Proving the equivalence of Action-GDL and DPOP

Technical Report RR-IIIA-2008-04

Abstract

Distributed Constraint Optimization Problems (DCOPs) are a general framework that can model a large class of Multi-agent Coordination Problems (MCPs). We propose a novel message-passing algorithm, the so-called Action-GDL, as an extension to the Generalized Distributive Law algorithm to efficiently solve MCPs. We show the generality of Action-GDL by proving that it has DPOP, the low-complexity, state-of-the-art algorithm to solve DCOPs, as a particular case. Finally, we provide empirical evidences to illustrate that Action-GDL can outperform DPOP in terms of computation, communication and parallelism.

1 Introduction

Multi-agent Coordination Problems (MCPs), also called distributed multi-agent decision making problems, are a class of problems in MAS focusing on how to coordinate agents' actions in order to yield a global desired behaviour for the MAS. Distributed Constraint Optimization Problems (DCOPs) are an extension of Constraint Optimization Problems (COPs) that can model a large class of MCPs [5].

State-of-the-art complete algorithms to solve DCOPs adopt two main approaches: search and dynamic programming. Search algorithms, like ADOPT [3], require linear-size messages, but an exponential number of messages. Dynamic programming algorithms, represented by the DPOP algorithm and its extensions [6], only require a linear number of messages, but their complexity lies on the message size, which may be very large.

In this paper, we formulate a new algorithm, the so-called Action-GDL, that takes inspiration on the GDL algorithm [1], extending and applying it to MCPs. GDL is a general message-passing algorithm that exploits the way a global function factors into a combination of local functions generalizing a large family of well-known algorithms (e.g. Viterbi's, Pearl's belief propagation, or Shafer-Shenoy algorithms). Therefore, GDL has a wide range of applicability. In our case, the rationale to apply (and extend) GDL is that a DCOP requires the maximization of a global function resulting from the combination of local functions.

In order to ensure optimality and convergence, Action-GDL must arrange, likewise GDL, the global function to optimise into a distributed junction tree (DJT) structure [2]. For this purpose, we introduce the Distributed Junction Tree Generator (DJTG) algorithm at the pre-processing phase of Action-GDL. DJTG is a message-passing algorithm, based on the one formulated in [4], that allows agents to *distributedly* compile a DJT keeping any distribution of relations among agents. Therefore, DJTG creates a DJT that adapts to the underlying distributed nature of the problem.

Thereafter, we show how DPOP, the low complexity, state-of-the-art algorithm to solve DCOPs is a particular case of Action-GDL. To do so, we: (1) prove that given a pseudotree there is a DJT such that Action-GDL mimics DPOP execution; and (2) show how the DJTG algorithm can compile, given the pseudotree, this DJT distributedly. Moreover, we provide empirical evidence to show that the generality of Action-GDL can be exploited to outperform DPOP in terms of amount of computation, communication, and parallelism.

This paper is structured as follows. Firstly, we provide a definition of DCOP (section 2) and the notation we will use through out this paper (section 3). Section 4 introduces Action-GDL, as well as its connection with GDL. Then, in section 6, we prove that Action-GDL extends DPOP (described in section 5). Section 7 introduces the DJTG algorithm and section 8 shows how the DJTG algorithm can distributedly compile a DJT such that Action-GDL mimics DPOP execution. Next, section 9 provides some evidences of how to exploit the generality of Action-GDL by showing how it can outperform DPOP when solving the same DCOPs. Finally, section 10 draws some conclusions.

2 DCOP

Here we introduce the Distributed Constraint Optimization Problem (DCOP), an extension to the Constraint Optimization Problem (COP). DCOPs can model a large class of MCPs [5]. These problems consist of a set of variables, each one taking on a value out of a finite discrete domain. Each constraint in this context has a set of variables as input specifying a cost, namely a relation. The goal of a COP algorithm is to assign values to these variables so that the total utility is maximized. A

DCOP [6, 3] is an extension to a COP where variables are distributed among agents. Let $\mathcal{X} = \{x_1, \ldots, x_n\}$ be a set of variables over domains $\mathcal{D}_1, \ldots, \mathcal{D}_n$. Let $r : \mathcal{D}_r \rightarrow \Re^+$, where $\mathcal{D}_r$ is the projection of the joint domain space $\mathcal{D} = \mathcal{D}_1 \times \ldots \times \mathcal{D}_n$ over variables in the domain of r, be a *utility relation* that assigns a utility value to each combination of values of its domain variables. Formally, a DCOP is a tuple $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{R}, \alpha \rangle$ where: $\mathcal{A}$ is a set of agents; $\mathcal{X}$ is a set of variables; $\mathcal{D}_n$ is the joint domain space for all variables; $\mathcal{R} = \{r_1, \ldots, r_p\}$ is a set of utility relations; and $\alpha : \mathcal{X} \rightarrow \mathcal{A}$ maps each variable to some agent.

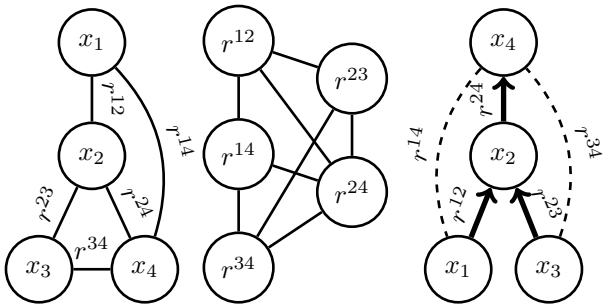
The objective function f is described as an aggregation (typically addition) over the set of relations. Formally:

$$f(d) = \sum_{i=1}^{p} r_i(d_{r_i}) \quad (1)$$

where d is an element of the joint domain space $\mathcal{D}$ and d_{r_i} is an element of $\mathcal{D}_{r_i}$. Solving a DCOP amounts to choosing values for the variables in $\mathcal{X}$ such that the objective function is maximized (minimized).

In a DCOP each agent receives knowledge about all relations that involve its variable(s) in addition to their domains. In general, DCOP algorithms do not impose any restriction regarding the number of variables that can be assigned to each agent or the arity of the relations. However, although all algorithms we refer to in this paper can deal with n-ary relations, for the sake of simplicity we mainly restrict them to unary and binary relations. Therefore, we will refer to unary relations involving variable $x_i \in \mathcal{X}$ as r^i, and to binary relations involving variables $x_i, x_j \in \mathcal{X}$ as r^{ij}.

A DCOP with binary relations is typically represented with its primal-constraint graph, whose vertices stand for variables and whose edges stand for binary relations, as shown by the example depicted in figure 1 (a). DCOPs can also be represented with its dual-constraint graph, whose vertices stand for relations and whose edges link relations that share some variable in their domains, as shown by the example depicted in figure 1 (b).



(a) Primal graph (b) Dual graph (c) Pseudotree

Figure 1: Different DCOP representations

3 Notation

Next we provide the definitions of a collection of functions and operators that we shall employ throughout the rest of this paper. Henceforth, given some variable set $X \subseteq \mathcal{X}$, $\mathcal{D}_X$ will stand for the joint domain space of variables in X. Furthermore, for exemplary purposes we assume that each domain $\mathcal{D}_i$ contains constant values $c_i^1, \ldots, c_i^{n_i}$.

Definition 1 (DV) *The domain variable function DV returns the domain variables of a given set of relations. Ex:* $DV(\{r^{31}\}) = \{x_3, x_1\}$, $DV(\{r^{31}, r^{34}\}) = \{x_1, x_3, x_4\}$.

Definition 2 (Complementary variables) *Given a set of variables X and a relation r, we define the complementary variables of X by r as the set of variables in r that are not in X . Formally,* $\bar{X}^r = DV(r) \setminus X$.

Definition 3 (Utility message) *A message from agent a_i to agent a_j is a utility message over $X \subseteq \mathcal{X}$, if the information sent is a utility relation over $\mathcal{D}_X$. Henceforth, we shall denote that utility relation as μ_{ij}.*

Definition 4 (Assignment) *Given a set of agents $X \in \mathcal{X}$, an assignment σ over X sets a value to each variable $x_k \in X$ and sets free the remaining variables.*
Ex: $X = \{x_1, x_2, x_3\}$, *σ an assignment over X,* $\sigma(x_1) = c_1^2$, $\sigma(x_3) = c_3^5$, *x_2 is free in σ*

Definition 5 (Value message) *A message from agent a_i to agent a_j is a value message over $X \subseteq \mathcal{X}$ if the information sent is an assignment over X. Henceforth, we shall denote such assignment by σ_{ij}.*

The combination operator joins the knowledge represented by two relations into a single one by adding their values.

Definition 6 (Combination) *Let r, s be two relations and $\mathcal{D}_{r \otimes s} = \times_{x_k \in DV(\{r,s\})} \mathcal{D}_k$ be their joint domain space. The combination of r and s (noted $r \otimes s$) is a utility relation over $\mathcal{D}_{r \otimes s}$ such that $(r \otimes s)(d) = r(d_r) + s(d_s)$ for all $d \in \mathcal{D}_{r \otimes s}$, where $d_r \in \mathcal{D}_r$ and $d_s \in \mathcal{D}_s$ are the projections of d over the domains of relations r and s respectively.*
Ex: $(r^{13} \otimes r^{14})(c_1^2, c_3^5, c_4^1) = r^{13}(c_1^2, c_3^5) + r^{14}(c_1^2, c_4^1)$.

We can readily generalize the combination operator over a finite set of relations: $\bigotimes_{\{r_1,\ldots,r_m\}} = r_1 \otimes (r_2 \otimes \ldots (r_{m-1} \otimes r_m) \ldots)$.

The summarization operator sums up the utility that a relation contains over a set of variables. Thus, the summarization operator over a relation r and a set of variables Xassesses the r maximum utility for the variables in X.

Definition 7 (Summarization) *The summarization operator of relation r over a set of variables X is a utility relation over $\mathcal{D}_X$ such that* $(\bigoplus_X r)(d_X) = \max_{d_{\bar{X}^r} \in \mathcal{D}_{\bar{X}^r}} r(d_X, d_{\bar{X}^r})$.
Ex: $(\bigoplus_{\{x_3\}} r^{13})(c_3^2) = \max_{k \in \mathcal{D}_1} r^{13}(k, c_3^2)$.

Notice that we can employ the summarization operator by specifying the variables to eliminate from a relation as follows $\bigoplus_{\setminus X} r = \bigoplus_{\bar{X}^r} r = \bigoplus_{DV(r) \setminus X} r$.

Definition 8 (Restriction) *The restriction of a relation r by an assignment σ over X is a utility relation over $\mathcal{D}_{\bar{X}^r}$ such that* $(\bigtriangledown_\sigma r)(d_{\bar{X}^r}) = r(d_X, d_{\bar{X}^r})$ *where $d_X \in \mathcal{D}_X$ contains the values set by σ to the variables in X.*
Ex: $X = \{x_3\}$, $\sigma(x_3) = c_3^2$, $(\bigtriangledown_\sigma r^{13})(c_1^1) = r^{13}(c_1^1, c_3^2)$.

It is straightforward to generalize the restriction operator over a finite set of assignments: $\bigtriangledown_{\{\sigma_1, \ldots, \sigma_m\}} r = \bigtriangledown_{\sigma_1} (\ldots (\bigtriangledown_{\sigma_m} r) \ldots)$.

4 The Action-GDL Algorithm

Notice that our objective is to solve MCPs, and in particular DCOP problems, namely to maximize some objective function, such as the one in equation 1, when the problem is distributed among agents. For this purpose, we propose to extend GDL [1].

4.1 The GDL Algorithm

GDL [1] is a general message-passing algorithm that exploits the way a global function factors into a combination of local functions to compute the objective function in an efficient manner. In order to ensure optimality and convergence, GDL arranges the objective function to assess in a *junction tree structure* (JT)[2].

Definition 9 *A **junction tree** (JT) is a tree of cliques that can be represented as a tuple $\langle \mathcal{X}, \mathcal{C}, \mathcal{S}, \Psi \rangle$ where: $\mathcal{X} = \{x_1, \ldots, x_n\}$ is a set of variables; $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$ is a set of cliques such that each clique $\mathcal{C}_i \subseteq \mathcal{X}$; $\mathcal{S}$ is a set of separators, where each separator is an edge between two cliques containing the intersection of the cliques[1]; and $\Psi = \{\psi_1, \ldots, \psi_m\}$ is a set of potentials, where potential ψ_i is a function assigned to clique $\mathcal{C}_i$ with domain $\Delta_i \subseteq \mathcal{X}$. Furthermore, the following properties must hold:*

- ***Single-connectedness.*** *Separators create exactly one path between each pair of cliques.*
- ***Covering.*** *Each potential domain is a subset of the clique to which it is assigned, namely $\Delta_i \subseteq \mathcal{C}_i$.*
- ***Running intersection.*** *If a variable x_i is in two cliques $\mathcal{C}_i$ and $\mathcal{C}_j$, then it must also be in all cliques on the (unique) path between $\mathcal{C}_i$ and $\mathcal{C}_j$.*

Likewise variables in DCOP, we assume that the variables in a junction tree are defined over domains $\mathcal{D}_1, \ldots, \mathcal{D}_n$. Moreover, $\mathcal{D}_{\mathcal{C}_i}$ stands for clique $\mathcal{C}_i$ domain space, namely the joint domain space of the variables in clique $\mathcal{C}_i$.

GDL defines a message-passing phase for cliques to exchange information about their variables. Once the message-passing phase is over, each clique can compute its state, namely its variables states. GDL is defined over two binary operations [1] that in our case, since we are concerned with the problem of maximizing an utility function, correspond to the addition and the maximization (the max-sum GDL).

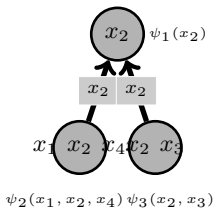


Figure 2: JT

#	Message/local knowledge ($\widehat{\mathcal{K}}$)
1.	$\mu_{21}(x_2) = max_{\{x_1, x_4\}} \psi_2(x_1, x_2, x_4)$
2.	$\mu_{31}(x_2) = max_{\{x_3\}} \psi_3(x_2, x_3)$
3.	$\widehat{\mathcal{K}}_1(x_2) = \psi_1(x_2) + \mu_{21}(x_2) + \mu_{31}(x_2)$
4.	$\mu_{12}(x_2) = \psi_1(x_2) + \mu_{31}(x_2)$
5.	$\mu_{13}(x_2) = \psi_1(x_2) + \mu_{21}(x_2)$
6.	$\widehat{\mathcal{K}}_2(x_1, x_2, x_4) = \psi_2(x_1, x_2, x_4) + \mu_{12}(x_2)$
7.	$\widehat{\mathcal{K}}_3(x_2, x_3) = \psi_3(x_2, x_3) + \mu_{13}(x_2)$

Table 1: Trace of GDL

To illustrate the way the max-sum GDL operates, consider the following example. Say that our goal is to distributedly maximize some objective function $f(x_1, x_2, x_3, x_4) = \psi_1(x_2) + \psi_2(x_1, x_2, x_4) + \psi_3(x_2, x_3)$, whose factors ($\psi_1$, ψ_2 and ψ_3) are arranged in the directed JT of figure 2. Nodes in the figure stand for cliques and edges for separators. Table 1 shows a trace of GDL when scheduled fully serially[2]. In that schedule, since the JT is directed, messages are send in two different message-passing phases: (i) one up the tree in which each clique sends a message to its clique parent when, for the first time, it has received messages from all of its children; (ii) one down the tree so that each clique sends a message to its children when it receives a message from its parent. At round 1, clique $\mathcal{C}_2 = \{x_1, x_2, x_4\}$ sends a message μ_{21} to clique $\mathcal{C}_1 = \{x_2\}$ with the values of its local function, ψ_2, after 'filtering out' dependence on all variables but those common to $\mathcal{C}_2$ and $\mathcal{C}_1$ (namely variables which are not in their separator). At round 3, after clique $\mathcal{C}_1$ receives the values of its children's local functions for its variable x_2, it combines those values into $\widehat{\mathcal{K}}_1$. $\widehat{\mathcal{K}}_1$ is a function that stands for $\mathcal{C}_1$ knowledge over its variables, namely x_2. At that point, since $\mathcal{C}_1$ has received messages from all its neighbors, $\widehat{\mathcal{K}}_1$ contains all the information related to x_2. At rounds 4 and 5, clique $\mathcal{C}_1$ sends messages to its

[1]Formally, a separator s^{ij} between clique $\mathcal{C}_i$ and $\mathcal{C}_j$ is defined as $s^{ij} = \mathcal{C}_i \cap \mathcal{C}_j$.

[2]Fully parallel and hybrid schedules are also feasible as discussed in [1].

children that contain the combination of its local function, ψ_1, with other children messages. Thus, $\mathcal{C}_2$ receives a message from $\mathcal{C}_1$ that contains the potential ψ_1 combined with μ_{31}. Then it can compute $\widehat{\mathcal{K}}_2$(round 6).

Finally, notice that GDL goes far beyond the example above. In fact, it generalizes a large family of well-known algorithms (e.g. the min-sum or Viterbi's, the max-product or Pearl's belief propagation, or Shafer-Shenoy). Moreover there is a wealth of theoretical and analytical results in approximated iterative versions of GDL when applied to different structures rather than JTs [1].

4.2 Extending GDL to solve DCOPs

In this section we introduce Action-GDL, a novel message-passing algorithm that takes inspiration on the GDL algorithm [1], extending and applying it to MCPs. Then, in section 6, we will show the generality of Action-GDL by proving that DPOP, the low complexity, state-of-the-art algorithm is a particular case of this algorithm.

Recall that our goal is to solve MCPs represented as DCOPs. Therefore, the capability of computing any objective function, as provided by GDL, is not enough. We need to go one step beyond GDL to allow a group of agents make a joint decision (regarding their variables' values) that maximizes any objective function. For this purpose, we propose an extension to GDL, the so-called Action-GDL algorithm.

Consider a MCP setting. As explained above in GDL, when a clique has received messages from all its neighbors, it has all information related to its variables and it can compute its objective function. In MCP, clique variables are decision variables and computing a clique objective function stands for assigning values to these decisions. Therefore, when a clique infers their state, there is no need to propagate more information related to its variables since we can propagate directly the decisions taken. In other words, there is no need to propagate messages containing relations down the tree because all a child requires to make a decision is its father's decisions (variables' assignments). It implies that in MCPs, when the first message-passing phase of GDL, up to the tree, is over, the second message-passing phase of GDL, down the tree, is no longer necessary. Thus, we require a second message-passing phase for cliques to exchange *decisions* down the tree, which is precisely the extension that Action-GDL introduces. Henceforth, we shall refer to the first message-passing phase as *utility propagation*, and to the second one as *value propagation*. It is relevant to notice that the value propagation phase ensures that whenever multiple optimal joint decisions are feasible, cliques converge to the very same joint decision, namely to the very same solution of a DCOP.

Table 2 displays a trace of Action-GDL over the JT in figure 2. Steps 1, 2 and 3 are the same as GDL. However, at step 4 the root clique assesses the optimal value for x_2 ($x_2^* = c_2^*$) and propagates this value down the tree through value messages to cliques $\mathcal{C}_2$ and $\mathcal{C}_3$ (steps 5 and 6). At steps 7-8 and 9-10, $\mathcal{C}_2$ and $\mathcal{C}_3$ assess the values of x_1, x_4 and x_3, respectively, using its parent decision value c_2^*.

#.Messages/local knowledge $\widehat{\mathcal{K}}$	#.Messages/local knowledge $\widehat{\mathcal{K}}$
1. $\mu_{21}(x_2)$=$\max_{\{x_1,x_4\}} \psi_2(x_1,x_2,x_4)$	6. $\sigma_{13}(x_2)$=c_2^*
2. $\mu_{31}(x_2)$=$\max_{\{x_3\}} \psi_3(x_2,x_3)$	7. $\widehat{\mathcal{K}}_2(x_1,x_4)$=$\psi_2(x_1,\sigma_{12},x_4)$
3. $\widehat{\mathcal{K}}_1(x_2)$=$\widehat{\mathcal{K}}_1(x_2)+\mu_{31}+\mu_{21}$	8. (c_1^*,c_4^*)=$arg\ \max_{d\in\mathcal{D}_2\times\mathcal{D}_4} \widehat{\mathcal{K}}_2(d)$
4. c_2^*=$arg\ \max_{d\in\mathcal{D}_2} \widehat{\mathcal{K}}_1(d)$	9. $\widehat{\mathcal{K}}_3(x_3)$=$\psi_3(\sigma_{13},x_3)$
5. $\sigma_{12}(x_2)$=c_2^*	10.c_3^*=$arg\ \max_{d\in\mathcal{D}_3} \widehat{\mathcal{K}}_3(d)$

Table 2: Trace of Action-GDL

Another major difference between Action-GDL and GDL has to do with the way they solve a problem. GDL runs over a JT as formalised by definition 9. Hence, all cliques are considered to be located in a single agent, which is in charge of running GDL. Under Action-GDL a set of (distributed) agents cooperatively exchange information to reach the joint decision that solves a MCP. That is, unlike GDL, Action-GDL supports the distribution of the problem, namely of a JT, so that cliques can be distributed to different agents. This is accomplished by running Action-GDL over a *distributed junction tree* (DJT):

Definition 10 *A **distributed junction tree** (DJT) is a tuple $\langle \mathcal{A}, \mathcal{X}, \mathcal{C}, \mathcal{S}, \Psi, \beta \rangle$ where $\langle \mathcal{X}, \mathcal{C}, \mathcal{S}, \Psi \rangle$ is a JT; $\mathcal{A} = \{a_1, \ldots, a_m\}$ is a set of agents; and $\beta : \mathcal{C} \rightarrow \mathcal{A}$ maps each clique to one agent.*

An example of DJT is given in figure 3(c). Likewise JTs, circles stand for cliques, and edges for separators, both labelled with their variables' indices. Moreover, the background of each clique is labelled with the agent's index to which is associated. However We also define $\widehat{N}(\mathcal{C}_i) = \{\mathcal{C}_j | s^{ij} \in \mathcal{S}\}$ as a function that returns the cliques connected by a separator to clique $\mathcal{C}_i$, namely its neighboring cliques. Since a DJT is a tree of cliques it can also be defined as a directed tree. In a directed DJT we define two additional relationships among cliques: $\widehat{P}(\mathcal{C}_i)$, which returns the parent of $\mathcal{C}_i$; and $\widehat{Ch}(\mathcal{C}_i)$, which returns the children of $\mathcal{C}_i$.

Algorithm 1 outlines Action-GDL. Given a DJT $\langle \mathcal{A}, \mathcal{X}, \mathcal{C}, \mathcal{S}, \Psi, \beta \rangle$, each agent $a \in \mathcal{A}$ involved in Action-GDL, only needs to know the subset of the DJT that involves the cliques it has assigned. Hence, every agent is assumed to start knowing a tuple $\langle \widehat{P}(\mathcal{C}_i), \widehat{Ch}(\mathcal{C}_i), \psi_i, \widehat{S}(\mathcal{C}_i), \beta_i \rangle$ for each one of its cliques $\mathcal{C}_i$, where $\widehat{\mathcal{S}}$ returns a clique's separators ($\widehat{\mathcal{S}}(\mathcal{C}_i) = \{s^{ik} | s^{ik} \in \mathcal{S}\}$), and β_i returns the agents assigned to clique $\mathcal{C}_i$'s parent or children.

During the utility propagation phase (lines 1-10), agents exchange utility messages. The initial knowledge for each clique is its potential (line 2). For each clique, its agent waits until receiving a utility message from each of its children cliques (lines 3-4). These messages contain a utility relation over the variables shared by both

cliques (their separator) and are sent by agents assigned to the children cliques. Every time that the agent receives a new utility message, it incorporates it (by using the combination operator) to its local knowledge (line 5). After combining utility messages from all the children of a clique, if that clique has a parent (line 7), its agent summarizes that part of its local knowledge that is of interest to the clique's parent (by means of a utility relation over its separator) and sends it to the agent associated to the parent's clique (line 9). Let $\widehat{\mathcal{M}}_i = \bigotimes_{\mathcal{C}_j \in \widehat{Ch}(\mathcal{C}_i)} \mu_{ji}(s^{ji})$ be the combination of all utility messages received for clique $\mathcal{C}_i$. Then the utility message that the agent assigned to $\mathcal{C}_i$ sends for its parent clique $\mathcal{C}_p = \widehat{P}(\mathcal{C}_i)$ (to the agent $\beta_i(\mathcal{C}_p)$) is the summary over its clique variables of the combination of its potential and the utility messages received from its children. Formally:

$$\mu_{ip}(s^{ip}) = \bigoplus_{s^{ip}} [\psi_i \otimes \widehat{\mathcal{M}}_i] \tag{2}$$

During the value propagation phase (lines 11-21), agents compute the optimal values for their variables and exchange value messages, namely decisions. Given a clique, its agent waits until receiving a value message (containing value assignments) for all variables in common (in the separator) with its clique parent (line 12-13). At that point, the agent has received all the knowledge, in form of utility (from children) and value (from the parent) messages, required for computing the objective function related to its clique variables. The agent restricts its knowledge by incorporating the already inferred decisions (line 14) and computes the optimal values for the rest of its clique variables (line 16). Formally, given clique $\mathcal{C}_i$, after its agent receives a value message $\sigma_{ji}(s^{ij})$ for the variables in its separator with $\mathcal{C}_j$, it can update its local knowledge as follows:

$$\widehat{\mathcal{K}}_i = \bigtriangledown_{\sigma_{ji}(s^{ij})} [\psi_i \otimes \widehat{\mathcal{M}}_i] \tag{3}$$

Hence, $\mathcal{C}_i$'s agent can assess the optimal assignments for unassigned variables by maximising its local knowledge as follows:

$$d^* = arg \max_{d \in \mathcal{D}_{\bar{\mathcal{C}}_i}} \widehat{\mathcal{K}}_i(d) \tag{4}$$

where $\bar{\mathcal{C}}_i = DV(\widehat{\mathcal{K}}) = \mathcal{C}_i \backslash s^{ij}$ is the set of clique variables excluding the variables in the separator between $\mathcal{C}_i$ and its parent $\mathcal{C}_j$.

Once an agent knows the variables' values for one of its cliques, it can propagate them down the tree (lines 18-21). Notice however that it only propagates variable assignments that are required by its children cliques, namely assignments for variables in their separator.

5 DPOP

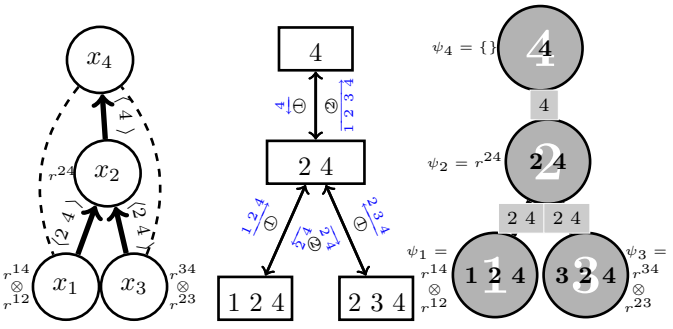
DPOP[6] is an optimal state-of-the-art dynammic programming algorithm to solve DCOPs. DPOP ar-

Algorithm 1 Action-GDL($\langle \mathcal{A}, \mathcal{X}, \mathcal{C}, \mathcal{S}, \Psi, \beta \rangle$)

Each agent $a \in \mathcal{A}$ for each one of its cliques $\mathcal{C}_i$ starts with $\langle \widehat{P}(\mathcal{C}_i), \widehat{Ch}(\mathcal{C}_i), \mathcal{C}_i, \psi_i, \widehat{S}(\mathcal{C}_i), \beta_i \rangle$ and runs:

1: **Phase I: UTILITY Propagation**
2: $\widehat{\mathcal{K}}_i = \psi_i$
3: **for all** $\mathcal{C}_j \in \widehat{Ch}(\mathcal{C}_i)$ **do**
4: Wait for utility message μ_{ji} from $\mathcal{C}_j$'s agent (that is $\beta_i(\mathcal{C}_j)$)
5: $\widehat{\mathcal{K}}_i = \widehat{\mathcal{K}}_i \otimes \mu_{ji}$
6: **end for**
7: **if** $\mathcal{C}_i$ is not the tree's root, let $\mathcal{C}_p = \widehat{P}(\mathcal{C}_i)$ **then**
8: Let $s^{ip} \in \widehat{S}(\mathcal{C}_i)$ be the separator between i and its parent
9: Send $\mu_{ip} = \bigoplus_{s^{ip}} \widehat{\mathcal{K}}_i$ to $\mathcal{C}_p$'s agent (that is $\beta_i(\mathcal{C}_p)$)
10: **end if**
11: **Phase II: VALUE propagation**
12: **if** $\mathcal{C}_i$ is not the tree's root, let $\mathcal{C}_p = \widehat{P}(\mathcal{C}_i)$ **then**
13: Wait for a value message σ_{pi} from $\mathcal{C}_p$'s agent (that is $\beta_i(\mathcal{C}_p)$)
14: $\widehat{\mathcal{K}}_i = \bigtriangledown_{\sigma_{pi}} \widehat{\mathcal{K}}_i$; /*Restrict $\widehat{\mathcal{K}}_i$ with the value message*/
15: **end if**
16: $d^* = arg \max_{d \in \mathcal{D}_{DV(\widehat{\mathcal{K}}_i)}} \widehat{\mathcal{K}}_i$; /*Fix the values for the free variables*/
17: $d^*_{\mathcal{C}_i} = d^* \cup \sigma_{pi}$; /*Put together the assesed values and the value message received. Assume the root gets an empty value message*/
18: **for all** $\mathcal{C}_j \in \widehat{Ch}(\mathcal{C}_i)$ **do**
19: Let $\sigma_{ij} = d^*_{\mathcal{C}^i}[s_{ij}]$; /*Project into the separator*/
20: Send σ_{ij} to $\mathcal{C}_j$'s agent (that is $\beta_i(\mathcal{C}_j)$)
21: **end for**
22: **return** $d^*_{\mathcal{C}_i}$;

ranges the DCOP problem in a pseudotree structure. From [6], a pseudotree arrangement of a graph G is a rooted tree with the same vertices as G and the property that adjacent vertices from the original graph fall in the same branch of the tree.



(a) DPOP (b) DJTG (c) DJT γ

Figure 3: DPOP & Action-GDL equivalent executions

Figure 1(c) shows a pseudotree for the constraint

graph in figure 1(a). A pseudotree of a constraint graph has two kinds of edges: *tree edges* (boldfaced lines); and *pseudoedges*, edges in the constraint graph (dashed lines). A pseudotree defines two relationships between variables: (1) parent/children for variables connected through an edge (e.g. in figure 1(c) x_2 is the parent of x_3); (2) pseudoparent/pseudochildren for variables connected through a pseudoedge (e.g. in figure figure 1(c) x_3 is a pseudochild of x_4). Therefore, we can represent a pseudotree as a pair $\langle P, PP\rangle$, where P and PP are functions that map each variable to its parent and pseudoparents, respectively. We also define the function *all parents* (AP) as follows $AP(x_i) = \{P(x_i)\} \cup PP(x_i)$. We obtain functions Ch and PCh, which return a variable's children and pseudochildren respectively, from the functions above as $Ch(x_i) = \{x_j \in \mathcal{A} | P(x_j) = x_i\}$ and $PCh(x_i) = \{x_j \in \mathcal{A} | PP(x_j) = x_i\}$.

DPOP has three phases: (1) a preprocessing phase to arrange a DCOP into a pseudotree; (2) a message-passing phase for agents to exchange utilities about their variables; and (3) a message-passing phase for agents to propagate value of their variables inferred. Algorithm 2 encodes the last two phases in terms of the operators introduced in section 3. Such encoding will ease the comparison with Action-GDL in section 6. Figure 3(a) depicts the DPOP execution over the pseudotree of figure 1(c).

Algorithm 2 DPOP($\langle \mathcal{A}, \mathcal{D}, \mathcal{R}\rangle$,$\langle P, PP\rangle$)

Each agent $a_i \in \mathcal{A}$, assigned to variable x_i, starts with $\langle P_i, Ch_i, PP_i, PCh_i, \mathcal{K}_i^0, \mathcal{D}_i\rangle$ where $\mathcal{K}_i^0 = r^i \otimes \bigotimes_{x_k \in AP(x_i)} r^{ik}$ and runs:

1: **Phase I: UTILITY Propagation**
2: **for all** $x_j \in Ch(x_i)$ **do**
3: Wait for the utility message μ_{ji} from a_j
4: $\mathcal{K}_i = \mathcal{K}_i \otimes \mu_{ji}$;
5: **end for**
6: **if** x_i is not the tree's root, let $x_p = P(x_i)$ **then**
7: Send $\mu_{ip} = \bigoplus_{\setminus x_i} \mathcal{K}_i$ to a_p
8: **end if**
9: **Phase II: VALUE propagation**
10: **if** x_i is not the tree's root, let $x_p = P(x_i)$ **then**
11: Wait for σ_{pi}
12: $\mathcal{K}_i = \bigtriangledown_{\sigma_{pi}} \mathcal{K}_i$;
13: **end if**
14: $d_i^* = arg \max_{d_i \in \mathcal{D}_i} \mathcal{K}_i$; /* Assess best value for x_i */
15: $d^* = d_i^* \cup \sigma_{pi}$; /* Put together the assessed value and the message received. Assume that the root gets an empty value message */
16: **for all** $x_j \in Ch(x_i)$ **do**
17: Send $\sigma_{ij} = d^*[DV(\mu_{ji})]$ to a_j /* Send the part of the assignment that is of interest to agent j*/
18: **end for**
19: **return** d_i^*;

In DPOP the initial local knowledge of an agent a_i about is variable x_i, namely $\mathcal{K}_i^0$, is a combination of some unary relation involving x_i and of some binary relations linking x_i with one of its parent or pseudoparent variables. Formally,

$$\mathcal{K}_i^0 = r^i \otimes [\bigotimes_{x_k \in AP(x_i)} r^{ik}] \quad (5)$$

Thus, in figure 3(a) the initial knowledge of a_1 about x_1 is composed of relations r^{14} and r^{12} (no unary relation in that case).

During the first message-passing phase, the *UTILITY propagation phase* (lines 1-8), each agent a receives, for each one of its assigned variables x_i, utility messages from all its children variables. Let $\mathcal{M}_i = \bigotimes_{x_j \in Ch(x_i)} \mu_{ji}$ be the combination of all utility messages received by agent a for variable x_i. Then, the utility message that agent a exchanges for each one of its variables, x_i, with the agent related to its parent variable, x_p, is the summarization of its current knowledge filtering out x_i (line 7). Formally:

$$\mu_{ip} = \bigoplus_{\setminus x_i} [\mathcal{K}_i^0 \otimes \mathcal{M}_i] \quad (6)$$

During the second message-passing phase, the *VALUE propagation phase* (lines 9-18), each agent a receives for each of its variables, x_i, a value message from the agent assigned to its parent variable, x_p, containing variable assignments for all variables in the domain of the utility message that this agent has sent to it in the previous phase, namely $\sigma_{ki}(DV(\mu_{ik}))$. Once agent a has received a value message for x_i from its parent x_p, the agent restricts its knowledge by incorporating the assigned variables (line 12) as follows:

$$\mathcal{K}_i = \bigtriangledown_{\sigma_{pi}(DV(\mu_{ip}))} [\mathcal{K}_i^0 \otimes \mathcal{M}_i] \quad (7)$$

Then, agent a can assesses the value of x_i as the one that maximizes its local knowledge (line 14) , namely:

$$d_i^* = arg \max_{d \in \mathcal{D}_i} \mathcal{K}_i(d) \quad (8)$$

Thereafter, agent a propagates to agents related to x_i' children a value message that contains value assignments for all variables in the domain of the utility message received from them in the previous phase (line 17). Formally:

$$\sigma_{ij}(DV(\mu_{ji})) = d^*_{DV(\mu_{ji})} \quad (9)$$

where $x_j \in Ch(x_i)$ and $d^*_{DV(\mu_{ji})}$ stands for the values to be assigned to the variables in the domain of the utility message μ_{ji}. Each agent a_i obtains these values from the assignment received from its parent along with its own assignment to x_i (line 15).

6 Generality of Action-GDL

In this section we prove that DPOP is a particular case of Action-GDL when Action-GDL is executed under

certain DJTs. We say that two distributed algorithms are equivalent if (i) agents perform the same computation and (ii) agents exchange the same messages. Recall that DPOP runs over pseudotrees. To show that Action-GDL generalizes DPOP, we prove that given any pseudotree we can build a DJT so that the execution of Action-GDL over this DJT is equivalent to the execution of DPOP over the pseudotree. To prove it, we firstly define a mapping from pseudotrees to DJTs. Secondly, we prove that given any pseudotree, the execution of DPOP over the pseudotree is equivalent to the execution of Action-GDL over the DJT produced by our mapping for the pseudotree.

In the next section we introduce the DJTG algorithm that distributedly computes mapping γ at a cost that is negligible with respect to the cost of solving the problem. Hence, the theoretical result provided in this section, together with the DJTG algorithm, ensure that Action-GDL is an efficient DCOP solver.

6.1 Mapping pseudotrees into junction trees

To define this mapping, we first need to define the notions of directly related variables (DRV) and inherited related variables (IRV) for a variable in a pseudotree. We say that a variable is directly related to another one if it is either: the very same variable, a variable of one of its parents, or a variable of one of its pseudoparents. We say that a variable is inherited related to another one if either: any of the children of the former is inherited related to the latter, or the latter is a pseudoparent of the former.

Definition 11 *Given a variable x_i in a pseudotree, its set of **directly related variables** is*

$$DRV(x_i) = \{x_i\} \cup AP(x_i) \tag{10}$$

Definition 12 *Given a variable x_i in a pseudotree, its set of **inherited related variables** is*

$$IRV(x_i) = \bigcup_{x_j \in Ch(x_i)} (IRV(x_j) \cup PP(x_j)) \setminus \{x_i\} \tag{11}$$

We can proceed to define a mapping, which we shall name γ, that builds a DJT from each pseudotree.

Definition 13 (γ) *Let γ be a function that given a DCOP $\Phi = \langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{R}, \alpha \rangle$ and a pseudotree $PT = \langle P, PP \rangle$ maps them to a DJT $\gamma(\Phi, PT) = \langle \mathcal{A}, \mathcal{X}, \mathcal{C}, \mathcal{S}, \Psi, \beta \rangle$, where:*

1) *The set of cliques $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_{|X|}\}$ contains one clique per variable in the pseudotree. The clique $\mathcal{C}_i$ contains all the variables directly or inherited related to variable x_i.*

$$\mathcal{C}_i = DRV(x_i) \ \cup \ IRV(x_i) \tag{12}$$

2) *The set of potentials Ψ contains one potential associated to each clique. Each clique potential ψ_i is the combination of: (i) a unary relation r_i that involves the clique decision variable x_i; and (ii) the binary relations that link x_i with its parent or one of its pseudoparents. Formally:*

$$\psi_i = r^i \otimes [\bigotimes_{x_j \in AP(x_i)} r^{ij}] \tag{13}$$

3) *The set of separators $\mathcal{S}$ contains one separator s^{ij} per pair of cliques $\mathcal{C}_i$ and $\mathcal{C}_j$ such that x_j is parent of x_i in the pseudotree. By definition of junction tree, each separator s^{ij} contains the intersection of its cliques ($s^{ij} = \mathcal{C}_i \cap \mathcal{C}_j$).*

4) *β is constructed as follows: each clique $\mathcal{C}_i$ is associated to the agent that controls variable x_i, formally $\beta(\mathcal{C}_i) = \alpha(x_i)$.*

Figure 3(a) shows a pseudotree PT over the DCOP Φ of figure 1(c) while figure 3(d) shows the DJT=$\gamma(PT, \Phi)$.

We also formulate, in addition to Eq. 12 a recursive definition for cliques in the mapping γ that will ease our proofs. Therefore we define each clique $\mathcal{C}_i$ in γ as the directly related variables of its variable, x_i, and the union of variables of each clique child excluding the child variable. Formally:

$$\mathcal{C}_i^{'} = DRV(x_i) \cup \left[\bigcup_{x_k \in Ch(x_i)} \mathcal{C}_k^{'} \setminus \{x_k\} \right] \tag{14}$$

Next we prove that the two definitions given for cliques in mapping γ, namely equations 12 and 14, are equivalent (Lemma 1).

Lemma 1 *Equations 12 and 14 are equivalent*

Proof 1 *We prove the lemma by induction on the depth of variable x_i. In the base case we consider a variable x_i whose depth is 1, a leaf in the pseudotree. Observe that in that case both equations, Eq. 12 and Eq. 14, define the x_i clique as its directed related variables $DRV(x_i)$.*

Induction Step: *Take a variable, x_i, whose depth is n+1.*

Then, by Eq. 14, x_i clique is defined as:

$$\mathcal{C}_i^{'} = DRV(x_i) \cup \left[\bigcup_{x_j \in Ch(x_i)} \mathcal{C}_j^{'} \setminus \{x_j\} \right]$$

By induction hypothesis we rewrite children cliques $\mathcal{C}'_j$ using Eq.12:

$$\mathcal{C}_i^{'} = DRV(x_i) \cup \left[\bigcup_{x_j \in Ch(x_i)} [DRV(x_j) \cup IRV(x_j)] \setminus \{x_j\} \right]$$

After eliminating $\{x_j\}$ from directed and inherited related variables:

$$\mathcal{C}_i^{'} = DRV(x_i) \cup \left[\bigcup_{x_j \in Ch(x_i)} AP(x_j) \cup IRV(x_j) \right]$$

Since $P(x_j)$ is already contained in $DRV(x_i)$:

$$\mathcal{C}_i^{'} = DRV(x_i) \cup \left[\bigcup_{x_j \in Ch(x_i)} IRV(x_j) \cup PP(x_j) \right]$$

Finally, by definition of inherited related variables (Eq. 12):

$$\mathcal{C}_i^{'} = DRV(x_i) \cup IRV(x_i)$$

Hence we have proved that both definitions, $\mathcal{C}_i$ (Eq. 12) and $\mathcal{C}_i^{'}$ (Eq. 14) are equivalent.

Before proving that the execution of DPOP over a pseudotree $\langle P, PP \rangle$ is equivalent to the execution of Action-GDL over the $DJT = \gamma(\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{R}, \alpha \rangle, \langle P, PP \rangle)$, we should provide with two observations about DJT that we will use in our proofs.

Observation 1 *Observe that due to the cliques composition (Eq. 12), the variables that compose a separator between x_i and x_j (the intersection of their cliques) contains all variables from clique $\mathcal{C}_i$ excluding x_i. Formally:*

$$\mathcal{C}_i \cap \mathcal{C}_j = \mathcal{C}_i \cap \left[DRV(x_j) \ \cup \ \left[\bigcup_{\mathcal{C}_k \in \widehat{Ch}(\mathcal{C}_j)} \mathcal{C}_k \setminus \{x_k\} \right] \right]$$

We can take $\mathcal{C}_i$ out of the union since it is child of $\mathcal{C}_j$:

$$\mathcal{C}_i \cap \mathcal{C}_j = \mathcal{C}_i \cap \left[DRV(x_j) \ \cup \ \mathcal{C}_i \setminus \{x_i\} \ \cup \bigcup_{\substack{\mathcal{C}_k \in \widehat{Ch}(\mathcal{C}_j) \\ k \neq i}} \mathcal{C}_k \setminus \{x_k\} \right]$$

Observe that the term $\mathcal{C}_i$ appears in both sides of the intersection. Therefore, since x_i it is not in $DRV(x_j)$ nor in other cliques children $\{\mathcal{C}_k \in \widehat{Ch}(\mathcal{C}_j) | k \neq i\}$:

$$\mathcal{C}_i \cap \mathcal{C}_j = \mathcal{C}_i \setminus \{x_i\}$$

Finally we can rewrite the definition of a separator s^{ij} in mapping γ, previously defined as the intersection of their cliques $\mathcal{C}_i \cap \mathcal{C}_j$, as :

$$s_{ij} = \mathcal{C}_i \setminus \{x_i\} \tag{15}$$

Observation 2 *Considering what is stated by Eq. 15, we can reformulate the variables that compose each clique C_i in a DJT defined by mapping γ (Eq. 14) as the union of the directly related variables of x_i with the union of all variables in the separators between clique C_i and each of its children. Formally:*

$$\mathcal{C}_i = DRV(x_i) \ \cup \ \left[\bigcup_{\mathcal{C}_k \in \widehat{Ch}(\mathcal{C}_i)} \mathcal{C}_k \setminus \{x_k\} \right] =$$

$$DRV(x_i) \cup \left[\bigcup_{\mathcal{C}_k \in \widehat{Ch}(\mathcal{C}_i)} s^{ki} \right]$$

Notice variables from a separator s^{ki} can also be expressed as the domain of the utility message $\mu_{ki}(s^{ki})$ sent through this separator. Hence, the equation above can be rewritten as:

$$\mathcal{C}_i = DRV(x_i) \cup \left[\bigcup_{\mathcal{C}_k \in \widehat{Ch}(\mathcal{C}_i)} DV(\mu_{ki}(s^{ki})) \right] \tag{16}$$

6.2 Proving equivalence

In this section we show how DPOP is a particular case of Action-GDL. To do so, we have to prove that given a pseudotree there is a DJT such that Action-GDL mimics DPOP execution. More formally, we have to prove that:

Theorem 1 *Given a DCOP Φ and a pseudotree* PT*, the execution of DPOP(Φ,* PT *) is equivalent to Action-GDL(γ(Φ,* PT *)).*

The proof of this theorem is immediate if we have previously demostrated that for both algorithms: (1) the computation performed and the messages exchanged during the utility propagation phase are the same (Lemma 2); and (2) the messages exchanged during the value propagation phase are also the same and that they converge to the same solution (Lemma 3).

Next sections contains the proofs of lemmas 2 and 3 as well as of theorem 1.

6.2.1 Utility propagation

Lemma 2 *Given a DCOP Φ and a pseudotree* PT*, the computation performed and the messages exchanged by each agent during the utility phase of DPOP(Φ,*PT*) and Action-GDL(γ(Φ,* PT *)) are the same.*

Proof 2 *We prove Lemma 2 by induction on d, the depth of variable x_i in the pseudotree* PT*.*

Consider case $d = 1$ (x_i is a leaf).

During a DPOP execution, x_i exchanges with its parent x_p in the PT *the following utility message (see Eq. 6):*

$$\mu_{ip} = \bigoplus_{\setminus x_i} [\mathcal{K}_i^0 \otimes \mathcal{M}_i]$$

Since x_i is a leaf:

$$\mu_{ip} = \bigoplus_{\setminus x_i} \mathcal{K}_i^0$$

During the Action-GDL execution, clique $\mathcal{C}_i$, related to variable x_i, exchanges with its clique parent $\mathcal{C}_p$, related to variable x_p, the following utility message (see Eq. 2):

$$\mu_{ip}(s^{ip}) = \bigoplus_{s^{ip}} [\psi_i \otimes \widehat{\mathcal{M}}_i]$$

Since $\mathcal{C}_i$ relates to x_i and by definition of γ (point 3) separators link cliques as are linked their variables in the PT*, $\mathcal{C}_i$ is a leaf. Therefore we can rewrite the above expression as:*

$$\mu_{ip}(s^{ip}) = \bigoplus_{s^{ip}} \psi_i$$

Replacing s^{ip} by the definition of separator in Eq. 15:

$$\mu_{ip}(s^{ip}) = \bigoplus_{\mathcal{C}_i \setminus \{x_i\}} \psi_i$$

Therefore, for both algorithms to exchange the same message in that case, two conditions must hold: (1) both messages must combine the same relations which is straightforward from observing ψ_i and $\mathcal{K}_i^0$ definitions (Equations 5 and 13 respectively); and (2) both messages must summarize over the same variables.

On the one hand, observe that the set of variables that the DPOP message summarizes on is composed of the variables in the $\mathcal{K}_i^0$ domain excluding x_i, namely $DV(\mathcal{K}_i^0) \setminus \{x_i\}$. $\mathcal{K}_i^0$ domain correspond with the set of directly related variables of x_i (see equation 5). Thus, the set of variables that DPOP message summarizes on is $DRV(x_i) \setminus \{x_i\}$. On the other hand, the set of variables that the Action-GDL message summarizes on is composed of $\mathcal{C}_i$ excluding x_i. $\mathcal{C}_i$ clique, since it is a leaf, is uniquely composed of the directly related variables of x_i (see Eq. 14). Thus, the Action-GDL message also summarizes on $DRV(x_i) \setminus \{x_i\}$ and messages are equal in that case.

Assume that lemma 2 holds for all variables whose depth is less than or equal to n in the pseudotree. Now consider variable x_i whose depth is $n+1$. We consider the subtree having x_i as root, with all variables under x_i in the pseudotree whose depth is at most n. Since by induction both algorithms exchange the same utility messages for variables in these subtrees whose depth is equal to or less than n, all the utility messages sent by variables under x_i are equal in both executions. Hence, we only have to prove that lemma 2 holds for utility messages sent by x_i.

During the execution of DPOP, variable x_i exchanges with its parent x_p the following utility message (see Eq. 6):

$$\mu_{ip} = \bigoplus_{\setminus x_i} [\mathcal{K}_i^0 \otimes \mathcal{M}_i]$$

Where $\mathcal{M}_i = \bigotimes_{x_j \in Ch(x_i)} \mu_{ji}$.

During Action-GDL execution clique $\mathcal{C}_i$, related to variable x_i, exchanges with its parent clique $\mathcal{C}_p$, related to variable x_p, the following utility message (see Eq. 2):

$$\mu_{ij}(s^{ip}) = \bigoplus_{s^{ip}} [\psi_i \otimes \widehat{\mathcal{M}}_i]$$

Where $\widehat{\mathcal{M}}_i = \bigotimes_{\mathcal{C}_j \in \widehat{Ch}(\mathcal{C}_i)} \mu_{ji}(s^{ji})$. *Replacing s^{ip} by the definition of separator in Eq. 15:*

$$\mu_{ij}(s^{ip}) = \bigoplus_{\mathcal{C}_i \setminus \{x_i\}} [\psi_i \otimes \widehat{\mathcal{M}}_i]$$

Again, for messages exchanged in DPOP and Action-GDL executions to be equal two conditions must hold: (1) both messages must combine same relations; and (2) both messages must summarize over the same variables.

That both messages combine the same relations is straightforward from: (1) the ψ_i and K_i^0 definitions from equations 5 and 13 respectively; and (2) considering that by induction the utility messages exchanged with their children are equal in both algorithms.

Now we will prove that both messages also summarize over the same set of variables. Firstly, observe that the set of variables that the DPOP message summarizes on is composed of the union of variables in the $\mathcal{K}_i^0$ domain excluding x_i and the domain of the utility messages exchanged with its children. As we have shown in the base case ($d = 1$), the $\mathcal{K}_i^0$ domain is composed of the directed related variables of x_i. Thus, the set of variables that the DPOP message summarizes on is composed of $\{DRV(x_i) \cup \bigcup_{x_j \in Ch(x_i)} DV(\mu_{ji})\} \setminus \{x_i\}$. On the other hand, the set of variables that the Action-GDL message summarizes on is composed of $\mathcal{C}_i$ excluding x_i. $\mathcal{C}_i$ clique, as specified by Eq. 16, is $\{DRV(x_i) \cup \bigcup_{x_k \in \widehat{Ch}(x_i)} DV(\mu_{ki}(s^{ki}))\} \setminus \{x_i\}$. Therefore, since by induction messages exchanged for variables under x_i are the same in both algorithms, messages exchanged for x_i are equal in both algorithms.

Thus, utility messages sent by agent at depth $n+1$ are equal in both execution and lemma 2 holds.

6.2.2 Value propagation

Lemma 3 *Given a DCOP Φ and a pseudotree* PT *the value assigned by each agent to its variable and the messages exchanged during the value propagation phase of DPOP(Φ,PT) and Action-GDL($\gamma(\Phi,$ PT$)$) are the same.*

Proof 3 *Note that it follows from lemma 2 that messages exchanged during the* UTIL propagation phase *are equal. Hence, to prove theorem 1 it only remains to prove that messages exchanged in the* VALUE propagation phase *are equal and both algorithms converge to the same solution.*

We prove this by induction on d, the depth of the pseudotree PT *defined over Φ.*

When $d = 1$, each variable x_i is a leaf that has neither parent nor pseudoparents in the pseudotree. Then, in DPOP algorithm, each variable x_i is inferred as (by Eq.17):

$$d_i^* = arg \max_{d \in \mathcal{D}_i} \mathcal{K}_i(d)$$

Where $\mathcal{K}_i$ is defined (see Eq. 7) as

$$\mathcal{K}_i = \underset{\sigma_{pi}(DV(\mu_{ip}))}{\bigtriangledown} [\mathcal{K}_i^0 \otimes \mathcal{M}_i]$$

Where x_p stands for the x_i parent in the pseudotree. Since x_i is a leaf it has no children and since $d = 1$ it has no parent. Therefore we can rewrite the above equation as:

$$d_i^* = arg \max_{d \in \mathcal{D}_i} \mathrm{r}^i(d)$$

On the other hand, following Action-GDL each clique $\mathcal{C}_i$, related to variable x_i, infers its clique variables as (by Eq. 4):

$$d_{\mathcal{I}_i}^* = arg \max_{d \in \mathcal{D}_{\mathcal{I}_i}} \widehat{\mathcal{K}}_i(d)$$

Where $\mathcal{I}_i = \mathcal{C}_i \setminus s^{ip}$ and $\widehat{\mathcal{K}}_i = \underset{\sigma_{pi}(s^{ip})}{\bigtriangledown} [\psi_i \otimes \widehat{\mathcal{M}}_i]$ where $\mathcal{C}_p = \widehat{\mathcal{P}}(\mathcal{C}_i)$ (see Eq. 3). Since $\mathcal{C}_i$ is a leaf it has no children and since $d = 1$ it has neither parent nor pseudoparents. Therefore $\widehat{\mathcal{K}}_i = \psi_i$ and the above equation can be rewritten as:

$$d_{\mathcal{C}_i}^* = arg \max_{d \in \mathcal{D}_{\mathcal{C}_i}} \psi_i(d)$$

*Taking the definition of $\mathcal{C}_i$ clique as the specified by Eq. 12, namely $\mathcal{C}_i = DRV(x_i) \cup IRV(x_i)$, we observe that since $\mathcal{C}_i$ is a leaf and has no parent its clique only contains variable x_i. Moreover, as is stated by the potential definition (Eq.*13*), in this case, the potential ψ_i is uniquely composed of a single relation r^i, the unary relation defined over x_i, being values inferred by both algorithms equal.*

Assume that the induction hypothesis holds for all pseudotrees whose depth is at most n. Now take PT *a pseudotree whose depth is $n + 1$. We can decompose this pseudotree as the root variable connected by directed edges and by a set of pseudoedges to a set of pseudotrees whose depth is at most n. The induction hypothesis holds in all these pseudotrees whose depth is at most n and we only need to prove it for the root variable x_i at level $n+1$.*

On the one hand, in DPOP, variable x_i is inferred as:

$$d_i^* = arg \max_{d \in \mathcal{D}_i} \mathcal{K}_i(d) \tag{17}$$

Once its variable have been inferred x_i propagates to each of its children $x_j \in Ch(x_i)$ a value message that contains the assignment for all variables in the domain of the utility message received from x_j (see Eq. 9). This assignment of variables comes from the assessed value x_i and the value message received from its parent. Since x_i is the root variable, the only assignment it propagates is $x_i = d_i^*$.

On the other hand, in Action-GDL, clique $\mathcal{C}_i$, related to variable x_i, infers the following variables (see Eq. 4):

$$d_{\mathcal{I}_i}^* = arg \max_{d \in \mathcal{D}_{\mathcal{I}_i}} \widehat{\mathcal{K}}_i(v) \tag{18}$$

Where $\mathcal{I}_i = \mathcal{C}_i \setminus s^{ip}$ is the set of clique variables excluding the variables in the separator between $\mathcal{C}_i$ and its parent $\mathcal{C}_p = \widehat{P}(\mathcal{C}_i)$ and since $\mathcal{C}_i$ is the root $\mathcal{I}_i = \mathcal{C}_i$.

Once its clique $\mathcal{C}_i$ has been inferred, it propagates to each one of its clique children $\mathcal{C}_j \in \widehat{Ch}(\mathcal{C}_i)$ a value message that contains an assignment for the variables in the separator s^{ij}.

Formally:

$$\varphi_{ij}(s^{ij}) = d_{s^{ij}}^* \tag{19}$$

Where $d_{s^{ij}}^*$ is a projection of $d_{\mathcal{C}_i}^*$ containing only the value for the variables in the separator s^{ij}.

To be equal both algorithms: (1) value messages exchanged must contain assignments for the same variables; and (2) they must infer the same variables and assign them the same values.

Firstly, to satisfy that value messages exchanged must contain assignments for the same variables is quite straightforward given above formulations. Observe that DPOP exchanges for x_i a value message with each one of its children that contains values for the variables in the domain of the utility message received from them and that this domain must be equal to x_i. Then in Action-GDL clique $\mathcal{C}_i$ exchanges a value message with each one of its children that contains an assignment for all variables in the separator. Since the domain of the utility message received from each one of its children in Action-GDL is equal to the separators and we have already proved that both algorithms exchange the same utility messages in the previous phase, then we can conclude that both algorithm exchange value messages with assignments for the same variables.

Secondly, to satisfy that that both algorithms infer the same variables and assign the same values, we have to prove that :(1) $\mathcal{C}_i$ is composed uniquely by variable x_i (they infer the same variables); and (2) $\mathcal{K}_i = \widehat{\mathcal{K}}_i$. (they infer variables with same values).

We prove (1) by defining $\mathcal{C}_i$ using Eq. 12:

$$\mathcal{C}_i = DRV(x_i) \cup IRV(x_i)\}$$

Since x_i is root variable it has no parent neither pseudoparents:

$$\mathcal{C}_i = \{x_i\} \cup IRV(x_i)$$

Since by the x_i is the root and inherited related variables, by its definition (see Definition 12), have always higher depth than x_i :

$$\mathcal{C}_i = \{x_i\}$$

Therefore $\mathcal{C}_i$ is composed uniquely by variable x_i.

Now we have to prove (2) that both algorithms also infer the same value for variable x_i, thus $\mathcal{K}_i = \widehat{\mathcal{K}}_i$.

Given the definition of $\mathcal{K}_i$ (see Eq.7) :

$$\mathcal{K}_i = \underset{\sigma_{pi}(DV(\mu_{ip}))}{\bigtriangledown} [\mathcal{K}_i^0 \otimes \mathcal{M}_i]$$

Where $x_p = P(x_i)$ *and* $\mathcal{M}_i = \bigotimes_{x_k \in Ch(x_i)} \mu_{ki}$. *Since* x_i *is the root variable it has no parent and* $\mathcal{K}_i = \mathcal{K}_i^0 \otimes \mathcal{M}_i$.

And the definition of $\widehat{\mathcal{K}}_i$ *(see Eq. 3)*

$$\widehat{\mathcal{K}}_i = \bigtriangledown_{\sigma_{pi}(s^{ip})} [\psi_i \otimes \widehat{\mathcal{M}}_i]$$

Where $\mathcal{C}_p = \widehat{P}(\mathcal{C}_i)$ *and* $\widehat{\mathcal{M}}_i = \bigotimes_{\mathcal{C}_j \in Ch(\mathcal{C}_i)} \mu_{ji}(s^{ji})$. *Since* $\mathcal{C}_i$ *is the root clique it has no parent and* $\widehat{\mathcal{K}}_i = \psi_i \otimes \widehat{\mathcal{M}}_i$.

We observe that both algorithms converge to the same value for variable x_i *by (1) observing that* $\mathcal{K}_i^0$ *and* ψ_i *definitions are equal (see Eq. 13 and Eq. 5 respectively); and (2) that utility messages exchanged in the previous phase are also equal in both algorithms (by lemma 2).*

Hence both algorithms exchange the same value messages and converge to the same solution and Theorem 1 holds.

7 The DJTG algorithm

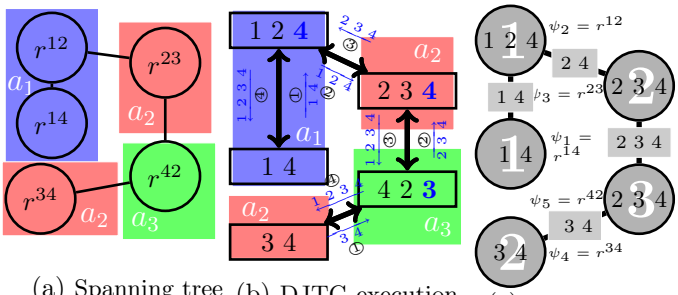
As explained in section 4, in order to ensure optimality and convergence, Action-GDL is restricted to be executed over a DJT (as given by definition 10). However, it has been argued [5] that traditional methods to compile JTs, these are triangulation methods based on the one proposed in [2], are not suitable when applied to problems that are distributed by nature because they produce JTs disregarding the structure of the problem. Therefore, here we propose an alternative algorithm, the so-called Distributed Junction Tree Generator (DJTG) that distributedly compiles a DJT, by exchanging a linear number of messages, that captures the distribution of control and relations required by the problem. Such DJT can be readily fed into, and hence solved by Action-GDL.

Given a spanning tree defined over a set of relations (a tree whose nodes are relations as depicted in figure 4(a)) distributed among agents, DJTG produces a DJT.

The DJTG algorithm receives as input a tuple $\langle \mathcal{A}, \mathcal{X}, \mathcal{R}, \kappa, ST \rangle$, where $\mathcal{A}$ is a set of agents, $\mathcal{X}$ is a set of variables, $\mathcal{R}$ is a set of relations, function $\kappa : \mathcal{R} \rightarrow \mathcal{A}$ maps relations to agents, and $ST \subseteq \mathcal{R} \times \mathcal{R}$ stands for the edges in the spanning tree defined over the relations in $\mathcal{R}$. Figure 4(a) illustrates an input to DJTG. Observe that relation r^{12} is assigned to agent a_1 and it is linked to relations r^{23} and r^{14} by edges in the $\mathcal{ST}$.

DJTG has two phases: 1) a preprocessing phase where agents create an arrangement with the same components as a DJT that may not satisfy the running intersection property (RIP); followed by 2) a message-passing phase that calculate the unique set of minimal cliques that satisfy the RIP.

In the preprocessing phase, each agent a creates individually a clique $\mathcal{C}_j$ for each one of its relations r_j



(a) Spanning tree (b) DJTG execution (c) Junction tree

Figure 4: DJTG execution

whose potential is each relation itself, namely $\psi_j = r_j$. Cliques are initially set to their potential domain, namely $C_i = \Delta_i$, in order to readily ensure the covering property. Moreover, for every two relations r_i, r_j connected in the $\mathcal{ST}$, agents create a separator s^{ij} linking their corresponding cliques $\mathcal{C}_i$ and $\mathcal{C}_j$. Figure 4 (b) shows the structure produced by the preprocessing phase. Boxes stand for cliques containing numbers that stand for variables' indices. Cliques are assigned to agents. For instance, clique $\mathcal{C}_1$ with variables x_1, x_4 and clique $\mathcal{C}_2$ with x_1, x_2 are assigned to a_1. The variables correspond to the domains of the cliques' potentials, namely the domains of r^{14} and r^{12} respectively. Cliques $\mathcal{C}_1$ and $\mathcal{C}_2$ are connected as their relations r^{14} and r^{12} in the $\mathcal{ST}$.

The second phase of DJTG is responsible for ensuring the RIP. In that phase, each agent exchanges for each one of its cliques, $\mathcal{C}_i$, *reachability* messages with agents related to $\mathcal{C}_i$'s neighbors that contain the set of reachable variables from $\mathcal{C}_i$. The set of reachable variables from a clique $\mathcal{C}_i$ to $\mathcal{C}_j$ is calculated as the union of $\mathcal{C}_i$ potential domain, Δ_i, with variables reachable from other $\mathcal{C}_i$ neighbors rather than $\mathcal{C}_j$.

Formally, a reachability message φ_{ij} from $\mathcal{C}_i$ to $\mathcal{C}_j$ is assessed as:

$$\varphi_{ij} = \Delta_i \cup \left[\bigcup_{\substack{\mathcal{C}_k \in \widehat{N}(\mathcal{C}_i) \\ k \neq j}} \varphi_{ki} \right] \tag{20}$$

where $\widehat{N}$ stands for a function that returns the neighbours of a clique. Figure 4(b) shows the messages exchanged over the preprocessed arrangement. Single-directed arrows between boxes stand for messages exchanged between cliques. Each arrow is labelled with some variables' indices and a circled number standing for the order of the message in the message-passing execution. Thus, agent a_1 sends a message to agent a_2 for clique $\mathcal{C}_3$ that contains variables (x_1, x_2, x_4), namely the variables that can be reached from clique $\mathcal{C}_2$. These variables are the result of the union of $\mathcal{C}_2$ potential domain, namely (x_1, x_2), with the reachable variables from $\mathcal{C}_2$, namely (x_1, x_4). Once an agent receives, for a given clique, reachability messages from all its neighbours, it redefines its clique adding variables that are in more than

one *reachability* message. Formally a clique $\mathcal{C}_i$ is redefined as follows:

$$\mathcal{C}_i = \Delta_i \cup \left[\bigcup_{\substack{\mathcal{C}_j, \mathcal{C}_k \in \widehat{N}(\mathcal{C}_i) \\ j \neq k}} \varphi_{ji} \cap \varphi_{ki} \right] \quad (21)$$

For instance, in figure 4(b) agent a_2 receives two reachability messages for clique $\mathcal{C}_3$: one with (x_1, x_2, x_4) from clique $\mathcal{C}_2$ associated to a_1, another one with (x_2, x_3, x_4) from clique $\mathcal{C}_5$ associated to a_3. Since both messages contain x_4, agent a_2 knows that its clique $\mathcal{C}_3$ must also carry x_4 to satisfy the RIP.

After computing cliques, it is straightforward to assess separators (see definition 9).

Finally, figure 4(c) depicts the DJT as produced by DJTG from the initial distribution of relations in figure 4(a). Circles stand for cliques, labelled with the variables each one contains, and edges between cliques stand for separators labelled with their variables. Notice that by creating a clique per relation and by assigning each clique to the agent associated to that relation, DJTG manages to preserve the initial distribution of the problem.

After computing cliques, it is straightforward to assess separators (see definition 9). Finally, figure 4(c) depicts the DJT as produced by DJTG from the initial distribution of relations in figure 4(a). Notice that by creating a clique per relation and by assigning each clique to the agent associated to that relation, DJTG manages to preserve the initial distribution of the problem. This alternative way of building a JT, by directly ensuring the RIP over a set of relations was initially formulated in [4] in the context of sensor networks. However, they restricted each agent to control a single clique whose potential results from the combination of relations located to the agent. DJTG extends the algorithm in [4] to: (1) allow each agent to be associated to more than one clique; and (2) accept as an input a spanning tree defined over some set of relations, without making any assumptions on their composition. In the next section we show how given a DCOP Φ and a pseudotree $\langle P, PP \rangle$, the DJTG algorithm can arrange the problem into a DJT $\gamma(\Phi, \langle P, PP \rangle)$ (see definition 13). This proof completes the proof of equivalence of DPOP and Action-GDL, given in section 6, by showing that there is a distributed algorithm that can compile a DCOP problem given a pseudotree into a DJT that makes the execution of DPOP and Action-GDL equal.

8 Building the mapping

In this section we show how, given a DCOP $\Phi = \langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{R}, \alpha \rangle$ and a pseudotree $\langle P, PP \rangle$, the DJTG algorithm, explained in section 7, arranges the problem into a $DJT = \gamma(\Phi, \langle P, PP \rangle)$ (see definition 13). Thus, it proves that exists a distributed algorithm that can compile a DCOP problem given a pseudotree arrangement over variables into a DJT that corresponds to the one specified by mapping γ.

As explained in section 7, the DJTG algorithm is executed over an spanning tree ST defined over a set of relations $\hat{\mathcal{R}}$ distributed among agents. We will show that the DJTG algorithm distributedly builds$\gamma(\Phi, \langle P, PP \rangle)$ given:

(1) A set of relations $\hat{\mathcal{R}}$ that contains one relation per variable in the pseudotree. Each relation $\hat{r}_i \in \hat{\mathcal{R}}$ is the result of combining all relations associated to its related variable x_i, where relations are associated to the lowest variable of its domain in the pseudotree. Thus, $\hat{r}_i = r^i \otimes \bigotimes_{r_j \in AP(r_i)} r^{ij}$; and

(2) An spanning tree ST defined over $\hat{\mathcal{R}}$ which is based on the pseudotree $\langle P, PP \rangle$ as follows: if two variables x_i,x_j are connected in the pseudotree so are their corresponding combined relations $\hat{r}_i$,$\hat{r}_j$ in the ST. We define the spanning tree ST as a function $N : \hat{\mathcal{R}} \rightarrow 2^{\hat{\mathcal{R}}}$ that returns for each relation $\hat{r}_i \in \hat{\mathcal{R}}$ its neighboring relations in the ST.

(3) A distribution κ that maps relations to agents as follows: each relation $\hat{r}_i \in \hat{\mathcal{R}}$ is assigned to the agent related to variable x_i in the DCOP by function α. Thus, $\kappa(\hat{r}_i) = \alpha(x_i)$.

Theorem 2 *Given a DCOP Φ and a pseudotree $\langle P, PP \rangle$ defined over Φ the DJTG compiles a $DJT = \gamma(\Phi, \langle P, PP \rangle)$*

Proof 4 *In the preprocessing phase of the DJTG, each agent a creates one clique $\mathcal{C}_i$ per relation assigned $\{\hat{r}_i \in \hat{\mathcal{R}} | \kappa(\hat{r}_i) = a\}$ whose potential is each relation itself, namely $\psi_i = \hat{r}_i$. Therefore, since $\hat{\mathcal{R}}$ contains one relation per variable in the pseudotree, the set of cliques $\mathcal{C}$ also contains one clique for variable in the pseudotree. Moreover, since $\hat{r}_i = r^i \otimes \bigotimes_{r_j \in AP(r_i)} r^{ij}$ for all $\hat{r}_i \in \hat{\mathcal{R}}$, potentials are defined as $\psi_i = r^i \otimes \bigotimes_{r_j \in AP(r_i)} r^{ij}$ for all $\psi_i \in \Psi$. Moreover agents create a separator s^{ij}, linking their corresponding cliques $\mathcal{C}_i$ and $\mathcal{C}_j$, for every two relations $\bar{r}_i$,$\bar{r}_j$ connected in the* ST. *Since these two relations correspond to two variables connected in the pseudotree, the set of separators $\mathcal{S}$ contains one separator s^{ij} per pair of cliques $\mathcal{C}_i$ and $\mathcal{C}_j$ such that x_j is parent of x_i in the pseudotree.*

Therefore, notice that the second, third and fourth condition of the mapping γ are straightforward since they are already fulfilled after the preprocessing phase of the DJTG algorithm. Thus, we only have to prove that after ensuring the running intersection property (RIP), the DJTG algorithm produces cliques as defined in equation 12.

During an DJTG execution, agent a computes clique $\mathcal{C}_i$ as:

$$\mathcal{C}_i = \Delta_i \cup \left[\bigcup_{\substack{\mathcal{C}_j, \mathcal{C}_k \in \widehat{N}(\mathcal{C}_i) \\ j \neq k}} (\varphi_{ji} \cap \varphi_{ki}) \right]$$

Among the neighbors of the clique $\mathcal{C}_i$ in the spanning tree we can distinct between the neighbor that correspond to the parent variable in the pseudotree, namely $P_i = \mathcal{C}_k | x_k = P(x_i)$ and these that correspond to one of the children's variables, namely $Ch_i = \{\mathcal{C}_j | x_j \in Ch(x_i)\}$. Therefore we can rewrite the above equation as follows:

$$\mathcal{C}_i = \Delta_i \cup \left[\bigcup_{\substack{\mathcal{C}_k, \mathcal{C}_t \in Ch_i \\ k \neq t}} (\varphi_{ti} \cap \varphi_{ki}) \right] \cup \left[\bigcup_{\mathcal{C}_k \in Ch_i} (\varphi_{kP_i} \cap \varphi_{kP_i}) \right]$$

Since potentials are defined following equation 13, we have that $\Delta_i = DV(r^i \otimes \bigotimes_{x_j \in AP(x_i)} r^{ij}) = DRV(x_i)$. After replacing Δ_i in the above equation:

$$\mathcal{C}_i = DRV(x_i) \cup \left[\bigcup_{\substack{\mathcal{C}_k, \mathcal{C}_t \in Ch_i \\ k \neq t}} (\varphi_{ti} \cap \varphi_{ki}) \right] \cup \left[\bigcup_{\mathcal{C}_k \in Ch_i} (\varphi_{kP_i} \cap \varphi_{kP_i}) \right]$$

By definition of pseudotree (see section 5), variables in different branches can not have direct dependencies among them. Therefore, as potentials are defined as equation 13, the intersection of reachable variables from different children of a clique, $\mathcal{C}_i$, in the spanning tree contain at most variable x_i which is already contained in $DRV(x_i)$. Therefore,

$$\mathcal{C}_i = DRV(x_i) \cup \left[\bigcup_{\mathcal{C}_k \in Ch_i} (\varphi_{kP_i} \cap \varphi_{kP_i}) \right] \tag{22}$$

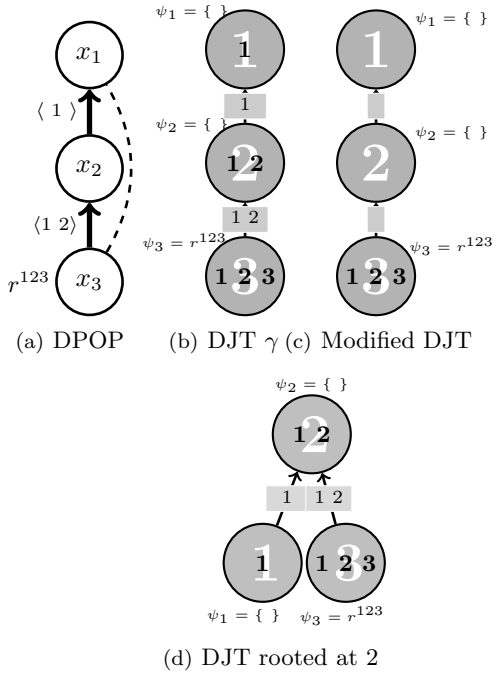
Moreover, as potentials are defined as equation 13, variables reachable from $\mathcal{C}_i$ children are: all variables that are under x_i in the pseudotree, its parents and pseudoparents. Furthermore, since the edges of the spanning tree used correspond to the edges of the pseudotree, all agents whose potential domain contains x_i are under $\mathcal{C}_i$ in the spanning tree, never in a higher level or in another branch. Notice that it means that x_i is only reachable from its $\mathcal{C}_i$'s children and not from its parent. Hence, all variables from agents under $\mathcal{C}_i$, along with its parents, can not be reachable from $\mathcal{C}_i$'s parent. Therefore the intersection between variables reachable from $\mathcal{C}_i$ children with variables reachable from $\mathcal{C}_i$ parent are the union of the pseudoparents of variables under x_i excluding all agents under x_i in the pseudotree. Observe that this set of agents exactly corresponds with the definition of inherited related variables of x_i (see definition 12). Finally, equation 22 can be rewritten as:

$$\mathcal{C}_i = DRV(x_i) \cup IRV(x_i)$$

The above expression corresponds to the clique definition given by equation 12 and theorem 2 holds.

9 Exploiting Action-GDL

At this point we have learned that Action-GDL generalises DPOP. It is now reasonable to wonder about the benefits that such generality delivers. In what follows we argue that Action-GDL can yield better algorithmic performance than DPOP. Action-GDL can achieve such improvement because: (i) DJTs allow to explore problem arrangements that cannot be represented via pseudotrees; and (ii) it can assess multiple variables' values at once. To show the benefits of Action-GDL with respect to DPOP, we empirically compare the computation and communication costs of both algorithms when solving the same DCOP. Moreover, we also compare the maximum degree of parallelism each algorithm can achieve.



(a) DPOP (b) DJT γ (c) Modified DJT

(d) DJT rooted at 2

Figure 5: Example of experimented rearrangements

Our first experiment is aimed at showing the communication and computation savings achieved by Action-GDL with respect to DPOP. Such savings are obtained by adequately transforming the problem arrangement represented by a pseudotree. Consider the example depicted in figure 5. Figure 5(a) shows a pseudotree. Observe that although variables x_1, x_2 do not have any relation, since DPOP can only eliminate variables one by one, its execution would propagate utility messages over these variables. Figure 5(b) depicts the DJT produced by mapping γ of definition 13 when applied to the pseudotree. Hence, according to theorem 1 the execution of Action-GDL over this DJT and the execution of

DPOP over the pseudotree are equivalent. However, we can further transform the DJT in figure 5(b) to obtain savings. Notice that the definition of mapping γ (from pseudotree to DJT) *forces* that each clique contains its variable although it is not part of its potential domain (equation 11). If we do not enforce such constraint, the DJTG algorithm generates the DJT of figure 5(c), which can be regarded as a rearrangement of the one in figure 5(b). When running over the DJT in figure 5(c), Action-GDL does not need to exchange any utility messages, reducing the computation required to solve the problem. Hence there is a rationale for the rearrangement that we propose. Notice that when running Action-GDL, a variable' value is assessed at the clique that concentrates all its information. In figure 5(c), the values of x_2, x_3 are assessed at the clique containing x_1, x_2, x_3. That clique is in charge of propagating its decisions. Hence, there is no need to propagate utility messages involving x_1 and x_2 up the tree.

Next we compare the size of the messages exchanged and the amount of computation required by Action-GDL and DPOP when solving the same DCOP as the number of variables grows. Given a number of variables $n \in \{10, 30, 50, 70, 90\}$, we generate 2000 DCOPs, each one with $1.5 \cdot n$ constraints whose arity is randomly picked from 2 to 4. We create pseudotrees for DPOP using the DFS-MCN heuristic [5] and DJTs for Action-GDL considering the rearrangement of the DJT produced by mapping γ as explained above. Figure 6 (upper) shows the average savings (in percentage) in communication and computation of Action-GDL with respect to DPOP[3]. Observe that a simple rearrangement of the DJT leads to significant savings in communication and computation costs, which increase as the number of variables grows.

In our second experiment we show that we can help Action-GDL to reduce the maximum degree of parallelism with respect to DPOP. We propose to found such improvement on another rearrangement of the DJT produced by mapping γ. This time we propose to change the root of the DJT. Figure 5(d) illustrates such rearrangement for the DJT in figure 5(b). Observe that changing the root of a DJT never changes either the computation or the communication costs because cliques and separators remain the same. Notice also that we cannot explore such an arrangement in DPOP because changing the root of a pseudotree can lead to a non-valid pseudotree. For instance, choosing x_2 as a root in the pseudotree of figure 5(a) makes it an non-valid pseudotree (because of the dependency between variables x_1 and x_3). Next we measure and compare the MPC, formally defined as $MPC = \max_{P_i \in P} \sum_{C_j \in P_i} d^{|C_j|}$, where P stands for the set of all paths, a path P_i contains all cliques from the i-th clique to the root, and d stands for the variable domain size. To run this experiment we employ the same pseudotrees generated for our first experiment above and we set $d = 2$. We rearrange DJTs for Action-GDL as explained above to select as clique root the one that reduces the MPC the most. Figure 6 (lower) shows our empirical results by depicting the average (in percentage) improvement in MPC[3] that Action-GDL achieves. Observe that the gain in parallelism can be very significant (from 25% to 40% of MCP reduction), and it increases as the number of variables grows.

[3]Percentage assessed as $\frac{(DPOP - ActionGDL)}{DPOP} \cdot 100$

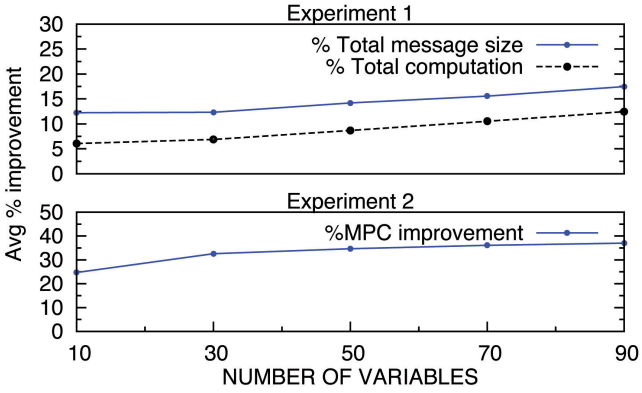


Figure 6: Experimental results

10 Conclusions

In this paper we show how the novel message-passing algorithm, the so-called Action-GDL, to solve MCP's has DPOP has a particular case. To do so, we prove that : (1) given a pseudotree there is a DJT such that Action-GDL mimics DPOP execution; and (2) that DJTG algorithm compiles, given the pseudotree, this DJT distributedly. Moreover, we provide empirical evidence to show how we can computationally exploit the generality of Action-GDL. Thus, we show that Action-GDL can outperform DPOP in terms of the amount of computation, communication and parallelism of the algorithm solving cost. Therefore, Action-GDL can efficiently solve DCOPs.

References

[1] S. M. Aji and R. J. McEliece. The generalized distributive law. *IEEE Transactions on Information Theory*, 46(2):325–343, 2000.

[2] F. V. Jensen and F. Jensen. Optimal junction trees. In *UAI*, pages 360–366, 1994.

[3] P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo. Adopt: asynchronous distributed constraint optimization with quality guarantees. *Artif. Intell.*, 161(1-2):149–180, 2005.

[4] M. A. Paskin, C. Guestrin, and J. McFadden. A robust architecture for distributed inference in sensor networks. In *IPSN*, pages 55–62, 2005.

[5] A. Petcu. *A Class of Algorithms for Distributed Constraint Optimization*. PhD thesis, EPFL, Lausanne, 2007.

[6] A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *IJCAI*, pages 266–271, 2005.