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 each could be of exponential size. This 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 filtering with the previous algorithms, producing a significative decrement in message size. As result, the improved algorithm is able to solve larger problems, keeping under control memory consumption. Experimental results show the benefits of this approach.

1 Introduction

Most of constraint reasoning work has been done under the implicit assumption that the constraint network is in the memory of a single agent, which performs the solving task in a centralized form. 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, of algorithms for distributed constraint solving, we mention ABT [13], ADOPT [6], DPOP [10]. As examples of problems for distributed solving, we mention distributed meeting scheduling [12] and sensor networks [1].

In the centralized case, there are different forms to decompose a problem instance [3]. 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 class of distributed synchronous 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

* This work has been partially supported by the project TIN2006-15387-C03-01.
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 substantial benefits. We have implemented and tested this approach on two different benchmarks; results show a clear reduction in largest message size, with respect the version without function filtering.

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 section 5, including distributed function filtering. The distributed algorithm to build cluster trees appears in section 6, with an example to clarify these new algorithms in section 7. Experimental results on two benchmarks appear 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 $\infty$). Formally, a finite COP is defined by a triple $(X, D, C)$, where

- $X = \{x_1, \ldots, x_n\}$ 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 $\text{var}(f_i) = (x_{i_1}, \ldots, x_{i_{r_i}})$ specifies the cost of every combination of values for the variables in $\text{var}(f_i)$, that is, $f_i : \prod_{j=i}^{r_i} D(x_j) \mapsto N^+$ (where $N^+$ is the set of natural numbers including 0 and $\infty$). The arity of $f_i$ is the cardinality of the set $\text{var}(f_i)$.

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 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 [5]. 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 $\{0, 1, \ldots, k\}$. That is, $f_i : \prod_{j=i}^{r_i} D(x_j) \mapsto \{0, 1, \ldots, k\}$.
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. 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 \(k\). The size of \(f\), denoted \(|f|\), is the cardinal of \(S_f\).

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 \(\alpha\) (resp. \(\beta\)) 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 \(T \subseteq S\), 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 \(T \subseteq \text{var}(f)\), 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 \(t_S \sqcup t'_T\), 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 \(f \in C, f(t_X) = f(t_X[\text{var}(f)])\)). 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 \((T, \chi, \psi)\), where \(T = \langle V, E\rangle\) is a tree, \(\chi\) and \(\psi\) 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 \(v \in V\) s.t. \(f \in \psi(v)\); \(\text{var}(f) \subseteq \chi(v)\);
- 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 \((T, \chi, \psi)\) is \(tw = \max_{v \in V} |\chi(v)|\). If \(u\) and \(v\) are adjacent vertices, \((u, v) \in E\), its **separator** is \(\text{sep}(u, v) = \chi(u) \cap \chi(v)\) [2]. **Summing** two functions \(f\)
and $g$ is a new function $f + g$ with scope $\text{var}(f) \cup \text{var}(g)$ and $\forall t \in \prod_{x_t \in \text{var}(f)} D_t$, $\forall t' \in \prod_{x_t \in \text{var}(g)} D_t$, $(f + g)(t \Join t') = f(t) \oplus g(t')$ ($t \Join t'$ is defined when $t$ and $t'$ share values of common variables). Function $g$ is a lower bound of $f$, denoted $g \leq f$, if $\text{var}(g) \subseteq \text{var}(f)$ 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 $\sum_{g \in G} g \leq f$; $\text{var}(G) = \bigcup_{g \in G} \text{var}(g)$. It is easy to check that for any $f, U \subset \text{var}(f)$, $f[U]$ is a lower bound of $f$, and $\sum_{f \in F} f[U] \leq \left( \sum_{f \in F} f[U] \right)$.

Two algorithms, Cluster-Tree Elimination and Mini-Cluster-Tree Elimination, respectively solve a WCSP instance exact or approximately. They work on a tree decomposition of the WCSP instance. Both are summarized in the following; for a more precise description see [2].

The Cluster-Tree Elimination algorithm (CTE) solves WCSP by sending messages along tree decomposition edges [2]. Edge $(u, v) \in E$ has associated two 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 messages except from $m(v, u)$ and projected on $\text{sep}(u, v)$. CTE is correct, with exponential complexity in time and space [4].

Mini-Cluster-Tree Elimination (MCTE($r$)) approximates CTE [2]. 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 $m(u, v)$ ($M(u, v) \subseteq m(u, v)$). It is computed as $m(u, v)$ but instead of summing all functions of $\psi(v)$ with all incoming messages except from $m(v, u)$, it computes a partition $P = \{ P_1, \ldots, P_q \}$ of those functions, such that the arity of the sum of functions in every $P_t$ does not exceed $r$. Message $M(u, v)$ is composed of $q$ functions; each is the result of summing the functions contained in each $P_t$.

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 [11]. In the context of cluster tree processing, the 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 easily integrates in the previous algorithms, generating their filtering versions. In addition, an iterative procedure of Mini-Cluster with filtering is produced, 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 never worse than 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 $(t \Join t', f(t) \oplus g(t'))$. If $f(t) \oplus g(t') \geq k$, tuple $t \Join t'$ 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 $\hat{T}^H$, as

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

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 $f + g$,

$$f + g = \hat{T}^G + g$$

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

$$S_{f+g} = \{\{t'' = t \not\Join t', f(t) \oplus g(t')\}|t_1 \in P\} \cup \{\{t_2'' = t_2 \not\Join t', f(t_2) \oplus g(t')|t_2 \in Q\}$$

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,

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

which is exactly $\hat{T}^G + g$.

How often do we know that $f$ will be added with $g$? 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, so we know that every $m_{(u,v)}$ will be added with $m_{(v,u)}$ in vertex $u$. 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.

Function filtering easily integrates into CTE. A filtering tree-decomposition is a tuple $(T, \chi, \psi, \phi)$, where $\phi(u,v)$ is a set of functions associated to edge $(u,v) \in E$ with scope included in $\text{sep}(u,v)$. $\phi(u,v)$ must be a lower bound of the corresponding $m_{(u,v)}$ (namely, $\phi(u,v) \leq m_{(v,u)}$). CTEf and MCTEf(r) algorithms 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 $M_{(u,v)}$. 
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. It executes MCTEf(r) using as lower bounds \( \phi(u,v) \) the messages \( M_{(v,u)}^{r-1} \) computed by MCTEf(r − 1) which, recursively, uses the messages \( M_{(v,u)}^{r-2} \) computed by MCTEf(r − 2), and so on.

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 \((X,D,C,A,\beta)\) to solve is arranged in a cluster tree \((T,\chi,\psi)\), where each vertex is a different agent (distributed algorithms to build a cluster tree exist, see section 6). Let us consider \( \text{self} \), a generic agent. It owns a specific vertex in the cluster tree, so \( \text{self} \) knows its position in the tree: it knows its neighboring agents and the separators with them. Besides, \( \text{self} \) also knows variables of \( \chi(\text{self}) \) and cost functions of \( \psi(\text{self}) \).

DCTE exchanges messages among agents. There is one message type, CF, to exchange cost functions. When \( \text{self} \) 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 \( \psi(\text{self}) \), producing a new cost function, which is projected on \( \text{sep}(\text{self},i) \) 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 \( \text{self} \) contains in its cluster (formed by \( \psi(\text{self}) \) and the CF messages received from its neighbors) enough information to solve the particular WCSP instance. When \( \text{self} \) minimizes \( \text{cluster}(\text{self}) \), it finds exactly the global minimum of all initial functions in \( \sum_{f \in \cup \psi(v), v \in V} f \). As consequence, when \( \text{self} \) builds its local optimal assignment, it will be part of a global optimal assignment (if \( O \) is such assignment, \( \text{self} \) will find \( O[\chi(\text{self})] \) as local optimal assignment). After executing DCTE, if there is a single global optimal assignment, then any agent will compute a local optimal assignment that is part of it. But it may happen that several global optimal assignments exist, all sharing the global optimum cost. Let us assume that \( s_1 \) and \( s_2 \) are global optimal assignments of a DWCSP instance distributed between agents \( a_1 \) and \( a_2 \). If \( a_1 \) finds \( s_1[\chi(a_1)] \) as local optimal assignment, and \( a_2 \) finds \( s_2[\chi(a_2)] \) as local optimal assignment, it may happen that each agent will assign different values to variables in \( \text{sep}(a_1, a_2) \). To solve this problem, after DCTE we use SS messages as explained below.

DCTE algorithm appears in Figure 1. It takes as input the tree decomposition \((T,\chi,\psi)\) on the vertex \( \text{self} \) (as explained in the first paragraph of this section), and returns the vertex \( \text{self} \) augmented with a number of cost functions, one per neighbor. These new cost functions plus the cost function of \( \psi(\text{self}) \) are enough to compute the local optimal
\begin{algorithm}
\begin{algorithmic}
  \Procedure{DCTE}{$T, \chi, \psi$}
  \If{$\text{neighbors}(\texttt{self}) = \{j\}$} \texttt{ComputeSendFunction}(\texttt{self}, j); \EndIf
  \While{\neg \text{received and send one CF msg per neighbor}}
    \State \texttt{msg} $\leftarrow$ \texttt{getMsg}();
    \If{$\texttt{msg}.\text{type} = \text{CF}$} \texttt{NewCostFunction}(\texttt{msg}); \EndIf
    \texttt{ComputeSolution}(\emptyset) \EndWhile
  \Procedure{NewCostFunction}{$\texttt{msg}$}
    \State $\texttt{function}[\texttt{msg}.\text{sender}]$ $\leftarrow$ \texttt{msg}.\text{functions};
    \ForEach{$j \in \text{neighbors}(\texttt{self})$ s.t. \texttt{self} has not sent CF to $j$}
      \If{\texttt{self} has received CF msg from all $i \in \text{neighbors}(\texttt{self}), i \neq j$}
        \texttt{ComputeSendFunction}(\texttt{self}, j); \EndIf
    \EndFor
  \EndProcedure
  \Procedure{ComputeSendFunction}{$\texttt{self}, \texttt{dest}$}
    \State $\texttt{Function} \leftarrow \sum_{i \in \text{neighbors}(\texttt{self}), i \neq \texttt{dest}} \texttt{function}[i] + \sum_{f \in \psi(\texttt{self})} f$;
    \State \texttt{sendMsg}($\text{CF}, \texttt{self}, \texttt{dest}, \texttt{Function}[\texttt{sep}(\texttt{self}, \texttt{dest})]$); \EndProcedure
  \Procedure{ComputeSolution}{$\texttt{vars}$}
    \State compute $\texttt{sol}$ minimizing $\text{cluster}(\texttt{self})$, but keeping unchanged in $\texttt{sol}$ the values of variables passed in $\texttt{vars}$; \EndProcedure
\EndProcedure
\end{algorithmic}
\end{algorithm}

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 [2]).

The main procedure is DCTE, which works as follows. If self has a single neighbor, j, 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 $\left(\sum_{f \in \psi(\texttt{self})} f[\text{separator}(\texttt{self}, j)]\right)$, that is sent to agent j. 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 $\psi(\texttt{self})$ 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.

At this point, we use SS messages to assure single assignments of variables in separators. The root agent computes a local optimal assignment minimizing \textit{cluster}(root) and sends a SS message to each child j, with the values of variables in \textit{sep}(root, j). When j receives such message, it computes a local optimal assignment minimizing \textit{cluster}(j) but keeping unchanged the values of variables in \textit{sep}(root, j). Then, j repeats the process, which ends when SS messages reach tree leaves. DCTE is a 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.

DCTE can be easily modified to produce the Distributed Mini-Cluster Elimination algorithm (DMCTE$(r)$). 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($r$) 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($r$) limits the arity of the resulting cost function. In consequence, DMCTE($r$) 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, DMCTE($r$) follows the same strategy as DCTE using $SS$ messages: the root computes its local optimal assignment and sends an $SS$ message to its children with the values of the variables in separators. When a child receives such a message, it computes its local optimal assignment keeping unchanged the received values, and it repeats the process with its children. Differently from DCTE, there is not guarantee that local optimal assignments computed in this way will be part of global optimal ones.

After exchanging $SS$ messages, all agents have agreed on a global assignment of variables in separators. This means that all agents have agreed on a global assignment, of which the values of variables in separators are known to more than one agent (the other values are known by their owning agent, which minimizes its cluster, keeping the values of variables in separators unchanged). The cost of this global assignment is an upper bound of the global optimum (the cost of any global assignment is an upper bound of the global optimum). To distributedly compute the cost of this global assignment, DMCTE($r$) uses $UB$ messages following the communication schema of $CF$ messages. A $UB$ message from $u$ to $v$ contains an upper bound of the cost of the global optimum in the subtree rooted at $u$ that does not include $v$. When $self$ has received $UB$ messages from all its neighbors except perhaps $i$, it adds the received upper bounds (excluding upper bound from $i$) with the cost of $\psi(self)$ on its computed local optimal assignment, producing a new upper bound which is sent to agent $i$.

In summary, DMCTE($r$) 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. They are sent down to the tree, from root to leaves.
- $UB$: 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(r) code due to space limitations. The very same idea of filtering cost functions can be applied to the distributed case, producing the DMCTEf(r) algorithm. 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) is an algorithm that iterates on DMCTEf(r), with increasing r. This algorithm uses previous messages as filters when computing new messages, that is, when computing \( M^i_{(u,v)} \) it is filtered with \( M^{i-1}_{(v,u)} \). The upper bound computed at the end of iteration \( i-1 \) is used as parameter \( ub \) in the iteration \( i \). This causes to decrease the size of messages computed at iteration \( i \), since the filtering process eliminates all tuples with cost higher or equal \( ub \).

6 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 [2]. In the distributed case, of interest here, there are also algorithms able to build a cluster tree in a distributed form. Next we provide a short description of the ERP algorithm [8], 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 \(\beta\) is a function that maps cost functions into agents. We assume that \(\beta\) 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 \(u\), \(\psi(u) = \{f \in C \mid \beta(f) = u\}; \chi^0(u) = \bigcup_{f \in \psi(u)} \text{var}(f)\).
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 \(G\) [9], 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,

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

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.

7 Example

On the instance depicted in Figure 2, with the indicated tree decomposition, we will detail the execution of DCTE, DMCTE and DIMCTE.
Fig. 2. 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 \( \{Z T\} \).

DCTE. Agent \( a_1 \) computes \( f_1 \leftarrow f_{TU} + f_{UV} + f_{VZ} \), projects on the separator and sends the result, \( f_2 = f_1[ZT] \), to \( a_2 \) in a CF message. Analogously, \( a_2 \) computes function \( f_3 \leftarrow f_{XY} + f_{YT} + f_{TZ} \), projects this function on the separator between \( a_1 \) and \( a_2 \), and sends the result, \( f_4 = f_3[ZT] \), to agent \( a_1 \) in a CF message.

Agent \( a_1 \) receives the CF message sent by \( a_2 \) and executes the procedure \texttt{NewCostFunction}, storing the received function and calling \texttt{ComputeSendFunction}. The very same process happens in \( a_2 \) when receiving the CF message from \( a_1 \): \texttt{NewCostFunction} is executed, the received cost function is stored. Now, \( a_2 \) computes its local optimum using the cost functions \( \{f_{XY}, f_{YT}, f_{TZ}, f_2\} \), obtaining \( XYZT \leftarrow bbaa \). Analogously, \( a_1 \) computes its local optimum using the cost functions \( \{f_{TU}, f_{UV}, f_{VZ}, f_4\} \), obtaining \( UVZT \leftarrow bbaa \). Since there is one optimal assignment only, there is no need to exchange SS messages. The global optimum cost is 20, that happens when all variables take value \( b \) except \( T \) that takes value \( a \).
**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 $g_1 = f_{TU}[T]$ and $g_2 = f_{ZV}[Z]$, builds a CF message containing $g_1$ and $g_2$ and sends it to $a_2$. Analogously, $a_2$ computes $g_3 = f_{VT}[T]$ and sends a CF message with $g_3$ and $f_{ZT}$ to $a_1$.

\[
\begin{array}{c|c|c|c|c}
T & g_1 & Z & g_2 & f_{ZT} \\hline
a & 2 & a & 2 & a \\hline
b & 0 & b & 0 & b
\end{array}
\]

\[
\begin{array}{c|c|c|c|c}
Z T & g_3 & f_{ZT} & a & 14 \\hline
& a & b & 10 & 10
\end{array}
\]

Agent $a_1$ computes its local optimum, that is $UVZT \leftarrow bbb$, while $a_2$ does the same, obtaining $XYZT \leftarrow bba$. Since there is discrepancy in the value of $T$, agent $a_1$ sends its optimal values of $Z$ and $T$ to $a_2$ in a SS message. Agent $a_2$ minimizes its cluster keeping the value $b$ for $Z$ and $T$. Now agents exchange their actual costs on the initial cost functions using $UB$ messages. At the end, each agent knows that 22 is the cost of the agreed solution (all $bs$), a true upper bound of the cost of the exact optimum (computed in the DCTE execution).

**DMCTEf.** We take 8 as the limit of the #tuples computed by each agent (8 = $2^3$, 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 $g_4 = f_{TU} + f_{UV}$, filtering with $g_3$ and $f_{ZT}[T]$ (received in the previous iteration, $r = 2$). Four tuples are eliminated because their cost reach the upper bound. It computes $g_5 = g_3(g_1 , f_{ZT}[T])[T]$. Agent $a_1$ filters $f_{ZV}$ with $f_{ZT}[Z]$, but no tuple is eliminated. Its projection on $Z$ generates $g_6 = f_{ZV}f_{ZT}[Z][Z]$. Agent $a_1$ sends a CF message with $g_5$ and $g_6$ to $a_2$. Agent $a_2$ computes $g_7 = f_{XY} + f_{VT}$, filtering its construction with $g_1$ (received in the previous iteration, $r = 2$). Three tuples are eliminated, since their cost is higher than or equal to the current $ub$. Then, it computes $g_8 = g_7^{DCTE_3}[T]$, builds a CF message with $g_8$ and $f_{ZT}$, and sends it to $a_1$.

\[
\begin{array}{c|c|c|c|c}
UVT & \\hline
a & a & a & 3 + 0 + 14 & \\hline
a & a & b & 2 + 4 + 10 & \\hline
a & b & a & 13 + 0 + 14 \geq ub & \\hline
\end{array}
\]

\[
\begin{array}{c|c|c|c|c}
XYT & \\hline
a & a & a & 20 + 2 \geq ub & \\hline
a & b & a & 24 + 0 \geq ub & \\hline
a & b & a & 14 + 2 & a & 14
\end{array}
\]

\[
\begin{array}{c|c|c|c|c}
g_4 : a & b & b & 12 + 4 + 10 \geq ub & \\hline
b & a & a & 12 + 0 + 14 \geq ub & \\hline
b & a & b & 10 + 4 + 10 \geq ub & \\hline
b & b & a & 2 + 0 + 14 & \\hline
b & b & b & 0 + 4 + 10 & \\hline
\end{array}
\]

\[
\begin{array}{c|c|c|c|c}
g_5 : a & 2 & g_6 : a & 2 & g_8 : a & 4 \\hline
b & 0 & b & 0 & b & 12
\end{array}
\]
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 \( SS \) messages cause no change. 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
\[
\begin{align*}
 f_1 &= f_{TU} + f_{UV} + f_{ZV}, \text{ filtering with } f_{ZT} \text{ and } g_8 \text{ (received in the previous iteration, } r = 3) \text{.}
\end{align*}
\]
It happens that all tuples are removed because their cost reach the upper bound.

Agent \( a_2 \) computes
\[
\begin{align*}
 f_2 &= f_{XY} + f_{YT} + f_{ZT}, \text{ filtering it with } g_5 \text{ and } g_6 \text{ (received in the previous iteration, } r = 3) \text{.}
\end{align*}
\]
The same situation happens: all tuples are removed because their cost reach the upper bound.

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 2 for \( \{ZT\} \), and sends it to \( a_2 \). Analogously, \( a_2 \) builds a \( CF \) message with only the tuple \( ba \) with cost 18 for \( \{ZT\} \), and sends it to \( a_2 \).

At this point, \( a_1 \) computes its local optimum \( XYZT \leftarrow bbba \) and \( a_2 \) does the same \( UVZT \leftarrow bbba \). \( SS \) messages cause no change in these optimums. There is no need of \( UB \) 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).
8 Experimental Results

We tested the algorithms presented in this article on two problems: random and meeting scheduling. We generated random problem instances according to the following parameters: number of agents, number of variables, size of value domains and number of unary and binary cost functions. We uniformly spread variables and cost functions among agents and variables, respectively. We randomly filled out cost functions with costs taken from the natural interval \{0 \ldots 9\}. For that problem, a solution is to assign values to variables in such a way the overall cost be minimal.

We generated instances of the distributed meeting scheduling problem considering department hierarchies [7]. Each department consists of a set of people working on it, which have to participate in a set of meetings. For the distributed meeting scheduling problem, a solution is to schedule the meetings in such a way the overall cost be minimal according to the preferences that people have of meetings and time-slots on their own agendas. Every agent represents one person. An agent has multiple variables:
one for the start time of each meeting the agent takes part in. Variable domains have 8
time-slots as values. All meetings last one time-slot. There exist two meeting types: in-
ternal meetings, involving people working on the same department, and external ones,
involving people from different departments. Variables of an agent share mutual exclu-
sion constraints and variables of all agents involved in the same meetings share equality
constraints. Unary constraints represent agents’ personal preferences. For all instances,
the number of attendants for meetings is at most 4.

Experimental results on random instances appear in Figure 3. We observe the largest
message size used by DCTE, the exact algorithm, which is \( d^n \) (domain-size\(\max\)-separator). About DMCTEf\((r)\), the increment in message size with \( r \) never achieves \( d^n \) for all in-
stances tested. DMCTEf\((r)\) provides savings in largest message size from 9.4% to 91%.

Further experimental results on the distributed meeting scheduling benchmark ap-
pear in Figure 4. We observe a similar picture here: DIMCTEf\((r)\) requires messages
of size substantially lower than DCTE. Savings in largest message size go from 62.6%
to 98.4 %. In addition, in three of the four tested instances, DCTE was unable to com-
pute the exact solution because it exhausted memory (messages of size \( 8^7 \) cannot be
handled by our simulator). In these instances, DMCTEf\((r)\) was able to compute the ex-
act solution using substantially footnotesizeer messages. These results illustrate clearly
our approach and show the applicability enhancements of DMCTEf\((r)\) with respect to
DCTE, which was unable to solve three of the four problems considered.

<table>
<thead>
<tr>
<th>#participants</th>
<th>40</th>
<th>60</th>
<th>80</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>#departments</td>
<td>10</td>
<td>20</td>
<td>20</td>
<td>25</td>
</tr>
<tr>
<td>#meetings</td>
<td>15</td>
<td>25</td>
<td>22</td>
<td>52</td>
</tr>
<tr>
<td>#calendar slots</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>#agents</td>
<td>50</td>
<td>98</td>
<td>104</td>
<td>160</td>
</tr>
<tr>
<td>#variables</td>
<td>171</td>
<td>429</td>
<td>408</td>
<td>730</td>
</tr>
<tr>
<td>domain size</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>#cost functions</td>
<td>210</td>
<td>570</td>
<td>536</td>
<td>730</td>
</tr>
<tr>
<td>largest separator</td>
<td>4</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>DCTE</td>
<td>4096(126)</td>
<td>( 8^n ) ('-')</td>
<td>( 8^n ) ('-')</td>
<td>( 8^n ) ('-')</td>
</tr>
<tr>
<td>DIMCTEf((k = 2))</td>
<td>64(141)</td>
<td>64(298)</td>
<td>64(330)</td>
<td>64(493)</td>
</tr>
<tr>
<td>DIMCTEf((k = 3))</td>
<td>512(126)</td>
<td>512(284)</td>
<td>512(313)</td>
<td>512(486)</td>
</tr>
<tr>
<td>DIMCTEf((k = 4))</td>
<td>1530(126)</td>
<td>4096(279)</td>
<td>4096(292)</td>
<td>4096(467)</td>
</tr>
<tr>
<td>DIMCTEf((k = 5))</td>
<td>3278(279)</td>
<td>32768(287)</td>
<td>32768(447)</td>
<td></td>
</tr>
<tr>
<td>DIMCTEf((k = 6))</td>
<td>1029(279)</td>
<td>259584(287)</td>
<td>30063(447)</td>
<td></td>
</tr>
<tr>
<td>DIMCTEf((k = 7))</td>
<td>3278(279)</td>
<td>512(287)</td>
<td>512(447)</td>
<td></td>
</tr>
<tr>
<td>Savings largest message</td>
<td>62.6%</td>
<td>98.4%</td>
<td>87.6%</td>
<td>98.4%</td>
</tr>
</tbody>
</table>

Fig. 4. Message size of DCTE and DIMCTEf\((k)\) for increasing \( k \) on meeting scheduling in-
stances. Between parenthesis, the optimum global cost found at the each iteration ('-' means that
the algorithm execution exhausted memory before reaching the solution).
9 Conclusions

We have presented DCTE, DMCTE$^r$, 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$^r$ limits 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$^r$, which increases its practical applicability. Experimental results confirm the benefits of this approach.

These algorithms are inspired in their centralized counterparts, but their extension requires some care. This is especially true for the DMCTE$^r$ 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 to assess the practical applicability of the presented algorithms on different benchmarks.

References