = 1) = -2.4$.

Once the standard DCOP $M^{\mathcal{B}}$ is generated, ER-DPOP agents solve this problem to find the *minimal* aggregated utility that is the *minimal expected regret* using standard DPOP.[2] To distinguish from the UTIL2 and VALUE2 messages that are produced in Phase 2, the UTIL and VALUE messages produced in Phase 3 by standard DPOP are referred to as UTIL3 and VALUE3 messages, respectively. The solution of the DCOP $M^{\mathcal{B}}$ is the solution of the original ER-DCOP. In Example 1, the solution x_b ($x_1 = x_2 = 0, x_3 = 1$) is the solution for $M^{\mathcal{B}}$ since it minimizes the expected regret of 0.48 and is the solution of the original ER-DCOP that models Example 1.

6. THEORETICAL ANALYSIS

We now present some theoretical properties of ER-DPOP including its correctness and complexity.

THEOREM 1. *ER-DPOP is correct, that is, given an ER-DCOP $\mathcal{E}$ over a belief space $\mathcal{B}$, ER-DPOP finds a solution $\mathbf{x}'$ that minimizes $ER(\mathcal{B}, \mathbf{x}')$.*

[2] In this phase, agents minimize their aggregated utilities rather than maximizing them.

Proof: In Phase 2, ER-DPOP transforms the original ER-DCOP into $|\mathcal{B}|$ standard DCOP $M^{B^q} = \langle \mathcal{X}, \mathcal{D}, \mathcal{F}^{B^q}, \mathcal{A}, \alpha \rangle$, where $\mathcal{F}^{B^q} = \{f_1^{B^q}, \ldots, f_m^{B^q}\}$ and $f_j^{B^q}$ is a constraint whose scope is identical to that of $f_j \in \mathcal{F}$ except that its constraint utilities are given by $f_j^{B^q}(x_{f_j}) = E(b_j^q, x_{f_j})$. Thus, the solution $\mathbf{x^{B^q}}$ of DCOP M^{B^q} will maximize the sum of the expected utility of the constraints in ER-DCOP w.r.t the joint belief B^q (i.e., $\mathbf{x^{B^q}} = \text{argmax}_x E(B^q, x)$). Moreover, ER-DPOP agents solve each M^{B^q} simultaneously but independently. Each agent performs the actions (i.e., aggregating the expected utilities and determining its assignment solution) $|\mathcal{B}|$ times independently—once is for each joint belief in $\mathcal{B}$—using DPOP to construct UTIL2 and VALUE2 messages. As a result, following the correctness of DPOP [25], at the end of Phase 2, ER-DPOP finds a solution $\mathbf{x^{B^q}}$ for each $B^q \in \mathcal{B}$ that maximizes $E(B^q, \mathbf{x^{B^q}})$.

Similarly, in Phase 3, ER-DPOP solves the DCOP $M^{\mathcal{B}}$ (whose constraints are computed as in Equation (3)) to minimize the aggregated expected regret. Thus, based on the correctness of DPOP [25], at the end of Phase 3, ER-DPOP finds the solution $\mathbf{x}'$ of $M^{\mathcal{B}}$ that minimizes $ER(\mathcal{B}, \mathbf{x}')$, which proves the correctness of ER-DPOP. □

Given $d = \max_{1 \leq i \leq n} |D_i|$ and $w = \max_{1 \leq i \leq n} |sep_i|$, we have the following properties:

PROPERTY 1. *The number of messages required by ER-DPOP is bounded by $\mathcal{O}(n)$ where n is the number of agents.*

Proof: In ER-DPOP, Phase 1 requires a linear number of messages in n [25]; Phase 2 requires $(n-1)$ UTIL2 messages and $(n-1)$ VALUE2 messages; and Phase 3 requires $(n-1)$ UTIL3 messages and $(n-1)$ VALUE3 messages. Thus, the number of messages required by ER-DPOP is bounded by $\mathcal{O}(n)$. □

PROPERTY 2. *The size of messages required by ER-DPOP is bounded by $\mathcal{O}(d^w \cdot |\mathcal{B}|)$.*

Proof: In ER-DPOP, Phase 1 produces messages whose size is linear in n [25]; Phase 2 produces UTIL2 and VALUE2 messages whose sizes are bounded by $\mathcal{O}(d^w \cdot |\mathcal{B}|)$ and $\mathcal{O}(w \cdot |\mathcal{B}|)$, respectively; and Phase 3 produces UTIL3 and VALUE3 messages whose sizes are bounded by $\mathcal{O}(d^w)$ and $\mathcal{O}(w)$, respectively [25]. Thus, the size of messages required by ER-DPOP is bounded by $\mathcal{O}(d^w \cdot |\mathcal{B}|)$ □

7. ER-DPOP IMPLEMENTATIONS

The main technical challenges of ER-DCOP are its exponential messages size and its exponential search space (Property 2). Thus, to address these challenges, in this section, we introduce a GPU-based and an ASP-based implementation of ER-DPOP to exploit SIMT-style parallelism and hard constraints, respectively.

7.1 GPU-ER-DPOP

The use of a GPU-based approach to solve ER-DCOPs is motivated by the observation that each utility vector of the UTIL2 messages and each entry of the UTIL3 messages in Phases 2 and 3, as well as constraints that are defined in Equation (3), can be computed independently. Thus, they can be computed in parallel. This observation finds a natural fit for SIMT processing and, as shown in recent works [4, 8], this is a viable approach to enhance the performance of belief propagation- and dynamic programming-based inference processes.

We thus develop GPU-ER-DPOP (GPU-based ER-DPOP implementation), which allows us to speed up the computation processes

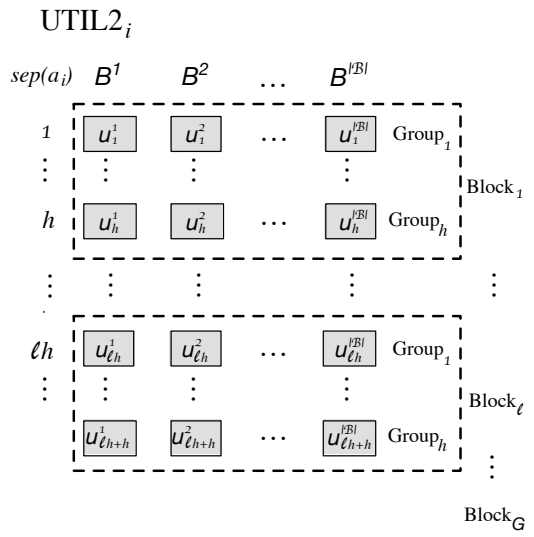


Figure 1: GPU Parallelization

required by Phases 2 and 3 and, as a consequence, reduces the overall runtime.

Figure 1 illustrates the high-level parallelization performed by each ER-DPOP agent when computing a UTIL2 message. Our GPU-ER-DPOP operates along three levels of parallelization:

- Each GPU-ER-DPOP agent a_i computes its UTIL2$_i$ message using G GPU-blocks: $b_1, \ldots, b_G$ with $G = \frac{|sep_i|}{h}$, and h being the number of utility vectors computed within a block.
- In constructing its UTIL2$_i$ message, since computing a utility vector for a value assignment of the variables in sep_i is independent of the computation of another utility vectors, we design a GPU-block b_j ($1 \leq j \leq G$) to compute h utility vectors in parallel. More precisely, the GPU-block b_1 computes h utility vectors corresponding to the first h different partial value assignments of variables in sep_i, sorted in lexicographical order; the GPU-block b_2 computes h utility vectors corresponding to the next h different partial value assignments of variables in sep_i; and so on.
- Within a block, we delegate the computation of each of the h utility vectors to a *group* of $|\mathcal{B}|$ GPU-threads, each of which is in charge of computing the optimal utility u^{B^q} for the agent a_i's subtree in the DCOP M^{B^q}. This is possible since the computations of different u^{B^q} in a utility vector is associated to a particular joint belief and, thus, are independent to each other.

Analogous parallelizations are performed by each GPU agent when constructing UTIL3 messages and computing Equation (3).

The underlying parallelization strategy adopted in our work is inspired to that explored in [8]. In that work, the focus is on the parallelization of the *join* and *projection* operators used in classical (D)COP inference-based resolution approaches. Similarly, GPU-ER-DPOP uses a GPU-thread to compute the optimal sum of utilities in an agent's subtree for a value combination of variables in its separator. In addition, different from the previous approaches, GPU-ER-DPOP heavily exploits the structure of the ER-DCOP model to compute the optimal utilities, with respect to different joint beliefs in parallel, as well as the expected regrets.

The number of threads T used in each block is a parameter that depends on the GPU architecture. It depends on the resource used by the GPU kernel and influences the number h of groups associated to each block. In our settings, we set $T = 256$ and, thus, each block computes at most $h = \lfloor \frac{256}{|\mathcal{B}|} \rfloor$ utility vectors. As a technical

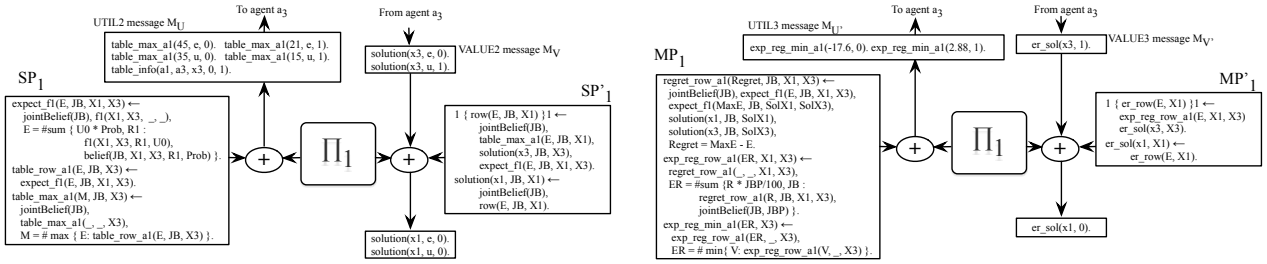


Figure 2: ASP Programs of Agent a_1 for Phase 2 (a) and Phase 3 (b) in Example 1

note, the number of blocks ℓ that can be scheduled in parallel also depends on the GPU architecture. In our settings $\ell = 14$. Thus, the number of utility vectors computed in parallel is bounded by $\ell \cdot h$.

7.2 ASP-ER-DPOP

Inspired by our ASP-based system that provides significant speedup and better scalability for standard DCOPs [15], we develop ASP-ER-DPOP, an ASP-based implementation of ER-DPOP. As we have previously noted, the ASP-based approach can capitalize on *(i)* the highly-expressive ASP language to more concisely represent constraint utilities as functions instead of explicitly enumerating them, and *(ii)* the highly-optimized ASP solvers to exploit problem structures such as pruning the search space based on hard constraints. We note that we did observe the same trend in our experiments (see Section 8).

We now present ASP-ER-DCOP. In this framework, each agent a_i is composed of the agent specification Π_i and its controller C_i.

Specifying an ER-DCOP: An ER-DCOP is encoded as a set of answer set programs $\{\Pi_i | a_i \in \mathcal{A}\}$. Each Π_i encodes information about: **(i)** the agent a_i, **(ii)** the variable x_i and its domain, **(iii)** the neighboring agents in the constraint graph, their variables, and their variables' domain, **(iv)** the constraints whose scope includes the variable x_i, **(v)** the belief space and its probabilistic model, and **(vi)** the beliefs for the constraints encoded in Π_i. Figure 3 shows the answer set program Π_1 for the ER-DCOP in Example 1.

agent(a1).	variable(x1, 0).
neighbor(a3).	variable(x3, 0..1).
scope(f1, x1).	scope(f1, x3).
f1(0, 0, 0, 0).	f1(0, 0, 1, 50).
f1(0, 1, 0, 0).	f1(0, 1, 1, 30).
jointBelief(e, 12).	jointBelief(u, 88).
belief(e, 0, 0, 0, 10).	belief(e, 0, 0, 1, 90).
belief(e, 0, 1, 0, 30).	belief(e, 0, 1, 0, 70).
belief(u, 0, 0, 0, 30).	belief(u, 0, 0, 1, 70).
belief(u, 0, 1, 0, 50).	belief(u, 0, 1, 0, 50).

Figure 3: Answer Set Program Π_1 for Example 1

Encoding Messages: Messages in Phases 2 and 3 (i.e., UTIL2, UTIL3, VALUE2, and VALUE3 messages) are encoded as ASP facts. For the UTIL2 and UTIL3 messages, each optimal utility, along with its corresponding *(i)* value assignment of the respective variables and *(ii)* joint belief (for only UTIL2 messages) is represented as a fact (see messages M_U and $M_{U'}$ in Figure 2). Upon receiving a UTIL2 or UTIL3 message, an agent is able to understand the mapping between the values in such facts to the corresponding variables via the information provided by another facts of the form *table_info* (see message M_U in Figure 2) included also in that message. Similarly, for VALUE2 and VALUE3 messages, a solution of a variable (along with its respective joint belief for VALUE2 messages) is encoded as a fact (see messages M_V and $M_{V'}$ in Figure 2).

Agent Controller: The agent controller C_i consists of a set of rules for communication (sending, receiving, and interpreting messages) and a set of rules for generating ASP programs that computes messages in Phases 2 and 3. Due to space constraints, we omit the detailed code of C_i, but we describe its functionality below.

ASP-ER-DPOP has the same phases as ER-DPOP. In Phase 1, the controllers C_i construct a pseudo-tree, and information about the parent, children, and pseudo-parents of a_i are added to Π_i. In Phase 2, after receiving UTIL2 (resp. VALUE2) messages as sets M_U (resp. M_V) of facts from children (resp. parent) agent(s), C_i generates a set SP_i (resp. SP_i') of rules (see Figure 2(a)), then computes an answer set of the program $\Pi_i \cup SP_i \cup M_U$ (resp. $\Pi_i \cup SP_i' \cup M_V$). Such an answer set is the encoded UTIL2 (resp. VALUE2) message of agent a_i to be sent to its parent (resp. children) agent(s). In Phase 3, after receiving UTIL3 (resp. VALUE3) messages as sets $M_{U'}$ (resp. $M_{V'}$) of facts from children (resp. parent) agent(s), C_i generates a set MP_i (resp. MP_i') of rules (see Figure 2(b)), then computes an answer set of the program $\Pi_i \cup MP_i \cup M_{U'}$ (resp. $\Pi_i \cup MP_i' \cup M_{V'}$). Such an answer set is the encoded UTIL3 (resp. VALUE3) message of agent a_i to sent to its parent (resp. children) agent(s). The original ER-DCOP's solution for variable owned by a_i is included in the VALUE3 message in this phase.

The controller is written in SICStus Prolog. We use CLASP [9] as our ASP solver and the SICStus *Linda* library for communication.

8. EXPERIMENTAL RESULTS

Since there are no other ER-DCOP solvers, a direct comparison with other systems is not feasible. We implemented another ER-DCOP solver, referred to as *Frodo-ER,* which differs from both ASP-ER-DPOP and GPU-ER-DPOP as follows: Frodo-ER generates a sequence of $|\mathcal{B}|$ DCOP problems M^{B^q} and solves them *sequentially* using DPOP in Phase 2. In contrast, ASP-ER-DPOP and GPU-ER-DPOP solves them *simultaneously*. Frodo-ER uses a publicly-available and widely-used implementation of DPOP [17].

In addition, to compare the actual regret of the solutions of the UR-DCOP and ER-DCOP models, we implemented *Iterative Constraint Generation Max-Sum* (ICG Max-Sum) [29], a UR-DCOP solver, using a publicly-available implementation of Max-Sum.[3]

We compare ASP-ER-DPOP (abbreviated to ASP-ER) and GPU-ER-DPOP (abbreviated to GPU-ER) against Frodo-ER. All

[3] https://code.google.com/p/jmaxsum/

Algorithm	$\lvert\mathcal{X}\rvert$				$\lvert D_i\rvert$					$\lvert\mathcal{B}\rvert$					p_1				
	8	13	18	23	4	6	8	10	12	5	30	50	70	90	0.4	0.5	0.6	0.7	0.8
ASP-ER	3.1	9.4	44.1	120.8	4.5	8.9	22.2	80.4	121.2	7.8	26.5	32.9	39.2	44.9	5.6	7.6	8.1	8.6	13.7
GPU-ER	0.1	0.2	–	–	0.1	0.1	1.2	4.8	15.4	0.2	0.5	0.8	1.1	2.4	0.1	0.1	0.2	0.7	1.7
Frodo-ER	0.3	61.1	–	–	1.8	33.6	143.2	–	–	42.2	86.5	113.7	117.1	131.5	1.1	4.8	45.1	118.5	196.1

(a)

Algorithm	p_2				
	0.4	0.5	0.6	0.7	0.8
ASP-ER	85.8	16.9	7.7	5.8	5.5
GPU-ER	0.2	0.2	0.3	0.2	0.3
Frodo-ER	42.6	42.3	39.6	41.3	41.2

(b)

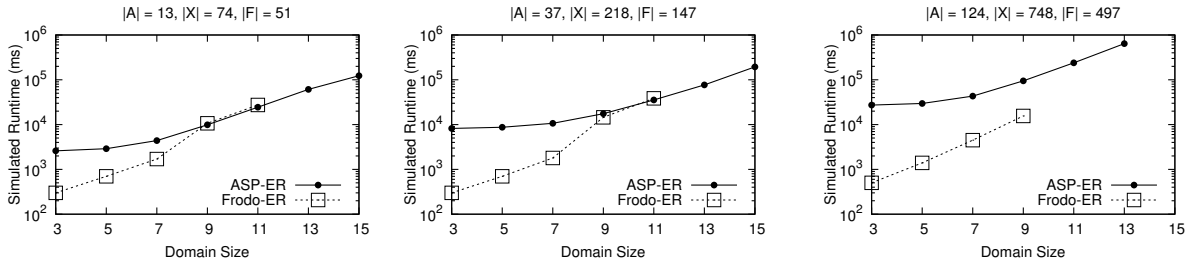
$\lvert\mathcal{B}\rvert$	Better	Worse	Equal	$\bar{V}_{UR-DCOP}$	$\bar{R}_{ER-DCOP}$
5	45 %	20 %	35 %	17.21	12.31
10	36 %	28 %	36 %	20.08	18.22
15	47 %	20 %	33 %	20.47	14.33

(c)

$\lvert\mathcal{B}\rvert$	ICG	ASP-ER
5	4.4	2.9
10	50.2	3.0
15	222.9	3.2

(d)

Table 2: Experimental Results of Random Graphs



(a) 13 Bus Topology (b) 37 Bus Topology (c) 124 Bus Topology

Figure 4: Experimental Results of Power Network Problems

algorithms use the same pseudo-tree for fair comparisons. We measure the runtime of the algorithms using the simulated runtime metric [27]. All experiments are performed on a Quadcore 2.2GHz machine with 8GB of memory. The GPU device adopted is a GeForce GTX TITAN with 14 multiprocessors, and a clock rate of 837 MHZ. If an algorithm fails to solve a problem, it is due to memory limitations. We conduct our experiments on random graphs [6] (for comparing ASP-ER and GPU-ER against Frodo-ER) and on comprehensive optimization problems in power networks [11] (for comparing ASP-ER against Frodo-ER).

Random Graphs: We create an n-node network, whose constraint density p_1 produces $\lfloor n\,(n-1)\,p_1 \rfloor$ edges in total [6]. In these experiments, we vary the number of variables $|\mathcal{X}|$; the domain size $|D_i|$; the constraint density p_1; the constraint tightness p_2, defined as the percentage of forbidden value combinations in a constraint; and the belief space size $|\mathcal{B}|$. For each experiment, we vary only one parameter and fix the others to their "default" values: $|\mathcal{A}| = |\mathcal{X}| = 13, |D_i| = 6, p_1 = p_2 = 0.6, |\mathcal{B}| = 5$. We set the timeout to 30 minutes. Tables 2(a) and 2(b) show the average runtimes (in seconds) for the solved instances (out of 30 instances). An algorithm fails to solve a configuration if it cannot solve at least 15 instances of that configuration.

We observe that ASP-ER is slower than Frodo-ER when the problem is less complex than the one with the default configuration. However, ASP-ER is able to solve more problems and is faster than Frodo-ER when the problem becomes more complex (i.e., increasing $|\mathcal{X}|, |D_i|, p_1, p_2$, or $|\mathcal{B}|$). The reason is that, unlike Frodo-ER agents, ASP-ER agents are able to prune a significant portion of the search space thanks to hard constraints. In detail, the search space does not include infeasible value combinations of the respective variables, and the size of the search space pruned increases with the complexity of the instances. The benefit of pruning is clearly seen at increasing p_2, i.e., while the runtimes of Frodo-ER are similar, those of ASP-ER decrease significantly as there are more hard constraints.

We also observe that GPU-ER is consistently faster than Frodo-ER and ASP-ER. Its use of SIMT-based parallelism is very effective at computing the independent utility vectors (or UTIL3 entries), which results in large speedup. However, as for Frodo-ER, it cannot scale to problems as large as those solved by ASP-ER (the settings with $|\mathcal{X}| \in \{18, 23\}$). This is due to the basic GPU-ER strategy adopted, which does not prune the search space based on hard constraints, resulting in analogous memory requirements as those of Frodo-ER.

We notice that, in Table 2(a), Frodo-ER reaches a timeout for $|D_i| > 8$ while both ASP-ER and GPU-ER can solve such experiments. This is due to that the size of the UTIL tables increases substantially when the domain size increases (Property 2), and ASP-ER and GPU-ER can scale to $|D_i| = 12$ (due to the effect of pruning in ASP-ER, and to the exploited thread-parallelism in GPU-ER), while Frodo-ER operates sequentially on such tables.

Power Network Problems: In this domain [11], each agent represents a node with consumption, generation, and transmission preferences, and a global cost function. Constraints include the power balance and no power loss principles, the generation and consumption limits, and the capacity of the power line between nodes. However, in reality, there is loss in power transmissions, and a stochastic constraint utility function is suited for representing this loss. The loss depends on how much power is transferred and on external factors (e.g., ambient temperature and the quality of power lines). If there are no losses, the objective is to minimize the global cost function. In the presence of losses, the objective is to minimize the regret of the achieved global cost function over different possible losses.

We use three network topologies, defined using the IEEE Distribution Test Feeder 2014 standards, and vary the domain size of the generation, load, and transmission variables of each agent from 3 to 15. We use three levels of losses ($|S_j| = 3$) corresponding to the loss of 10%, 30%, and 60% of the power transferred, and $|\mathcal{B}| = 3$. Figure 4 shows the runtime of ASP-ER and Frodo-ER

in milliseconds. The results in Figure 4 are consistent with those shown earlier.

Despite ER-DPOP having space-exponential message sizes, we believe that it is scalable in solving real application domains (e.g., in the power network domain with a 124 Bus topology, we can solve problems with $|\mathcal{X}| = 748$ and $|\mathcal{F}| = 497$). We also observe that ASP-ER has better scalability in the power network problems than in random graphs because the power network problems exhibit more structure than random graphs, and their underlying constraint graphs have small induced width and, thus, small separator sets.

Comparison Between UR-DCOP and ER-DCOP Models: In addition, we compare the solution quality of ER-DCOP and UR-DCOP models in terms of actual regret. After generating UR-DCOP instances, we augment a probability for each joint belief according to a normal distribution, as: $\mathcal{P}_{\mathcal{B}} \sim \mathcal{N}(\frac{|\mathcal{B}|}{2}, \frac{|\mathcal{B}|}{5})$. For each instance, the two solutions—one is with respect to the UR-DCOP model, and the other is with respect to the ER-DCOP model—are computed. The actual regrets of those solutions are achieved by *(i)* picking a joint belief in the belief space based on the distribution $\mathcal{P}_{\mathcal{B}}$, and *(ii)* calculating the actual regret of each of those solutions relative to the picked joint belief.

In this experiment, we generate 1000 instances for each configuration with $|\mathcal{A}| = |\mathcal{X}| = 8, |D_i| = 3, p_1 = 0.5, p_2 = 0$, and $|\mathcal{B}| \in \{5, 10, 15\}$. Table 2(c) compares the quality of ER-DCOP and UR-DCOP solutions over 1000 instances where "better", "worse", and "equal" columns indicate the number of times, in percentage, that the regret of the ER-DCOP solution is less than, greater than, and equal to the regret of the UR-DCOP solution of the same instance, respectively. It also reports the average regret of UR-DCOP ($\bar{V}_{UR-DCOP}$) and ER-DCOP ($\bar{R}_{ER-DCOP}$) solutions over those 1000 instances. Table 2(d) shows the average runtimes (in seconds) for ICG Max-Sum (abbreviated to ICG) and ASP-ER to solve those instances.[4] The ER-DCOP model yield solutions with smaller average regret than the UR-DCOP model when the probabilities of joint beliefs have a normal distribution. We also notice that ER-DCOP solutions can be worse than UR-DCOP solutions. This happens whenever a joint belief associated to a low probability, yet producing a high regret, actually happens. Furthermore, ASP-ER is consistently faster than ICG, due to the presence of cycles in the constraint graphs, which affect the convergence time of ICG (which is based on Max-Sum).

9. DISCUSSION

ER-DCOPs is closely related to UR-DCOPs [29] as both extend DCOPs to deal with stochastic utilities. It is therefore worth to discuss their main differences. The main difference lies in that ER-DCOP removes the independence assumption between the belief of a random variable and the values of decision variables. We observe that ER-DCOPs can be used to model UR-DCOP instances by considering, in every joint belief, the conditional probabilities of a random variable given different value assignments of the respective variables are identical (i.e., $b_j(r_j|x_{f_j}) = b_j(r_j|x'_{f_j})$ where x_{f_j} and x'_{f_j} are two arbitrary different value assignments of variables in $scp(f_j)$). In this sense, ER-DCOPs are more expressive than UR-DCOPs. The second main difference between UR-DCOP and ER-DCOP is the notion of a solution in each framework. While UR-DCOPs minimize the worst-case loss, ER-DCOPs aim to minimize the expected regret. Although we believe that each approach has its own merit, we note that ER-DCOPs might yield better results in term of actual regrets as shown in our experiments.

In general, abstract distributed optimization problems can be formulated as n-player coordination games. Thus, *Bayesian Games* (BGs) and *Potential Games* (PGs) may be used to model same scenarios as ER-DCOP. However, for BGs, the common prior belief is independent from the joint actions of the respective players (e.g., Harsanyi's [13] and Aumann's [2] models). Thus, as beliefs of random variables can be dependent to the decision variables in ER-DCOP, representing ER-DCOPs using BGs is not a straightforward matter. Moreover, for PGs, key differences between DCOPs and PGs are as follows: *(i)* DCOPs assume that the agents are cooperative while agents may be competitive in PGs; *(ii)* DCOPs aim at finding a (global) optimal solution for the potential function, while PGs aim at finding an equilibrium outcome, which correspond to a local optima of the potential function; and *(iii)* DCOPs require a distributed solution approach while PGs do not have such a strict requirement (though solution approaches like best response can also be thought of as distributed approaches).

Finally, we observe that researchers have also used *Graphical Models* (GMs) to model conditional independence/dependencies between variables in combinatorial optimization problems (e.g., [21]). GMs and ER-DCOPs are different as: *(i)* ER-DCOP's random variables exhibit different probabilities for each joint belief, while in typical GMs, this cannot be straightforwardly applied, and *(ii)* GMs are typically used to capture the conditional dependence between random variables, while ER-DCOPs represent conditional dependence between decision variables that are controlled by agents and random variables that are beyond the direct control of agents.

10. CONCLUSIONS

In this paper, we proposed ER-DCOPs to model DCOPs with uncertainty in constraint utilities. Differently from another approaches, it allows to represent the beliefs about exogenous factors, which possibly depend on the decision variables, and it focuses on minimizing the expected regret. To solve ER-DCOPs, we proposed a distributed algorithm, called ER-DPOP, which is complete and requires a linear number of messages in the number of agents in the problem. In addition, we presented two implementations for ER-DPOP: One which harnesses the parallelism offered by GPUs to speed up the process of solving ER-DCOPs, and another which uses ASP and exploits logic programming-based inference rules to prune the solution search space. Such approaches take advantage from two orthogonal means of exploiting the ER-DCOP structure. Our experimental evaluation shows that ER-DCOP solutions outperform corresponding UR-DCOP solutions in terms of the actual regret, and that both ER-DPOP implementations outperform a straightforward repeated application of state-of-the-art DCOP solver (i.e., DPOP) in terms of better scalability and runtime.

ACKNOWLEDGMENT

This research is partially supported by NSF grants 1345232 and 0947465. 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] J. Atlas and K. Decker. Coordination for uncertain outcomes using distributed neighbor exchange. In *Proceedings of the*

[4] Although ICG Max-Sum does not solve UR-DCOPs optimally [29], we use it as it is the only known UR-DCOP solver. The UR-DCOP results in Table 2(c) are computed using an optimal centralized solver.

International Conference on Autonomous Agents and Multiagent Systems (AAMAS), pages 1047–1054, 2010.

[2] R. J. Aumann, M. Maschler, and R. E. Stearns. *Repeated games with incomplete information*. MIT press, 1995.

[3] C. Baral. *Knowledge Representation, reasoning, and declarative problem solving with Answer sets*. Cambridge University Press, Cambridge, MA, 2003.

[4] F. Bistaffa, A. Farinelli, and N. Bombieri. Optimising memory management for belief propagation in junction trees using gpgpus. In *Proceedings of the International Conference on Parallel and Distributed Systems (ICPADS)*, pages 526–533, 2014.

[5] S. Citrigno, T. Eiter, W. Faber, G. Gottlob, C. Koch, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. The dlv system: Model generator and application frontends. In *Proceedings of the Workshop on Logic Programming*, pages 128–137, 1997.

[6] P. Erdös and A. Rényi. On random graphs i. *Publicationes Mathematicae Debrecen*, 6:290, 1959.

[7] 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.

[8] F. Fioretto, T. Le, E. Pontelli, W. Yeoh, and T. C. Son. Exploiting GPUs in solving (distributed) constraint optimization problems with dynamic programming. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP)*, 2015.

[9] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. clasp: A conflict-driven answer set solver. In *Proceedings of the International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pages 260–265, 2007.

[10] M. Gelfond and V. Lifschitz. Logic programs with classical negation. In *Proceedings of the Interntional Conference on Logic Programming (ICLP)*, pages 579–597, 1990.

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

[12] Y. Hamadi, C. Bessière, and J. Quinqueton. Distributed intelligent backtracking. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pages 219–223, 1998.

[13] J. Harsanyi. Games with incomplete information played by "Bayesian" players, I-III: Part I. the basic model. *Management Science*, pages 1804–1817, 2004.

[14] K. Hoang, F. Fioretto, P. Hou, M. Yokoo, W. Yeoh, and R. Zivan. Proactive dynamic distributed constraint optimization. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2016.

[15] T. Le, T. C. Son, E. Pontelli, and W. Yeoh. Solving distributed constraint optimization problems with logic programming. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2015.

[16] T. Léauté and B. Faltings. Distributed constraint optimization under stochastic uncertainty. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 68–73, 2011.

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

[18] R. Maheswaran, M. Tambe, E. Bowring, J. Pearce, and P. Varakantham. Taking DCOP to the real world: Efficient complete solutions for distributed event scheduling. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 310–317, 2004.

[19] R. Mailler and V. Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 438–445, 2004.

[20] V. Marek and M. Truszczyński. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-year Perspective*, pages 375–398, 1999.

[21] R. Marinescu and R. Dechter. AND/OR branch-and-bound search for combinatorial optimization in graphical models. *Artificial Intelligence*, 173(16-17):1457–1491, 2009.

[22] 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.

[23] D. T. Nguyen, W. Yeoh, and H. C. Lau. Stochastic dominance in stochastic DCOPs for risk-sensitive applications. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 257–264, 2012.

[24] I. Niemelä. Logic programming with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3–4):241–273, 1999.

[25] 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.

[26] J. Sanders and E. Kandrot. *CUDA by Example. An Introduction to General-Purpose GPU Programming*. Addison Wesley, 2010.

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

[28] S. Ueda, A. Iwasaki, and M. Yokoo. Coalition structure generation based on distributed constraint optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 197–203, 2010.

[29] F. Wu and N. Jennings. Regret-based multi-agent coordination with uncertain task rewards. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 1492–1499, 2014.

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

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

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