Improving DPOP with Function Filtering

Ismel Brito
IIIA - CSIC
Campus UAB
08193 Bellaterra, Spain
ismel@iiia.csic.es

Pedro Meseguer
IIIA - CSIC
Campus UAB
08193 Bellaterra, Spain
pedro@iiia.csic.es

ABSTRACT

DPOP is an algorithm for distributed constraint optimization which has, as main drawback, the exponential size of some of its messages. Recently, some algorithms for distributed cluster tree elimination have been proposed. They also suffer from exponential size messages. However, using the strategy of cost function filtering, in practice these algorithms obtain important reductions in maximum message size and total communication cost. In this paper, we explain the relation between DPOP and these algorithms, and show how cost function filtering can be combined with DPOP. We present experimental evidence of the benefits of this new approach.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: Problem Solving, Search

General Terms

Algorithms

Keywords

distributed constraint optimization, agent coordination

1. INTRODUCTION

In the last years, there is an increasing interest for distributed problem solving. When several agents related by constraints look for a global consistent assignment satisfying every constraint, this problem can be seen as distributed constraint reasoning, where each agent owns a part of the instance but no agent knows the whole instance. Agents want to achieve a global consistent solution without joining all information into a single agent. 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 reasoning, we mention ABT [16], ADOPT [7], DPOP [9]. As examples of problems requiring distributed solving, we mention –among many others– distributed meeting scheduling [14] and sensor networks [1].

Considering distributed constraint optimization, the DPOP algorithm [9] represents a useful step forward to distributedly compute the global optimum. Its main drawback is the exponential complexity of its *UTIL* messages, which could reach size $\Theta(d^{w^*})$, where d is the maximum domain size and w^* is the induced width of the DFS tree used. Independently, some algorithms for distributed cluster-tree elimination (DCTE) [2] have been developed. They suffer from the same kind of exponentially large messages. In practice, using the cost function filtering strategy, these algorithms have greatly reduced the maximum message size. In this paper, we analyze the relation of DPOP with DCTE, and we show that the cost function filtering strategy can also be applied in the DPOP context, producing a new algorithm BT-IDPOPf. Experimental results show that BT-IDPOPf achieves substantial reductions in maximum message size and total data exchanged with respect to the original DPOP, especially for medium and high values of w^*. BT-IDPOPf is an anytime algorithm able to provide better solutions as the running time increases, until eventually computing the optimum.

The paper is organized as follows. In section 2, we provide a summary of existing dynamic programming approaches for the centralized and distributed cases. In section 3, we show that DPOP can be seen as a part of DCTE. From this fact, we combine DPOP with cost function filtering, producing the new algorithm BT-IDPOPf in section 4. We provide an example of its execution in section 5. BT-IDPOPf is evaluated empirically in section 6, showing important benefits compared to DPOP. Finally, we conclude in section 7. For space reasons, theorem proofs have been removed.

Cite as: Improving DPOP with Function Filtering, Brito and Meseguer, *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, van der Hoek, Kaminka, Lespérance, Luck and Sen (eds.), May, 10–14, 2010, Toronto, Canada, pp. XXX-XXX.

2. PRELIMINARIES

2.1 Centralized COPs

In a centralized setting, a *Constraint Optimization Problem* (COP) involves a finite set of variables, each 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 positive natural numbers (including 0 and ∞). A finite COP is (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 set of x_i possible values;
- C is a set of cost functions; $f_V \in C$ on the ordered set of variables $V = (x_{i_1}, \ldots, x_{i_r})$ (called its scope) specifies the cost of every combination of values of variables of V, that is, $f_V : \prod_{x_j \in V} D(x_j) \mapsto N$. The arity of f_V is $|V|$. The scope of a cost function $f \in C$ is also written $var(f)$.

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 with an overall cost lower than a threshold provided by the user. It is *optimal* if its overall cost is minimal.

procedure BE(X, D, C)
  for each $i : n, \ldots, 1$ **do**
    $B_i \leftarrow \{f_V \in C | x_i \in V\}$;
    $g_i \leftarrow (\sum_{f_V \in B_i} f_V)[-x_i]$;
    $X \leftarrow X - \{x_i\}$;
    $C \leftarrow (C \cup \{g_i\}) - B_i$;

Figure 1: The BE algorithm.

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 by removing the values of variables that do not appear in T. This idea can be extended to cost functions: the projection of f_V on $T \subset V$, is a new cost function $f_V[T]$ formed by the tuples of f_V removing the values of variables that do not appear in T, removing duplicates and keeping the minimum cost of the original tuples in f_V. The *concatenation* of two tuples t_S and t'_T, written $t_S \cdot t'_T$, is a new tuple with scope $S \cup T$, formed by the values appearing in t_S and t'_T. This concatenation 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 $\sum_{f \in C} f(t_X)$, that is, the addition of the individual cost functions evaluated on t_X (implicitly, it is assumed that, for each $f \in C, f(t_X) = f(t_X[var(f)])$). We define two operations on functions,

1. *Projecting out* a variable $x \in V$ from f_V, denoted $f_V[-x]$, is a new function with scope $V - \{x\}$ defined as projecting f_V on $V - \{x\}$, $f_V[-x] = f_V[V - \{x\}]$.

2. *Summing* two functions f_V and g_W is a new function $f + g$ with scope $V \cup W$ and $\forall t \in \prod_{x_i \in V} D_i$, $\forall t' \in \prod_{x_j \in W} D_j$ s.t. $t \cdot t'$ is defined, $(f + g)_{V \cup W}(t \cdot t') = f_V(t) + g_W(t')$.

We say that 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 $(\sum_{g \in G} g) \leq f$. It is easy to check that for any $f, Y \subset var(f)$, $f[Y]$ is a lower bound of f, and $\sum_{f \in F} f[Y] \leq (\sum_{f \in F} f)[Y]$.

The *primal graph* G of a COP is a graph where each node represents a variable and there exists a link between two nodes if the corresponding variables appear together in the scope of at least one cost function. Given an ordering d of variables, the induced graph G^* is computed as follows: (i) it is initialized with G and (ii) variables are processed from last to first: if the ancestors of variable x (variables connected with x and previous to x in the ordering d) are not connected in G^*, new links connecting them are added [3]. The width of a node x in G^* along the ordering d is the number of variables connected with x in G^* and previous to it in d. The induced width w^* of G is the maximum width of all nodes of G^* along the ordering d.

Bucket Elimination (BE). COPs can be solved by the bucket elimination algorithm (BE) [3], that appears in Figure 1. Given a COP (X, D, C) and an ordering $d = \{x_1, x_2, \ldots, x_n\}$, BE processes variables from last to first. When processing variable x_i, it builds the bucket B_i as the set of all cost functions (original and produced during processing other variables after x_i along d) for which x_i is the highest variable in its scope. A new cost function g_i is computed, by adding cost functions in B_i and projecting out x_i. Cost function g_i replaces the bucket B_i in C and x_i is eliminated from X. This process is repeated n times, eliminating one variable per iteration, until all variables have been processed.

procedure CTE$(T = (V, E), \chi, \psi)$
  for each $(u, v) \in E$ s.t. all $m_{(i,u)}, i \neq v$ have arrived **do**
    $B \leftarrow \psi(u) \cup \{m_{(i,u)} \mid (i, u) \in E, i \neq v\}$;
    $m_{(u,v)} \leftarrow (\sum_{f \in B} f)[sep(u, v)]$;
    send $m_{(u,v)}$ to v;

Figure 2: The CTE algorithm.

After BE, an optimal solution is computed processing variables from first to last. Variable x_i takes the D_i value that is the best extension of $x_1, ..., x_{i-1}$ with respect to B_i.

Cluster Tree Elimination (CTE). A *tree decomposition* (TD) of a COP (X, D, C) is a triple (T, χ, ψ), where $T = (V, E)$ is a tree, χ and ψ are labeling functions which associate with each node $v \in V$ two sets, $\chi(v) \subseteq X$ and $\psi(v) \subseteq C$ such that

- for each $f \in C$, there is exactly one node $v \in V$ s. t. $f \in \psi(v)$; $var(f) \subseteq \chi(v)$;
- for each $x \in X$, the set $\{v \in V | x \in \chi(v)\}$ induces a connected subtree of T.

Its *tree-width* is $tw = max_{v \in V} |\chi(v)|$. If u and v are adjacent nodes, its *separator* is $sep(u, v) = \chi(u) \cap \chi(v)$. The maximum separator size of the TD is s. Finding the TD with the smallest tree width is NP-hard [3].

A COP formulated as a TD can be solved by the *cluster tree elimination* (CTE) algorithm, which appears in Figure 2. CTE sends messages along TD edges [4]. 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 $sep(u, v)$. Its approximate version, MCTE(r), limits by r the maximum arity of new computed functions.

CTE can also be seen as a two phase algorithm. In the first phase, starting from the leaves, each leaf node sends message m to its parent, which in turn receives all messages from its children, performs the necessary computation and sends a message to its parent. This phase terminates when the information reaches the root. In the second phase, the root computes the CTE messages to be sent to its children. Upon reception, each child performs the same process, which terminates when messages reach tree leaves.

After CTE, the optimum of variables in $\chi(u)$ is computed optimizing $\psi(u) \cup \{m_{(v,u)} | v \in neighbors(u)\}$, assigning the same value to variables that appear in more than one node of the TD.

2.2 Distributed COPs

Moving into a distributed context, a *Distributed Constraint Optimization Problem* (DCOP), is a COP where variables, domains and cost functions are distributed among automated agents. A *variable-based* (resp. *cost-function-based*) DCOP is a 5-tuple (X, D, C, A, α) (resp. β), where X, D, C define a COP, A is a set of p agents and α maps each variable to one agent (resp. β maps each cost function to one agent).

DCOPs can be solved either using distributed search or by using distributed inference. About distributed search we mention the reference algorithm ADOPT [7] and their improved versions BnB-ADOPT [15] and ADOPT-ng [13]. About distributed inference, we mention DPOP [9] and DCTE [2]. In the following these two algorithms are summarized (for a more complete description the reader should consult the original sources).

Distributed Cluster Tree Elimination (DCTE). Distributed cluster tree elimination algorithms have been proposed in [2], working on a tree decomposition (TD). A TD can be built in a distributed form using the ERP algorithm [**?**]. A generic agent *self* is a node in the TD, it owns variables in $\chi(self)$ and cost functions in $\psi(self)$. It also knows the separators with its parent and children in the TD.

Basic DCTE works as follows. In the first place, *self* communicates with its neighbors with *CF* messages, that contain cost functions computed as in the centralized case and following the same communication strategy (each arc (u, v) of the tree has associated two messages, one from u to v and other from v to u). When all these messages have been exchanged, $cluster(self)$ is a set of cost functions formed by $\psi(self)$ and the cost functions received from its neighbors. Minimizing $cluster(self)$, each agent obtains the global minimum and a tuple that is part of a global tuple of optimum cost. Secondly, to assure that common variables to several agents take the same value, *SS* messages are used, from root to leaves. Each *SS* message contains the optimal values of variables in $\chi(self)$ which should be maintained by its children.

DMCTE(r) is its approximated version, when the arity of cost functions in *CF* messages is bounded by r. Differently from DCTE, there is no guarantee that DMCTE(r) will find the global optimum. This algorithm returns an interval $[LB, UB]$ where LB and UB are lower and upper bounds of the optimum cost, respectively.

Dynamic Programming Optimization Protocol (DPOP). DPOP works on a DFS-tree arrangement of the constraint graph. Given a graph $G(V, A)$, a rooted DFS-tree is a triple (V, E, PE) where E (edges)$\cup PE$ (pseudoedges) $= A$ (arcs of G), $E \cap PE = \phi$ and (V, E) forms a rooted tree. Nodes in different branches of the DFS-tree (a special case of pseudotree) have no links between them. There are distributed algorithms to compute the DFS-tree. Its induced width w^* is the induced width of the original constraint graph, assuming a depth-first order traversal of the DFS-tree [9].

DPOP performs three phases in sequence: (1) DFS phase. An agent is selected as root (for instance, by a leader election process). From this agent, a distributed DFS traversal of the constraint graph is started. At the end, each agent labels its neighbors as parents, pseudoparents, children or pseudochildren. (2) UTIL phase. Each agent (starting from leaves) sends a *UTIL* message to its parent, that contain an aggregated cost function computed adding received *UTIL* messages from its children with its own cost functions with parent and pseudoparents. The sent cost function does not contain the agent's variable, which is projected out. (3) VALUE phase. Each agent determines its optimal value using the cost function computed in phase 2 and the *VALUE* message received from its parent. Then, it informs its children using *VALUE* messages. The agent at the root starts this phase. In DPOP, *UTIL* and *VALUE* messages play the same role as *CF* and *SS* messages in DCTE.

We identify each phase as DPOP(*phase*). We focus on DPOP(*util*), which is the phase responsible for using exponentially large messages. DPOP(*dfs*) is a preprocess to compute the DFS-tree arrangement, while DPOP(*value*) is a one-pass process from root to leaves, with the best assignments for variables in early levels of the DFS-tree. Both require a polynomial number of linear size messages.

Originally DPOP considers that agents have utilities associated with value tuples, so an optimal solution is the tuple that maximizes the overall utility. In this paper, we consider that agents have costs associated with value tuples, so an optimal solution is the tuple that minimizes the overall cost. Keeping this in mind, and for homogeneity with DCTE, in the following we consider that in the UTIL phase DPOP agents exchange *CF* messages and in the VALUE phase DPOP agents exchange *SS* messages.

3. RELATION BETWEEN DPOP AND DCTE

In this section we study the relations between dynamic programming algorithms in the centralized and distributed cases. A summary of such relations appears in Figure 3.

3.1 DPOP(*util*) and BE

We can see that DPOP(*util*) performs the same process as BE in the centralized case, when BE uses an ordering related to the DFS-tree used by DPOP.

Given a rooted DFS-tree, we define its *depth-first order* as the linear sequence of tree nodes in which they would be first visited by depth-first search (also known as the preorder listing). DPOP on the rooted DFS-tree performs the same kind of computation as BE in the constraint graph with the depth-first order of the rooted DFS-tree, as stated in the next result.

THEOREM 1. *Given a rooted DFS-tree, where DPOP operates. When DPOP(util) processes variable x_i, it computes the same cost function and sends it to the same node as BE when it processes x_i along the depth-first order of the rooted DFS-tree.*

3.2 BE and CTE

In the centralized case, the relation between BE and CTE has been widely studied by Dechter and colleagues [4, 3, 5]. In short, BE can be seen as a part of CTE working on the bucket tree, a particular tree decomposition. In the following, we summarize the main steps to achieve this conclusion.

Following [3], we define the *bucket tree* (BT) of the induced graph G^* along the ordering $d = \{x_1, x_2, \ldots, x_n\}$ of a COP instance as the tuple $(T = (V, E), \chi, \psi)$, where

- there is a node in V per bucket B_j; in total V has n nodes;
- there is an edge in E from B_i to B_j (B_i is parent of B_j) iff x_i is the closest ancestor of x_j in G^* along the ordering d;
- the set $\chi(B_j)$ is composed of x_j and all its previous neighbors in G^* along the ordering d;
- the set $\psi(B_j)$ is composed of all cost functions with x_j as the highest variable in its scope.

The size of the largest separator s of the BT is the induced width w^* of G^* [3] (we use them interchangeably in the BT). It can be proved [3], that this BT is a legal TD, so it can be processed by CTE. Keeping the view of CTE as two phase algorithm, we can prove that BE on d is equivalent to the 1st phase of CTE.

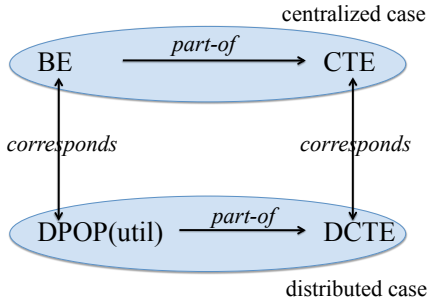

Figure 3: Relations between centralized/distributed dynamic programming algorithms.

THEOREM 2. *(from [3]) When BE processes variable x along the ordering d, it computes the same cost function as CTE(phase 1) when processing B_x. CTE(phase 1) sends this function to the bucket of the variable to which BE sends the function.*

3.3 CTE and DCTE

DCTE [2] mimics the behavior of CTE in the distributed case. Both work on a TD of the considered instance, which can be computed in centralized or distributed form. CTE is a message-passing algorithm, and its extension to the distributed case is as follows: DCTE uses *CF* messages (that play the same role as m messages in CTE), and *SS* messages, to propagate the solution chosen by agents at higher levels of the TD to agents in lower levels (*SS* messages are not needed in the centralized case because all the information is accessible to CTE, executed in a single computer).

Its approximate version, DMCTE(r), is closely related to the centralized MCTE(r) with this difference: while MCTE(r) limits the arity of new cost functions computed in a TD node, DMCTE(r) limits the size of messages sent by this node. DMCTE(r) also uses *BB* messages, to compute and propagate the lower and upper bounds in the TD. For a detailed description of DCTE-based algorithms, see [2].

Cost function filtering was originally proposed for the centralized case [12], and its extension to the distributed case appears in [2], jointly with its usage inside DCTE-based algorithms. It is described in detail in section 4.1.

3.4 DPOP(*util*) and DCTE

As BE can be seen as a part of CTE in the centralized case, DPOP(*util*) can be seen as a part of DCTE in the distributed case (since DPOP(*util*) performs the same process as BE in the centralized case, BE is part of CTE and CTE performs the same process as DCTE in the distributed case). In the following, we formally prove that DPOP can be seen as a part of DCTE.

Given a rooted DFS-tree, we have defined the depth-first order of its nodes. Given a node ordering, we have defined the BT (a TD of this ordering with n nodes). So from the DFS-tree we define the corresponding BT. Based on this we connect DPOP(*util*) with DCTE, as stated next.

THEOREM 3. *Given a rooted DFS-tree, where DPOP operates. When DPOP(util) processes variable x, it computes the same cost function as DCTE(1 phase) when it processes B_x. DCTE(phase 1) sends this function to the bucket of the variable to which DPOP(util) sends the function.*

Given a DFS-tree computed in distributed form, can we built the corresponding BT also in a distributed way? (or should we pass through centralized formats?) Yes, BT can be built in distributed form, and the procedure to do it appears below. From the DFS-tree, we define the induced DFS*-tree as follows:

- nodes of the DFS-tree are processed per branch, in the order from leaf to root;
- the process of node x_i is as follows: for each pair $x_j, x_k \in \{parent(x_i)\} \cup pseudoparents(x_i)$ with x_j higher in the tree than x_k, send a message to both asking that if x_j is not parent of x_k, a pseudoedge connecting x_j and x_k should be included (x_j should be included in the pseudoparents of x_k and x_k in the pseudochildren of x_j).

From the DFS*-tree, we build the corresponding BT:

- there is one bucket per node of the DFS*-tree: variable x_i corresponds to B_i;
- the parent of B_i in the BT is the bucket corresponding to the variable parent of x_i in the DFS*-tree (which is the same as in the DFS-tree);
- $\chi(B_i)$ is composed by the parent and pseudoparents of x_i;
- $\psi(B_i)$ is composed of all cost functions that x_i has with its parent and pseudoparents, such that x_i is the last variable of its scope in the branch of the DFS*-tree.

The whole process requires exchanging a linear number of constant size messages. It is easy to see that this is the corresponding BT (it follows the guidelines presented in section 3.2, replacing G^* by DFS*-tree and ordering d by depth-first order). It is direct to check that, given a DFS-tree, there is only one DFS*-tree and BT associated with it.

4. DPOP + FUNCTION FILTERING

In practice, cost function filtering has provided substantial benefits when used inside DCTE-based algorithms [2]. Given the relation between DPOP and DCTE, we accommodate cost function filtering inside DPOP.

4.1 Cost Function Filtering

The strategy of cost function filtering was proposed in [12] to improve centralized COP solving. In the distributed context, its usage has been proposed in [2] inside DCTE-based algorithms. The basics of cost function filtering are as follows.

Let us consider a DCOP and a generic agent *self* which knows a global upper bound UB on the maximum acceptable cost of any solution. A cost function f is stored as a set containing all pairs $(t, f(t))$ with cost less than UB. Imagine that *self* knows that cost function f will be added (in the future) with cost function g, and it also knows a set of functions G that is a lower bound of g. We define the filtering of f from G, noted $\overline{f}^G$, as

$$\overline{f}^G(t) = \begin{cases} f(t) & if \quad (\sum_{h \in G} h(t)) + f(t) < UB \\ UB & otherwise \end{cases}$$

where $\overline{f}^G(t)$ is the *filtered cost* of tuple t (by functions f and G).

THEOREM 4. *Let f and g be two cost functions, $var(g) \subseteq var(f)$, and G a set of functions that is a lower bound of g. Filtering f with G before adding f to g is equal to $f + g$,*

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

Therefore, if *self* has to send cost function f to another agent, instead of sending f it can send $\overline{f}^G$, which is smaller (or equal in the worst case). The idea is not sending those elements of f which, when added with g, will exceed UB so they will be removed from further computation. Not sending them will allow us to save communication effort, while keeping the correctness and complenetess of the base algorithm on top of which cost function filtering is implemented. This strategy also works with lower bounds of the exact cost functions, as stated in the next result.

THEOREM 5. *Let f and g be two cost functions, $var(g) \subseteq var(f)$, and G a set of functions that is a lower bound of g. Let f_{LB} and g_{LB} be two lower bounds of f and g respectively. Then,*

$$\overline{f_{LB}}^G + g_{LB} \leq f + g$$

How does cost function filtering help in DCTE-based algorithms? Let us consider two agents u and v, that exchange cost function messages. The cost function from u to v will be added with other cost functions of $cluster(v)$ to compute the minimization. So any of the functions that are already in $cluster(v)$ can filter the cost function from u to v (obviously before arriving at v). A similar situation happens in agent u with cost function from v. DMCTE(r) exchanges lower bounds of the exact cost functions and DMCTEf(r) uses cost function filtering with these lower bounds. Expecting better lower bounds the DIMCTEf algorithm has been proposed. It is an iterative algorithm that executes DMCTEf(r) with increasing r, using the *CF* message from u to v of iteration $r-1$ as filter for the *CF* message from v to u at iteration r [2].

4.2 New DPOP Versions

Since DPOP can be seen as part of DCTE-based algorithms, cost function filtering can also be applied to DPOP, producing new algorithms which are presented in this section. These algorithms no longer work on the DFS-tree of original DPOP, but on the BT defined by this DFS-tree, following the guidelines of section 3.4. Now, an agent *self* is a node of the BT, which knows variables in $\chi(self)$ and cost functions in $\psi(self)$. It also knows its parent and children nodes, with their corresponding separators.

The main difference of these algorithms with DCTE-based ones is the ordering of *CF* messages: while in DCTE-based algorithms agents send *CF* messages as soon as the sending condition holds, in the new DPOP versions *CF* messages are sent in two phases: first, from leaves to root, and second from root to leaves (of the BT). In the first phase, an internal node has to wait until receiving *CF* messages from all its children; in the second phase it has to wait until receiving the *CF* message from its parent.

Ordering *CF* messages could seem a minor element in the algorithm design. However, it is crucial to extract the maximum power from cost function filtering. When the $CF(u,v)$ message is sent from the parent u to v (second phase), u has already received the $CF(v,u)$ message sent in the first phase (from v), so it can use $CF(v,u)$ to filter $CF(u,v)$. In DIMCTEf, since messages are not ordered, in iteration r a $CF(u,v)$ message is filtered by the corresponding $CF(v,u)$ message of previous iteration $r-1$.

The basic algorithm is BT-DPOP(r) (bucket tree DPOP(r)), which appears in Figure 4. It is composed of the following phases (it is described for a generic agent *self*, we assume that the BT has already been built in a preprocessing step):

1. First phase: let B^1_{self} be the set of all cost functions received by *self* from its children union $\psi(self)$. B^1_{self} is partitioned in $P_1, P_2, \ldots, P_q$ classes, such that the function obtained adding all cost functions of a class projected on the separator $sep(self, parent(self))$ has arity lower than or equal to r. The cost functions of each class are added, producing q cost functions, which form a new *CF* message which is sent to $parent(self)$.

2. Second phase: let u be a child of *self* and let B^2_{self} be the set of all cost functions received by $self$ from its children except u union $\psi(self)$ union the cost functions received from its parent. B^2_{self} is partitioned in $P_1, P_2, \ldots, P_q$ classes, such that the function obtained adding all cost functions of a class, projected over $sep(self, u)$ has arity lower than or equal to r. All cost functions of each class are added, producing q cost functions, which form a new CF message which is sent to u. A similar process is done for all children of $self$.

3. Third phase: it is DPOP($value$) (it exchanges SS messages).

4. Fourth phase: after executing DPOP($value$), agents have assigned their variables. Agent $self$ computes a partial upper bound minimizing $\psi(self)$ but keeping the values received in SS messages. In addition, $self$ minimizes the set $cluster(self)$ without any restriction in the value of its variables, computing a cost that is a lower bound lb_{self} of the optimum cost. To compute a global upper bound and take the maximum global lower bound, information is exchanged using BB messages. A BB message from u to v contains

procedure BT-DPOP-1(T, χ, ψ, r, ub) /* *CF* messages up*/
  $thereIsSol \leftarrow$ false;
  if $self = leaf(T)$ **then** ComputeSendFunction($self, parent(self), r$);
  while $\neg$ (received one CF per children and sent one CF to parent) **do**
    $msg \leftarrow$ getMsg(); **if** $(msg.type) = CF$ **then** NewCostFunction(msg, r);

procedure BT-DPOP-2(T, χ, ψ, r, ub) /* CF messages down*/
  if $self = root(T)$ **then**
    for each $j \in children(self)$ ComputeSendFunction($self, j, r$);
  else $msg \leftarrow$ getMsg(); **if** $(msg.type) = CF$ **then** NewCostFunction(msg, r);

procedure BT-DPOP-3(T, χ, ψ) /* SS messages down*/
  if $self = root(T)$ **then**
    $sol \leftarrow$ ComputeSolution($\emptyset$);
    for each $j \in children(self)$ **do** SendSolutionSeparator($self, j, sol$);
  else
    $msg \leftarrow$ getMsg();
    if $(msg.type) = SS$ **then**
      $sol \leftarrow$ ComputeSolution($msg.solsep$);
      for each $j \in children(self)$ **do** SendSolutionSeparator($self, j, sol$);

procedure BT-DPOP-4(T, χ, ψ) /* BB messages first up, second down*/
  if $self = leaf(T)$ **then**
    $LB \leftarrow$ minimum $cluster(self)$; ComputeSendBounds($self, parent(self)$);
  else
    while $\neg$ (received/sent one *BB* per neighbor) **do**
      $msg \leftarrow$ getMsg(); **if** $(msg.type) = BB$ **then** NewBounds(msg);
  return $(LB, \sum_{j \in neighbors(self)} ub[j] + \sum_{f \in \psi(self)} f(sol))$;

procedure NewCostFunction(msg, r)
  $functions[msg.sender] \leftarrow msg.functions$;
  for each $j \in neighbors(self)$ s.t. *self* has not sent *CF* to j **do**
    if *self* has received *CF* msg from all $i \in neighbors(self), i \neq j$ **then**
      ComputeSendFunction($self, j, r$);

procedure NewSolutionSeparator(msg)
  $sol \leftarrow$ ComputeSolution($msg.solsep$);
  for each $j \in children(self)$ **do** SendSolutionSeparator($self, j$);
  if $neighbors(self) = \{k\}$ **then**
    $LB \leftarrow$ minimum $cluster(self)$; ComputeSendBounds($self, k$);

procedure NewBounds(msg)
  $ub[msg.sender] \leftarrow msg.upperBound$; $LB \leftarrow max\{msg.lowerBound, LB\}$;
  if *thereIsSol* **then**
    for each $j \in neighbors(self)$ s.t. *self* has not sent *BB* to j **do**
      if *self* has received *BB* msg from all $i \in neighbors(self), i \neq j$ **then**
        ComputeSendBounds($self, j$);

procedure ComputeSendFunction($self, dest, r$)
  $B \leftarrow \{functions[i] | i \in neighbors(self), i \neq dest\} \bigcup \psi(self)$;
  $\{P_1 \ldots P_q\} \leftarrow$ partition($B, r, sep(self, dest)$);
  for each $k : 1 \ldots q$ **do**
    sendMsg($CF, self, dest, \{(\sum_{g \in P_k} g)[sep(self, dest) \cap (\cup_{g \in P_k} var(g))]\}$);

procedure SendSolutionSeparator($self, dest$)
  sendMsg($SS, self, dest, \{sol[x] \mid x \in sep(self, dest)\}$);

procedure ComputeSendBounds($self, dest$)
  $UB \leftarrow \sum_{j \in neighbors(self), j \neq dest} ub[j] + \sum_{f \in \psi(self)} f(sol)$;
  sendMsg($BB, self, dest, UB, LB$);

function ComputeSolution($vars$)
  $thereIsSol \leftarrow$ true;
  return assignment minimizing $cluster(self)$, keeping the values of $vars$;

Figure 4: The BT-DPOP(r) algorithm.

```
function BT-IDPOPf(T, χ, ψ, δ)
  for each j ∈ neighbors(self) do filter[j] ← ∅;
  UB ← ∞; Δ ← 0; r ← 0;
  repeat
    r ← r + 1;
    BT-DPOPf-1(T, χ, ψ, r, UB − Δ);
    if not received an empty function then
      filter[j] ← functions[j];
      if r < s then BT-DPOPf-2(T, χ, ψ, r, UB − Δ);
      if not received an empty function then
        BT-DPOP-3(T, χ, ψ);
        if r < s then
          [lb_r, ub_r] ← BT-DPOP-4(T, χ, ψ);
          LB ← max_k{lb_k | k : 1, ..., r};
          UB ← min_k{ub_k | k : 1, ..., r};
          Δ ← (UB × δ)/100;
          for each j ∈ neighbors(self) do filter[j] ← functions[j];
  until r = s ∨ UB − Δ ≤ LB ∨ received an empty function;
  return UB;
```

Figure 5: The BT-IDPOPf algorithm.

two parts: an upper bound of the cost of the global optimum in the subtree rooted at u that does not include v, and a global lower bound. When u has received BB messages from all its neighbors except perhaps v, it adds the received upper bounds (excluding upper bound from v) with its partial upper bound, producing a new upper bound. As lower bound, it takes the maximum between its lower bound and the lower bound contained in the message. A new BB message is formed containing these new upper and lower bounds, which is sent to agent v. The process terminates when two BB messages have been exchanged in each arc of the BT. At every agent, the global upper bound is computed as the sum of the partial upper bound (the sum of cost functions in $\psi(self)$) with the received upper bounds. Every agent has the same global lower lb and upper ub bounds. This algorithm returns the interval $[lb, ub]$.

Provided with the corresponding filters for CF messages computed in phase 1 and phase 2, this algorithm becomes BT-DPOPf(r). From it, we propose BT-IDPOPf, an iterative algorithm that executes BT-DPOPf(r) with increasing r. In phase 1 of iteration r, to filter a CF message from u to its parent v, it uses the CF message from v to u of iteration $r-1$. In phase 2 of iteration r, to filter a CF message from v to u, it uses the CF message from u to v of iteration r (which has been sent in phase 1). In addition, when $r = s$ (the maximum separator size), only phases 1 and 3 are executed. BT-IDPOPf appears in Figure 5, a detailed description follows:

1. First phase: as phase 1 of BT-DPOP(r), filtering a CF message from u to v with the CF message from v to u of iteration $r-1$. As UB it uses the lowest ub returned at previous iterations.
2. Second phase: if $r < s$, as phase 2 of BT-DPOP(r), filtering a CF message from v to u with the CF message from u to v of current iteration. Same UB as first phase.
3. Third phase: as phase 3 of BT-DPOP(r).
4. Fourth phase: if $r < s$, as phase 4 of BT-DPOP(r).

BT-IDPOPf terminates when (i) $r = s$, or (ii) $LB = UB$, or (iii) an agent computes an empty cost function. In cases (i) and (ii), the assignment computed at iteration r is an optimal solution and its cost is the optimum. In case (iii), the solution is the assignment computed at iteration $r-1$. This algorithm computes the optimum cost and terminates, as stated in the next result.

THEOREM 6. *The algorithm BT-IDPOPf computes the optimum cost, provides an optimal solution and terminates.*

In fact, the termination condition (ii) is $UB - \frac{UB \times \delta}{100} \leq LB$, that is, BT-IDPOPf terminates when the computed cost UB is within a δ percentage of the optimum, using the cost interval returned by BT-DPOPf(r). When $\delta = 0$, it computes the optimum cost, but when $\delta > 0$ the returned cost is, at most, δ percentage above the optimum. This property allows the algorithm to stop execution before reaching the optimum. This is very useful in practice, because many applications require to achieve good solutions (with low cost) but they do not require to achieve the optimum (minimum) cost. Besides, BT-IDPOPf can also be seen as an *anytime* algorithm, able to improve the solution quality over time.

It is worth comparing BT-IDPOPf with the approximated DPOP version A-DPOP [10]. They present the following differences. Each iteration of BT-IDPOPf requires two *CF* messages per link (up, down), while A-DPOP requires one *UTIL* per link (up). If A-DPOP looks for the exact solution, there is no gain in the message size of the original DPOP. However, if BT-IDPOPf looks for the exact solution, large gains in message size may occur. In a hypothetical iterative A-DPOP, there is no direct way to make messages shorter.

5. EXAMPLE

Let us consider the instance depicted in Figure 6. It is formed by seven agents, each owning a variable. Domains are $\{a, b\}$ and binary cost functions are indicated. We provide a DFS tree of the constraint graph and the corresponding BT.

An example of the execution of BT-IDPOPf appears in Figure 7. For $r = 2$ message size is limited to 2^2, so in some cases cost functions have to be partitioned ($B_R \rightarrow B_V$ and $B_V \rightarrow B_R$). Function filtering causes no pruning, because in the first iteration the upper bound is initialized to ∞. In phase 4, agents exchange their partial upper bounds and global lower bounds. At the end, each agent returns the interval $[44, 45]$. For $r = 3$ message size is limited to 2^3, but this does not limit exchanged cost functions (the size of maximum separator is 3). Now the upper bound is 45, and function filtering causes real pruning. In Figure 7 there are a number of cost functions for which not every value combination

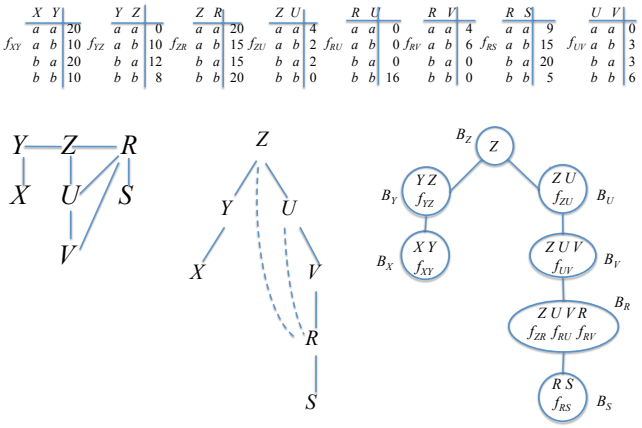

Figure 6: DCOP instance, each agent holds a variable, domains are $\{a, b\}$, cost functions are indicated. From left to right: constraint graph, DFS-tree rooted at Z, the corresponding bucket tree.

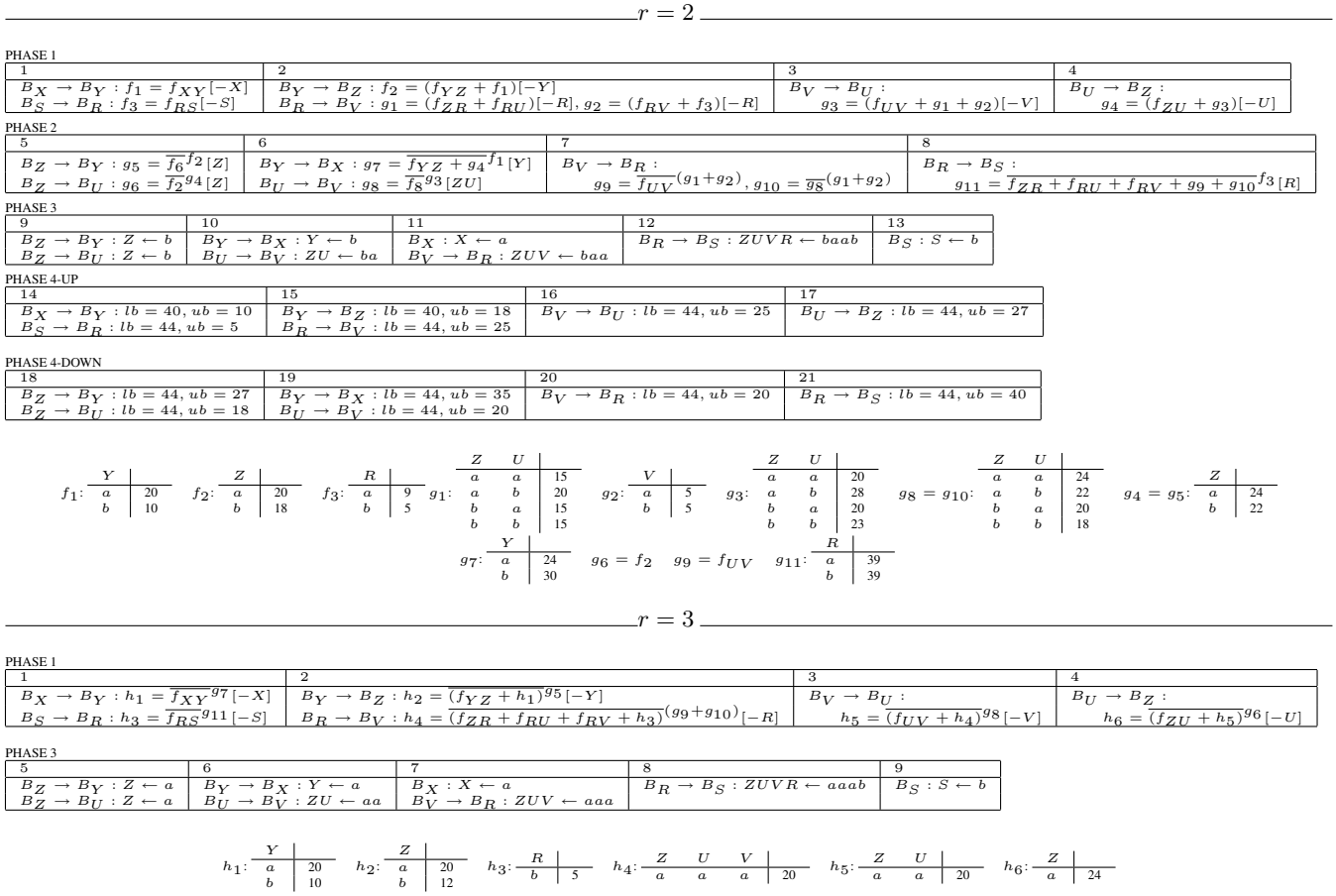

Figure 7: Trace of BT-IDPOPf on the instance of Figure 6, for $r = 2$ and $r = 3$. Exchanged cost functions appear below. For $r = 3$ phases 2 and 4 are not needed because 3 is the size of the largest separator. The optimum costs 44 and the assignment $ZYXUVRS \leftarrow aaaaabb$ is optimal.

appears. The missing combinations have been pruned because their filtered cost exceeded the upper bound. Phases 2 and 4 are not needed, because $r = 3$ equals the highest separator size. We have computed the optimum (cost 44) and an optimal assignment. It is worth noting that this execution looks for optimal solutions. If the user accepts solutions within 5% of the optimum, BT-IDPOPf would not have executed this iteration. Effectively, since it reported in the previous $r = 2$ iteration a solution $ZYXUVRS \leftarrow bbaaabb$ with cost 45 which is within 5% of the optimum (computed using the interval [44,45]), $r = 3$ iteration would have not been executed.

6. EXPERIMENTAL RESULTS

We tested DPOP and BT-IDPOPf on two benchmarks: random DCOPs and distributed meeting scheduling problems. We generated random instances according to the following parameters: number of agents = number of variables, size of domains and number of unary and binary cost functions. We randomly filled out cost functions with costs taken from $\{0, \ldots, 9\}$. An optimal solution is to assign values to variables such that the overall cost is minimum.

We generated instances of the distributed meeting scheduling problem considering department hierarchies [6]. Each department consists of a set of people working in it, which have to participate in a set of meetings. An optimal solution is to schedule the meetings in such a way that the overall cost is minimum according to the preferences that people have of meetings and time-slots on their own agendas. A person 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: internal meetings, involving people working on the same department, and external ones, involving people from different departments. A person's variables share mutual exclusion constraints and variables of people involved in the same meeting share equality constraints. Unary constraints represent personal preferences. A meeting involves at most 4 people.

Experimental results on random DCOPs are similar to those of meeting scheduling. For space reasons we focus on meeting scheduling results, that appear in Figure 8. We provide the largest message size and the total data exchanged (in Kbytes) for DPOP, and BT-IDPOPf for each iteration (until a termination condition is satisfied). We consider two scenarios: one where we accept optimal solutions only ($\delta = 0\%$), and other where we accept solutions which are at most 5% distant from the optimum ($\delta = 5\%$). For each BT-IDPOPf iteration, we provide the interval $[lb, ub]$. When LB is higher than or equal to $UB(1 - \frac{\delta}{100})$, the computed solution is within δ distance from the optimum so BT-IDPOPf terminates.

First, we consider optimal solutions ($\delta = 0\%$). Regarding largest message size, BT-IDPOPf causes a substantial improvement with respect to DPOP. Regarding total data exchanged, we observe a

instance	A		B		C		E		F		G		H	
#participants	40		50		70		40		100		60		90	
#departments	10		10		10		8		25		20		15	
#meetings	15		22		24		18		52		25		32	
#agents=#vars	66		96		96		80		246		144		112	
#cost functions	210		340		342		324		968		570		606	
largest separator	4		5		5		6		7		8		8	
optimum cost	126		183		187		137		458		279		263	
Algorithm	Largest message	Total data	Largest message	Total data	Largest message	Total data	Largest message	Total data	Largest message	Total data	Largest message	Total data	Largest message	Total data
DPOP	16(126)	179	128(183)	461.2	128(187)	855.2	1024(137)	6363.9	8192(458)	16011.8	65536	$(-)$	65536	$(-)$
BT-IDPOPf														
$(r = 2)$	0.3[114,135]	36.7	0.3[177,188]	56.4	0.3[175,191]	48.5	0.3[132,144]	72.4	0.3[350,516]	188.6	0.3[240,345]	155.3	0.3[235,280]	123.8
$(r = 3)$	2.0[121,130]	75.9	2.0[177,184]	85.3	2.0[181,191]	102.1	2.0[135,141]	109.9	2.0[429,476]	649.1	2.0[256,321]	527.0	2.0[245,266]	407.4
$(r = 4)$	1.4[126,126]	19.5	*4.1[182,184]	*72.4	12.9[182,191]	102.9	*12.5[137,137]	*87.0	16.0[431,476]	2192.9	16.0[273,302]	2120.9	16.0[257,268]	786.9
$(r = 5)$			*0.5[183,183]	*9.3	*1.4[187,187]	*29.0			128.0[458,467]	5046.2	123.7[279,281]	4065.6	*67.9[262,268]	*632.4
$(r = 6)$									*17.6[458,458]	*123.7	*24.0[279,279]	*92.8	*821.3[262,263]	*4202.4
$(r = 7)$													*987.6(empty f)	*2930.6
Savings $\delta = 0\%$	88%	26%	97%	52%	90%	67%	99%	96%	98%	49%	99%		99%	
Savings $\delta = 5\%$	88%	26%	98%	69%	90%	70%	99%	97%	98%	50%	99%		99%	

Figure 8: Largest message and total data transfer of DPOP and BT-IDPOPf for increasing r on meeting scheduling instances, in Kbytes. All domains (any variable, any instance) have size 8. For DPOP, the optimum global cost appears between parenthesis ("$-$" means that the algorithm execution exhausted memory before reaching the solution). For BT-IDPOPf, the lower and upper bounds returned at each iteration appear between brackets. The user is willing to accept solutions whose cost surpasses the optimum up to δ ($\delta = 0\%$ means optimum only). With asterisk (*) we denote iterations that are executed for $\delta = 0\%$ but not for $\delta = 5\%$.

similar picture: BT-IDPOPf requires exchanging far less data than DPOP. In fact, there are two instances that cannot be solved by DPOP (instances G and H, our simulator cannot handle messages of size 65536 Kbytes), which is solved by BT-IDPOPf using much shorter messages. Considering MB-DPOP, the bounded memory version of DPOP [11], reduction in message size is limited to $\Theta(d^r)$, which would cause messages of size 0.25Kb, 2Kb, 16Kb, 128Kb, 1024Kb, 8192Kb, 65536Kb, for $r = 2, 3, 4, 5, 6, 7, 8$ respectively. However, BT-IDPOPf uses messages of the same size for low r but shorter for high r (see Figure 8). For the total data exchanged, MB-DPOP requires more data than DPOP (Figure 3 of [11]), but BT-IDPOPf requires less data than DPOP.

Secondly, if we allow for a solution with a cost within 5% of the optimum, benefits increase because the termination condition on bounds becomes looser: it passes from $LB = UB$ to $LB \geq UB \times \frac{95}{100}$. For many problem instances, BT-IDPOPf stops before computing the optimum, saving several iterations for the highest values of r (the most expensive ones), which represents further savings in communication and computation. Regarding largest message size, BT-IDPOPf savings with respect to DPOP are really good. Regarding total data exchanged, a similar picture appears: BT-IDPOPf exchanges much less data than DPOP. It is really illustrative to observe the capacity of reasoning with bounds, able to stop execution when bounds are close enough and, in many cases, saving some of the most costly iterations (marked with * in Figure 8).

7. CONCLUSIONS

We have improved the DPOP algorithm with cost function filtering. In practice, we obtain substantial reductions in maximum message size and total data exchanged. These benefits are significant for low w^* but increase as w^* increases, reaching orders of magnitude savings. Our approach provides a bounded interval of costs, which allows us to stop execution when the cost of the current solution is not more than a percentage away from the optimum.

8. ACKNOWLEDGEMENTS

This work is partially supported by the project TIN2009-13591-C02-02. We thank reviewers for their useful criticisms.

9. REFERENCES

[1] Bejar R., Fernandez C., Valls M., Domshlak C., Gomes C., Selman S., Krishnamachari B. Sensor networks and distributed CSP: Communication, computation and complexity. *Artificial Intelligence*, 161:117–147, 2005.

[2] Brito I., Meseguer P., Distributed Cluster Tree Elimination. *IJCAI-09* workshop on DCR, 2009.

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

[4] Dechter R., Kask K., Larrosa J. A general scheme for multiple lower bound computation in constraint optimization. *Proc. CP-01*, 2001.

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

[6] Maheswaran R., Tambe M., Bowring E., Pearce J., Varakantham P. Taking DCOP to the Real World: Efficient Complete Solutions for Distributed Multi-Event Scheduling *Proc. AAMAS-04*, 2004.

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

[8] Paskin M., Guestrin C., McFadden J.A robust architecture for distributed inference in sensor networks. *Proc. IPSN*, 2005.

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

[10] Petcu A., Faltings B. Approximations in Distributed Optimization *CP-05* workshop on DCR, 2005.

[11] Petcu A., Faltings B. MB-DPOP: A new memory-bounded algorithm for distributed optimization *Proc. IJCAI-07*, 2007.

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

[13] Silaghi M. Yokoo M. Nogood-based Asynchronous Distributed Optimization. *Proc. AAMAS-06*, 2006.

[14] Wallace R., Freuder E. The Distributed Constraint-based reasoning and privacy/efficiency tradeoffs in multi-agent problem solving. *Artificial Intelligence*, 161:209–227, 2005.

[15] Yeoh W., Felner A. Koenig S. BnB-ADOPT: An Asynchronous Branch-and-Bound DCOP Algorithm *Proc. AAMAS-08*, 591–598, 2008.

[16] 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.