Distributed tree decomposition with privacy

Vincent Armant, Laurent Simon, Philippe Dague

Distributed tree decomposition with privacy

Vincent Armant, Laurent Simon, and Philippe Dague

Laboratoire de Recherche en Informatique, LRI, Univ. Paris-Sud & CNRS, Orsay, F-91405

Abstract. Tree Decomposition of Graphical Models is a well known method for mapping a graph into a tree, that is commonly used to speed up solving many problems. However, in a distributed case, one may have to respect the privacy rules (a subset of variables may have to be kept secret in a peer), and the initial network architecture (no link can be dynamically added). In this context, we propose a new distributed method, based on token passing and local elections, that shows performances (in the jointree quality) close to the state of the art Bucket Elimination in a centralized case (*i.e.* when used without these two restrictions). Until now, the state of the art in a distributed context was using a Depth-First traversal with a clever heuristic. It is outperformed by our method on two families of problems sharing the small-world property.

1 Introduction

Tree decomposition was introduced in [21]. It aims at mapping the graphical representation of a problem into a tree such that all linked variables in the initial graph stay linked (directly, or indirectly) in any node (also called *cluster* of the new tree). Once decomposed, the solving time of the initial problem can be bounded for a large class of problems. This nice property explains why tree decomposition has been widely studied, in many centralized applications (graph theory [21], constraints optimization [16, 5, 15], planning [13], databases [11, 6], knowledge representation [14, 22, 10]), but also in distributed ones (distributed constraints optimization [19, 3, 7, 17], ...). Intuitively, the decomposition guides the reasoning mechanism through the network, and can bound the number of messages. When dealing with distributed systems, it is indeed essential to bound the maximal number of messages, for instance to ensure some quality of services.

When the problem is tree-decomposed, its complexity can be bounded by an exponent of its *width*, which is the size of the largest cluster in the tree (minus 1). Many applications rely on good tree decompositions, and many polynomial classes are based on the existence of a good decomposition. Because of its exponential impact on the bound, even a small improvement in the quality of the decomposition may lead to large improvements in practice.

However, when the system is intrinsically distributed, or subject to privacy settings, no peer can have a global view of the system, and new algorithms must be explored. For instance, adding a link between two peers may not be feasible (only already existing links can be allowed). In all previous approaches, the initial distributed system was supposed to fulfil an additional strong characteristic: two peers that share a common variable must be directly linked by the network.

In this paper, we propose to explore the general case, *i.e.* when links between peers are not forced to follow the above characteristic. In this more general case, we propose a new distributed algorithm for distributed tree decomposition, based on local elections with a token. Using a token was necessary to prevent concurrent decisions, which was identified as one of the main issues for good jointree construction. We conclude this paper by an experimental analysis of our algorithm on families of small-world networks, and show that the produced jointrees are significantly better than state of the art distributed algorithms, while allowing more general networks.

2 Distributed Tree Decomposition

Introduced By Robertson and Seymour [21], the tree decomposition was applied in many problems. It was noticeably used in circuit diagnosis in [9], in the centralized case. In this work, Dechter and El Fattah considered a graphical model of the set of equations describing the circuit. Each node of the graph is labelled by a variable, and each edge models a semantic dependency between variables in the same equation.

Figure 1 shows a problem described by a conjunction of f_i formulae signatures (e.g. constraints, properties, theories). On the right we show the corresponding interaction graph s.t. each node is labelled by a variable and there exists a link between two nodes iff they appear in the scope of a same function. For example, in light grey, we surround parts of the interaction graph modeling f_1, f_2 and f_3 of the initial problem.

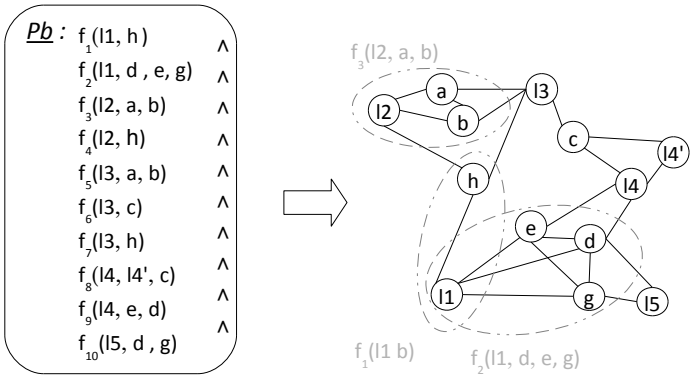


Fig. 1: Left: Centralized problem description. Right: The corresponding problem interaction graph

The tree decomposition of an interaction graph is a tree of clusters of variables having the running intersection property and preserving the initial dependency schema of variables. Here we recall the definition of [21], keeping in mind that a node corresponds to a variable.

Definition 1 (tree decomposition [21]). *A tree decomposition of a graph G is a pair (χ, T) in which $T = (CT, F)$ is a tree where the nodes $ct_i \in CT$ are clusters names*

and the edges F model the inter-dependencies between clusters, and $\chi = \{\chi_{ct_i} : ct_i \in CT\}$ is a set of subsets of $vertices(G)$ s.t. each subset χ_{ct_i} is the set of vertices of the cluster ct_i. The tree decomposition fulfils the following properties:

1. $\bigcup_{ct_i \in CT} \chi_{ct_i} = vertices(G)$,
   All nodes in the initial graph are at least in a cluster, and the tree contains no new variables ***(vertices compliance)****.*
2. $\forall \{x, y\} \in Edges(G), \exists ct_i \in CT$ *with* $\{x, y\} \in \chi_{ct_i}$
   Each pair of variables connected by an edge in the initial graph must be found together in a cluster ***(dependencies compliance)****.*
3. $\forall ct_i, ct_j, ct_k \in CT$, *if* ct_k *is on the path from* ct_i *to* ct_j *in* T, *then* $\chi_{ct_i} \cap \chi_{ct_j} \subseteq \chi_{ct_k}$,
   Two clusters that contain the same node are connected by clusters that also contain this node ***(running intersection)****.*

The result of a tree decomposition is also called a jointree. Figure 2 represents the tree decomposition of the interaction graph of Figure 1. In this figure, one can check that each node labelled by a variable in the interaction graph belongs to at least one cluster of the tree-decomposition. In addition, we can notice that clusters $ct_2, ct_5, ct_7, ct_6, ct_3$ that contain $l1$ are connected in the tree. Then, the running intersection property is satisfied for $l1$. It is easy to check that the running intersection is satisfied for all variables.

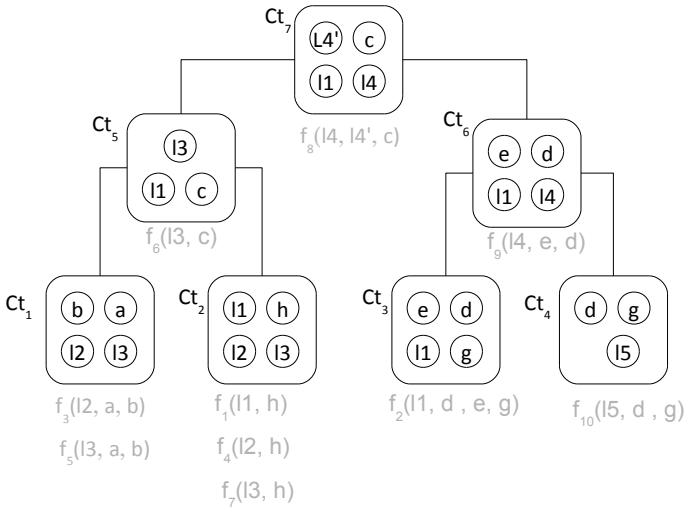


Fig. 2: Tree decomposition of the interaction graph of Figure 1

2.1 Distributed Theories and Privacy

Let us point out that, in the interaction graph, each variable is directly connected to all other variables that appear with it in at least one formula. Now, in a distributed context,

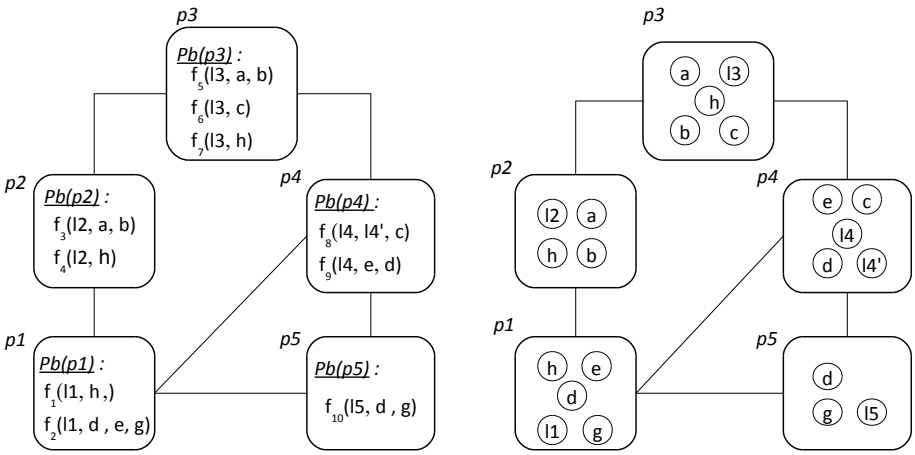


Fig. 3: Left: distributed problem of figure 1. Right: Its acquaintance graph

the framework is made up of peers, where each peer knows a subset of formulae and can interact with its neighbourhood by messages in order to solve a global problem. In that case we need to consider at the same time the network interactions between peers defined by peers' acquaintances and the semantic interactions between variables defined by formulae (from a global point of view). In our approach, we model a distributed setting of *peers* and formulae by the notion of acquaintance graph.

Definition 2 (acquaintance graph). *Given a distributed problem setting defined on variables $Vars$, an acquaintance graph $G((P, V), ACQ)$ is a graph defined by:*

1. *$P = \{P_i\}_{i=1..n}$ is the set of peers,*
2. *$V : P \rightarrow 2^{Vars}$ is the node labelling function where $V(p_i)$ represents the vocabulary used by p_i to describe its problem.*
3. *$ACQ \subseteq P \times P$ defines the peers' acquaintances i.e. the neighbourhood with which each peer can exchange messages.*

This definition of an acquaintance graph differs from [1], which represents a multi-graph where edges are labelled by shared variables.

Figure 3 (right) shows an example of an acquaintance graph of a distributed theory (left). We can emphasize that the acquaintance graph differs from the interaction graph. p_1 and p_3 have in common the variable h, but they do not share it in the network via a direct link like it would be the case in an interaction graph. Let us also point out that, without loss of generality, we can assume that two peers sharing an acquaintance link share also one or several variables. In addition, **we will not allow the creation of new acquaintance links** once the acquaintance graph is given. This is a first restriction for building a tree decomposition. One may also notice, figure 3, that a particular set of variables (written l_i), are only in one peer. We will consider these variables "local", and the other ones "shared". The privacy rule of our framework ensures that **local variables stay local**, *i.e.* no local variable is sent to any other peer in the network. This

is our second restriction for building a tree decomposition. We can notice that the tree decomposition shown in figure 4 does respect privacy while the one in figure 2 does not.

2.2 Distributed Tree Decomposition

Because an acquaintance link between two peers may not follow the interaction graph of variables, we need to adapt Definition 1 for the distributed context.

Definition 3 (Distributed Tree Decomposition (DTD)).
Let $G((P, V), ACQ)$ be the acquaintance graph of a distributed system. A tree of clusters $T((CT, \chi), F)$ s.t. each cluster $ct \in CT$ is labelled by a set of variables $\chi(ct)$, is a distributed tree decomposition of G iff:

1. $\bigcup_{p \in P} V(p) = \bigcup_{ct \in CT} \chi(ct)$ ***(compliance of the vocabulary)***
2. $\forall p \in P, \exists ct \in CT$ *s.t.* $V(p) \subseteq \chi(ct)$ ***(compliance of variables dependency)***
3. $\forall ct, ct', ct" \in CT$, *if* ct' *is on the path from* ct *to* $ct"$ *in* T, *then* $\chi(ct) \cap \chi(ct") \subseteq \chi(ct')$ ***(running intersection)***
4. *There exists a function* $\gamma : CT \rightarrow P$ *that represents the cluster provenance s.t.* $\forall \{ct, ct'\} \in F$:
   - $\gamma(ct) = \gamma(ct')$ *or*
   - $\{\gamma(ct), \gamma(ct')\} \in ACQ$

   Each cluster is hosted by one peer. Neighbouring clusters are hosted by the same peer or by neighbour peers in the acquaintance graph. ***(compliance of acquaintances)***
5. $\forall v \in V(p)$ *s.t.* $\forall p' \neq p, v \notin V(p')$, $\forall ct$ *s.t.* $v \in \chi(ct)$: $\gamma(ct) = p$
   Local variables belonging to p *can only appear in a cluster hosted by* p. ***(privacy rule)***

In the above definition, the first three properties are similar to the classical tree decomposition. If the later preserves the initial set of nodes and their dependencies, in our definition, the distributed tree decomposition of an acquaintance graph preserves the set of peer variables and their dependencies. We also need to add the fourth one, to adapt our DTD definition to the distributed case. The compliance of acquaintances also forces DTDs to have the following property: a cluster is hosted by one (and only one) peer, and a peer hosts at least one cluster (if its formula is not empty), and possibly many. In addition, all interactions between clusters are following the initial peer acquaintance. Our privacy rule, in the context of DTD is expressed by the fifth property. All clusters hosted by one peer p can contain any shared variable from any peer, but only local variables of p itself. If we look at figure 4, we see that all variables l_i stay at their initial peer. At the opposite, shared variables h and g are sent to p_4 for ensuring the running intersection.

2.3 Tree Decomposition: choose your father

Tree decomposition aims at directing the reasoning during search, or fixing a bound on the reasoning task. Then, the complexity of most classical reasoning tasks is bounded by the size of the largest obtained cluster. There is a strong link between the problem of finding a good decomposition, a good forgetting order, and the graph triangulation problem. However, in a distributed approach, only orders based on depth-first (DFS) and breadth-first (BFS) searches have been proposed so far. Building a distributed decomposition tree by BFS was first proposed by Elzahir et al. [7] for distributed CSPs. GrunBach and Wu [12] proposed to apply the tree decomposition for software verification, based on a BFS. However, we experimentally checked (not reported here) that the quality of the decompositions by BFS order was significantly worse than those based on DFS. So, we limit this section to the presentation of DFS-based orders and Bucket-Elimination (BE) orders.

Distributed, Depth-First, Tree Decomposition In distributed constraints optimization, Distributed DFS (DDFS) is the state of the art [19, 8]. Initially, a peer is chosen as the root of the tree (this is a global decision). It chooses one of its neighbours and sends it a special message. When a peer receives such a message for the first time, it records the sender as its father. Then, it forwards this message to one of its neighbours that have not yet been chosen by it. If all its neighbours have been visited, then it sends the return message to its father. The DDFS algorithm sends messages through all edges of the network of peers. Its complexity (in the number of non concurrent messages) is $2 * |E|$, where E is the set of edges.

Distributed Bucket Elimination Bucket Elimination is a general method that can be used for decomposition. It builds a tree of clusters from the leaves to the root of the tree. When applied with the DFS order, BE does not have to build the tree, it follows the DFS one: DFS induces a total order on viewed peers s.t., for each peer, each of its neighbours is an ancestor or a descendant. Initially, an empty cluster χ_{ct_p} is associated to each peer; then, for each peer p_i, if it is a descendant of some peer p_j then the vocabulary common to the two peers p_i, p_j is added to the cluster $\chi_{ct_{p_i}}$. Then, the algorithm typically proceeds bottom-up in two stages: (1) a peer p_i sends to its father p_j a projection $pjct_{p_i}$ of the variables contained in $\chi_{ct_{p_i}}$ and belonging to one of its ancestor; (2) when p_j receives $pjct_{p_i}$ from p_i, it adds $pjct_{p_i}$ in its cluster $\chi_{ct_{p_i}}$. When it has received a projection from all its children it projects to his father the variables of $\chi_{ct_{p_j}}$ shared with one of its ancestors. For more details, the piece of work [3] assumes a DTD based on Cluster Tree Elimination [4] applied to a decentralized context.

We show the jointree built by Distributed BE with DDFS and the Maximum Cardinality Set (MSC) heuristic in figure 4. In the latter, privacy is preserved. One may notice here that variables are shared among peers if needed. The notion of projection ($pjct_{p_i}$ in the above algorithm) will also appear in our algorithm. This notion is essential for ensuring the running intersection property in the final jointree.

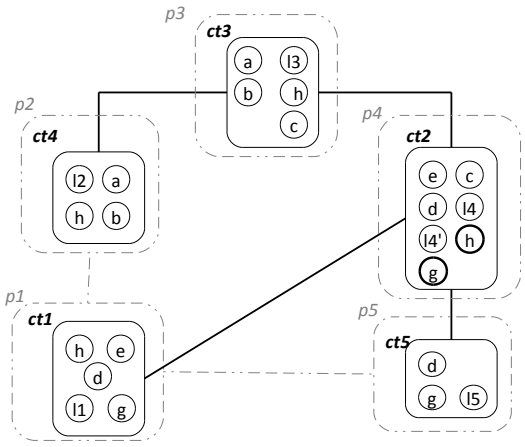


Fig. 4: Tree Decomposition given by BE and driven by DFS and MCS heuristic

2.4 Good Properties of Centralized Methods

Centralized methods often rely on equivalent problems for tree decomposition, such as graph triangulation. In this problem, one wants to add as few edges as possible to a given graph in order to obtain a chordal graph (all cycles of length four or more have a chord, which is an edge joining two non-adjacent nodes in the cycle). The triangulation algorithm iteratively eliminates all nodes. At each iteration, (1) a variable is chosen; (2) edges are added between pairs of nodes from its neighbours that are not already connected (clique-fill); (3) the node is removed. The quality of a triangulation can be trivially measured by the number of new edges. The node chosen at step (1) must add as few edges as possible, in a short and in a long-term point of view. For instance, a node in a clique with its entire neighbours is a very good candidate. A node with a poorly connected neighbourhood is a bad choice. The heuristic *Min-Fill-In* is based on this very simple observation. Each node is evaluated w.r.t. the number of new edges one may have to add to fill its neighbourhood as a clique.

In order to get a global elimination order from the above algorithm, we must add a fourth step: a cluster containing all the neighbourhood of the removed variable is built (and memorized) at the new step. In this context, the Min-Fill-In algorithm can be viewed as trying to reduce the size of the clusters. However, when adapting this idea to the distributed case, it may not be possible to add links to any pair of peers. In our formulation, the acquaintance graph is given, and no network link can be added "on the fly".

About concurrency, it is trivial to say that centralized methods are not concurrent. However, we think that the non-concurrency property is essential to obtain good join-trees. Let us consider a simple path $v_1 - ... - v_n$. One can check that the sequential elimination of one or two variables (i.e. (v_1, v_n)) at ends of the path will lead to the creation of clusters with at most two variables. Unfortunately, the concurrent eliminations of at least three variables at the same time will lead to the creation of a cluster of at least three variables. This example can easily be generalized for tree structured graph

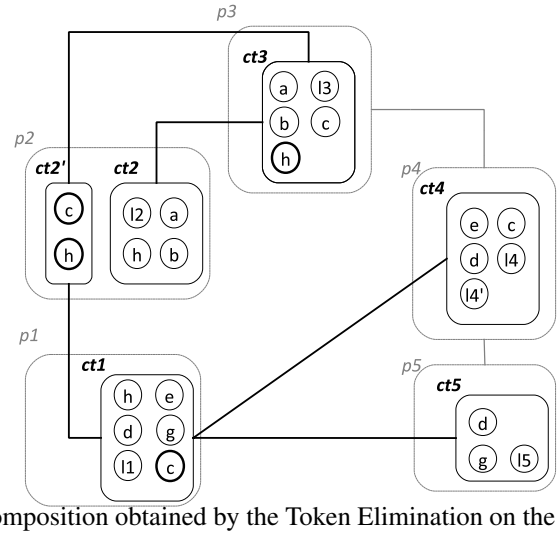


Fig. 5: Tree Decomposition obtained by the Token Elimination on the running example

or cyclic graph, and clearly illustrates the fact that even few concurrent decisions can get any algorithm away from the optimal decomposition. In order to prevent those bad decisions to be taken, we propose to use a token to limit concurrent choices.

3 Distributed Token Elimination

The idea of the algorithm is the following. Each peer votes for its best neighbour (including itself). It has to decide, locally, which peer should be removed first. A peer that has been chosen by all its neighbours can be removed (it is a sink node in the graph of "best" choices). When removed, a peer builds a cluster of variables for the distributed jointree, and memorizes it locally. This peer will be the root of a sub-jointree in the final jointree. All the variables that are not local to this sub-tree are in its *projection* (exactly in the same sense as it is used in BE, see 2.3). A removed peer remains active, and participates to the elections until all its neighbourhood has also been removed (after that, it will only be active for routing token messages).

In order to build a distributed jointree (DJT) at the end of the algorithm, we consider the first removed peers as the leaves of the DJT. The final DJT is built bottom-up to the root. This is done when all the peers have been eliminated. The algorithm has to connect all the sub-jointrees together. Because we do not want to allow any new link in the network, while ensuring the running intersection, this stage is a little bit tricky. Each peer asks the network for its son by flooding its name and its projection, and when the son answers, all peers on the path between them create new "connection" clusters with all the variables needed for ensuring the running intersection property in the final DJT. So, we have the following properties: First, during the whole algorithm, each cluster is only in one peer. Second, before this last reconnection stage, each peer can only contains one cluster (called the "main" cluster). Third, at the end of the algorithm, each

peer contains exactly one "main" cluster, and an unbounded number of "connection" clusters.

An illustration of the final solution of our algorithm is shown in figure 5. One can especially notice that privacy is preserved, and that clusters, thanks to connection clusters, are connected on the top of the initial peers network (see the $P2$ clusters).

Additionally, as reported section 2.4, we need to carefully handle concurrent decisions. For this, a single token is circulating on the network, following the current votes. Because the directed graph of "best" choices is not guaranteed to be strongly connected, we must add an escape strategy. For this, we suppose a cycle that goes through all nodes exactly once. In our approach, we build this cycle by a DFS traversal of the peers.

3.1 Data Structures and Notations

The $Score$ of a peer is the estimated size of its main cluster (the algorithm tries to choose the next peer with minimal score). In addition, each peer maintains an array $score[]$ of the "cluster" score of its neighbours. This "cluster" score takes into account the current projections of orphan sub-jointrees. More generally, this is where the heuristics take place.

The $Token$ circulating in the network is also decorated with precious information. It contains, intuitively, the set of current orphan sub-jointrees that are being built, rooted in removed peers so far. To allow any peer in the network to smartly choose which sub-jointree to attach as a son, each sub-jointree also contains its projection, *i.e.* all the non local variables of the sub-jointree.

$neighbours*$ is the set of all the neighbours of a peer, including removed peers, and itself. $neighbours+$ is the set of all the neighbours of a peer, including itself, but not including removed peers. $best$ is the "best" neighbour of a peer p, *i.e.* the peer having the smallest value in the $score[]$ array, including p. $Children$ is the set of direct children a peer has registered so far.

3.2 The Distributed Token Elimination Algorithm

DTE is given as Algorithm 1. Initially, the token is sent to one peer.

Local Elections Each time a peer p_{token} receives (or initially has) the token (line 2), it organizes local elections (line 4) and waits for the votes of all its neighbours (including removed ones). The reception of this message is treated in line 31. The computed score of p takes into account the size of the set of variables made up from current projections in the token (set of non local variables of sub-jointrees built so far), local variables and shared variables with ancestors. The underlying idea of this score is to estimate the size of the cluster if p, the peer that received the $ELEC$ message (line 31), would be removed at this point. The computed score of the peer p is sent back to p_{token} but also to all its other neighbours, line 33, and received line 37. So, if we get back to the initial peer p_{token} that organized the local election, it has updated its own $score[]$ values for all its neighbours, taking into account the current open projections in the token, and the local election is closed.

One may notice here that $score[]$ arrays have been updated only in the neighbourhood of p_{token}, and thus the votes of the neighbours of p_{token} are based only on partial information: if p is a neighbour of p_{token}, p only updates the scores of its neighbours that are also connected to p_{token}. All other peers are ranked according to an old value of $score[]$, that can be based on an old set of orphans sub-jointrees. At some point, we must accept this, because this is just a heuristic, and trying to have a better estimation will cost too much. However, one may also notice here that if no peer is elected, the token remains the same, and follows the "best" links or the global cycle, eventually touching all the peers at the end of the cycle. The algorithm will end because once a peer has updated its own score according to the token, if the token does not change (no peer is removed), then its score will remain the same. If the token touches all the peers, all $score[]$ values will have been updated according to the same orphan sub-jointrees (this remark is important for termination, we have the guarantee to find a node to remove after at most one cycle, and in any case after visiting at most two times each edge).

At this step, after line 5, the peer p_{token} has now two possibilities, removing itself (see section 3.2), or giving the token to its best neighbour/cycle (see section 3.2).

Elected and Removed When p_{token} is a local minimum, *i.e.* all its neighbourhood, including itself, has elected it, then it has two steps to perform before being removed from the network. First, it needs to compute the new projection, and register itself as a new potential orphan sub-jointree with this projection.

This is done from lines 9 to line 17 in the algorithm. Pj_p is the projection of p over its alive neighbourhood. $Children$ is the set of peers that p has chosen as sons: all the peers that are roots in the current orphan sub-jointrees with which p shares at least one variable (these allow one to preserve the running intersection). New_χ is the set of variables in the projections of orphan sub-jointrees that p has to take into account when it will be removed, in addition to its own projection variables. So, for the new sub-jointree rooted in p, its projection χ_p is computed in line 12. Now, p has to update the set of sub-jointrees attached to the token. First, it removes its sons, then it adds itself as a new sub-jointree. At last, it is important that p tells its neighbourhood it has been removed (sending and acknowledging $ELIM$ messages). During this stage, it is also important to notice that all neighbours of p update their $best$ values (including p itself).

We postpone the explanation of lines regarding the reconnection of sub-jointrees to the end of the section.

Sending the Token The idea of lines 24 to 29 is to let the token following the "best" links in the graph, and rely on the static DFS order for escaping. So, if p is not its best choice (not in a local minimum), then, if p still has alive neighbours, it sends the token to the best of its alive neighbours. Otherwise, because $neighbours^+$ initially contains p itself, p has been removed with all its neighbours. In this case, we rely on the static cycle built on the top of DFS to send the token and escape.

If p was its own best choice, we are in a local minimum, and we also need to escape. Observing the fact that at least one of the alive neighbours is not considering p as the best choice (if p was the best choice for all its neighbourhood, then it has just been eliminated, and has sent the $ELIM$ messages to its neighbourhood forcing its

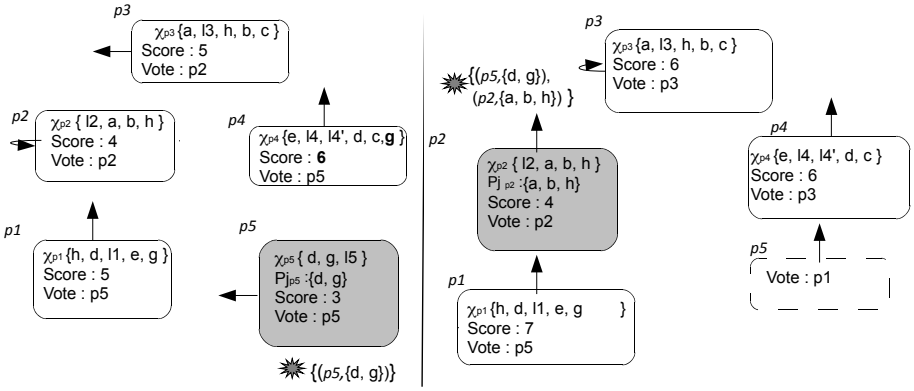


Fig. 6: Two iterations of Distributed Token Elimination on our example.

neighbours to vote for another peer), we know that there is a peer with a better local score than p if we follow any $best$ link from any neighbour of p that has not voted for it. So, in our solution, we propose to use this heuristic, as shown line 29.

Running Example An example (limited to two steps) of the algorithm is given in figure 6. First, suppose that the token was in p_1. According to the votes, the token is sent to p_5. Then, as shown in figure 6 (left), p_5 is selected for elimination (see the scores in the figure). It adds its own projection set $\{d, g\}$ (all its variable except local ones) to the token and removes itself. The token then goes through p_1 and p_2 (following elections results), and then (Figure 6 right) p_2 is removed and updates the token accordingly. In the next step (not shown on the figure), the token will be sent to p_3 which will remove the couple $(p_2, \{a, b, h\})$ from the token (p_3 shares at least one variable in this set) and will add the couple $(p_3, \{h, c\})$ in the token (a and b are not anymore shared).

Final Stage: re-connecting all sub-jointrees When the above algorithm finishes, it ends with a forest of sub-jointrees, distributed on the network. Each node has its own main cluster, fixed when the node was removed, but all the sub-jointrees have to be merged in one jointree, by reconnecting them while respecting the running intersection.

The first peer that detects the termination broadcasts the $Reconnection$ message in the network (line 20). Then, (not shown here), each non-leaf peer p that receives this message initiates the flooding algorithm 2: each p broadcasts a new message $ReqCON$ to the network with the set of its children. When receiving a message $ReqCON$ for the first time, a peer p records the sender p' as a father link and broadcasts the message to all its neighbourhood (excluding p'). Of course, due to the flooding mechanism, the message $ReqCON$ visits all links of the network (lines 4 to 6).

Now, when a peer has received the $ReqCON$ message from all its neighbours (line 10), it initiates the answer by sending back the message $AnsCON$ with the list of children of the initial peer p (noted $Wanted$ in the algorithm, lines 12 to 14). The message then goes bottom-up in the tree, following all the fathers links created above. Now, each

Algorithm 1: Distributed Token Elimination

(2) p **receives** $TOKEN(SubT)$ **from** p'
(3) *// local election*
(4) $\forall p' \in neighbours^*$, send $ELEC(SubT)$ to p'
(5) wait $VOTE(b_{p'})$ from all $p' \in neighbours^*$
(6)
(7) **if** (p has received $VOTE(\top)$ from all $p' \in neighbours^*$)
(8) *// Eliminate p (see section 3.2)*
(9) $Pj_p \leftarrow \bigcup_{p' \in neighbours^+} V_p \cap V_{p'}$
(10) $Children \leftarrow \bigcup_{(p',V') \in SubT} p'$ s.t. $V' \cap V_p \neq \emptyset$
(11) $New_\chi \leftarrow \bigcup_{(p',V') \in SubT} V'$ s.t. $p' \in Children$
(12) $\chi_p \leftarrow Pj_p \cup New_\chi \setminus \bigcup_{v \in V_p}$ s.t. $v \notin Pj_p$
(13) *// Update Orphan Sub-Jointrees*
(14) $SubT \leftarrow SubT \setminus \bigcup_{(p',V') \in SubT} (p', V')$ s.t. $p' \in Children$
(15) $SubT \leftarrow SubT \cup (p, \chi_p)$
(16) send $ELIM$ to all $p' \in neighbours^*$
(17) wait $AckELIM$ from all $p' \in neighbours^*$
(18) **if** ($Pjct = \{\{\}\}$)
(19) *// re-connecting sub-jointrees (see section 3.2)*
(20) broadcasts the $Reconnection$ message in the network.
(21)
(22) *// sending the token(see section 3.2)*
(23) **if** ($best \neq p$)
(24) **if** ($neighbours^+ \neq \emptyset$) *// token passing*
(25) send $TOKEN(SubT)$ to $best$
(26) **else**
(27) send $TOKEN(SubT)$ to $succ_{DFS}(p')$
(28) **else**
(29) send $TOKEN(SubT)$ to p" s.t. p as received $VOTE(\perp)$ from p"

(31) p **receives** $ELEC(SubT)$ **from** p'
(32) $score[p] \leftarrow eval(SubT)$
(33) send $SCORE(score[p])$ to its neighbours and wait $AckSCORE$
(34) $best \leftarrow min(score[])$
(35) send $VOTE(best == p')$ to p'

(37) p **receives** $SCORE(i)$ **from** p'
(38) $score[p'] \leftarrow i$
(39) send $AckSCORE$ to p'

(41) p **receives** $ELIM$ **from** p'
(42) remove p' from $score$ and $neighbours^+$
(43) $best \leftarrow min(score[])$
(44) send $AckELIM$ to p'

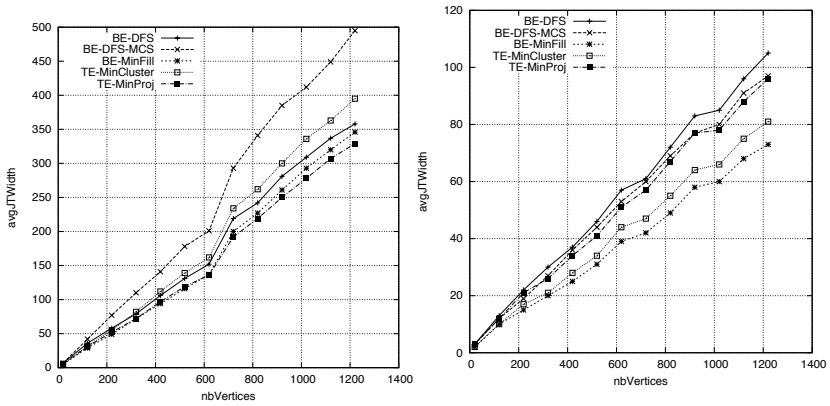
Algorithm 2: Sub-jointrees reconnection

(2) p **receives** $ReqCON(Wanted)$ **from** p'
(3) *// top-down flooding*
(4) **if** $(father(Wanted) \neq \emptyset)$
(5) $father(Wanted) \leftarrow p'$
(6) send $ReqCON(Wanted)$ to $neighbours - p'$
(7)
(8) *// bottom up connections*
(9) $nbMsg(Wanted) + +$
(10) **if** $(nbMsg(Wanted) == |neighbours|)$
(11) **if** $(p \in Children)$
(12) send $AnsCON(Wanted, \{Pj_p\})$ to $father(Wanted)$
(13) **else**
(14) send $AnsCON(Wanted, \{\})$ to $father(Wanted)$

(16) p **receives** $AnsCON(Wanted, projChild)$ **from** p'
(17)
(18) **if** $(Wanted == Children)$
(19) $sonLinks \leftarrow sonLinks \cup p'$ *//for future use*
(20) **else**
(21) **foreach** $V' \in projChild$
(22) $conClusters(Wanted) \leftarrow conClusters(Wanted) \cup V'$
(23) $nbMsg(Wanted) + +$
(24) **if** $(nbMsg(Wanted) == |neighbours|)$
(25) **if** $(p \in Wanted)$
(26) send $AnsCON(Wanted, conClusters(Wanted) \cup Pj_p)$ to $father(Wanted)$
(27) **else**
(28) send $AnsCON(Wanted, conClusters(Wanted))$ to $father(Wanted)$

peer that recognizes itself in the list attached to $AnsCON$ when the message goes up adds its own shared variables to the list of new shared variables (noted $ProjChild$ in the algorithm, lines 25 to 26). Those variables need to be added along the path to ensure the running intersection property. They are shared between the peer and its direct neighbourhood. In addition, each time a peer receives $AnsCON$ with a non empty list $projChild$, it creates a new "connection" cluster containing all variables of $projChild$ (named $conClusters(Wanted)$ in the algorithm for simplicity, lines 21 to 22).

Complexity Analysis At the end of the Distributed Token Elimination n peers have been eliminated. For each elimination the token will potentially visit all edges in both directions (in the worst case) making a local election at each step. Afterwards, during the sub jointree reconnection phase, each peer will be required. Since the local election requires a constant number of non concurrent messages and the time complexity of the sub-jointrees connection is bound by the diameter of the graph, the time complexity of the whole algorithm is in $O(N * E)$. In addition, the algorithm is complete and correct as soon as the initial graph respects the running intersection. The correctness and the completeness of our algorithm can be ensured by the following remark: the execution of



(a) WS Graph (b) BA Graph

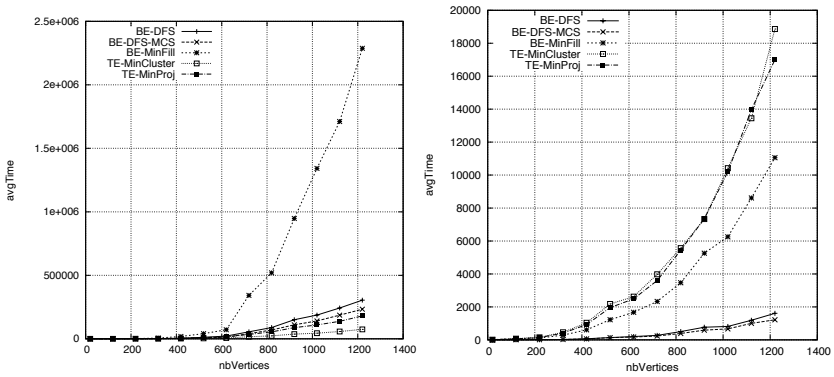
Fig. 7: Width of obtained jointrees, average values over 10 experiments per point

our algorithm can be easily reduced to a particular execution of a centralized, classical, decomposition tree algorithm.

4 Experimental evaluation

We implemented a java-based simulator of a distributed system for analysing the performance of our Distributed Token Elimination algorithm. We used a generator of small-world networks, following the model generation of Barabassi and Albert [2] (called BA Graphs) and Watts and Strogatz [24] (called WS Graphs). Let us point out that BA Graphs are extremely heterogeneous. The degree of nodes follows a power law, a strong characteristic of many real world problems (World Wide Web, email exchanges, scientific citations, ...) [18]. WS Graphs are more homogeneous and have been used for modelling diagnosis circuits [20] or various CSP instances [23]. On this kind of networks, we were able to scale up to more than thousands of variables. Before analyzing the results, let us also point out that we do not consider the privacy characteristics of our algorithms in this experiment: all variables are shared; and we consider that each peer has exactly one and only one variable for simplicity. However, as shown in the algorithm, line 12, private variables are treated apart. They are automatically removed from any projection the peer can send away. Thus, our experiment does not necessarily have to take privacy into account.

We report in figure 7 the quality of the jointrees produced by our algorithm, Distributed Token Elimination, with a set of selected state of the art algorithms. Bucket Elimination with the MinFill heuristic (BE-MinFill) is well known and represents the current state of the art for centralized methods. We also report two DFS variants (Maximum Cardinality Set and no heuristic). The first one is certainly the most used in distributed systems. We report the Token Elimination algorithm with two variants. The first one is the MinCluster (the one previously reported), and the second one MinProjection (instead of scoring the peers with their potential cluster size, we score them according to the size of the set of projection variables).



(a) WS Graph (b) BA Graph

Fig. 8: Average CPU time for compilation of jointrees over 10 experiments per point

The reported results show a significant improvement of Token Elimination over DFS. The obtained results, with the MinCluster variant, are now very close to BE on BA Graph. Surprisingly, after 700 nodes TE-Minproj gives better decompositions than BE on WS-Graph. Let us recall here that the jointrees built by Token Elimination are forced to follow the original links in the initial graph, which is not the case of BE.

The above results must be nuanced by the cost of the distributed compilation itself. As shown in figure 8, if DFS does not produce very good jointrees, it can produce them very quickly. BE needs more time, but produces the best jointrees. The total time needed by Token Elimination is clearly the worst on BA Graph figure, and quickly increasing. However, let us point out that for WS Graph TE is faster than BE. Given the improvement in the quality of the obtained jointrees, we strongly believe Token Elimination is a good solution, particularly suited to a distributed compilation approach of the network.

5 Conclusion

In this paper, we proposed a new distributed method for building distributed jointrees. Our method can handle privacy rules by keeping secrets secret (local variables stay local), and allow the jointree to be built only on the top of existing links between nodes. We show that this method can handle large structured instances and, even if the compilation cost is clearly above the simple Distributed DFS algorithm, the quality of the obtained jointree clearly outperforms DFS, even with a clever heuristic, and can even surpass the quality of Bucket Elimination in its centralized version. We believe that our algorithm can improved previous results in many distributed applications.

6 Acknowledgments

We want to thank the reviewers for the very useful comments they provided during the reviewing process and Philippe Jégou for his fruitful remarks.

References

1. Philippe Adjiman, Philippe Chatalic, François Goasdoué, Marie-Christine Rousset, and Laurent Simon. Distributed reasoning in a peer-to-peer setting: Application to the semantic web. *Journal of Artificial Intelligence Research (JAIR)*, 25:269–314, 2006.
2. Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, Oct 1999.
3. Ismel Brito and Pedro Meseguer. Cluster tree elimination for distributed constraint optimization with quality guarantees. *Fundam. Inform.*, 102(3-4):263–286, 2010.
4. Rina Dechter. *Constraint processing*. Elsevier Morgan Kaufmann, 2003.
5. Rina Dechter. Tractable structures for constraint satisfaction problems. In *Handbook of Constraint Programming, part I, chapter 7*, pages 209–244. Elsevier, 2006.
6. Artan Dermaku, Tobias Ganzow, Georg Gottlob, Benjamin J. McMahan, Nysret Musliu, and Marko Samer. Heuristic methods for hypertree decomposition. In *MICAI*, pages 1–11, 2008.
7. Redouane Ezzahir, Christian Bessiere, Imade Benelallam, and Mustapha Belaissaoui. Asynchronous breadth first search dcop algorithm. *Applied Mathematical Sciences*, 2(37-40):1837–1854, 2008.
8. Boi Faltings, Thomas Léauté, and Adrian Petcu. Privacy guarantees through distributed constraint satisfaction. In *IAT*, pages 350–358, 2008.
9. Yousri El Fattah and Rina Dechter. Diagnosing tree-decomposable circuits. In *IJCAI*, pages 1742–1749, 1995.
10. Aurélie Favier, Simon de Givry, and Philippe Jégou. Exploiting problem structure for solution counting. In *CP*, pages 335–343, 2009.
11. Georg Gottlob, Nicola Leone, and Francesco Scarcello. A comparison of structural csp decomposition methods. *Artificial Intelligence*, 124:2000, 2000.
12. Stéphane Grumbach and Zhilin Wu. Distributed tree decomposition of graphs and applications to verification. In *IPDPS Workshops*, pages 1–8, 2010.
13. Carlos Guestrin and Geoffrey J. Gordon. Distributed planning in hierarchical factored mdps. In *UAI*, pages 197–206, 2002.
14. Jinbo Huang and Adnan Darwiche. The language of search. *J. Artif. Intell. Res. (JAIR)*, 29:191–219, 2007.
15. Philippe Jégou, Samba Ndiaye, and Cyril Terrioux. Dynamic heuristics for backtrack search on tree-decomposition of csps. In *IJCAI*, pages 112–117, 2007.
16. Kalev Kask, Rina Dechter, Javier Larrosa, and Avi Dechter. Unifying tree decompositions for reasoning in graphical models. *Artif. Intell.*, 166(1-2):165–193, 2005.
17. Thomas Léauté and Boi Faltings. Coordinating logistics operations with privacy guarantees. In *IJCAI*, pages 2482–2487, 2011.
18. M. E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45(2):167–256, Jun 2003.
19. Adrian Petcu and Boi Faltings. A scalable method for multiagent constraint optimization. In *IJCAI*, pages 266–271, 2005.
20. Gregory Provan and Jun Wuang. Automated benchmark model generators for model-based diagnostic inference. In *IJCAI*, pages 513–518, 2007.
21. Neil Robertson and P. D. Seymour. Graph minors. ii. algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309 – 322, 1986.
22. Sathiamoorthy Subbarayan, Lucas Bordeaux, and Youssef Hamadi. Knowledge compilation properties of tree-of-bdds. In *AAAI*, pages 502–507, 2007.
23. Toby Walsh. Search in a small world. In *IJCAI*, pages 1172–1177, 1999.
24. Duncan J. Watts and Steven H. Strogatz. Collective dynamics of s̀mall-worldńetworks. *Nature*, 393(6684):440–442, Jun 1998.