Resource-Aware Junction Trees for Efficient Multi-Agent Coordination

N. Stefanovitch
LIP6 - UPMC
75016 Paris, France
stefanovitch@poleia.lip6.fr

A. Farinelli
University of Verona
Verona, I-37134, Italy
alessandro.farinelli@univr.it

A. Rogers, N.R. Jennings
University of Southampton
Southampton, SO17 1BJ, UK
{acr,nrj}@ecs.soton.ac.uk

ABSTRACT

In this paper we address efficient decentralised coordination of cooperative multi-agent systems by taking into account the actual computation and communication capabilities of the agents. We consider coordination problems that can be framed as Distributed Constraint Optimisation Problems, and as such, are suitable to be deployed on large scale multi-agent systems such as sensor networks or multiple unmanned aerial vehicles. Specifically, we focus on techniques that exploit structural independence among agents' actions to provide optimal solutions to the coordination problem, and, in particular, we use the Generalized Distributive Law (GDL) algorithm. In this settings, we propose a novel resource aware heuristic to build junction trees and to schedule GDL computations across the agents. Our goal is to minimise the total running time of the coordination process, rather than the theoretical complexity of the computation, by explicitly considering the computation and communication capabilities of agents. We evaluate our proposed approach against DPOP, RDPI and a centralized solver on a number of benchmark coordination problems, and show that our approach is able to provide optimal solutions for DCOPs faster than previous approaches. Specifically, in the settings considered, when resources are scarce our approach is up to three times faster than DPOP (which proved to be the best among the competitors in our settings).

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: [Coherence and coordination, Multiagent systems]

General Terms

Algorithms, Performance, Experimentation

Keywords

multiagent coordination, junction tree, treewidth, variable elimination, heuristic algorithm, GDL, DCOP

1. INTRODUCTION

Many practical applications require the development of effective decentralised coordination techniques for cooperative multi-agent systems. For example, agent-based techniques have been widely used to control physical devices which can acquire and process information from the environment, such as sensor networks deployed to collect environmental data [12] or multiple unmanned aerial vehicles deployed to collectively patrol and map a defined area [16]. The development of decentralised coordination techniques is particularly challenging in these domains because such devices usually have constrained computational resources (due to the requirement of minimising power consumption) and because communication is usually limited in bandwidth and is dependent on the physical distance and mutual positions of the devices (due to the wireless communication technology frequently used). Moreover, to develop more cost effective systems, and to manage legacy, it is expected that the devices within such networks will be heterogeneous; having different computation and communication capabilities.

Recent work has shown that to develop effective and efficient coordination techniques it is crucial to exploit the structural independence between the agents' utility functions (i.e. the fact that the utility of each agent only depends on its own choice of action and that of a small number of locally interacting neighbours) [16, 14]. Doing so allows the decentralised coordination problem to be framed as a Distributed Constraint Optimisation Problem (DCOP), enabling a number of optimal algorithms to be used as solution techniques, e.g., ADOPT [9], OptAPO [8] and DPOP [15].

However, these algorithms take no account of the heterogeneous computational and communication resources available to the different agents within the system. In many settings, and particularly in the cooperative settings we focus on here, it may be beneficial to delegate computations such that (i) we take advantage of agents with greater than average computational capabilities, and (ii) we minimise communication between agents with poor communication links. Current algorithms for solving DCOPs do not consider such strategies. For example, DPOP arranges the constraint network into a pseudo-tree using a Depth First Search (DFS) method. While the DFS can be conveniently performed using distributed algorithms, it does not take into account agents' individual computation and communication capabilities, and it can result in an inefficient allocation of computations to the agents. In contrast Paskin and Guestrin developed an approach to cope with networks that have poor quality communication links (as it is frequently the case with wireless networks)[14]. Their approach uses the routing tree of the communication network to arrange agents into a *junction tree*, which is then further optimised in order to minimise communications. While this work takes computation and communication into account, it forces the junction tree structure to be a spanning tree of the communication network, which can, as we shall show later, significantly reduce the efficiency of the coordination process.

Thus, against this background, in this paper we address these shortcomings by proposing a *resource aware* solution technique for DCOPs. Our approach pre-processes the constraint network by

Cite as: Resource-Aware Junction Trees for Efficient Multi-Agent Coordination, N. Stefanovitch, A. Farinelli, A. Rogers and N. R. Jennings, *Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, Tumer, Yolum, Sonenberg and Stone (eds.), May, 2–6, 2011, Taipei, Taiwan, pp. 363-370.

building a junction tree, over which optimal inference is performed using standard message passing techniques, such as those provided by the Generalised Distributive Law (GDL) framework [1]. Junction trees are well known structures, frequently used in graphical models and constraint processing [7] and while finding the optimal junction tree (in the sense of a minimal size of the largest clique) is NP-hard, a number of heuristics that build near optimal junction trees are well known. Here, we propose a distributed approach to build the junction tree that is based on the variable elimination algorithm [5], but is extended to consider the heterogeneous nature of both computational and communication resources within the network. In this context, the optimal tree is not the one that minimises the theoretical complexity of the computation (as is the case within the standard literature of junction trees), but is the one that minimises the total running time of the coordination algorithm (including both the time required by the agents to individually compute their partial solutions, and the time for these solutions to propagate up and down the junction tree).

In doing the above, this paper makes the following contributions to the state of the art:

- We present the first model of decentralised coordination that explicitly considers the total running time required by a coordination algorithm that operates in heterogeneous multi-agent system.
- We propose a novel distributed algorithm, based on the variable elimination algorithm, which uses a novel resource aware heuristic to minimise the running time of the coordination process (as defined above).
- We empirically evaluate the proposed technique on a simulated environment, comparing it with three state of the art approaches for multi-agent systems: the pseudo-tree built by DPOP, the junction tree formation algorithm of Paskin and Guestrin, and a benchmarking centralised approach. Our results show that, when communication resources are scarce, our resource aware heuristic improves upon previous techniques being up to 3 times faster than DPOP, which proved to be the best competitors in our settings.

The rest of the paper is organised as follows: Section 2 provides basic background knowledge on DCOPs and graphical models while Section 3 formalises the problem we are addressing. Section 4 details our approach and the resource aware heuristic we propose. Section 5 presents the empirical analysis of our approach, Section 6 discusses related work and Section 7 concludes.

2. BACKGROUND

We provide here a brief review of background knowledge concerning DCOPs, junction trees and the GDL framework.

2.1 Distributed Constraint Optimisation Problems

Formally a DCOP can be defined as a tuple $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \Psi \rangle$, where $\mathcal{A} = \{A_1, \ldots, A_k\}$ is a set of agents, $\mathcal{X} = \{1, \ldots, n\}$ is a set of variables. Each variable is owned by exactly one agent, but an agent can potentially own more than one variable. An agent is responsible for assigning values to the variables it owns. $\mathcal{D} = \{D_1, \cdots, D_n\}$ is a set of discrete and finite variable domains, each variable i can take value in the domain D_i. Finally, $\Psi = \{\psi_1, \ldots, \psi_m\}$ is a set of constraint functions that describe the constraints among variables. Each function $\psi_i : D_{i_1} \times \cdots \times D_{i_{r_i}} \rightarrow \mathbb{R}$ depends on a set of variable $\mathbf{X_i} \subseteq \mathcal{X}$, where $r_i = |\mathbf{X_i}|$ is the arity of the function. Each function assigns a real value to each possible assignment of the variables it depends on. Given this settings, we wish to find the variable assignment $\mathbf{X}^*$ such that the sum of all constraint functions is maximised:

$$\mathbf{X}^* = \arg\max_{\mathbf{X}} \sum_{i=1}^{m} \Psi_i(\mathbf{X}_i) \quad (1)$$

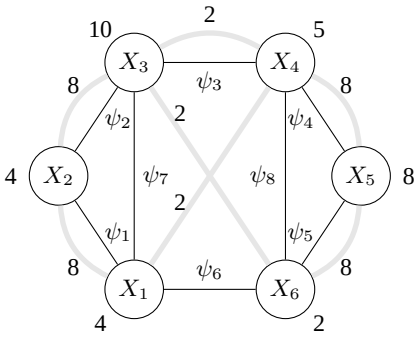


Figure 1: Example of a MAS coordination problem : communication (wide grey edges) and constraint network (thin black edges), nodes represent both agents and variables

2.2 Junction Trees

DCOPs can be solved using message passing algorithms across the *constraint network*; a graph representation of a DCOP where nodes are variables and edges are constraint functions. However, in order to ensure completeness and termination, the constraint network must be a tree. If this is not the case, then it is necessary to transform the original constraint network into a special graphical structure called a junction tree. This is done by forming a clique graph whose nodes (cliques) and edges (separators) are clusters of variables. A junction tree is simply a clique graph which satisfies the following four properties: single-connectedness, running intersection, covering and maximality. Single-connectedness ensures that the graph is a tree, yielding termination. Running intersection ensures that any variable present in the intersection of the domain of two cliques is also present in every clique of the path joining them, yielding correctness. Covering ensures that the domain of every constraint function of the DCOP is the subset of at least one clique. Maximality states that a clique can not have a domain which is a subset of the domain of another clique. One of the most well-known algorithms to transform a constraint graph into a junction tree is the variable elimination algorithm. This algorithm works by sequentially selecting variables of the constraint graph, eliminating them and forming cliques accordingly (see [7] for more details).

2.3 GDL

Having formed a junction tree, the optimal solution of the DCOP can be found using a suitable message passing algorithm, of which GDL is the most general [1]. The GDL algorithm uses two operators, $\otimes$ for function combination and $\oplus$ for function marginalisation, and exploits the distribution property of $\otimes$ over $\oplus$. It works by passing messages along the edges of the junction tree and performing computation at the level of the nodes. DCOPs can be solved by GDL using the sum and max operators respectively.

A message from a clique n to a clique m is a utility function defined recursively over the intersection of the domains of n and m

by the formula: $\forall \mathbf{y} \in d(m) \cap d(n)$, where the function d gives the set of variables associated to a clique or separator (collect phase):

$$\psi_{n\to m}(\mathbf{y}) = \oplus_{\mathbf{x}\in d(n)\setminus d(m)}(\otimes_{i\in\Gamma(n)\setminus m}\psi_{i\to n}\otimes\psi_n)(\mathbf{x}\cup\mathbf{y})$$

A special node, called the root, receives the information from all of its neighbours. It computes the optimal instantiation of its variables and then starts the recursive phase of local optimisation yielding the global optima (propagation phase). Specifically hen a clique m receives the instantiation from its parent n it conditionally optimise the instantiation of its variable and then propagates the set of instantiation it knows:

$$\mathbf{x}^*_n = (arg\oplus_{\mathbf{x}\in d(n)\setminus d(m)}(\otimes_{i\in\Gamma(n)\setminus m}\psi_{i\to n}\otimes\psi_n)(\mathbf{x}\cup\mathbf{x}^*_{m\cap n}))\cup\mathbf{x}^*_{m\cap n}$$

Both the collect and propagation phases can be performed with a linear number of messages. However, during the collect phase, the computation of the messages and the size of the messages are exponential in the size of the clique sending them, and the cardinality of the corresponding separator respectively.

The use of junction trees in combination with the GDL algorithms is attractive as they inherently work in a distributed way by message passing. Moreover such an approach is efficient as it is exponential only in the treewidth rather than in the total number of variables. The treewidth is a parameter of a tree decomposition which is the size of the largest clique, and is usually far smaller than the total number of variables. Finding a junction tree of minimal treewidth is however an NP-hard problem [7].

3. PROBLEM DESCRIPTION

Having presented the necessary background, we now formally describe the problem that we tackle; that of, minimising the total running time of a coordination algorithm when faced with heterogeneous computational resources, and a bandwidth constrained communication structure. To this end, we introduce the concept of a *computational* task to model the GDL solution process for DCOP. A computational task $\tau(\psi_{n\to m}(\mathbf{y}))$ describes the amount of computation that an agent has to perform in order to *compute* the message $\psi_{n\to m}(\mathbf{y})$ defined in Section 2.3.

Since the computation of a message $\psi_{n\to m}(\mathbf{y})$ recursively depends on the computation of messages from neighbouring cliques, these computational tasks are tied by the set of execution constraints $EC(\tau_i) = \{\tau^i_1, \cdots, \tau^i_k\}$ where τ^i_j are tasks that must be executed before τ_i. We denote the set of all the computational tasks as T, and this consists of the set of cliques of the junction tree and the set of constraint functions[1]. Execution of a computational task involves the processing of constraint functions and received messages (i.e., the summation of those functions and the maximisation over some subsets of the decision variables) in order to compute the message. Each computational task is assigned to a single agent, which is responsible for all the aspects of the execution of this task. We denote $\alpha : \mathrm{T} \to \mathcal{A}$ the function representing this allocation.

The *completion time* of a task τ depends on its size (given by the $size$ function), on the computational power of the agent $\alpha(\tau)$ expressed as the number of constraint checks per unit of time (given by the $speed$ function), and on the characteristics of the communication links used to route the messages produced by the execution of $EC(\tau)$ to the agent $\alpha(\tau)$ (given by the $trans$ function). The completion time of a single computational task, can then be defined as a function $CT : \mathrm{T} \to \mathbb{N}$, whose expression is:

$$\begin{aligned} CT(\tau) &= max_{\tau_i\in EC(\tau)}CT(\tau_i) + trans_{\alpha(\tau_i),\alpha(\tau)}(\tau_i) \\ &\quad + size(\tau)/speed(\alpha(\tau)) \end{aligned}$$

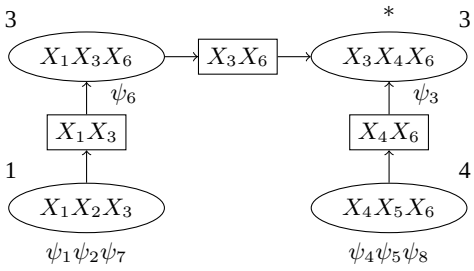


Figure 2: Instance of a solution obtained on the MAS coordination problem of Figure 1

Note that, we assume that agents are not multitasking and tasks are not preemptive, such that agents can either compute, send or receive messages at any time, and can not interrupt one of those activities if already started. We consider a restricted communication structure composed of pairwise communication links. For each link we consider a symmetric limited bandwidth that defines the time required to transmit messages over the link. If an agent on the shortest path between two agents is performing one of the above activities, messages between these two agents have either to wait or to find a new route. Such a setting models some important aspects of wireless sensor networks such as limited bandwidth, limited connectivity, multi-hop communication and possible network congestions. Small bandwidth values can represent both low throughput reliable communication links or high throughput unreliable communication links.

Figure 1 shows an exemplar instance of a constraint network associated with a restricted communication network. The nodes represent decision variables, the thin black edges are constraint dependencies between the variables. Constraints that hold between variables are associated with constraint functions ψ_i as shown in the figure. The thick grey edges represent the underlying communication network between the agents responsible for the variables. Numbers next to those edges correspond to the bandwidth of the communication links while numbers next to the agents represent computational speed of the agents.

Now, given any specific constraint network, we can define the set of computational tasks and the set of messages that need to be computed and propagated according to the GDL algorithm. The GDL algorithm can then be described as the execution of the associated computational tasks using agents as computational resources. Specifically, given a DCOP instance $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \Psi \rangle$, to obtain the set of associated computational tasks we have to choose a set $\mathcal{C}$ of cliques, an allocation $\alpha_\Psi : \Psi \to \mathcal{C}$ of constraints functions to cliques, and an allocation $\alpha_\mathcal{C} : \mathcal{C} \to \mathcal{A}$ of the cliques onto the agents. We can now define $MS(\mathcal{C}, \alpha_\Psi, \alpha_\mathcal{C})$ as the time to complete all tasks subject to the execution constraints. Such a criterion is known as the makespan in the scheduling literature[2], and in our setting it corresponds to the time at which the last task has been computed (which is the maximisation of the root clique). Therefore, representing the special task associated to the root clique of the GDL algorithm as τ_r, the makespan can be defined as: $MS =$

[1]Constraint functions can be seen as computational tasks requiring no processing from the agent side, but requiring some non-zero transmission time to the agent responsible of the clique to which the constraint function is allocated.

[2]The makespan is a concept more general but akin to the number of non concurrent constraint checks in DCOP literature since it explicitly takes into account communication delays

$CT(\tau_r)$. Our aim is then to find the junction tree, the allocation of constraints to cliques and cliques to agents that will minimise this makespan, such that:

$$\arg \min_{\mathcal{C},\alpha_\Psi,\alpha_\mathcal{C}} MS(\mathcal{C},\alpha_\Psi,\alpha_\mathcal{C})$$

Unfortunately, optimally scheduling a set of tasks onto a set of heterogeneous processors, even without considering communication, is known to be NP-hard [18]. Therefore, in this paper, we aim to design an effective heuristic that works well in practise.

4. RESOURCE-AWARE JUNCTION TREES

Here we present our approach to build junction trees and allocate computational tasks to agents in order to heuristically minimise the makespan of the coordination process. We first note that the minimisation problem stated in the previous section can be divided into two sub-problems: (i) finding a suitable junction tree decomposition of the problem, by defining cliques and allocating constraint functions to cliques; (ii) allocating the resulting computational tasks to agents to minimise the makespan. However, the two subproblems are interconnected because grouping of variables in cliques induced by the junction tree impacts on the computation and communication that agents need to perform. The approach we propose treats both subproblems at the same time, and it is divided into two key parts: a distributed protocol that implements variable elimination and a novel resource aware heuristic (RAH) that seeks to select variables in order to heuristically minimise the makespan of the coordination process.

Distributed Protocol for Variable Elimination

This protocol is a distributed negotiation protocol that extends the variable elimination algorithm by making use of calls for proposals (CFP) and bids in order to determine the next variable to eliminate and the agent responsible for the associated clique maximisation. Our protocol proceeds by each agent broadcasting one CFP for each variable it is responsible for. Each agent then replies to a CFP by estimating the makespan associated with itself being responsible for the computation of the clique that would be created if the variable specified in the CFP was eliminated. Given this description, we now formally define the key elements of our protocol.

• A *constraint* is a tuple $\langle f, \mathcal{X}_f, \psi_f, a_f \rangle$, where f is the identifier of the constraint, $\mathcal{X}_f \subseteq \mathcal{X}$ is the domain (or label) of the constraint function, $\psi_f : 2^{\mathcal{X}_f} \rightarrow \mathbb{R}$ is the constraint function and $a_f \in \mathcal{A}$ is the identity of the agent owning the constraint.

• A *variable* is a tuple $\langle X, \Gamma_X, \Gamma_X^{past}, a_X, c_X \rangle$, where X is the identifier of the variable, $\Gamma_X \subseteq \mathcal{X}$ is the set of neighbours of this variable in the constraint graph, $\Gamma_X^{past} \subseteq \mathcal{X}$ is the set of former neighbours that have already been eliminated, $a_X \in \mathcal{A}$ is the agent responsible for this variable and $c_X \in \mathcal{C}$ is the clique related to the elimination of this variable (initially void). For instance, in the Figure 1 variable X_2 is represented by the following tuple $\langle X_2, \{1,2,3\}, \{\}, 2, \emptyset \rangle$.

• A *clique* is a tuple $\langle c, \mathcal{X}_c, \Psi_c, \Gamma_c^\mathcal{C}, X_c, a_c \rangle$, where c is the identifier of the clique, $\mathcal{X}_c \subseteq \mathcal{X}$ is the domain (or label) of the clique, $\Psi_c \subseteq \Psi$ is the set of constraint functions allocated to the clique (initially void), $\Gamma_c^\mathcal{C} \subseteq \mathcal{C}$ is the set of neighbours in the junction tree (initially void), $X_c \in \mathcal{X}$ is the variable whose elimination led to the creation of c and $a_c \in \mathcal{A}$ is the identity of the agent responsible of the clique. For instance in the Figure 2 the clique $X_1X_2X_3$ is represented by the following tuple $\langle X_1X_2X_3, \{X_1, X_2, X_3\}, \{\psi_1, \psi_2, \psi_7\}, \{X_1X_3X_6\}, X_2, 1 \rangle$.

• An *agent* is a tuple $\langle a, \mathcal{X}_a, \Psi_a, \mathcal{C}_a, \Gamma_a^{com} \rangle$, where a is the identifier of the agent, $\mathcal{X}_a \subseteq \mathcal{X}$ is the subset of variables the agent owns, $\Psi_a \subseteq \Psi$ is the set of constraint functions allocated to the agent, $\mathcal{C}_a \subseteq \mathcal{C}$ is the set of cliques allocated to the agent (initially void) and $\Gamma_a^{com} \subseteq \mathcal{A}$ is the set of neighbours of the agent in the communication graph. For instance in the Figure 2, agent 3 is represented by the following tuple $\langle 3, \{X_3\}, \{\psi_3, \psi_6\}, \{X_1X_3X_6, X_3X_4X_6\}, \{1,2,3\} \rangle$.

• A *CFP* is initiated by one agent which proposes to another to take the responsibility of a clique (which is only *partially* created and initialised). Formally a CFP is a tuple $\langle s, c, d \rangle$, where $s \in \mathcal{A}$ is the sender agent, d is a date (the number of the bidding turn), and c is a clique, whose function (of exponential size) is not created. The initialised fields of c are Ψ_c - with the list of subfunctions that would be allocated to c, $\Gamma_c^\mathcal{C}$ - with the list of neighbouring cliques in the junction tree (this set is needed to determine the set of separators, which in turns define the scope of messages that would have to be transmitted or received by c and X_c - which identifies the variable for which the CFP has been sent.

• A *bid* is formally a tuple $\langle s, r, cfp, clique, v \rangle$ where $s \in \mathcal{A}$ is the sender agent, $r \in \mathcal{A}$ is the addressee agent, cfp is the CFP for which the bid is a reply, $clique$ is the identity of the clique to which the constraint functions of the clique of the CFP will be allocated and $v \in \mathbb{R}^+$ is an evaluation of the time it would take the agent s to compute the clique proposed by agent r in his CFP.

Algorithm 1 Variable selection and elimination ($selection$)

```
1: parallel { treat_CFP() }
2: date ← 0
3: while date < |X| do
4:   // compute variables' heuristic values
5:   send_CFP()
6:   treat_bids()
7:   // select the next variable to eliminate
8:   selected_bid ← consensus_select(best_bid)
9:   c ← selected_bid.cfp.c
10:  X ← X_c
11:  // allocate the clique
12:  if selected_bid.s = a then
13:    C_a ← C_a ∪ {c_X}
14:  end if
15:  // update the set of constraint
     functions
16:  if X ∈ X_a then
17:    Ψ_a ← Ψ_a \ Ψ_c
18:  end if
19:  // add new constraint dependencies
     between each pair of neighbours
20:  for Y ∈ X_a ∩ X_c do
21:    for Z ∈ X_c : Y ≠ Z do
22:      if Y ∉ Γ_Z then
23:        // extend variables' neighbourhood
24:        Γ_Z ← Γ_Z ∪ {Y}
25:        Γ_Y ← Γ_Y ∪ {Z}
26:      end if
27:    end for
28:  end for
29:  // update variables neighbourhood with
     respect to X
30:  for all X_a ∈ X_a do
31:    // remove variables' constraint edges
32:    Γ_{X_a} ← Γ_{X_a} \ {X}
33:    // memorise variables' past constraint
       edges
34:    Γ^{past}_{X_a} ← Γ^{past}_{X_a} ∪ {X}
35:  end for
36:  // reset the algorithm's local variables
37:  best_bid ← ∅
38:  date ← date + 1
39: end while
40: compute a maximum spanning tree and connect the cliques accordingly
41: start the GDL message passing inference algorithm
```

Having defined our terms we now present our protocol in algorithms 1-5. Variable elimination works first by computing a heuristic evaluation for each variable. This is done by concurrently executing the functions $treat\_CFP$ (line 1 of Algorithm 1), $send\_CFP$ and $treat\_bids$ (lines 5-6 of Algorithm 1). CFP are used in order to assess the impact on the makespan of the elimination of a variable. In order to do so a CFP contains the clique corresponding to the elimination of this variable. No clique is added to the junction tree until a consensus has been reached on the variable to eliminate (lines 9-14 of Algorithm 1), at which time only one clique is actually created.

• $send\_CFP$ computes and sends a CFP for each of the variable an agent is responsible for. The field Ψ_c is filled with the set of constraint functions[3] that would be allocated to this clique if actually created, and the set $\Gamma_c^{\mathcal{C}}$ is filled accordingly with the neighbourhood of this clique. Notice that this is a conservative estimation as the actual neighbours will be computed in Algorithm 1 (line 40). A CFP is compiled only with the information available in the direct neighbourhood of the variable in the constraint graph (lines 4, 6 and 8). The CFP is then propagated to all the agents.[4]

• $treat\_CFP$ waits for incoming CFP and calls upon receiving the procedure $RAH\_evaluate\_clique$. This function returns an heuristic evaluation of the impact on the makespan for this agent to compute the clique in the CFP. If the clique of the CFP has a domain which is a subset of one of the cliques associated with a variable of the agent, the evaluation is then set to 0, and the clique that would be created is set to this already existing clique. Otherwise, the value is left untouched and the clique that would be created is the one in the CFP (lines 4-9). This is done in order to enforce the use of maximal cliques only. The bid is then sent to the sender agent.

• $treat\_bids$ handles received bid messages. If an incoming bid has an evaluation lower than the current best bid, it is selected as the best bid (lines 5-8). The agent corresponding to the lowest evaluation is stored inside the bid data structure. When all the bids have been received, the flow of control returns to Algorithm 1.

• $selection$ is the function implementing the actual distributed variable elimination algorithms. Once the previous algorithms complete, it selects a variable to eliminate according to the heuristic evaluation computed (line 7). This step is made through a consensus, where each agent proposes the best bid it has received. Various algorithms such as a wave propagation algorithm [17] could be used in order to perform a consensus, however the simplest way is to propagate each message to all the agents. The variable and the clique are then extracted (lines 8-9). The agent selected by the winning bid adds the new clique to the set of cliques that it is responsible for (lines 11-13). The agent owning the selected variable first updates the set of its constraint functions by removing the ones allocated to the new clique (line 17). All the neighbouring agents of the selected variables update concurrently both their neighbourhood (lines 31-34) and the constraint neighbourhood of their variables (lines 24-25). It is necessary to do so in order to store both the deleted edges (used to compute the heuristic) and the full set of dependencies between variables (used to communicate with all the relevant agents). When all the variables have been eliminated, the cliques are connected together using a maximum spanning tree, which enforces the tree structure and the running intersection property of the junction tree [3]. This can be done efficiently in a distributed way using the approach of [6]. Finally, Algorithm 1 starts the GDL messages-passing algorithm (line 41). The root node is selected as the one in the middle of the diameter of the junction tree.

[3]The actual function is not transmitted as it is of exponential size. The heuristic can be computed by considering only its domain (see line 13 of Algorithm 5)

[4]While each agent might not be able to reach all other agents directly, messages will be propagated to all the agents possibly though multi-hop communication.

Algorithm 2 Outgoing CFP management ($send\_CFP$)

1: // compute and send CFP_X
2: **for all** $X \in \mathcal{X}_a$ **do**
3: // compute the domain of the clique
4: $\mathcal{X}_c \leftarrow \{X\} \cup \Gamma_X$
5: // compute the set of related constraint functions
6: $\Psi_c \leftarrow \cup_{Y \in \Gamma_X \cup \Gamma_X^{past}} \{\langle f, \mathcal{X}_f, \emptyset, a_Y \rangle : \psi_f \in \Psi_{a_Y}, \mathcal{X}_f \subseteq \mathcal{X}_c\}$
7: // compute the set of related cliques
8: $\Gamma_c^{\mathcal{C}} \leftarrow \cup_{Y \in \Gamma_X^{past}} \{c_Y\}$
9: // partially create the clique
10: $clique \leftarrow \langle c, \mathcal{X}_c, \Psi_c, \Gamma_c^{\mathcal{C}}, X, \emptyset \rangle$
11: // create the CFP
12: $CFP_X \leftarrow \langle a, clique, date \rangle$
13: *broadcast* CFP_X
14: **end for**

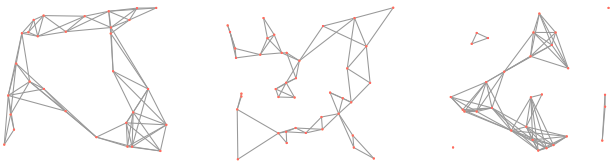
Algorithm 3 Incoming CFP management ($treat\_CFP$)

1: **if** $receive(cfp = \langle s, c, d \rangle) \wedge d = date$ **then**
2: $v \leftarrow RAH\_evaluate\_clique(c)$
3: // enforce the use of maximal cliques only
4: **if** $\exists X \in \mathcal{X}_a : \mathcal{X}_c \subseteq \mathcal{X}_{c_X}$ **then**
5: $clique \leftarrow c_X$
6: $v \leftarrow 0$
7: **else**
8: $clique \leftarrow c$
9: **end if**
10: $bid \leftarrow \langle a, s, cfp, clique, v \rangle$
11: *send* bid
12: **end if**

Algorithm 4 Incoming bid management ($treat\_bids$)

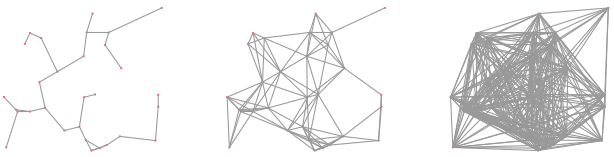
1: $bids \leftarrow \emptyset$
2: **while** $|bids| < |\mathcal{A}|$ **do**
3: **if** $receive(bid = \langle s, r, cfp, clique, v \rangle) \wedge r = a \wedge cfp.d = date$ **then**
4: $bids \leftarrow bids \cup \{bid\}$
5: **if** $best\_bid.v > v$ **then**
6: $best\_bid \leftarrow bid$
7: $v \leftarrow 0$
8: **end if**
9: // select the next variable
10: **end if**
11: **end while**

As there are a finite number of variables and one variable is always eliminated at the end of each turn, this algorithm will always terminate provided that there is no message loss. While we do not deal with such an issue here, we note that such issue could be addressed by using other consensus approaches [4] or specific communication protocol (such as for example TCP). Our protocol is fully distributed as an agent only needs to know the constraints it is involved with and the total number of agents in the system, at no moment does an agent know the full set of constraints or variables. In contrast, each agent executes the part of the variable elimination relative to their variables. The consensus protocol ensures that at each step, all the agents are synchronised on the identity of the eliminated variable. Therefore, our protocol is correct, and results in the same junction tree that would be created through a conventional centralized variable elimination using our resource aware heuristic.



(a) (b) (c)

Figure 3: Constraint graphs of (a) *ring*, (b) *tree* and (c) *cluster* instances



(a) (b) (c)

Figure 4: Three communication graphs of the *cluster* instance, where (a) r=100, (b) r=300 and (c) r=700

Resource Aware Heuristic

Algorithm 5 RAH heuristic ($RAH\_evaluate\_clique$(c : clique))

```
1: v ← X_c
2: eval ← +∞
3: a ← identity of the current agent
4: timeComp ← s(d(c))/speed(a)
5: // estimate trans for τ ∈ C (separators)
6: timeSep ← 0
7: for all c' ∈ Ψ_c do
8:    timeSep ← max(timeSep,sp(a,a_c',s(X_c ∩ X_c')))
9: end for
10: // estimate trans for τ ∈ Ψ (constraint functions)
11: timeSub ← 0
12: for all f ∈ Γ_c^C do
13:    timeSub ← max(timeSub,sp(a,a_f,s(X_f)))
14: end for
15: eval ← min(eval,timeComp+timeSep+timeSub)
16: return eval
```

The $RAH\_evaluate\_clique$ procedure takes as input a clique and gives an heuristic estimate of the impact on the makespan of the computation of the clique on the current agent. This is done by greedily allocating constraint functions to this clique and allocating the clique to one agent. In more detail, this procedure computes the sum of three values: the time to compute the task, the time to transfer the allocated constraint functions, and the time to transfer the messages of the execution constraints (lines 5-14 of Algorithm 5). The evaluation of the time to transfer messages is computed as the size of the message divided by the maximal bandwidth between the involved agents. This computation is done by the sp function (lines 8, 13 of Algorithm 5), where the function s returns the number of elements in the utility table representing a function given its domains. Note that this is an approximation since congestion in the communication network will impact this result, however as discussed in Section 3 we focus here on an effective heuristic approach rather than an optimal allocation which is known to be NP-hard [10].

Figure 2 depicts the results of our approach applied to the MAS coordination problem of Figure 1. The constraint function allocation are indicated beneath the cliques, and clique allocation to agents are indicated on top of each clique with the number of the responsible agent. The asterisk denotes the root. In this instance, agent 3 is responsible for two cliques as the heuristic estimates that the makespan could best be reduced by saving communication rather than exploiting distribution of computations.

5. EMPIRICAL EVALUATION

We empirically evaluate our RAH algorithm against two closely related state of the art distributed inference algorithms: (i) DPOP, whose pseudo-tree is built with a distributed DFS approach[5], and (ii) RDPI (Robust Distributed Probabilistic Inference), the initial junction tree construction of Paskin and Guestrin which builds a junction tree over the minimum spanning tree of the communication network [14]. We also compare against a centralised benchmarking approach, which generates a near-optimal junction tree using the standard variable elimination algorithm and the *minimum size* (MS) heuristic, and then allocates all tasks to the fastest agent in the system. We refer to this algorithm as MS.

We benchmark these algorithms on a set of three scenarios with different constraint network structures. Agents are located in a square of fixed size of 1000 unit and we consider three different topologies: rings, trees and clusters with 30, 40 and 30 agents respectively. For simplicity we consider that there are as many variables as agent; constraints are n-ary. Figure 3 shows the structure of the three constraint networks considered.

For each scenario we perform a set of experiments by varying the communication range (i.e., the distance within which two agents can communicate) in order to study the impact of the availability of communication resources on the coordination procedure. The communication range describes the availability of the communication resources. A communication range lower than 100 indicates that only a few communication links between neighbouring agents exist, while a communication range greater than 500 implies that each agent is roughly connected to at least half of the agents in the system. The wider is the communication range, the more likely it is to find direct high bandwidth communication links between any two agents. In the limit, such a case is equivalent to having no communication constraint at all.

Figure 4 represents the structure of the communication network for three different ranges of communication in the *cluster* instance. For each experiment (i.e. a fixed constraint and communication network structure) the agents' computational speed and links' bandwidth are randomly drawn from the set {2,4,8}.

Results

We measure the makespan and the *treewidth* (the size of a maximal clique [7]) for each algorithm. The makespan is empirically computed on a simulation environment matching the full characteristics described in Section 3 (i.e., blocking communications, non multitask agents and non preemptive tasks) using the same routing policy for all the algorithms (except RDPI which has its own). Note that in these simulations we include the full effect of network congestion. The unit of makespan measurement is the time step of the simulator. For each experiment we performed 15 runs, and we report the mean and the standard error of the mean in Figure 5. The treewidth measurements are reported in Table 1. In both case r notes the communication range parameter.

In the *ring* and *cluster* scenario, the resource aware heuristic performs up to three times faster than DPOP when resources are

[5]We use here the most connected node (MCN) heuristics, which as discussed in [15] drives the DFS to obtain pseudo-trees with low treewidth

scarce (communication range below 100). The performance then stabilises (two times faster than DPOP) when the communication range increases. In the *tree* scenario, because the communication and constraint networks are both tree structured and generated according to a distance measure between agents, communication along the DPOP pseudo-tree matches the communication network more closely than in other instances, where the agent responsible for neighbouring cliques are less likely to be neighbours in the communication graph. As a result the performances of RAH and DPOP are similar until the point where communication is no longer a scarce resource (communication range of 400) where DPOP is able to perform better. These results show the importance of taking into account the differences between the communication and constraint networks in DCOPs.

The comparison between RAH and MS shows that the difference in performance is not only highly contrasted but is reversed when communication is no longer a scarce resource. Specifically in such a case MS is able to perform better than the resource aware heuristic we propose as all the agents are able to directly communicate with fast communication links with the centralising agent, indicating that in this case centralising the solution is more efficient. Conversely when resources are highly constrained the RAH is able to perform up to 3 times faster than MS.

The makespan obtained with the RDPI algorithm was extremely high (around two orders of magnitude higher than MS) in our settings, and are thus not reported in the Figure 5. This is related to the high treewidth (see Table 1) of the junction trees which is up to six times the treewidth of a near-optimal junction tree and two to three times larger on average.This is due to the fact that RDPI forces the junction tree to be built on top of a spanning tree of the communication network and this can result in junction trees with very large treewidth. In order to tackle this problem, Paskin proposes to use simulated annealing in order to optimise the junction tree. However, such a procedure requires an expensive distributed evaluation procedure in order to evaluate the cost of a local move and an unbounded number of messages [13]. As we focus here on the efficiency of junction trees that can be obtained with simple preprocessing techniques, we only report for RDPI the performance of the initial junction tree. The reported experimental evidences suggest that the cost of RDPI when the optimisation procedures converge, is within a factor of two of hypothesized optimal junction tree, which was built using an off-line centralised procedure, while the initial tree is up to seven times worst than that. However, notice that in our experiments RDPI results were orders of magnitude worst than competitors.

Furthermore notice that the treewidth for MS and RAH are very similar (see Table 1) but RAH clearly outperforms MS when communication is scarce (see Figure 5). These results again show the importance of taking into account agents' communication and computation capabilities when building the junction tree.

Summarising, our results show that while the treewidth of the junction tree remains an important parameter as it has an exponential impact on the efficiency of the algorithm, junction trees with higher treewidths can still result in better overall performances in such heterogeneous distributed settings if computations are appropriately scheduled across agents.

Complexity

While the running time depicted in Figure 5 only shows the relative performance of the different junction trees, it is important for real-world applications to also take into account the distributed running time of the preprocessing steps of all those algorithms. We discuss here the complexity, in terms of number of messages exchanged for the different approaches. For ease of notation, let us assume there

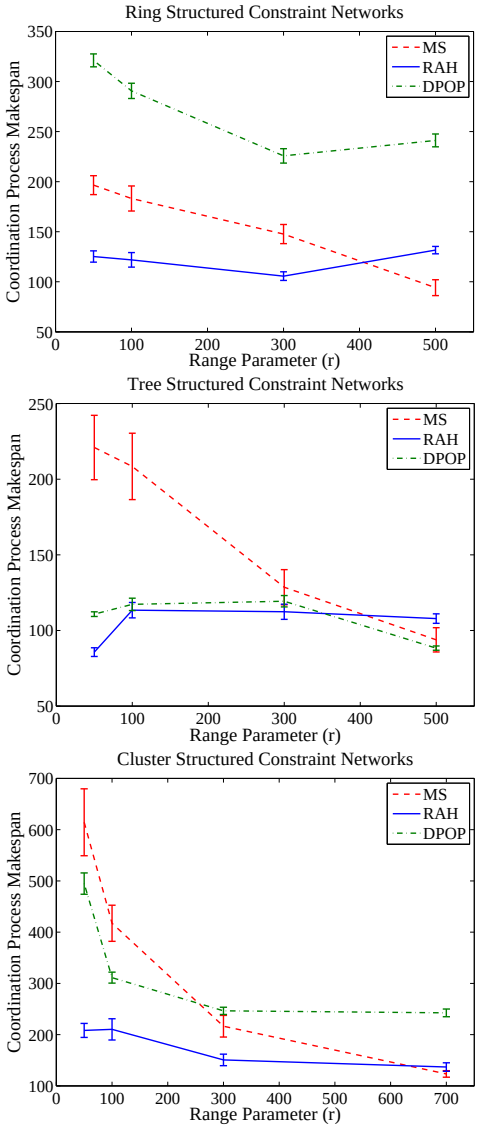


Figure 5: Coordination process makespan for the a) *ring* b) *tree* and c) *cluster* structured constraint graph instance

	r	MS	RAH	DPOP	RDPI
cycle	50	5	5.8 ± 0.1	6	10.4 ± 0.3
	100	5	5.4 ± 0.1	6	8.8 ± 0.2
	300	5	5.2 ± 0.1	6	11.1 ± 0.5
	500	5	5.1 ± 0.1	6	19.4 ± 0.5
tree	50	4	4.4 ± 0.1	5	13.3 ± 0.7
	100	4	4.4 ± 0.1	5	18.5 ± 0.7
	300	4	4.2 ± 0.1	5	20.6 ± 1.0
	500	4	4.1 ± 0.1	5	20.6 ± 1.0
cluster	50	7	7.0 ± 0	7	8 ± 0
	100	7	7.4 ± 0.1	7	8 ± 0
	300	7	7.1 ± 0.1	7	13 ± 0.7
	700	7	7.0 ± 0	7	24.2 ± 0.5

Table 1: Benchmarked treewidths

are as many agents as variable, where n is this number, tw is the treewidth, and that each agent possesses exactly one variable.

In terms of number of messages, DFS exploration uses $2n$ messages, and the MCN heuristic uses tw messages at each steps, yielding a number of messages for our DPOP implementation in $O(ntw)$. RDPI uses $O(n \log n)$ messages in order to build a spanning tree and then $2(n-1)$ messages in order to build the junction tree on top of it, yielding a number of messages in $O(n \log n)$. The MS algorithm is centralised and therefore each agent sends to the centralising agent its information regarding the variables and variables neighborhood, requiring the exchange of n messages. Finally the number of messages of our approach is the following. During the step $0 < k < n$ of Algorithm 1 $n-k$ CFP and $n(n-k)$ bids are sent, yielding a total number of messages for RAH in $O(n^3)$.

While the number of messages of our algorithm is higher than the others, as the results show, our approach can yield better running time for the DCOP solving algorithm for the solution phase. For real applications the measure we are interested in is the combined running time of the preprocessing phase and the actual solution phase. Such a running time depends on various parameters, the number of variables n, the tree-width tw and the cardinality d of the variables and also depends on the computation and communication capabilities of the multi-agent system. Now, the number of messages of the preprocessing phase for RAH is higher than competitors, but notice that messages sent in this preprocessing phase are of fixed size with respect to tw and d, while the complexity of the junction tree solution phase is exponential in tw with a basis of d. Therefore, depending on the values of the above parameters, the time required to send the messages for the preprocessing phase can be negligible with respect to the gain obtained in the running time for the solution phase. For instance if we consider the *cluster* experiment, we have $n = 30, tw = 7, d = 2$. The maximal time to compute a clique in such a case is $2^7 = 128$ times steps, while the RAH algorithm needs to exchanges $27 \cdot 10^3$ messages. However, if we consider $d = 10$, the complexity of the junction tree solution phase become prevalent with $10 \cdot 10^6$ times steps while the number of messages sent by RAH does not change.

Thus, depending on the settings of a coordination problem our algorithm can provide substantial gains in terms of total running time despite having a preprocessing overhead greater than the ones currently used in DPOP, MS and RDPI.

6. RELATED WORK

The use of junction trees (and other related graphical models) for solving DCOPs and the development of distributed approaches for junction tree compilation is a recent research topic that is gaining increasing attention. For example, Xia and Prassana use a distributed junction tree creation algorithm based on a DFS tree and propose to select the root so as to minimise the makespan [19], Otten and Dechter propose an heuristic for graphical models based on problem size measure that aims at load balancing efficiently the junction tree inference on as set of processors [11], Allouche and al. explicitly consider the problem of using distributed variable elimination in order to solve hard constraint optimisation problems [2]. However, none of these approaches address the problem from a MAS perspective, and as such they do not consider heterogeneity of computation and communication and they do not focus on having a distributed approach

7. CONCLUSION

In this work we take a first important step to explicitly consider multi-agent system specific issues (such as heterogeneity of computation and communication across the agents) when applying solution techniques developed in the graphical model community to decentralised constraint optimisation.

Specifically, we show the importance of taking into account the actual resources of a multi-agent system when solving combinatorial optimisation problems across it, and validate our approach on benchmark coordination problems

Future work are divided in two directions. The first is to empirically validate our approach on a deployed wireless sensor network. The second aims to investigate bounded approximate algorithms. Addressing the trade-off among communication, computation and the bound that can be provided on solution quality.

8. REFERENCES

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

[2] D. Allouche, S. de Givry, and T. Schiex. Towards parallel non serial dynamic programming for solving hard weighted csp. In *Principles and Practice of Constraint Programming*, volume 6308 of *Lecture Notes in Computer Science*, pages 53–60, 2010.

[3] J. R. S. Blair and B. Peyton. An introduction to chordal graphs and clique trees. *Institute for Mathematics and Its Applications*, 56, 1993.

[4] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, 1996.

[5] R. Dechter and J. Pearl. Network-based heuristics for constraint-satisfaction problems. *Artif. Intell.*, 34(1):1–38, 1987.

[6] R. G. Gallager, P. A. Humblet, and P. M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. Program. Lang. Syst.*, 5(1):66–77, 1983.

[7] K. Kask, R. Dechter, J. Larrosa, and A. Dechter. Unifying tree decompositions for reasoning in graphical models. *Artif. Intell.*, 166(1-2):165–193, 2005.

[8] R. Mailler and V. Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *Proc. of the 3rd Int. Conf. on Autonomous Agents and MultiAgent Systems*, pages 438–445, 2004.

[9] P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, (161):149–180, 2005.

[10] A. Moukrim and A. Quilliot. Scheduling with communication delays and data routing in message passing architectures. In *Parallel and Distributed Processing*, volume 1388 of *Lecture Notes in Computer Science*, pages 438–451, 1998.

[11] L. Otten and R. Dechter. Towards parallel search for optimization in graphical models. In *Proc. of the 11th Int. Symposium on Artificial Intelligence and Mathematics*, 2010.

[12] P. Padhy, R. K. Dash, K. Martinez, and N. R. Jennings. A utility-based sensing and communication model for a glacial sensor network. In *Proc. of 5th Int. Conf. on Autonomous Agents and Multi-Agent Systems*, pages 1353–1360, 2006.

[13] M. A. Paskin. *Exploiting locality in probabilistic inference*. PhD thesis, University of California at Berkeley, 2004.

[14] M. A. Paskin and C. E. Guestrin. Robust probabilistic inference in distributed systems. In *Proceedings of the 20th Conf. on Uncertainty in Artificial Intelligence*, pages 436–445, 2004.

[15] A. Petcu. *A Class of Algorithms for Distributed Constraint Optimization*. Phd. thesis no. 3942, Swiss Federal Institute of Technology (EPFL), 2007.

[16] R. Stranders, A. Farinelli, A. Rogers, and N. Jennings. Decentralised coordination of mobile sensors using the max-sum algorithm. In *Proc. of the 21st Int. Joint Conf. on Artificial Intelligence*, pages 292–298, 2009.

[17] G. Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, 2000.

[18] H. Topcuoglu, S. Hariri, and M.-Y. Wu. Task scheduling algorithms for heterogeneous processors. *Proc. of Heterogeneous Computing Workshop*, pages 3–14, 1999.

[19] Y. Xia and V. K. Prasanna. Parallel exact inference on the cell broadband engine processor. In *Proc. of the 2008 ACM/IEEE Conf. on Supercomputing*, pages 1–12, 2008.