Multi-Variable Agent Decomposition for DCOPs

Ferdinando Fioretto
Department of Computer Science
New Mexico State University
Las Cruces, NM 88003, U.S.
ffiorett@cs.nmsu.edu

William Yeoh
Department of Computer Science
New Mexico State University
Las Cruces, NM 88003, U.S.
wyeoh@cs.nmsu.edu

Enrico Pontelli
Department of Computer Science
New Mexico State University
Las Cruces, NM 88003, U.S.
epontell@cs.nmsu.edu

Abstract

The application of DCOP models to large problems faces two main limitations: *(i) Modeling limitations*, as each agent can handle only a single variable of the problem; and *(ii) Resolution limitations*, as current approaches do not exploit the local problem structure within each agent. This paper proposes a novel *Multi-Variable Agent (MVA)* DCOP decomposition technique, which: *(i)* Exploits the co-locality of each agent's variables, allowing us to adopt efficient centralized techniques within each agent; *(ii)* Enables the use of hierarchical parallel models and proposes the use of GPUs; and *(iii)* Reduces the amount of computation and communication required in several classes of DCOP algorithms.

Introduction

The *Distributed Constraint Optimization Problem (DCOP)* model is an elegant formalism to describe cooperative multi-agent problems and has been applied to solve coordination and resource allocation problems (Maheswaran et al. 2004; Léauté and Faltings 2011; Zivan et al. 2015; Miller, Ramchurn, and Rogers 2012). However, the applicability of DCOPs to more general classes of problems, where agents control multiple variables, faces two important challenges: *(i) Modeling limitations*, due to the common assumption that each agent controls only one variable; and *(ii) Resolution limitations*, which prevent agents from exploiting local problem structure. To cope with such restrictions, reformulation techniques are commonly used to transform a general DCOP into one where each agent controls exclusively one variable. There are two commonly used reformulation techniques (Burke and Brown 2006; Yokoo 2001): *(i) Compilation*, where each agent creates a new *pseudo-variable*, whose domain is the Cartesian product of the domains of all variables of the agent; and *(ii) Decomposition*, where each agent creates a *pseudo-agent* for each of its variables. While both techniques are relatively simple, they can be inefficient. In compilation, the memory requirement for each agent grows exponentially with the number of variables that it controls. In decomposition, the DCOP algorithms will treat two pseudo-agents as independent entities, resulting in unnecessary computation and communication costs.

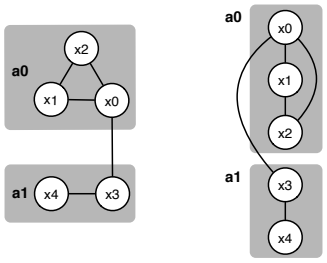
Copyright © 2016, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

In this paper, we propose a novel decomposition method, called *Multi-Variable Agent (MVA)* DCOP decomposition, to overcome these limitations; MVA decomposition enables a separation between the agents' *local subproblems*, which can be solved independently using centralized solvers, and the DCOP *global problem*, which requires coordination of the agents. The decomposition does not lead to any additional privacy loss. The MVA framework enables the use of different centralized and distributed solvers in a hierarchical and parallel way. We explore the use of two centralized solvers, *Depth-First Branch and Bound (DF-BnB)* and *Gibbs*, to solve the agents' local subproblems. For the global coordination, we consider three representative DCOP algorithms: *Asynchronous Forward Bounding (AFB)* (Gershman, Meisels, and Zivan 2009), as an example of a search algorithm, *Distributed Pseudo-tree Optimization Procedure (DPOP)* (Petcu and Faltings 2005), as an example of an inference algorithm, and *Distributed Gibbs (D-Gibbs)* (Nguyen, Yeoh, and Lau 2013), as an example of a sampling algorithm. Our work is motivated by the increasing interest in modeling complex distributed multi-agent applications, where each agent needs to solve complex subproblems (Kim and Lesser 2013; Giuliani et al. 2014; Amigoni, Castelletti, and Giuliani 2015).

This paper advances the state of the art by: **(1)** defining a separation between the distributed DCOP resolution process and the centralized agent subproblem resolution process, enabling the use of efficient centralized solvers to solve independent subproblems; **(2)** enabling the use of hierarchical parallel models during DCOP resolution and proposing a general model to exploit *Graphics Processing Units (GPUs)* for DCOP resolution; **(3)** proposing search-based and sampling-based GPU-accelerated local solvers to speed up the DCOP resolution process; and **(4)** providing empirical evidence of the efficiency in computation and communication of MVA decomposition with respect to existing DCOP reformulation techniques.

Background

DCOP: A DCOP (Modi et al. 2005; Yeoh and Yokoo 2012) is described 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., $x_i \in D_i$); $\mathcal{F} = \{f_1, \ldots, f_m\}$ is a set of *cost functions*, where $f_i : \prod_{x_j \in \mathbf{x}^i} D_i \rightarrow \mathbb{N} \cup \{\infty\}$



for $i < j$

x_i	x_j	Costs
0	0	7
0	1	10
1	0	2
1	1	3

(a) Constraint Graph (b) Linear Ordering (c) Cost Function

Figure 1: Example DCOP.

Sender	Message Type	Receiver	Message Content
a_0^0	[CPA_MSG]	a_0^1	[0 - - - -] (0)
a_0^0	[FB_CPA]	$a_0^0, a_0^2, a_1^3, a_1^4$	[0 - - - -] (0)
a_0^1	[FB_ESTIMATE]	a_0^0	(7)
a_0^1	[CPA_MSG]	a_0^2	[0 0 - - -] (7)
a_0^0	[FB_CPA]	a_0^2, a_1^3, a_1^4	[0 0 - - -] (7)
a_0^2	[FB_ESTIMATE]	a_0^0	(9)
a_0^2	[FB_ESTIMATE]	a_0^1	(14)
a_0^2	[CPA_MSG]	a_1^3	[0 0 0 - -] (21)
a_0^0	[FB_CPA]	a_1^3, a_1^4	[0 0 0 - -] (21)
a_1^3	[FB_ESTIMATE]	a_0^0, a_0^1, a_0^2	(7)
a_1^3	[CPA_MSG]	a_1^4	[0 0 0 0 -] (28)
a_1^3	[FB_CPA]	a_1^4	[0 0 0 0 -] (28)
a_1^3	[FB_ESTIMATE]	a_0^0, a_0^1, a_0^2	(2)
a_1^4	[NEW_SOLUTION]	$a_0^0, a_0^1, a_0^2, a_0^3, a_1^3$	[0 0 0 0 0] (35)
...		...	...

Figure 2: Partial Trace of AFB after Decomposition.

and $\mathbf{x}^i \subseteq \mathcal{X}$ is the set of the variables relevant to f_i; $\mathcal{A} = \{a_1, \ldots, a_p\}$ is a set of *agents*; and $\alpha : \mathcal{X} \rightarrow \mathcal{A}$ maps each variable to one agent. Given a DCOP P, $G_P = (\mathcal{X}, E_\mathcal{F})$ is the *constraint graph*[1] of P, where $\{x, y\} \in E_\mathcal{F}$ *iff* $\exists f_i \in \mathcal{F}$ such that $\{x, y\} \subseteq \mathbf{x}^i$. 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 cost $F(\sigma) = \sum_{f_i \in \mathcal{F}, \mathbf{x}^i \subseteq X_\sigma} f_i(\sigma)$ is the sum of the costs across all the applicable cost functions in σ. A solution σ is *complete* if $X_\sigma = \mathcal{X}$. The goal is to find an optimal complete solution $\mathbf{x}^* = \text{argmin}_\mathbf{x} F(\mathbf{x})$. Fig. 1(a) shows the constraint graph of a simple DCOP with 2 agents a_0 and a_1, where each variable can be assigned the values 0 or 1. Fig. 1(c) shows the cost functions.

In the following algorithms, we assume that each agent controls exactly one variable, which is the situation after reformulating the DCOP via compilation or decomposition.

AFB: *Asynchronous Forward Bounding (AFB)* (Gershman, Meisels, and Zivan 2009) is a complete *search algorithm*, where each agent maintains a solution (called a CPA) and its corresponding cost. The agents first order themselves according to a linear ordering. The head agent assigns a value to its variable, calculates its (unary) cost functions, and sends this CPA to its successor agent, using a *CPA* message. Each agent also sends its CPA, in a *FB_CPA* message, to each of its descendants (i.e., all lower priority agents according to the linear ordering) to request the recipients to compute a cost estimate for that CPA. The cost estimates are returned in *FB_ESTIMATE* messages, which are used to prune the search space. This process continues down the linear ordering until a tail agent finds a complete solution, which is sent to all the agents. When the head agent exhausts all its value assignments, it broadcasts a termination message.

Fig. 2 shows a snippet of the messages sent by the agents in our example DCOP after a decomposition reformulation, where agent a_i^j is the pseudo-agent that controls variable x_j of agent a_i. We assume that the pseudo-agents are ordered as in Fig. 1(b). AFB requires 98 messages between pseudo-agents controlled by different agents (i.e., actual agent-to-agent messages) and 60 messages between pseudo-agents controlled by the same agent (i.e., internal agent messages).

DPOP: The *Distributed Pseudo-tree Optimization Procedure (DPOP)* (Petcu and Faltings 2005) is a complete *inference algorithm* composed of three phases. During *Pseudo-tree Generation,* agents coordinate to build a pseudo-tree (Hamadi, Bessière, and Quinqueton 1998). During *UTIL Propagation,* each agent, starting from the pseudo-tree leafs, computes the optimal sum of costs in its subtree for each value combination of variables in its separator.[2] It does so by adding the costs of its functions with the variables in its separator and the costs in the UTIL messages received from its children, and projecting out its own variables by optimizing over them. During *VALUE Propagation,* each agent, starting from the pseudo-tree root, determines the optimal value for its variables. The root agent does so by choosing the values of its variables from its UTIL computations.

D-Gibbs: *Gibbs* (Geman and Geman 1984) is a Markov chain Monte Carlo algorithm that can be used to approximate joint probability distributions. It generates a Markov chain of samples, each of which is correlated with previous samples. It does so by iteratively sampling one variable from the conditional probability distribution, assuming that all the other variables take their previously sampled values. Once the joint probability distribution is found, one can identify a complete solution with the maximum likelihood. The Gibbs sampling algorithm can be used to solve a DCOP by mapping the DCOP to a maximum a-posteriori estimation problem (Nguyen, Yeoh, and Lau 2013). The *Distributed Gibbs (D-Gibbs)* algorithm extends Gibbs by tailoring it to solve DCOPs in a *decentralized manner*.

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*, and have access to several memory levels, each with different properties in terms of speed and capacity. GPUs support *Single-Instruction Multiple-Thread (SIMT)* processing. In SIMT, the same instruction is executed by different threads, while handling different data.

MVA Decomposition

Definition 1 (Local and Interface Variables) *For each agent* $a_i \in \mathcal{A}$, $L_i = \{x_j \in \mathcal{X} \mid \alpha(x_j) = a_i\}$ *is the set of its* local *variables.* $B_i = \{x_j \in L_i \mid \exists x_k \in \mathcal{X} \wedge \exists f_s \in \mathcal{F} :$

[1]Technically, G_P is a hyper-graph; we refer to it as a constraint graph to simplify the discussion.

[2]The separator of x_i contains all ancestors of x_i in the pseudo-tree that are connected to x_i or one of its descendants.

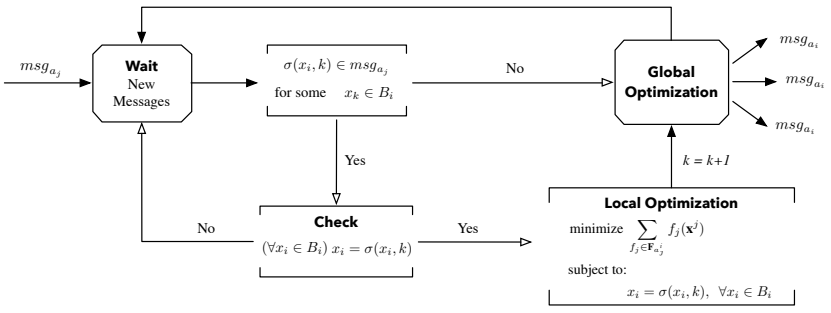


Figure 3: MVA Execution Flow Chart.

Sender	Message Type	Receiver	Message Content
a_0	[CPA_MSG]	a_1	[0 1 0 - -] (19)
a_0	[FB_CPA]	a_1	[0 1 0 - -] (19)
a_1	[FB_ESTIMATE]	a_0	(9)
a_1	[NEW_SOLUTION]	a_0	[0 1 0 0 0] (33)
a_1	[NEW_SOLUTION]	a_0	[0 1 0 1 0] (31)
a_1	[CPA_MSG]	a_0	[0 1 0 - -]
a_0	[CPA_MSG]	a_1	[1 1 0 - -] (7)
a_0	[FB_CPA]	a_1	[1 1 0 - -] (7)
a_1	[FB_ESTIMATE]	a_0	(4)
a_1	[NEW_SOLUTION]	a_0	[1 1 0 0 0] (16)
a_1	[NEW_SOLUTION]	a_0	[1 1 0 1 0] (12)
a_1	[CPA_MSG]	a_0	[1 1 0 - -]
a_0	[TERMINATE]	a_1	

Figure 4: Complete trace of MVA-AFB.

$\alpha(x_k) \neq a_i \wedge \{x_j, x_k\} \subseteq \mathbf{x}^s\}$ *is the set of its* interface *variables.*

Definition 2 (Local Constraint Graph) *For each agent* $a_i \in \mathcal{A}$*, its* local constraint graph $G_i = (L_i, E_{\mathcal{F}_i})$ *is a subgraph of the constraint graph, where* $\mathcal{F}_i = \{f_j \in \mathcal{F} | \mathbf{x}^j \subseteq L_i\}$.

In Fig. 1(a), $L_0 = \{x_0, x_1, x_2\}$, $L_1 = \{x_3, x_4\}$, $B_0 = \{x_0\}$, $B_1 = \{x_3\}$. We use $\sigma(x_i, k) \in D_i$ to denote the k^{th} value assignment to variable x_i. Next, we describe how the MVA decomposition is applied to solve a general DCOP, exploiting the combination of decentralized DCOP algorithms, off-the-shelf centralized solvers, and their GPU parallel versions.

Description of the MVA Decomposition

In the MVA decomposition, a DCOP problem P is decomposed into $|\mathcal{A}|$ subproblems $P_i = (L_i, B_i, \mathcal{F}_i)$, where P_i is associated to agent $a_i \in \mathcal{A}$. In addition to the decomposed problem P_i, each agent receives: **(a)** the *global* DCOP algorithm P_G, which is common to all agents in the problem and defines the agent's coordination protocol and the behavior associated to the receipt of a message, and **(b)** the *local* algorithm P_L, which can differ between agents and is used to solve the agent's subproblem. Fig. 3 shows a flow chart illustrating the four conceptual phases in the execution of the MVA framework for each agent a_i:

Phase 1—Wait: The agent waits for a message to arrive. If the received message results in a new value assignment $\sigma(x_r, k)$ for a interface variable x_r of B_i, then the agent will proceed to Phase 2. If not, it will proceed to Phase 4.

Phase 2—Check: The agent checks if it has performed a complete new assignment for all its interface variables, indexed with $k \in \mathbb{N}$, which establishes an enumeration of the interface variables' assignments. If it has, then the agent will proceed to Phase 3, otherwise it will return to Phase 1.

Phase 3—Local Optimization: When a complete assignment is given, the agent passes the control to a local solver, which solves the following problem:

$$\min \sum_{f_j \in \mathcal{F}_i} f_j(\mathbf{x}^j) \quad \text{subject to: } x_r = \sigma(x_r, k) \quad \forall x_r \in B_i$$

Solving this problem results in finding the best assignment for the agent's local variables given the particular assignment for its interface variables. Notice that the local solver P_L is independent from the DCOP structure and it can be customized based on the agent's local requirements. For example, agents can use GPU-optimized solvers, if they have access to GPUs, or use off-the-shelf CP, MIP, or ASP solvers. Once the agent solves its subproblem, it proceeds to Phase 4.

x_0	x_3	Costs
0	0	7
0	1	10
1	0	2
1	1	3

(a) Constraint Table of Interface Variables

x_0	Best Local Solutions	Costs
0	$[x_1 = 1, x_2 = 0]$	19
1	$[x_1 = 1, x_2 = 1]$	7

(b) a_0's MVA_TABLE

x_3	Best Local Solutions	Costs
0	$[x_4 = 0]$	7
1	$[x_4 = 0]$	2

(c) a_1's MVA_TABLE

Figure 5: MVA_TABLES.

Phase 4—Global Optimization: The agent processes the new assignment as established by the DCOP algorithm P_G, executes the necessary communications, and returns to Phase 1. The agents can execute these phases independently of one another because they exploit the co-locality of their local variables without any additional privacy loss, which is a fundamental aspect in DCOPs (Greenstadt, Pearce, and Tambe 2006).

In addition, the local optimization process can operate on $m \geq 1$ combinations of value assignments of the interface variables, before passing control to the next phase. This is the case when the agent explores m different assignments for its interface variables in Phases 2 and 3. These operations are performed by storing the best local solution and their corresponding costs in a cost table of size m, which we call MVA_TABLE, and can be visualized as a *cache memory*. The minimum value of m depends on the choice of the global DCOP algorithm P_G. For example, for common search-based algorithms such as AFB, it is 1, while for common inference-based algorithms such as DPOP, it is exponential in size of the separator set. Figs. 5(b) and 5(c) show the MVA_TABLES of the two agents in our example DCOP with $m = 2$. Using the MVA decomposition, each agent computes only the necessary rows of the table on demand. Fig. 4 shows the messages sent by agents in our example DCOP with the MVA framework. In total, AFB requires only 13 messages (compared to 98 messages with the decomposi-

tion reformulation) between agents. Additionally, since the local subproblem of each agent is solved using a local search engine, the agents do not need to send any internal agent messages (compared to 60 messages with the decomposition reformulation).

Hierarchical Parallel Local Optimization

We use DFBnB and Gibbs as representative complete and incomplete algorithms for the local optimization process within each agent. DFBnB is correct and complete. Thus, it does not affect the correctness and completeness of the global complete DCOP algorithms used (Theorem 1). In addition, it allows us to exploit the problem structure through bound propagations. Gibbs provides quality guarantees and can be used in combination with D-Gibbs to provide good approximated solutions (Theorem 4).

The use of hierarchical parallel solutions is motivated by the observation that the search for the best local solution for each row of the MVA_TABLE is independent of the search for another row and, as such, they can be performed in parallel. This observation finds a natural fit for SIMT processing and, therefore, in addition to the CPU versions of DFBnB and Gibbs, we provide their GPU counterparts. The use of GPUs allows us to speed up the local optimization process and, consequently, reduces the overall DCOP solving time.

We now describe the GPU versions and leave out the simpler CPU versions due to space constraints. Without loss of generality, in the following description, we assume that all variables $x_i \in L_i$ have the same domains, denoted to as D_i.

Gibbs: At a high level, it performs these parallelizations:

- Due to the stochastic nature of the Gibbs process, a set of R sampling processes, each with a different seed, can be carried out in parallel. The best cost and corresponding solution across all runs is thus selected. Each parallel process is executed by a *group of GPU blocks*.
- Computing the costs in each row of the MVA_TABLE U_i is independent of the computation of the costs in the other rows. Thus, we compute the sample costs using (at most) $|D_i|^{|B_i|}$ parallel *groups of GPU threads*, one for each row.
- The computation of the conditional probabilities for each possible value of the variables involved in the Gibbs process is independent of the computation of the probability for the other values. Thus, we compute the probabilities using $|D_i|$ parallel *threads*, one for each possible value.

The procedure GPU-GIBBS is the core of the local sampling algorithm and corresponds to one of the R independent sampling processes executed in parallel. It executes T sampling trials for the subset of non-interface local variables $L_i \setminus B_i$ of agent a_i. Its pseudocode, executed by each thread in a *group*, is shown in lines 1-17. We use the symbols $\leftarrow$ and $\leftleftarrows_k$ to denote sequential and parallel operations, respectively. The former is performed by a single thread in each group of threads in a block, and is often associated to assignment operations performed in shared memory—a memory area shared among all threads in a block—while the latter is performed in parallel by k threads of the group.

The function first stores the following structures in shared memory: The local constraint graph G_i; the array P, which

Procedure GPU-Gibbs(L_i, B_i, D_i, G_i, T, R)

1 $\langle G_i, P, Z, \text{sample}, \text{sample}^* \rangle \leftarrow$ ASSIGNSHAREDMEM()
2 $r_{id} \leftarrow$ the thread row index of MVA_TABLE$_i$
3 $d_{id} \leftarrow$ the thread value index of D_i
4 sample$[0..|B_i|-1] \leftleftarrows_{|B_i|}$ MVA_TABLE$_i[r_{id}][0 \ldots |B_i|-1]$
5 sample$[|B_i| \ldots |L_i|-1] \leftleftarrows_{|L_i|-|B_i|}$ RANDOM$(0 \ldots |D_i|-1)$
6 cost $\leftarrow \sum_{f_j \in G_i} f_j(\text{sample}\,|_{S_j})$
7 $\langle \text{sample}^*, \text{cost}^* \rangle \leftarrow \langle \text{sample}, \text{cost} \rangle$
8 **for** $t = 1$ **to** T **do**
9 **for** $k = |B_i|$ **to** $|L_i|-1$ **do**
10 $P[d_{id}] \leftleftarrows_{|D_i|} \exp\left[\sum_{f_j \in G_i} \frac{1}{f_j(d_{id} \cup \text{sample}|_{S_j})}\right]$
11 $Z \leftarrow \sum_{j=0}^{|D_i|-1} P[j]$
12 $P[d_{id}] \leftleftarrows_{|D_k|} P[d_{id}] \cdot \frac{1}{Z}$
13 sample$[k] \leftarrow$ SAMPLE(P)
14 cost $\leftarrow \sum_{f_j \in \mathcal{F}_i} f_j(\text{sample}\,|_{S_j})$
15 **if** *cost* $<$ *cost** **then**
16 $\langle \text{sample}^*, \text{cost}^* \rangle \leftarrow \langle \text{sample}, \text{cost} \rangle$
17 $\langle V_i^R[r_{id}], U_i^R[r_{id}] \rangle \leftarrow \langle \text{sample}^*, \text{cost}^* \rangle$

is used to store the probabilities for each value of the non-interface local variables; the normalization constant Z, which is used to normalize the probabilities; the array sample, which is used to store the current sample of value assignments for all local variables; and the array sample*, which is the best sample found so far (line 1). The thread then identifies the row index r_{id} in the MVA_TABLE$_i$ and the value d_{id} in D_i that it is in charge of sampling (lines 2-3). The function determines its initial sample, where the interface variables receive their respective values from row r_{id} in MVA_TABLE$_i$, while the other variables receive random values (lines 4-5). Finally, it calculates the cost for that sample (line 6) and stores the initial sample and cost as the best sample and cost found so far (line 7).

The procedure performs T sampling trials, iterating through all the non-interface local variables (lines 8-16). For each variable, it *(i)* computes in parallel the probability $P[d_{id}]$ of each value d_{id} for the variable x_k according to the equation:

$$P[d_{id}] = P(x_k = d_{id} \mid x_l \in L_i \setminus \{x_k\}) = \frac{1}{Z} \exp \sum_{f_j \in \mathcal{F}_i} \frac{1}{f_j(\text{sample}\,|_{S_j})}$$

where sample $|_{S_j}$ is the set of value assignments for the variables in the scope S_j of function f_j and Z is the normalization constant (lines 10-12); *(ii)* samples the value for that variable according to the probabilities (line 13); *(iii)* computes the cost of the updated sample (line 14); and *(iv)* updates the best sample and cost if necessary (lines 15-16). After all its sampling trials, it stores the best sample and cost in the r_{id}-th row in V_i^R and U_i^R, respectively (line 17).

DFBnB: The GPU version of DFBnB operates by performing a complete DFBnB search on the local non-interface variables $x_i \in L_i \setminus B_i$. Unlike Gibbs, the sequential nature of the DFBnB operations makes such process less amenable to multi-thread parallelism within each GPU block. However,

we can exploit the multitude of GPU threads to perform a parallel exploration of the space of value assignments to the interface variables. Thus, we delegate each row of the MVA-table to a thread, which executes a complete search through the solution space of the non-local interface variables, reporting the best solution with its associated cost.

Theoretical Results

Theorem 1 *The MVA framework with P_G and P_L is correct and complete* iff *P_G and P_L are both correct and complete.*

Proof Sketch: Let us prove the forward direction for soundness. Assume that the combination P_G and P_L with the MVA framework is correct and that it finds an optimal complete solution $\mathcal{V}^*$. Now assume that P_G is not correct. Then, an agent a_i might not explore the combination of values $\langle v^i_1, \ldots, v^i_{b_i}\rangle \in \mathcal{V}^*$ for its interface variables $x^i_j \in B_i$ $(j = 1, \ldots, b_i)$, which contradicts the assumption that the MVA framework finds the optimal complete solution $\mathcal{V}^*$. Therefore, P_G is correct. The argument for P_L is similar to that of P_G. Assume that P_L is not correct. Thus, an agent a_i might not explore the combination of values $\langle v^i_{b_i+1}, \ldots, v^i_{l_i}\rangle$ for its non-interface local variables $x^i_j \in L_i \backslash B_i$, $(j = b_i+1, \ldots, l_i)$, which contradicts the assumption that the MVA framework finds the optimal complete solution $\mathcal{V}^*$. Thus, P_L is correct. The reverse direction and completeness' proof are similar. □

Theorem 2 *The additional space requirement for the MVA framework (compared to resolution of the decomposed problem using P_G) is $O(M \cdot l)$, where M is the maximum number of rows of* MVA_TABLE *needed on demand by each agent a_i and $l = \max_{i \in \{j \mid a_j \in \mathcal{A}\}} |L_i|$.*

Theorem 3 *The message requirements of MVA framework have the same order-complexity as the P_G procedure.*

The above refers to the worst case message complexity, where all local variables are interface variables. In problems with non-interface local variables and non-empty subproblems, the message requirements of search algorithms are often significantly smaller in the MVA framework ($O(d^{|\mathcal{A}|})$) than with the decomposition technique ($O(d^{|\mathcal{X}|})$). This is shown in our example in Figs. 2 and 4.

Theorem 4 *The sampling processes of both MVA-DG (i.e., the MVA framework with D-Gibbs as P_G and Gibbs as P_L) and D-Gibbs converge to the same solution.*

Related Work

The concept of solving localized constrained subproblems within a distributed framework has been previously explored in (Distributed) CSPs (Armstrong and Durfee 1997; Yokoo and Hirayama 1998). The main differences are that existing work has focused on the development of specific algorithms rather than a general framework and they have not been generalized to DCOPs. To the best of our knowledge, the only algorithm able to deal with agent subproblems without the use of decomposition techniques is Adopt-MVA (Davin and Modi 2006), an extension of Adopt (Modi et al. 2005). The MVA decomposition presented here allows the integration of *any* global DCOP coordination algorithm and *any* local optimization procedure. As such, it subsumes AdoptMVA. Another line of work that solves subproblems in a centralized manner can be found in the work on *partially-centralized* DCOP algorithms (Petcu, Faltings, and Mailler 2007; Vinyals et al. 2010). The main difference with our approach is that the subproblems defined by the MVA decomposition are confined within the agents local variables, and therefore are privacy-preserving. In contrast, subproblems solved by the partially-centralized algorithms are defined over variables that can be owned by different agents, which is undesirable in several application domains.

Experimental Results

We evaluate our MVA decomposition with three global DCOP algorithms (AFB, DPOP, and D-Gibbs) and two local centralized solvers (DFBnB and Gibbs) implemented on CPUs and GPUs. In addition to the *lazy* version (*MVA-lazy*) described in this paper, where agents solve their local subproblems on demand during the resolution process, we also implemented an *eager* version (*MVA-eager*), where agents populate their complete MVA table in a pre-processing step. We compare them against the Compilation and Decomposition pre-processing techniques on random graph and radar coordination instances. In our version of Compilation, agents retain exclusively the solutions of the local problem, whose search space is explored via DFBnB. All experiments are performed on an Intel i7 Quadcore 3.4GHz machine with 16GB of RAM. The GPU solvers are run on an GeForce GTX TITAN with 14 multiprocessors, 2688 cores, and a clock rate of 837MHz. We report runtime measured using the simulated time (Sultanik, Modi, and Regli 2007) metric as well as the number of external agent-to-agent messages and internal agent messages. We impose a timeout (t.o.) of 600sec of simulated time and a memory limit of 2GB. Results report the average over 50 runs, and are statistically significant with p-values < 0.001.[3]

Random Graph Instances: We create an n-node network, whose local constraint graphs density p^l_1 produces $\lfloor |L_i|(|L_i|-1)p^l_1 \rfloor$ edges among the local variables of each agent a_i, and whose (global) density p^g_1 produces $\lfloor b(b-1)p^g_1 \rfloor$ edges among interface non-local variables, where b is the total number of interface variables of the problem. Fig. 6 shows the results, where AFB and DPOP use DFBnB as local solver, while D-Gibbs uses Gibbs. We conducted two experiments, where we set $|D_i| = 4$, $p^l_1 = 0.6$, and the constraint tightness $p_2 = 0.4$. For the first experiment, we set $p^g_1 = 0.4$ and vary the number of agents $|\mathcal{A}|$ (Fig. 6(top)). For the second experiment, we set $|\mathcal{A}| = 4$ and vary the ratio $|B_i|/|L_i|$ (Fig. 6(bottom)). In this experiment, we build a subgraph for each agent with $p^l_1 = 0.6$ and create as many inter-agent constraints as necessary to reach the desired ratio. In both experiments, starting from the highest function arity, we transform each clique involving k variables to a k-ary function with costs randomly chosen from $[1, 1000]$.

- Unlike Decomposition, MVA and Compilation do not need internal agent communication since agent subproblems are solved locally within each agent.

[3] t-test performed with null hypothesis: MVA-lazy-decomposed algorithms are faster than non-MVA-decomposed ones.

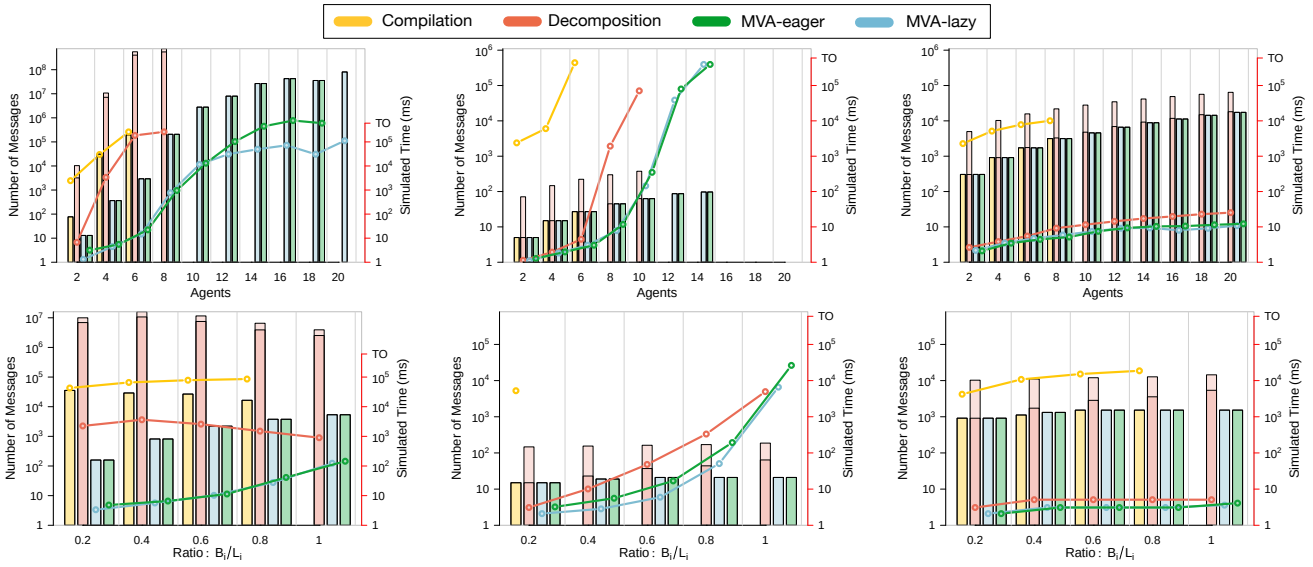


Figure 6: Random Graph Instances with CPUs: AFB-DFBnB (left), DPOP-DFBnB (middle), and D-Gibbs-Gibbs (right). Dark (light) bars indicate the number of external (internal) messages, and lines indicate runtime, all in logarithmic scale.

- The number of external messages required by each framework is similar for DPOP and D-Gibbs because they are linear in the number of agents, and in the number of samples as well for D-Gibbs. Both of these factors are independent of the number of local variables.
- AFB on MVA requires up to one order of magnitude fewer external messages compared to Compilation, and several orders of magnitude fewer compared to Decomposition. The reason is that AFB agents broadcast messages to request for cost estimates and announce complete solutions. These broadcasts occur more regularly with Decomposition and Compilation than with the MVA decomposition.
- The number of messages and runtimes of both MVA versions are similar to each other, indicating that agents in both versions ultimately construct the entire MVA table.
- The additional space requirement for AFB and D-Gibbs on MVA is negligible. In both algorithms, the agents require a single row for the MVA_TABLE (i.e., $M = 1$ in Theorem 2). In contrast, MVA-decomposed DPOP agents have to cache a number of the MVA_TABLE's rows that is exponential in the size of their interface variables.
- The runtimes of the algorithms on MVA tend to their runtimes with Decomposition as the ratio of interface variables increases. When all variables are interface variables ($|B_i|/|L_i| = 1$), AFB and D-Gibbs are faster on MVA than with Decomposition, as the agents can solve their local subproblems quicker with centralized algorithms; DPOP is slower on MVA due to overhead.
- In general, all the algorithms are faster on MVA than Decomposition and Compilation.

Table 1 illustrates the runtimes obtained by parallelizing DFBnB and Gibbs on GPUs, using DPOP as the global algorithm. We fix $|\mathcal{A}| = 4$ and vary $|L_i|$ and $|D_i|$. The speedup is at least one order of magnitude in most configurations.

	D_i and L_i size:	6	12	18	24	30	
DPOP-Gibbs	CPU	45	211	447	1271	2275	$\|L_i\|$
	GPU	3	4	6	17	36	
	speedup	15	52.7	74.5	74.8	63.2	
	CPU	44	315	836	1946	5853	$\|D_i\|$
	GPU	8	22	80	204	432	
	speedup	5.5	14.3	10.45	9.5	13.6	
DPOP-DFBnB	CPU	17000	t.o.	t.o.	t.o.	t.o.	$\|L_i\|$
	GPU	2000	$2.28{\cdot}10^7$	t.o.	t.o.	t.o.	
	speedup	8.5	N/A	–	–	–	
	CPU	3400	61000	129000	437000	988000	$\|D_i\|$
	GPU	450	5000	11200	39300	94100	
	speedup	7.5	12.2	11.5	11.2	10.5	

simulated time (ms) and CPU vs. GPU speedup

Table 1: Random Graph Instances with CPUs and GPUs.

Radar Coordination Instances: This problem models a set of radars, which collect real-time data on the location and importance of atmospheric phenomena, and a set of controllers, which can operate on a subset of the radars (Kim and Lesser 2013). Each phenomenon is characterized by size and weight (i.e., importance). Radars have limited sensing ranges, which determine their scanning regions. The goal is to find a radar configuration that maximizes the utility associated with the scanned phenomena. Controllers are modeled as agents whose variables they control are radars. The domain of each variable represents the scanning regions of a radar. The utilities (which, in our case, are modeled as costs by taking their negative values) are functions involving all radars that may detect a given phenomenon.

In our experiments, radars are equally spaced onto a grid, and each controller coordinates 16 radars (arranged in a 4×4 grid). Radars have four possible scanning directions, and phenomena are randomly generated across the grid until the underling constraint graph results connected. Table 2 tabulates the results. The first two rows report the grid configurations and the number of agents. We omit the results for Compilation because it failed to solve any of the instances. We

configuration	8x4	8x8	12x8	16x8	20x8
# agents	2	4	6	8	10
MVA-lazy	11	213	3186	33212	42090
MVA-eager	732	10391	27549	77489	82637
Decomposition	213	108818	178431	t.o.	t.o.

AFB-DFBnB simulated time (ms)

MVA-lazy	844	30312	225761	478955	t.o.
MVA-eager	823	30396	225538	477697	t.o.
Decomposition	61978	t.o.	t.o.	t.o.	t.o.

DPOP-DFBnB simulated time (ms)

Table 2: Radar Coordination Instances.

also omit results for D-Gibbs as it cannot handle hard constraints. Similar to random graph instances, the MVA-based algorithms are faster than Decomposition-based algorithms. Unlike random graph instances, AFB with MVA-lazy is up to one order of magnitude faster than MVA-eager. AFB can successfully prune portions of the search space using the hard constraints. As a result, AFB agents with MVA-lazy, in contrast to MVA-eager, do not need to construct the entire MVA table. DPOP with both MVA-lazy and MVA-eager have similar runtimes, as DPOP does not perform any pruning being based on dynamic programming.

Conclusions

In this paper, we proposed the MVA decomposition for DCOPs with multi-variable agents. This decomposition provides a hierarchical parallel model that defines a clear separation between the distributed agent coordination and the centralized agent subproblem resolution, while preserving agent privacy. This separation allows the use of efficient centralized solvers to solve agent subproblems as well as the use of potentially different solvers for different agents, each designed to exploit domain-specific properties. Experimental results show that the use of MVA speeds up several DCOP algorithms by up to several orders of magnitude and reduces their communication requirements with respect to existing techniques. In the future, we plan to investigate the application of propagation schemes (e.g., as in (Fioretto et al. 2014)) to further reduce agent-to-agent communication.

Acknowledgments

This research is partially supported by NSF grants 1345232, 0947465, and 1540168. 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

Amigoni, F.; Castelletti, A.; and Giuliani, M. 2015. Modeling the management of water resources systems using multi-objective dcops. In *AAMAS*, 821–829.

Armstrong, A. A., and Durfee, E. H. 1997. Dynamic prioritization of complex agents in distributed constraint satisfaction problems. In *AAAI*, 822.

Burke, D., and Brown, K. 2006. Efficiently handling complex local problems in distributed constraint optimisation. In *ECAI*, 701–702.

Davin, J., and Modi, P. 2006. Hierarchical variable ordering for multiagent agreement problems. In *AAMAS*, 1433–1435.

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

Geman, S., and Geman, D. 1984. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Trans. PAMI* 6(6):721–741.

Gershman, A.; Meisels, A.; and Zivan, R. 2009. Asynchronous Forward-Bounding for distributed COPs. *JAIR* 34:61–88.

Giuliani, M.; Castelletti, A.; Amigoni, F.; and Cai, X. 2014. Multiagent systems and distributed constraint reasoning for regulatory mechanism design in water management. *Journal of Water Resources Planning and Management*.

Greenstadt, R.; Pearce, J.; and Tambe, M. 2006. Analysis of privacy loss in DCOP algorithms. In *AAAI*, 647–653.

Hamadi, Y.; Bessière, C.; and Quinqueton, J. 1998. Distributed intelligent backtracking. In *ECAI*, 219–223.

Kim, Y., and Lesser, V. 2013. Improved max-sum algorithm for dcop with n-ary constraints. In *AAMAS*, 191–198.

Léauté, T., and Faltings, B. 2011. Coordinating logistics operations with privacy guarantees. In *IJCAI*, 2482–2487.

Maheswaran, R.; Tambe, M.; Bowring, E.; Pearce, J.; and Varakantham, P. 2004. Taking DCOP to the real world: Efficient complete solutions for distributed event scheduling. In *AAMAS*, 310–317.

Miller, S.; Ramchurn, S.; and Rogers, A. 2012. Optimal decentralised dispatch of embedded generation in the smart grid. In *AAMAS*, 281–288.

Modi, P.; Shen, W.-M.; Tambe, M.; and Yokoo, M. 2005. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *AIJ* 161(1–2):149–180.

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

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

Petcu, A.; Faltings, B.; and Mailler, R. 2007. PC-DPOP: A new partial centralization algorithm for distributed optimization. In *IJCAI*, 167–172.

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

Vinyals, M.; Pujol, M.; Rodriguez-Aguilarhas, J.; and Cerquides, J. 2010. Divide-and-coordinate: DCOPs by agreement. In *AAMAS*, 149–156.

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

Yokoo, M., and Hirayama, K. 1998. Distributed constraint satisfaction algorithm for complex local problems. In *ICMAS*, 372–379.

Yokoo, M., ed. 2001. *Distributed Constraint Satisfaction: Foundation of Cooperation in Multi-agent Systems*. Springer.

Zivan, R.; Yedidsion, H.; Okamoto, S.; Glinton, R.; and Sycara, K. 2015. Distributed constraint optimization for teams of mobile sensing agents. *JAAMAS* 29(3):495–536.