Distributed Cluster Tree Elimination*

Ismel Brito and Pedro Meseguer

IIIA, Artificial Intelligence Research Institute
CSIC, Spanish National Research Council
Campus UAB, 08193 Bellaterra, Spain
ismel|pedro@iiia.csic.es

Abstract

Cluster and mini-cluster tree elimination are well-known solving methods for constrained optimization problems, developed for the centralized case. These methods, based on cost function combination, can be easily reformulated as synchronous algorithms to solve the distributed versions of the above mentioned problems. During solving they exchange a linear number of messages, but these messages could be of exponential size, which is their main drawback that often limits their practical application. Filtering is a general technique to decrease the size of cost function combination when using upper and lower bounds. We combine this technique with the previous algorithms, producing a significative decrement in message size and improving their practical memory usage. As result, the improved algorithm is able to solve larger problems, keeping under control memory consumption.

1 Introduction

Most of constraint reasoning methods have been developed under the implicit assumption that the constraint network is in the memory of a single agent, which performs the solving task. This is called centralized solving. In the last years, there is an increasing interest to solve these problems in a distributed form, when different problem parts are in the memory of different agents and they cannot be joined into a single one (because incompatible formats, privacy, etc.). New solving algorithms have been developed for this distributed model, where communication between agents is done by message passing. As examples, we mention ABT [10], ADOPT [5], DPOP [8].

In the centralized case, there are different forms to decompose a problem instance [2]. In particular, several algorithms work on a special structure: the cluster tree. These algorithms can be extended to distributed constraint solving, assuming that the distributed instance is arranged in a cluster tree. Interestingly, there are distributed algorithms able to build a cluster tree from a distribution of constraints into agents. So the extension of cluster tree solving algorithms to the distributed case seems feasible. The first goal of the paper is to show that a new set of distributed synchronous

*This work has been partially supported by the project TIN2006-15387-C03-01.

Proceedings of the 15th International RCRA workshop (RCRA 2008):
Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion
Udine, Italy, 12–13 December 2008

algorithms, inspired in the centralized algorithms working on the cluster tree, can be developed to solve distributed constraint problems. In the centralized case, these algorithms exhibit a exponential complexity in time and memory. Exponential memory is often the most restrictive limitation for their practical applicability. Some versions limit the memory usage, at the cost of achieving approximate solutions. Function filtering is an strategy to overcome this fact, allowing a better use of memory and achieving, in many cases, the exact solution. The second goal of the paper is to show that this idea can be applied to distributed constraint solving, causing some benefits.

The paper is organized as follows. In section 2, we provide a precise definition of the problems we consider. To make a self-contained paper, we summarize some centralized solving algorithms in section 3, while the idea of function filtering appears in section 4. Moving into a distributed context, we present new solving algorithms based on these ideas in sections 5 and 6, including distributed function filtering. The distributed algorithm to build cluster trees appears in section 7, with an example to clarify these new algorithms in section 8. Finally, section 9 contains some conclusions.

2 Preliminaries

In a centralized setting, a *Constraint Optimization Problem* (COP) involves a finite set of variables, each one taking a value in a finite domain. Variables are related by cost functions that specify the cost of value tuples on some variable subsets. Costs are natural numbers (including zero and ∞). Formally, a finite COP is defined by a triple (X, D, C),

- $X = \{x_1, \ldots, x_n\}$ is a set of n variables;
- $D = \{D(x_1), \ldots, D(x_n)\}$ is a collection of finite domains; $D(x_i)$ is the initial set of possible values for x_i;
- C is a set of cost functions; $f_i \in C$ on the ordered set of variables $var(f_i) = (x_{i_1}, \ldots, x_{i_{r(i)}})$ specifies the cost of every combination of values for the variables in $var(f_i)$, that is, $f_i : \prod_{j=i_1}^{i_{r_i}} D(x_j) \mapsto N^+$ (where N^+ is the set of natural numbers including 0 and ∞). The arity of f_i is the cardinality of the set $var(f_i)$.

The *overall cost* of a complete tuple (involving all variables) is the addition of all individual cost functions on that particular tuple. A *solution* is a complete tuple whose overall cost is not unacceptable. A solution is *optimal* if its overall cost is minimal.

Previous COP definition does not make explicit the fact that there is an upper bound in the cost of acceptable value tuples, so those value tuples whose cost exceeds this upper bound can be safely removed. In addition,

this upper bound may change during problem resolution. To make explicit these ideas, COP definition is refined to produce the so called *Weighted Constraint Satisfaction Problem* (WCSP). Formally, a WCSP is defined as a four tuple $(X, D, C, S(k))$, where X and D are as in the previous definition, C is a set of cost functions and $S(k)$ is a valuation structure [4]. While in COPs a cost function maps value combinations into natural numbers, in a WCSP a cost function maps value combinations into a special set $\{0, 1, ..., k\}$. That is, $f_i : \prod_{j=i_1}^{i_{r_i}} D(x_j) \mapsto \{0, 1, ..., k\}$. Costs are elements of the set $\{0, 1, ..., k\}$, where 0 is the minimum cost and k is the minimum unacceptable cost. All costs lower than k are acceptable, while all costs higher or equal to k are equally unacceptable. Costs are combined with the $\oplus$ operation: $a \oplus b = min\{a + b, k\}$, meaning that if the addition of two costs exceeds k, it automatically equals k. Costs are totally ordered with the standard order among naturals. The *size* of f, denoted $|f|$, is the cardinal of S_f. Observe that this definition includes purely satisfaction instances (classical CSP), where tuples are either permitted or forbidden: a permitted tuple costs 0, a forbidden tuple costs 1, and k must be 1. We store cost function f as a set S_f containing all pairs $(t, f(t))$ with cost less than k.

This definition can be extended to a distributed context. A *Distributed Weighted Constraint Satisfaction Problem* (DWCSP), is a WCSP where variables, domains and cost functions are distributed among automated agents. Formally, we define a *variable-based* (resp. *cost-function-based*) DWCSP as a 6-tuple $(X, D, C, S(k), A, \alpha$ (resp. $\beta))$, where X, D, C and $S(k)$ define a WCSP, A is a set of p agents and α (resp. β) maps each variable (resp. cost function) to one agent. Here we assume the DWCSP model: it is a refined version of distributed constraint optimization, where the notion of unacceptable cost is explicitly handled. In the rest of the paper, we will assume the cost-function-based definition of DWCSP.

Next, some terminology to be used in the rest of the paper. An *assignment or tuple* t_S with scope S is an ordered sequence of values, each corresponding to a variable of $S \subseteq X$. The *projection* of t_S on a subset of variables $T \subset S$, written $t_S[T]$, is formed from t_S removing the values of variables that do not appear in T. This idea can be extended to cost functions: the projection of f on $T \subset var(f)$, is a new cost function $f[T]$ formed by the tuples of f removing the values of variables that do not appear in T, removing duplicates and keeping the minimum cost of the original tuples in f. The *join* of two tuples t_S and t'_T, written $t_S \bowtie t'_T$, is a new tuple with scope $S \cup T$, formed by the values appearing in t_S and t'_T; it is only defined when common variables have the same values in t_S and t'_T. The *cost* of a tuple t_X (involving all variables) is $\oplus_{f \in C} f(t_X)$, that is, the addition of the individual cost functions evaluated on t_X (implicitly, it is assumed that, for each $f \in C, f(t_X) = f(t_X[var(f)])$). A *solution* is a tuple with cost lower than k. A solution is *optimal* if its cost is minimal.

3 Cluster Tree Elimination

Centralized WCSPs can be solved using tree decomposition methods. A *tree decomposition* of a WCSP $\langle X, D, C, S(k)\rangle$ is a triple $\langle T, \chi, \psi\rangle$, where $T = \langle V, E\rangle$ is a tree, χ and ψ are labeling functions which associate with each vertex $v \in V$ two sets, $\chi(v) \subseteq X$ and $\psi(v) \subseteq C$ such that

- for each cost function $f \in C$, there is exactly one vertex $v \in V$ such that $f \in \psi(v)$; in addition, $var(f) \subseteq \chi(v)$;
- for each variable $x \in X$, the set $\{v \in V | x \in \chi(v)\}$ induces a connected subtree of T.

The *tree-width* of $\langle T, \chi, \psi\rangle$ is $tw = max_{v\in V}|\chi(v)|$. If u and v are adjacent vertices, $(u, v) \in E$, its *separator* is $sep(u, v) = \chi(u) \cap \chi(v)$ [1]. *Summing* two functions f and g is a new function $f + g$ with scope $var(f) \cup var(g)$ and $\forall t \in \prod_{x_i\in var(f)} D_i$, $\forall t' \in \prod_{x_j\in var(g)} D_j$ $(f + g)(t \bowtie t') = f(t) \oplus g(t')$ ($t \bowtie t'$ is defined when t and t' share values of common variables). Function g is a *lower bound* of f, denoted $g \leq f$, if $var(g) \subseteq var(f)$ and for all possible tuples t of f, $g(t) \leq f(t)$. A set of functions G is a *lower bound* of f iff $(\sum_{g\in G} g) \leq f$; $var(G) = \cup_{g\in G} var(g)$. It is easy to check that for any $f, U \subset var(f)$, $f[U]$ is a lower bound of f, and $\sum_{f\in F} f[U] \leq (\sum_{f\in F} f)[U]$.

The Cluster-Tree Elimination algorithm (CTE) solves WCSP by sending messages along tree decomposition edges [1]. Edge $(u, v) \in E$ has associated two CTE messages $m_{(u,v)}$, from u to v, and $m_{(v,u)}$, from v to u. $m_{(u,v)}$ is a function computed summing all functions in $\psi(v)$ with all incoming CTE messages except from $m_{(v,u)}$ and projected on $sep(u, v)$. CTE appears in Figure 1. It is correct, with exponential complexity in time and space [3].

Mini-Cluster-Tree Elimination (MCTE(r)) approximates CTE. If the number of variables in u is high, it may be impossible to compute $m_{(u,v)}$ due to memory limitations. MCTE(r) computes a lower bound by limiting to r the maximum arity of the functions sent in the messages. A MCTE(r) message, $M_{(u,v)}$, is a set of functions that approximate the corresponding CTE message $m_{(u,v)}$ ($M_{(u,v)} \leq m_{(u,v)}$). It is computed as $m_{(u,v)}$ but instead of summing all functions of set B (see Figure 1), it computes a partition $P = \{P_1, \ldots, P_q\}$ of B such that the arity of the sum of functions in every

procedure CTE($\langle X, D, C, S(k)\rangle$, $\langle\langle V, E\rangle, \chi, \psi\rangle$)
1 **for each** $(u, v) \in E$ s.t. all $m_{(i,u)}, i \neq v$ have arrived **do**
2 $B \leftarrow \psi(u) \cup \{m_{(i,u)} \mid (i, u) \in E, i \neq v\}$;
3 $m_{(u,v)} \leftarrow (\sum_{f\in B} f)[sep(u, v)]$;
4 send $m_{(u,v)}$;

Figure 1: The CTE algorithm.

P_i does not exceed r. The MCTE(r) algorithm is obtained replacing line 3 of CTE by the following lines (where the projection is done on the variables of the separator that appear in the scope of the functions in the partition class),

3.1 $\{P_1, ..., P_q\} \leftarrow partition(B, r)$;
3.2 $M_{(u,v)} \leftarrow \{(\sum_{f \in P_i} f)[sep(u,v) \cap (\cup_{f \in P_i} var(f))] \mid i : 1...q\}$;

4 Filtering Cost Functions

Filtering cost functions is a clever strategy to decrease the size of messages sent by CTE and MCTE(r). It was introduced for the centralized case in [9]. The basic idea is to detect tuples that, although having acceptable cost in their vertex, they will always generate tuples with unacceptable cost when combined with other cost functions coming from other vertices. These initial tuples are removed before they are sent, decreasing the size of exchanged cost functions. This idea generates IMCTEf(r), an iterative version of MCTEf(r), which nicely decreases the size of exchanged functions at each iteration. In the best case, this strategy would allow for an exact computation of the optimal solution. If not, it will compute an approximated solution closer to the optimal one than the one computed by MCTE(r).

A *nogood* is a tuple t that cannot be extended into a complete assignment with acceptable cost. Nogoods are useless for solution generation, so they can be eliminated as soon as are detected. For summing $f + g$, we iterate over all the combinations $(t, f(t)) \in S_f$ and $(t', g(t')) \in S_g$ and, if they match, compute $(t \bowtie t', f(t) \oplus g(t'))$. If $f(t) \oplus g(t') \geq k$, tuple $t \bowtie t'$ is a nogood so it is not stored in S_{f+g}.

Filtering cost functions consists of anticipating the nogoods on cost functions, removing them before real operation. Imagine that we know that cost function f will be added (in the future) with cost function g, and we know that the set of functions H is a lower bound of g. We define the filtering of f from H, noted $\overline{f}^H$, as

$$\overline{f}^H(t) = \begin{cases} f(t) & if \quad (\bigoplus_{h \in H} h(t)) \oplus f(t) < k \\ k & otherwise \end{cases}$$

Tuples reaching the upper bound k are removed because they will be generate unacceptable tuples when f will be added with g (remember that H is a lower bound of g). This causes to reduce $|f|$ before operating with it.

Let f and g be two cost functions, and G set of functions that is a lower bound of g. Filtering f with G before adding with g it is equal to $f + g$,

$$f + g = \overline{f}^G + g$$

To see this result, it is enough to decompose the function f, stored as the set of tuples that do not reach the upper bound k, in the following partition,

$$S_f = \{(t_1, f(t_1)) | t_1 \in P\} \cup \{(t_2, f(t_2)) | t_2 \in Q\}$$

where $P = \{t | t \in \prod_{x_i \in var(f)} D(x_i),\ \exists t' \in \prod_{x_j \in var(G)} D(x_j), t \bowtie t' \text{ is defined, such that } (\bigoplus_{h \in G}\ h(t')) \oplus f(t) < k\}$, and $Q = \{t | t \in \prod_{x_i \in var(f)} D(x_i),\ \forall t' \in \prod_{x_j \in var(G)} D(x_j), t \bowtie t' \text{ is defined, such that } (\bigoplus_{h \in G}\ h(t')) \oplus f(t) \geq k\}$. Assuming that $var(G) = var(g)$, function $f + g$ is stored as,

$$S_{f+g} = \{(t'' = t \bowtie t', f(t) \oplus g(t'))\} =$$

$$\{(t''_1 = t_1 \bowtie t', f(t_1) \oplus g(t'))|t_1 \in P\} \cup \{(t''_2 = t_2 \bowtie t', f(t_2) \oplus g(t'))|t_2 \in Q\}$$

but the second set is empty because for any $t', f(t_2) \oplus (\bigoplus_{h \in G}\ h(t')) \leq f(t_2) \oplus g(t')$, since G is a lower bound of g, so $f(t_2) \oplus g(t') \geq k, \forall t_2 \in Q$. Then, $f + g$ is stored as,

$$\{(t''_1 = t_1 \bowtie t', f(t_1) \oplus g(t'))|t_1 \in P\}$$

which is exactly $\overline{f}^G + g$.

How often do we know that function f will be added with function g? In vertex u, function $m_{(u,v)}$ summarizes the effect of the part of the cluster tree rooted at u, while $m_{(v,u)}$ summarizes the effect of the rest of the cluster tree. To compute the solution in vertex u, these two functions must be added. Therefore, if we are able to compute a lower bound of $m_{(v,u)}$ before $m_{(u,v)}$ is sent, we can filter $m_{(u,v)}$ with that lower bound and reduce its size.

With this idea, function filtering easily integrates into CTE. A *filtering tree-decomposition* is a tuple $\langle T, \chi, \psi, \phi \rangle$, where $\phi(u, v)$ is a set of functions associated to edge $(u, v) \in E$ with scope included in $sep(u, v)$. $\phi(u, v)$ must be a lower bound of the corresponding $m_{(v,u)}$ (namely, $\phi(u, v) \leq m_{(v,u)}$). The algorithms CTEf and MCTEf(r) use a filtering tree decomposition. They are equivalent to CTE and MCTE(r) except in that they use $\phi(u, v)$ for filtering functions before computing $m_{(u,v)}$ or $M_{(u,v)}$. For CTEf, we replace line 3 by,

3 $\quad m_{(u,v)} \leftarrow \overline{\sum_{f \in B} f}^{\phi(u,v)}[sep(u, v)];$

Similarly for MCTEf(r) we replace line 3 by two lines,

3.1 $\quad \{P_1, ..., P_p\} \leftarrow partitioning(B, r);$

3.2 $\quad M_{(u,v)} \leftarrow \{(\overline{\sum_{f \in P_i} f}^{\phi(u,v)})[sep(u, v) \cap (\cup_{f \in P_i} var(f))] \mid i : 1...p\};$

An option for CTEf is to include in $\phi(u, v)$ a message $M_{(v,u)}$ from a previous execution of MCTE(r). Applying this idea to MCTEf, we obtain a recursive algorithm which naturally produces an elegant iterative approximating method called IMCTEf (Figure 2). It executes MCTEf(r) using as lower bounds $\phi(u, v)$ the messages $M^{r-1}_{(v,u)}$ computed by MCTEf($r-1$) which, recursively, uses the messages $M^{r-2}_{(v,u)}$ computed by MCTEf($r-2$), an so on.

procedure IMCTE($\langle X, D, C, k\rangle$, $\langle\langle V, E\rangle, \chi, \psi\rangle$)
  for each $(u, v) \in E$ **do** $\phi(u, v) := \{\emptyset\}$; $r := 1$;
  repeat
    MCTEf(r); $r := r + 1$;
    for each $(u, v) \in E$ **do** $\phi(u, v) := M_{(u,v)}$;
  until exact solution or exhausted resources

Figure 2: The IMCTE algorithm.

5 Distributed Cluster Tree Elimination

The CTE algorithm can be easily adapted to the distributed case, producing the *Distributed CTE* (DCTE) algorithm. We assume that the DWCSP instance (X, D, C, A, β) to solve is arranged in a cluster tree (T, χ, ψ), where each vertex is a different agent (distributed algorithms to build a cluster tree exist, see section 7). Let us consider $self$, a generic agent. It owns a specific vertex in the cluster tree, so $self$ knows its position in the tree: it knows its neighboring agents and the separators with them. Besides, $self$ also knows variables of $\chi(self)$ and cost functions of $\psi(self)$.

DCTE exchanges messages among agents. There is one message type, CF, to exchange cost functions. When $self$ has received function messages from all its neighbors except perhaps i, it performs the summation of the received cost functions (excluding cost function from i) with the cost functions of $\psi(self)$, producing a new cost function, which is projected on $sep(self, i)$ and sent to agent i. This process is repeated for all neighbors. Agents having a single neighbor are the first computing and sending CF messages.

CF messages play the same role as function messages in centralized CTE. For each edge (i, j) in the tree (i and j are neighboring agents) there are two CF messages: one from i to j and other from j to i. Once these two messages have been exchanged for all edges in the tree, agent $self$ contains in its cluster (formed by $\psi(self)$ and the CF messages received from its neighbors) enough information to solve the particular WCSP instance, assuring that local optimal solutions at different vertices are part of a global optimal solution. To break ties, a total ordering on the values of each variable is needed, which has to be common to all agents.

DCTE algorithm appears in Figure 3. It takes as input the tree decomposition (T, χ, ψ) on the vertex $self$ (as explained in the first paragraph of this section), and returns the vertex $self$ augmented with a number of cost functions, one per neighbor. These new cost functions plus the cost function of $\psi(self)$ are enough to compute the local optimal solutions of each vertex such that they are compatible with solutions of other vertices and share the global optimum (they form the minimal subproblem, see [1]). DCTE records the cost functions exchanged among agents. When a cost function

procedure DCTE(T, χ, ψ)
compute $neighbors(self)$, $\chi(self)$, $\psi(self)$;
for each $j \in neighbors(self)$ **do**
  compute separator($self, j$); $recvCF[j] \leftarrow sentCF[j] \leftarrow$ false;
$end \leftarrow$ false;
if $neighbors(self) = \{other\}$ **then** ComputeSendFunction($self, other$);
while ($\neg end$) **do**
  $msg \leftarrow$ getMsg();
  if ($msg.type = CF$) **then** NewCostFunction(msg);
  $end \leftarrow \wedge_{j \in neighbors(self)}(sentCF[j] \wedge recvCF[j])$;
compute *solution* from $\psi(self) \cup \{recvFunction[j] \mid j \in neighbors(self)\}$;

procedure NewCostFunction(msg)
$recvFunction[msg.sender] \leftarrow msg.function$; $recv[msg.sender] \leftarrow$ true;
for each $j \in neighbors(self)$ s.t. $sent[j] =$ false **do**
  if $\wedge_{i \in neighbors(self), i \neq j} recv[i]$ **then** ComputeSendFunction($self, j$);

procedure ComputeSendFunction($self, dest$)
$function \leftarrow \sum_{i \in neighbors(self), i \neq dest} recvFunction[i] + \sum_{f \in \psi(self)} f$;
sendMsg($self, dest, function$[separator($self, dest$)]); $sent[dest] \leftarrow$ true;

Figure 3: The Distributed Cluster Tree Elimination algorithm.

is received from neighbor j, it is stored in $recvFunction[j]$ and the boolean $recvCF[j]$ is set to true. When a cost function is sent to neighbor j, the boolean $sentCF[j]$ is set to true.

The main procedure is DCTE, which works as follows. First, some elements of the cluster tree required for $self$ are computed ($neighbors(self)$, $\chi(self), \psi(self)$, separators between $self$ and its neighbors), and some variables (vectors $recvCF$ and $sendCF$, variable end) are initialized. If $self$ has a single neighbor, $other$, $self$ does not have to wait for any incoming cost function. So the procedure ComputeSentFunction is called, computing the corresponding summation of functions projected on the separator $(\sum_{f \in \psi(self)} f)$[separator($self, other$)], that is sent to agent $other$. Next, there is a loop that reads a message, processes it and checks the final condition, when $self$ has received/sent a cost function from/to each neighbor. Finally, the *solution* is computed using the cost functions in $\psi(self)$ and the received cost functions.

Procedure NewCostFunction processes CF messages. It records the cost function contained in the message, and if this function allows for computing a cost function to be sent to another neighbor, it is done by the ComputeSentFunction procedure.

DCTE is a distributed synchronous algorithm, since $self$ has to wait to receive CF messages from all its neighbors but j, to be able to compute (and send) its CF message to j.

6 Distributed Mini-Cluster Tree Elimination

DCTE can be easily modified to produce the Mini-cluster version (DMCTE). We have two new parameters here: r is the maximum arity of the cost functions that can be sent to neighbors, and ub is the initial upper bound.

DMCTE is conceptually close to DCTE, but its practical implementation is more involved. While DCTE adds all cost functions of an agent (no matter the resulting arity), and sends the projection on the separator, DMCTE limits the arity of the resulting cost function. In consequence, DMCTE exchanges cost functions (via CF messages) which are approximations of the exact cost functions (those exchanged by DCTE). When each agent has sent to/received from a CF message to each of its neighbors, these approximate cost functions have been propagated. While this approximation allow for computing an approximate solution at each agent, there is no guarantee that these solutions will be compatible each other (that is, with the same values for common variables in the separators). To assure this, once CF messages have been exchanged, values of common variables in the separators are exchanged via SS messages. If a discrepancy exists, the value coming from the agent with lower id prevails. Finally, a global upper bound is computed by exchanging UB messages, containing the cost of the current solution on the original cost functions of each agent.

DMCTE uses three message types,

- CF: cost function messages. They work as the CF messages of the DCTE algorithm, with the following exception. The set of cost functions to be added is partitioned, such that the cost function resulting from the addition of the cost functions of each class do not exceed arity r. Each resulting function is projected on the separator and sent to the corresponding agent. A CF message contains not a single cost function (as in DCTE) but a set of cost functions.

- SS: solution separator messages. When $self$ has received all approximate cost functions and computed an approximate solution, it exchanges the values of variables in the separators with its neighbors. The value sent by the agent with lower id prevails. These messages are sent/received following the same strategy as CF messages.

- UB: upper bound messages. Once a compatible solution has been found, agents exchange the cost of this solution via UB messages. They are sent/received following the same strategy as CF messages.

We do not provide the DMCTE code due to space limitations. The very same idea of filtering cost functions can be applied to the distributed case. The iterative version of mini-cluster tree elimination can be extended to the

distributed case, where messages of the previous iteration are used as filters to the messages of the current iteration.

The DMCTEf(r) algorithm performs filtering when adding cost functions. Its only novelty with respect DMCTE(r) is that new cost functions to be send to other agents j are filtered when they are computed. The *Distributed IMCTEf* (DIMCTEf) algorithm is a direct extension of IMCTEf (Figure 2) to the distributed case, where previous messages are recorded and used as filters. The upper bound computed at the end of iteration i is used as parameter ub in the next iteration. We do not provide the code of DIMCTEf due to space limitations.

7 Distributed Cluster Tree Formation

Throughout this paper it is assumed the existence of a cluster tree where the problem instance is arranged. In the centralized case, it is well-known the existence of algorithms to build such a tree [1]. In the distributed case, of interest here, there are also algorithms able to build a cluster tree in a distributed form. In the following, we provide a short description of the ERP algorithm [6], able to compute the cluster tree from a DWCSP instance.

Initially, we have $(X, D, C, S(k), A, \beta)$, where X is the set of variables, D is the collection of corresponding domains, C is the set of cost functions, $S(k)$ is a valuation structure, A is a set of agents and β is a function that maps cost functions into agents. We assume that β covers the whole set of agents (if there is some agent without cost function, it is removed from A). The construction of the cluster tree $(T(V, E), \chi, \psi)$ has the following steps (where $\chi^0(v)$ denotes the initial value of $\chi(v)$) :

1. The set of vertices is the set of agents.
2. For each vertex $u, \psi(u) = \{f \in C \mid \beta(f) = u\}; \chi^0(u) = \cup_{f \in \psi(u)} var(f)$.
3. Two vertices u and v are considered adjacent if they share one or more variables, that is, $\chi^0(u) \cap \chi^0(v) \neq \emptyset$. This criterion defines a graph G.
4. Using a distributed spanning tree algorithm on G [7], we obtain a spanning tree $T(V = A, E)$. It does not necessarily satisfy the connectness (also called running intersection) property.
5. The connectness property is assured as follows. Each agent i sends to its neighbors in the tree the variables that initially appear in is $\chi^0(i)$. Once agent i has received from its neighbors all these messages, it updates $\chi(i)$ as follows,

$$\chi(i) \leftarrow \chi^0(i) \bigcup_{j,k \in neighbors(i), j \neq k} (\chi^0(j) \cap \chi^0(k))$$

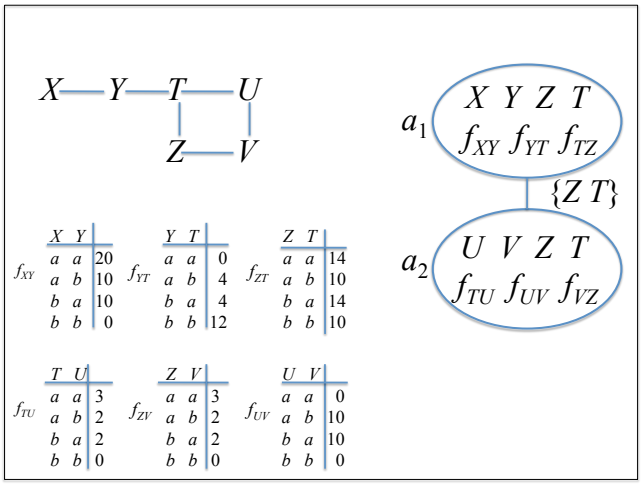

Figure 4: Problem instance, involving 6 variables and 6 cost functions. Its tree decomposition, formed by two vertices a_1 and a_2, appears in the right. The separator between them is $\{Z\ T\}$.

which has a clear meaning: if a variable x appears in two neighbors j and k, it must also appear in the vertex itself, to assure connectness.

8 Example

On the instance depicted in Figure 4, with the indicated tree decomposition, we will detail the execution of DCTE, DMCTE and DIMCTE.

DCTE. Agent a_1 computes function $f_1 \leftarrow f_{XY} + f_{YT} + f_{TZ}$, projects this function on the separator between a_1 and a_2, and sends the result, $f_2 = f_1[ZT]$, to agent a_2 in a CF message,

f_1:

X	Y	T	Z	
a	a	a	a	34
a	a	a	b	34
a	a	b	a	34
a	a	b	b	34
a	b	a	a	28
a	b	a	b	28
a	b	b	a	32
a	b	b	b	32

X	Y	T	Z	
b	a	a	a	24
b	a	a	b	24
b	a	b	a	24
b	a	b	b	24
b	b	a	a	18
b	b	a	b	18
b	b	b	a	22
b	b	b	b	22

f_2:

Z	T	
a	a	18
a	b	22
b	a	18
b	b	22

Analogously, a_2 computes $f_3 \leftarrow f_{TU} + f_{UV} + f_{VZ}$, projects on the separator and sends the result, $f_4 = f_3[ZT]$, to a_1 in a CF message,

f_3:

U	V	Z	T	
a	a	a	a	6
a	a	a	b	5
a	a	b	a	5
a	a	b	b	4
a	b	a	a	15
a	b	a	b	14
a	b	b	a	13
a	b	b	b	12

U	V	Z	T	
b	a	a	a	15
b	a	a	b	13
b	a	b	a	14
b	a	b	b	12
b	b	a	a	4
b	b	a	b	2
b	b	b	a	2
b	b	b	b	0

f_4:

Z	T	
a	a	4
a	b	2
b	a	2
b	b	0

Agent a_2 receives the CF message sent by a_1 and executes the procedure `NewCostFunction`, storing the received function and calling `ComputeSendUB`. The very same process happens in a_1 when receiving the CF message from a_2: `NewCostFunction` is executed, the received cost function is stored. Now, a_1 computes its local optimum using the cost functions $\{f_{XY}, f_{YT}, f_{TZ}, f_4\}$, obtaining $XYZT \leftarrow bbba$. Analogously, a_2 computes its local optimum using the cost functions $\{f_{TU}, f_{UV}, f_{VZ}, f_2\}$, obtaining $UVZT \leftarrow bbba$.

DMCTE($r = 2$)**.** If $r = 2$, agents cannot compute cost functions of arity greater than 2. Since original cost functions are binary, agents do not perform any addition on them, they just project on the separator and send the resulting cost functions. Thus, a_1 computes $g_1 = f_{YT}[T]$ and sends a CF message with g_1 and f_{ZT} to a_2.

g_1:

T	
a	0
b	4

f_{ZT}:

Z	T	
a	a	14
a	b	10
b	a	14
b	b	10

Analogously, a_2 computes $g_2 = f_{TU}[T]$ and $g_3 = f_{ZV}[Z]$, builds a CF message containing g_2 and g_3 and sends it to a_1.

g_2:

T	
a	2
b	0

g_3:

Z	
a	2
b	0

Agent a_1 computes its local optimum, that is $XYZT \leftarrow bbba$, while a_2 does the same, obtaining $UVZT \leftarrow bbbb$. Since there is discrepancy in the value of T, values of T and Z are exchanged between the agents. Assuming that a_2 has smaller identifier than a_1, its value for T prevails, so a_1 changes its optimum to $XYTZ \leftarrow bbbb$. Now agents exchange their actual costs on the initial cost functions using UB messages. At the end, each agent knows that a true upper bound of the cost of the exact optimum is 22, the cost of the all b's solution in the whole instance.

DIMCTEf. We take 9 as the limit of the #tuples computed by agent ($9 = 2^3$, we use the same memory as DMCTE($r = 3$)). Let us start with $r = 2$ ($r = 1$ has no sense here because all initial cost functions are binary).

This is exactly the DMCTE execution indicated above. Then, we move to $r = 3$ taking as new $ub = 22$, the upper bound computed in the previous iteration. Agent a_1 computes $g_4 = f_{XY} + f_{YT}$, filtering its construction with g_2. Then, it computes $g_5 = \overline{g_4}^{g_2}[T]$, builds a CF message with g_5 and f_{ZT}, and sends it to a_2. It is worth noting that three tuples are eliminated, since they are nogoods (their cost is higher or equal the current ub).

g_4:

X	Y	T	
a	a	a	$20 + 2 \geq ub$
a	a	b	$24 + 0 \geq ub$
a	b	a	$14 + 2$
a	b	b	$22 + 0 \geq ub$
b	a	a	$10 + 2$
b	a	b	$14 + 0$
b	b	a	$0 + 2$
b	b	b	$12 + 0$

g_5:

T	
a	0
b	12

f_{ZT} :

Z	T	
a	a	14
a	b	10
b	a	14
b	b	10

Agent a_2 computes $g_6 = f_{TU} + f_{UV}$, filtering with g_1 and $f_{ZT}[T]$. Four tuples are eliminated because their cost reach the upper bound. It computes $g_7 = \overline{g_6}^{\{g_1, f_{ZT}[T]\}}[T]$.

g_6:

U	V	T	
a	a	a	$3 + 0 + 14$
a	a	b	$2 + 3 + 10$
a	b	a	$13 + 0 + 14 \geq ub$
a	b	b	$12 + 3 + 10 \geq ub$
b	a	a	$12 + 0 + 14 \geq ub$
b	a	b	$10 + 3 + 10 \geq ub$
b	b	a	$2 + 0 + 14$
b	b	b	$0 + 3 + 10$

g_7:

T	
a	2
b	0

Agent a_2 filters f_{ZV} with $f_{ZT}[Z]$, but no tuple is eliminated. Its projection on Z generates $g_8 = \overline{f_{ZV}}^{f_{ZT}[Z]}[Z]$.

g_8:

Z	
a	2
b	0

At this point, a_1 and a_2 compute the approximate optimum in their respective vertices. It happens that a_1 computes $XYZT \leftarrow bbba$, while a_2 computes $UVZT \leftarrow bbba$. No discrepancy exists on values of variables in separators, so they exchange UB messages on the cost of the optimum, obtaining 20 as upper bound (in fact, this is the exact optimum).

Then, we move with $r = 4$, expecting to have enough memory to compute the exact optimum. Agent a_1 computes $f_1 = f_{XY} + f_{YT} + f_{ZT}$, filtering it with g_7 and g_8. It happens that all tuples are removed because their cost reach the upper bound.

	X	Y	T	Z		X	Y	T	Z	
	a	a	a	a	$34+2+2 \geq ub$	b	a	a	a	$24+2+2 \geq ub$
	a	a	a	b	$34+2+0 \geq ub$	b	a	a	b	$24+2+0 \geq ub$
	a	a	b	a	$34+0+2 \geq ub$	b	a	b	a	$24+0+2 \geq ub$
f_1:	a	a	b	b	$34+0+0 \geq ub$	b	a	b	b	$24+0+0 \geq ub$
	a	b	a	a	$28+2+2 \geq ub$	b	b	a	a	$18+2+2 \geq ub$
	a	b	a	b	$28+2+0 \geq ub$	b	b	a	b	$18+2+0 \geq ub$
	a	b	b	a	$32+0+2 \geq ub$	b	b	b	a	$22+0+2 \geq ub$
	a	b	b	b	$32+0+0 \geq ub$	b	b	b	b	$22+0+0 \geq ub$

This means that the previous *approximate* solution is in fact the true optimum, and its cost the optimum cost. We observe that in the $r = 4$ iteration, DIMCTEf uses no extra memory since each tuple is discarded as it is generated. To satisfy the communication protocol, a_1 builds a CF message with only the tuple ba with cost 20 for $\{ZT\}$, and sends it to a_2.

Agent a_2 computes $f_3 = f_{TU} + f_{UV} + f_{ZV}$, filtering with f_{ZT} and g_5. Then, it computes $f_5 = \overline{f_3}^{\{f_{ZT}, g_5\}}[ZT]$.

	U	V	Z	T		U	V	Z	T	
	a	a	a	a	$6+14+0 \geq ub$	b	a	a	a	$15+14+0 \geq ub$
	a	a	a	b	$5+10+12 \geq ub$	b	a	a	b	$13+10+12 \geq ub$
	a	a	b	a	$5+14+0$	b	a	b	a	$14+14+0 \geq ub$
f_3:	a	a	b	b	$4+10+12 \geq ub$	b	a	b	b	$12+10+12 \geq ub$
	a	b	a	a	$15+14+0 \geq ub$	b	b	a	a	$4+14+0$
	a	b	a	b	$14+10+12 \geq ub$	b	b	a	b	$2+10+12 \geq ub$
	a	b	b	a	$13+14+0 \geq ub$	b	b	b	a	$2+14+0$
	a	b	b	b	$12+10+12 \geq ub$	b	b	b	b	$0+10+12 \geq ub$

f_5:

Z	T	
a	a	4
b	a	2

All but three tuples are removed because they reach the upper bound. Agent a_2 builds a CF message with f_5 and sends it to a_1. At this point, a_1 computes its local optimum $XYZT \leftarrow bbba$ and a_2 does the same $UVZT \leftarrow bbba$. There is no need to exchange SS and UB messages, because the problem has been solved exactly. The solution inside each agent is the optimal one.

In this case, DIMCTEf exchanges messages of the same size as DMCTE($r = 3$) (maximum of 2^3 tuples), but it is able to solve exactly this instance, while the exact algorithm DCTE requires larger messages (maximum of 2^4 tuples).

9 Conclusions

We have presented DCTE, DMCTE and DIMCTEf, distributed synchronous algorithms for solving distributed WCSPs (a more precise version of the well-known distributed COPs). The DCTE algorithm solves the problem exactly, but requires messages of exponential size. DMCTE and DIMCTEf limit the used memory, at the cost of achieving an approximated solving. Using

the function filtering strategy, that also holds in the distributed context, DIMCTEf performs a better memory usage than DMCTE, which increases its practical applicability.

These algorithms are inspired in their centralized counterparts, but their extension requires some care. This is especially true for the DMCTE algorithm that, in addition to the CF message type used for DCTE, requires two new types of messages SS and UB to deal with the subtleties of approximated solving. More work is needed, especially of experimental nature, to assess the practical applicability of the presented algorithms on different benchmarks.

Acknowledgments

Authors thank reviewers for their constructive comments and criticisms.

References

[1] Dechter R. *Constraint Processing*. Morgan Kaufmann, 2003.

[2] Gottlob G., Leone N., Scarcello F. A comparison of structural CSP decomposition methods. *Artificial Intelligence*, 124, 243–282, 2000.

[3] Kask K., Dechter R. Larrosa J., Dechter A. Unifying cluster-tree decompositions for reasoning in graphical models. *Artificial Intelligence*, 166(1-2), 2005.

[4] Larrosa J. Node and arc consistency in weighted CSP *Proc. of AAAI-02*, 2002.

[5] Modi P. J., Shen W.M., Tambe M., Yokoo M. Adopt: asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, **161**, 149–180, 2005.

[6] Paskin M., Guestrin C. McFadden J. A robust architecture for distributed inference in sensor networks. *Proc of IPSN*, 55–62, 2005.

[7] Perlman R. An algorithm for distributed computation of a spanning tree in an extended LAN. *ACM SIGCOMM Computer Communication Review*, 44–53, 1985.

[8] Petcu A., Faltings B. A scalable method for multiagent constraint optimization *Proc. of IJCAI-05*, 266–271, 2005.

[9] Sanchez M., Larrosa J., Meseguer P. Improving Tree Decomposition Methods with Function Filtering. *Proc. of IJCAI-05*, 2005.

[10] Yokoo M., Durfee E., Ishida T., Kuwabara K. The Distributed Constraint Satisfaction Problem: Formalization and Algorithms. *IEEE Trans. Know. and Data Engin.*, **10**, 673–685, 1998.