Including Soft Global Constraints in DCOPs

Patricia Gutierrez, Pedro Meseguer, Christian Bessiere

Including Soft Global Constraints in DCOPs*

Christian Bessiere[1], Patricia Gutierrez[2], and Pedro Meseguer[2]

[1] University of Montpellier, France
bessiere@lirmm.fr
[2] IIIA - CSIC, Universitat Autònoma de Barcelona
08193 Bellaterra, Spain.
{patricia|pedro}@iiia.csic.es

Abstract. In the centralized context, global constraints have been essential for the advancement of constraint reasoning. In this paper we propose to include soft global constraints in distributed constraint optimization problems (DCOPs). Looking for efficiency, we study possible decompositions of global constraints, including the use of extra variables. We extend the distributed search algorithm BnB-ADOPT$^+$ to support these representations of global constraints. In addition, we explore the relation of global constraints with soft local consistency in DCOPs, in particular for the generalized soft arc consistency (GAC) level. We include specific propagators for some well-known soft global constraints. Finally, we provide empirical results on several benchmarks.

1 Introduction

Distributed Constraint Optimization Problems (DCOPs) are commonly used for modeling many multi-agent coordination problems. DCOPs are formalized in terms of agents, variables with finite domains and cost functions (a particular case of soft constraints [12]). Cost functions are used to evaluate the cost of variable assignments. Agents should find a complete value assignment with minimum sum of costs. It is usually assumed that each agent handles a single variable, and it also knows about the domain and cost functions associated with that variable.

In the centralized context, global constraints have been essential for the advancement of constraint reasoning. The well-known *alldifferent(T)* global constraint means that all the variables in the set T must assign a different value (independently of the cardinality of T). Soft global constraints are associated with a violation measure that defines the costs of value assignments. For example, the *soft-alldifferent(T)* is associated with the violation measure μ_{var} (the number of variables in T that have to change their value to satisfy that all are different), or with μ_{dec} (the number of pairs of variables in T with the same value [14]).

In the distributed context, global constraints have been studied in the satisfaction case [2]. However, to the best of our knowledge, no relation between DCOPs and soft

* Christian Bessiere is partially supported by the FP7-FET ICON project 284715 and by the "Agence Nationale de la Recherche" project ANR-10-BLA-0214. Patricia Gutierrez and Pedro Meseguer are partially supported by the projects TIN2009-13591-C02-02 and Generalitat de Catalunya 2009-SGR-1434. Patricia Gutierrez has an FPI scholarship BES-2008-006653.

$D_{x_1} = \{a\}$ $D_{x_2} = \{a, b\}$ $D_{x_3} = \{a, b\}$

x_1	x_2	x_3	μ_{var}
a	a	a	2
a	a	b	1
a	b	a	1
a	b	b	1

x_1	x_2	μ_{var}
a	a	1
a	b	0

x_1	x_3	μ_{var}
a	a	1
a	b	0

x_2	x_3	μ_{var}
a	a	1
a	b	0
b	a	0
b	b	1

Fig. 1. (*Left*) *soft-alldifferent* global constraint with μ_{var} violation measure; (*right*) a decomposition in binary constraints. However, *soft-alldifferent* is not binary decomposable with μ_{var}, because $\mu_{var}^{x_1,x_2,x_3}(a,a,a) = 2 \neq \mu_{var}^{x_1,x_2}(a,a) + \mu_{var}^{x_1,x_3}(a,a) + \mu_{var}^{x_2,x_3}(a,a) = 3$.

global constraints have been established. In this paper, we advocate for the inclusion of soft global constraints in DCOPs. We assume that a soft constraint instance can be expressed as a cost function in the weighted model [12], so these terms are used interchangeably in the paper. In DCOPs it is a common assumption that cost functions are binary, that is, defined over two variables. However, not every cost function relation can be decomposed into an equivalent set of binary ones. For example, consider the *soft-alldifferent* constraint with the violation measure μ_{var} (represented by the cost function that appears in Figure 1 left). Observe that the tuple $(x_1 = a, x_2 = a, x_3 = a)$ has a different cost in the global formulation –involving all variables– and in the binary formulation (three cost functions in Figure 1 right). Hence, this soft constraint is not binary decomposable with violation measure μ_{var}.[1] In general, most soft global constraints are not binary decomposable, so working with their original formulations is crucial for their effective inclusion in DCOPs.

Our proposal enhances DCOP expressivity since not every cost function can be expressed as a set of binary cost functions. We also investigate several decompositions of soft global constraints, including decompositions with extra variables, looking for the one that provides the best performance in DCOP solving. We extend the distributed search algorithm BnB-ADOPT$^+$ to support different decompositions of soft global constraints. In addition, we explore the relation of global constraints with soft local consistency in DCOPs, in particular with the generalized soft arc consistency (GAC) level. On the one hand, the quality of the bounds obtained as result of applying local consistency is often better when the problem contains global constraints than when it contains an equivalent binary formulation. On the other hand, enforcing GAC on global constraints can be expensive using generic propagators. In the worst case, this is exponential in the number of variables. However, efficient propagators have been proposed for some global constraints that exploit constraint semantics, reaching the consistency level with lower complexity (usually polynomial) than with generic propagators.

The paper is structured as follows. In Section 2 we define some concepts needed for the rest of the paper. We analyze the decomposition of global constraints in a polynomial number of fixed arity constraints in Section 3. We explain how to include global constraints in DCOPs in Section 4, jointly with the extension of the distributed search algorithm BnB-ADOPT$^+$ to handle them (without and with GAC enforcement). We provide an empirical evaluation of the proposed techniques in Section 5. Finally, we conclude the paper in Section 6.

[1] However *soft-alldifferent* is binary decomposable with violation measure μ_{dec} [14].

2 Preliminaries

Concepts such as constraint optimization, soft global constraints and soft arc consistency have been defined using a centralized point of view, but they can easily be generalized to the distributed context. Here we recall the original definitions, the DCOP generalization and a short description of the BnB-ADOPT$^+$ algorithm.

COP. A *Constraint Optimization Problem* (COP) is defined by $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ where: $\mathcal{X} = \{x_1, \ldots, x_n\}$ is a set of variables. $\mathcal{D} = \{D_{x_1}, \ldots, D_{x_n}\}$ is a set of finite domains such that D_{x_i} is the value set for x_i. $\mathcal{C}$ is a finite set of cost functions, where every cost function $C(T) : \prod_{x_i \in T} D_{x_i} \mapsto N \cup \{0, \infty\}$ on the ordered subset of variables $T = (x_1, \ldots, x_r)$ specifies the costs of every combination of values on T. It is worth noting that hard constraints can be modelled in this formalism associating $0/\infty$ costs with permitted/forbidden tuples, respectively. When a cost function $C(T)$ is evaluated on a value tuple t we follow the notation: $C_T(t)$. The cost of a tuple t is calculated adding all individual cost functions evaluated on t. If $\top$ is the lowest unacceptable cost, a *solution* is a tuple t containing a complete variable assignment with cost lower than $\top$. An *optimal solution* is a solution with minimum cost.

These problems are in many cases solved using a branch-and-bound schema, usually enhanced with sophisticated methods to improve lower bound computation (maintaining some forms of local consistency at each node). This facilitates pruning of the current branch and removal of future values, which improves performance.

Soft Global Constraints. A soft global constraint C is a class of soft constraints whose arity is not fixed. Constraints with different arities can be defined by the same class. For instance, *soft-alldifferent*(x_1, x_2, x_3) and *soft-alldifferent*(x_1, x_4, x_5, x_6) are two instances of the *soft-alldifferent* global constraint. The cost of global constraints is evaluated using a violation measure μ. A soft global constraint C with violation measure μ is *contractible* iff μ is a non-decreasing function [10].[2] A soft global constraint C with violation measure μ admits a *binary decomposition without extra variables* iff for any instance $C(x_1, \ldots, x_p)$ of C, there exists a set S of binary soft constraints involving only variables $x_1, \ldots, x_p$ such that for any value tuple t on $x_1, \ldots, x_p$, $\sum_{C(x_i,x_j) \in S} C_{x_i,x_j}(t[x_i, x_j]) = \mu(t)$. We also say that C is semantically decomposable in S.

Soft Arc Consistency. We consider a COP: (i, a) means x_i taking value a, $\top$ is the lowest unacceptable cost, $C(x_i)$ is the unary cost function on x_i values, C_ϕ is a zero arity cost function that represents a lower bound of the cost of any solution. As [8, 9], we consider the following local consistencies:

- *Node Consistency**: (i, a) is node consistent* (NC*) if $C_\phi + C_{x_i}(a) < \top$; x_i is NC* if all its values are NC* and there is $a \in D_{x_i}$ s.t. $C_{x_i}(a) = 0$; a problem is NC* if every variable is NC*.
- *Generalized Arc Consistency**: (i, a) is generalized arc consistent (GAC) wrt. a non-unary cost function $C(T)$, if there exist a value tuple t on T such that $(i, a) \in t$

[2] Function f on a sequence is non-decreasing if $f(\mathbf{a}) \leq f(\mathbf{b})$, for every sequence $\mathbf{a}$ and $\mathbf{b}$ such that $\mathbf{a}$ is a prefix of $\mathbf{b}$ [10]. The intuition behind is as follows: C with μ is contractible when $\mu(a, b, c) \leq \mu(a, b, c, d) \leq \mu(a, b, c, d, e)$...., so shortening by the right the sequence of variables on which C is defined gives a valid lower bound to the cost of C. This is in relation with the nested representation, defined in Section 4.

and $C_T(t) = 0$; x_i is GAC if all its values are GAC wrt. every cost function involving x_i; a problem is GAC* if every variable is GAC and NC*.

In the following we refer to NC* and GAC* as NC and GAC, without asterisk. GAC can be reached by shifting costs from the problem and deleting values not NC. Cost are shifted with *equivalent preserving transformations* in the following way: first projecting the minimum cost from non-unary cost functions to unary costs functions, and then projecting the minimum cost from unary cost functions into C_ϕ. After projection, node inconsistent values are deleted. When a value is deleted in x_i, GAC is rechecked on every variable that x_i is constrained with, so a deleted value might cause further deletions. The GAC check must be performed until no further values are deleted. The systematic application of these operations (projection and deletion of node inconsistent values) does not change the optimum (for details on projections and optimality, see [8]).

DCOP. A *Distributed Constraint Optimization Problem* (DCOP) [13] is defined by $(\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{A}, \alpha)$ where $\mathcal{X}, \mathcal{D}$ and $\mathcal{C}$ define a COP and: $\mathcal{A} = \{a_1, \ldots, a_p\}$ is a set of agents. $\alpha : \mathcal{X} \rightarrow \mathcal{A}$ maps each variable to one agent. Solving a DCOP is an NP-hard task. Agents communicate and coordinate while looking for the optimal solution through messages. In this paper, it is assumed that: messages are never lost; messages sent from one agent to another are delivered in the same order they were sent; α maps a single variable to each agent, so we use the terms variable and agent interchangeably.

BnB-ADOPT$^+$. BnB-ADOPT [16] is a reference algorithm for optimal DCOP solving. Agents are arranged in a depth-first search (DFS) pseudo-tree and asynchronously perform a depth-first-branch-and-bound search until an optimal solution is found. Agents may have a parent, children (connected by tree edges of the pseudo-tree), pseudoparents and pseudochildren (connected by back-edges of the pseudo-tree) [13]. Each agent $self$ holds a $context$ that is updated with message exchange. The $context$ holds a set of assignments involving $self$ ancestors. Agents exchange the following messages:

- VALUE(i, j, val, th): agent i informs child or pseudochild j that it has taken value val with threshold th;
- COST$(k, j, context, lb, ub)$: agent k informs parent j that with $context$ its bounds are lb and ub;
- TERMINATE(i, j): agent i informs child j that agent i terminates.

A BnB-ADOPT agent executes the following loop: it reads and processes all incoming messages and assigns a value. Then, it sends a VALUE to each child or pseudochild and a COST to its parent. When BnB-ADOPT terminates, each agent has assigned the optimum value for its variable. We use the BnB-ADOPT$^+$ version [7], which saves redundant messages. For more details, see [16, 7].

3 Soft Global Constraint Decompositions in DCOP

In this Section we analyze the different forms of decomposing a soft global constraint into a polynomial number of smaller constraints of fixed arity, in the DCOP context (with the standard assumption that each agent owns a single variable).

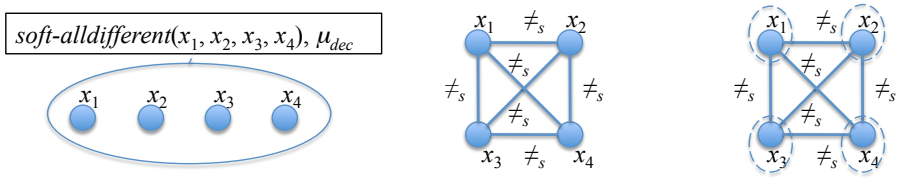

Fig. 2. *Left*: The *soft-alldifferent*(x_1, x_2, x_3, x_4) global constraint with the μ_{dec} violation measure. *Center*: Its binary decomposition, $\neq_s$ stands for soft binary inequality. *Right*: Binary decomposition in DCOP; agents are represented with discontinuous lines.

3.1 Decompositions without Extra Variables

As previously mentioned, some global constraints are semantically decomposable in a set of binary constraints on the variables of the global constraint. For example, in the hard case *alldifferent* is semantically decomposable in a clique of binary inequality constraints between the variables involved in the global constraint. Passing to the soft case, the *soft-alldifferent* global constraint with the violation measure μ_{dec} is semantically equivalent to a clique of soft binary inequalities. A soft binary inequality has 0 cost if the involved variables have different values and a cost of 1 if they have the same value. Including the binary decomposition of a soft global constraint does not cause extra difficulties in most DCOP solving algorithms (you are simply adding some extra soft binary constraints that are treated as any other soft constraint). Figure 2 shows the decomposition of *soft-alldifferent* into a clique of soft binary inequalities.

3.2 Decompositions with Extra Variables

In the hard case, there are global constraints that are not binary decomposable but they can be decomposed in a polynomial number of smaller, fixed arity constraints [4, 3], if we allow a polynomial number of *extra variables*. For example, the hard $atmost[k, v](y_1, ..., y_p)$ global constraint establishes that value v cannot appear more than k times in $\{y_1, ..., y_p\}$. Allowing $p+1$ extra variables $\{z_0, z_1, ..., z_p\}$ with domains $D_{z_j} = \{0, 1, ..., j\}$, p new ternary constraints:

$$\text{if } y_i = v \text{ then } z_i = z_{i-1} + 1 \text{ else } z_i = z_{i-1} \qquad i : 1, ..., p$$

and one unary constraint:

$$z_p \leq k$$

It is easy to see that the original constraint is semantically equivalent to this set of new constraints. Variables $\{z_0, z_1, ..., z_p\}$ are acting as counters: z_i contains the number of times value v appears in the original variables $y_1, ..., y_i$. Variables $\{z_0, z_1, ..., z_p\}$ are called extra variables because they are not present in the original problem definition. However, they are treated as any other problem variable.

Passing to the soft case, the *soft-atmost*$[k, v](y_1, ..., y_p)$ has the following meaning: if value v appears less than or k times in the set $\{y_1, ..., y_p\}$ that assignment costs 0, otherwise it costs the number of times v appears minus k. This soft constraint can be decomposed with extra variables as follows. We keep the same extra variables as in the hard decomposition $\{z_0, z_1, ..., z_p\}$ with the same domains $D_{z_j} = \{0, 1, ..., j\}$. Previous p ternary constraints remain as hard constraints modelled in the soft formalism (permitted/forbidden tuples 0/∞ cost) plus the unary constraint that becomes the soft one:

$$\text{if } z_p \leq k \text{ then cost} = 0 \text{ else cost} = z_p - k$$

In tabular form with μ as cost, each ternary constraint generates a table similar to the table on the left, while the unary constraint generates a table similar to the table on the right:

z_{i-1}	y_i	z_i	μ
c	v	$c+1$	0
c	$\neq v$	c	0
otherwise			∞

z_p	μ
$\leq k$	0
$> k$	$z_p - k$

The proposed decomposition appears in Figure 3.[3]

Allowing extra variables in DCOP, a question naturally follows: which agent owns these extra variables, which have no real existence? To solve this issue we propose to add a number of *virtual agents*, to own these extra variables. While this approach allows to keep the assumption that each agent owns a single variable, a new issue appears on the existence and activity of virtual agents with respect to real agents. Previous approaches have used the idea of virtual agents to accommodate modifications or extensions that deviate from original problem structure [13]. In addition, all variables are treated in the same way, one variable per agent, so no preference is given to a particular subset of variables in front of others. Implementation maintains uniformity for all variables.

Virtual agents can be simulated by real agents. If some real agents have substantial computational/communication resources, they can host some virtual agents. The precise allocation of virtual agents depends on the nature of the particular application to solve.

4 Adding Soft Global Constraints in DCOP

We consider several ways to model the inclusion of a soft global constraint in DCOPs, looking for the one that gives the best performance. The user chooses one of the three representations and the solving is done on that representation. In the rest of this Section, the term "constraint" always mean "soft constraint" (either global or not).

We assume that agents are ordered. The evaluation of a global constraint $C(T)$ by every agent depends on the selected model. We analyze the three following representations:

[3] Observe that local consistencies on decompositions of global cost functions has recently been explored in [1].

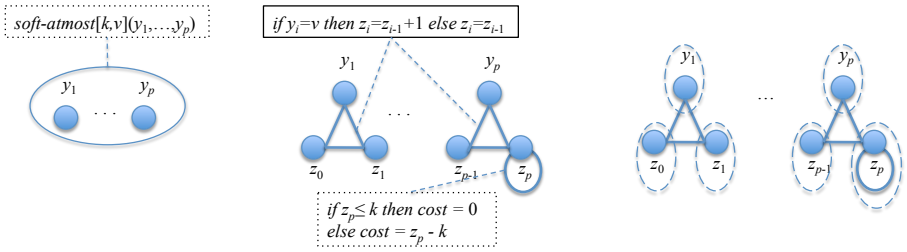

Fig. 3. *Left*: The *soft-atmost*$[k, v](y_1, ..., y_p)$ soft global constraint. *Center*: Its decomposition in p ternary and one unary constraint. *Right*: This decomposition in the distributed context; agents are represented with discontinuous lines.

- Direct representation. C is treated as a generic constraint of arity $|T|$. Only one agent involved in the constraint evaluates it: the one that appears last in pseudo-tree ordering.
- Nested representation. If C is contractible, then C allows nested representation. The nested representation of $C(T)$ with $T = \{x_{i_1}, \ldots, x_{i_p}\}$ is the set of constraints $\{C(x_{i_1}, \ldots, x_{i_j})$ with $j \in 2 \ldots p\}$. For instance, the nested representation of *soft-alldifferent*(x_1, x_2, x_3, x_4) is the set $S = \{$*soft-alldifferent*(x_1, x_2), *soft-alldifferent*(x_1, x_2, x_3), *soft-alldifferent* $(x_1, x_2, x_3, x_4)\}$. The nested representation has the following benefit. Since x_2, x_3 and x_4 are the last agent of a constraint in S, any of them is able to evaluate that particular constraint. When assignments are made following the order x_1, x_2, x_3, x_4, every intermediate agent is able to aggregate costs and calculate a lower bound of the current partial solution. Since C is contractible, this bound increases monotonically on every agent. By this, it is possible to calculate updated lower bounds during search and backtrack earlier if the current solution has unacceptable cost.
- Bounded arity representation. If C is binary decomposable without extra variables, agent $self$ includes all constraints of the binary decomposition of C that involve x_{self} in their scope. Otherwise, if C is decomposable with extra variables, agent $self$ includes all constraints of the decomposition of C that involve x_{self} in their scope. On the contrary with previous representations (direct and nested), constraints included by *self* are non-global.

Since the nested representation allows to calculate updated bounds and performs efficient backtracking, it is expected to be more efficient than the direct representation. However not all global constraints are contractible, so the direct representation has to be analyzed. In the bounded arity representation every intermediate agent is an evaluator, as in the nested representation.

4.1 Search with BnB-ADOPT$^+$

From now on, we differentiate between two types of constraint instances: global constraints (as a result of the direct and nested representations) and non-global constraints

(coming from bounded arity representation of global constraints, as well as particular constraints that may exist in the considered problem).

BnB-ADOPT$^+$ can be generalized to handle constraints of any arity. It is required that each constraint is evaluated by the last of the agents involved in the constraint in the partial ordering of the pseudo-tree, while other agents have to send their values to the evaluator. This simple strategy is mentioned in [16, 13]. We assume that our version of BnB-ADOPT$^+$ includes this generalization.

We have extended the distributed search algorithm BnB-ADOPT$^+$ [7] to support global constraints. The following modifications are needed:

1. ($self$ denotes a generic agent) $self$ keeps a set of global constraints, separated from the set of non-global constraints it is involved. Agent $self$ knows about (and stores) a constraint C iff $self$ is involved in C. Every constraint $C(T)$ implicitly contains the agents involved in T (neighbors of $self$). For some global constraints, additional information can be stored. For example, for the *soft-atmost*$[k, v]$ constraint, parameters k (number of repetitions) and v (value) are stored.
2. During the search process, every time $self$ needs to evaluate the cost of a given value v, all local costs are aggregated. Non-global constraints are evaluated as usual, and global constraints are evaluated according to their violation measure.
3. VALUE messages are sent to agents, depending on the constraint type:
   - For a non-global constraint, VALUE messages are sent to all the children and the last pseudochild in the ordering (the deepest agent in the DFS tree involved in the constraint evaluates it; VALUE to children are needed because they include a threshold required in BnB-ADOPT; observe that for binary constraints this is the original BnB-ADOPT behavior).
   - For a global constraint, there are two options:
     - For the direct representation, VALUE messages are sent to all the children and the last pseudochild in the ordering (the deepest agent in the DFS tree involved in the constraint evaluates it; VALUE to children are needed because they include a threshold required in BnB-ADOPT). [4]
     - For the nested representation, VALUE messages are sent to all children and all pseudochildren (any child or pseudochild is able to evaluate a constraint of the nested representation).
4. COST messages include a list of all the agents that have evaluated a global constraint. This is done to prevent duplication of costs when using the nested representation and it is explained in the next paragraphs.

Figure 4 shows the pseudocode for cost aggregation in BnB-ADOPT$^+$ (lines [1-4]). Costs coming from non-global constraints are calculated as usual, aggregating all non-global constraint costs evaluated on $self$ value and the assignments of the current context (lines [5-13]). Costs coming from global constraints are calculated in lines [14-27]. Although there is no need to separate non-global from global cost aggregation,

[4] In distributed search, a global constraint in the direct representation has the same treatment as a non-global one. However, when GAC is enforced, global and non-global constraints are treated differently (Section 4.2).

```
(1) procedure CalculateCost(value)
(2)   cost = cost + NonGlobalCostWithValue(value);
(3)   cost = cost + GlobalCostWithValue(value);
(4)   return cost;

(5) function NonGlobalCostWithValue(value)
(6)   cost = 0;
(7)   for each nonGlobal ∈ nonGlobalConstraintSet do
(8)     assignments = new list(); assignments.add(self, value);
(9)     for each (x_i, d_i) ∈ context do
(10)      if x_i ∈ nonGlobal.vars then assignments.add(x_i, d_i);
(11)    if assignments.size == nonGlobal.vars.size then          //self is the last evaluator
(12)      cost = cost + nonGlobal.Evaluate(assignments);
(13)  return cost;

(14)  function GlobalCostWithValue(value)
(15)    cost = 0;
(16)    for each global ∈ globalConstraintSet do
(17)      assignments = new list(); assignments.add(self, value);
(18)      for each (x_i, d_i) ∈ context do
(19)        if x_i ∈ global.vars then assignments.add(x_i, d_i);
(20)      if assignments.size == global.vars.size then                    //self is the last evaluator
(21)        cost = cost + global.μ.Evaluate(assignments);
(22)      else                                                  //self is an intermediate agent in the restriction
(23)        if NESTED representation then
(24)          for each x_i ∈ global.vars do
(25)            if lowerGlobalEvaluators.contain(x_i) then cost = cost + 0;
(26)            else cost = cost + global.μ.Evaluate(assignments);
(27)    return cost;
```

Fig. 4. Aggregating costs of binary and global cost functions.

we have presented them in separate procedures for a better understanding of the new modifications.

For every global constraint of the set (*globalConstraintSet*) $self$ creates a tuple with the assignments in its current context (*assignments*, lines [17-19]). If $self$ is the deepest agent in the DFS tree (taking into account the variables involved in the global constraint) then $self$ evaluates the constraint (lines [20-21]). If $self$ is an intermediate agent, it does the following. If representation is direct, $self$ cannot evaluate the global constraint: it does nothing and cost remains unchanged. If representation is nested, it requires some care. A nested global constraint is evaluated more than once by intermediate agents and if these costs are simply aggregated duplication of costs may occur. To prevent this, COST messages include the set of agents that have evaluated global constraints (*lowerGlobalEvaluators*). When a COST message arrives, *self* knows which agents have evaluated its global constraints and contributed to the lower bound. If some of them appear in the scope of C, then *self* does not evaluate C (lines [25-26]). By doing this, the deepest agent in the DFS tree evaluating the global constraint precludes any other agent in the same branch to evaluate the constraint, avoiding cost duplication. Preference is given to the deepest agent because it is the one that receives more value assignments and can perform a more informed evaluation. When bounds coming from a branch of the DFS are reinitialized (this happens under certain conditions in BnB-ADOPT, for details see [16]), the agents in the set *lowerGlobalEvaluators* lying on that branch are removed.

BnB-ADOPT$^+$-UGAC messages:
VALUE($sender$, $destination$, $value$, $threshold$, $\top$, C_ϕ)
COST($sender$, $destination$, $context[]$, lb, ub, $\underline{subtreeContr}$, $\underline{lowerGlobalEvaluators}$)
STOP($sender$, $destination$, $emptydomain$)
$\underline{\text{DEL}(sender, destination, value)}$

Fig. 5. Messages of BnB-ADOPT$^+$-UGAC. New parts wrt. BnB-ADOPT$^+$ are underlined.

4.2 Propagation with BnB-ADOPT$^+$

Specific propagators exploiting the semantics of global constraints have been proposed in the centralized case [9]. These propagators allow to achieve generalized arc consistency in polynomial time whereas a generic propagator is exponential in the number of variables in the scope of the constraint.

Soft local consistency is based on *equivalent preserving transformations* where costs are shifted from non-unary cost functions to unary cost functions. The same technique can be applied in distributed. We project costs from non-global/global cost functions to unary cost functions and finally project unary costs to C_ϕ. After projections are made agents check their domains searching for inconsistent values. For this, some modifications are needed:

- The domain of neighboring agents (agents connected with $self$ by soft constraints) are represented in $self$.
- A new DEL message is added to notify value deletions.
- COST and VALUE messages include extra information.

Following the technique proposed in [6], we maintain GAC during search performing only unconditional deletions, so we call it unconditional generalized arc consistency (UGAC). [5] An agent $self$ deletes a value v *unconditionally* if this value is assured to be sub-optimal and does not need to be restored again during the search process. If $self$contains a value v not NC ($C_{self}(v) + C_\phi > \top$) then v can be deleted unconditionally because the cost of a solution containing the assignment $self = v$ is necessarily greater than $\top$. We also detect unconditional deletions in the following way. Let us consider agent $self$ executing BnB-ADOPT$^+$. Suppose $self$ assigns value v and sends the corresponding VALUE messages. As response, COST messages arrive. We consider those COST messages whose context is simply $(self, v)$. This means that the bounds informed in these COST messages only depend on $self$ assignment (observe that the $root$ agent always receives such COST messages). If the sum of the lower bounds contained in those COST messages exceeds $\top$, v can be deleted unconditionally because the cost of a solution containing the assignment $self = v$ is necessarily greater than $\top$.

As in [6], messages include information required to perform deletions, namely $\top$ (the lowest unacceptable cost), C_ϕ (the minimum cost of any complete assignment), the subtree contribution to C_ϕ (each node k computes the contribution to the C_ϕ of the subtree rooted at k), and the set of agents lower than the current one that are evaluators

[5] Previous results [5] indicate that conditional deletions do not always pay off in terms of communication cost. Because of that, we concentrate here on unconditional deletions.

of global constraints in which the current is involved. These four elements travel in existing BnB-ADOPT$^+$ messages (the first two in VALUE messages, the last two in COST messages). In addition, a new message DEL($self, k, v$) is added, to notify agent k that $self$ deletes value v. The structure of these new messages appears in Figure 5. When $self$ receives a VALUE message, $self$ updates its local copies of $\top$ and C_ϕ if the values contained in the received message are better (lower $\top$ or higher C_ϕ). When $self$ receives a COST message from a child c, $self$ records c subtree contribution to C_ϕ and the list of lower agent global evaluators. When $self$ receives a DEL message, $self$ removes the deleted value from its domain copy of the sender agent and performs projections from the soft constraints involving the sender agent to its unary costs and to C_ϕ. When $\top$ or C_ϕ change, D_{self} is tested for possible deletions.

This mechanism described to detect and propagate unconditional deletions is similar to the one proposed in [6]. However, to reach the GAC level agents need to project costs not only from binary cost functions, but from global cost functions as well. In the following, we describe how to project binary and global costs specifically from the *soft-alldifferent* and *soft-atmost* global constraints.

Projecting cost with bounded arity constraints. The projection of costs from cost function $C(T)$ to the unary cost function $C_{x_i}(a)$, where T is a fixed set of variables, $x_i \in T$ and $a \in D_{x_i}$ is a flow of costs defined as follows. Let α_v be the minimum cost in the set of tuples of $C(T)$ where $x_i = a$ (namely $\alpha_a = min_{t \in tuples\ s.t.\ x_i = a} C_T(t)$). The projection consists in adding α_a to $C_{x_i}(a)$ (namely, $C_{x_i}(a) = C_{x_i}(a) + \alpha_a, \forall a \in D_{x_i}$) and subtracting α_a from $C_T(t)$ (namely, $C_T(t) = C_T(t) - \alpha_a, \forall t \in tuples\ s.t.\ x_i = a, \forall a \in D_{x_i}$). Every agent in T performs projections following a fixed order (projections are done first over higher agents in the pseudo tree). As a result, cost functions are updated the same way in all agents.

Projecting costs with *soft-alldifferent*. We follow the approach described in [9] for the centralized case, where GAC is enforced on the *soft-alldifferent* constraint in polynomial time, whereas it is exponential when a generic algorithm is used.

A graph for every *soft-alldifferent* constraint is constructed following [15]. This graph is stored by the agent and updated during execution. Every time a projection operation is required, instead of exhaustively looking at all tuples of the global constraint, the minimum cost that can be projected is computed as the flow of minimum cost of the graph associated with the constraint [9]. Minimum flow cost computation is based on the successive shortest path algorithm, which searches shortest paths in the graph until no more flows can be added to the graph. Pseudocode appears in Figure 6.

Evaluation of these propagators in the distributed context is an extra issue because it is not based on table look-ups. In the centralized case, they are usually evaluated by their CPU time. An evaluation proposal appears in Section 5.

Projecting costs with *soft-atmost*. For the *soft-atmost* global constraint we propose the following technique to project costs from the global constraint *soft-atmost*$[k, v](T)$ to the unary cost functions $C_{x_i}(v)$. Agent x_i counts how many agents in T have a singleton domain $\{v\}$. If the number of singleton domains $\{v\}$ is greater than k, a minimum cost equal to the number of singleton domains $\{v\}$ minus k can be added to the unary cost $C_{x_i}(v)$ in one of the agents of the global constraint. We always project on the first agent of the constraint (we choose the first agent because in case of value deletion the search

```
(1)  procedure ProjectFromAllDiffToUnary(global, v)
(2)  graph = graphsSet.get(global);                    //the graph associated with global is fetched
(3)  minCost = minCost + getMinCostFlow(graph);
(4)  for each x_i ∈ global.vars do
(5)    minCost = minCost + CostWithFlow(graph, x_i, v);
(6)    if minCost > 0 then
(7)      graph.getArc(x_i, v).cost = cost − minCost;
(8)      if x_i = self then C_self(v) = C_self(v) + minCost;

(9)  function getMinCostFlow(graph)
(10)   graph.SuccesiveShortestPath();
(11)   minCostFlow = 0;
(12)   for each arc ∈ graph.arcs do
(13)     minCostFlow = minCostFlow + (arc.flow ∗ arc.cost);
(14)   return minCostFlow;

(15) function CostWithFlow(graph, x_i, v)
(17)   if graph.arc(x_i, v).flow = 0 then return 0;
(19)   path = graph.residualGraph.FindShortestPath(x_i, v);//shortest path from v to x_i in the residual graph
(20)   flow = ∞; cost = 0;                              //calculate flow as the minimum capacity in this path
(21)   for each arc ∈ path do
(22)     if arc.capacity < flow then flow = arc.capacity;
(24)   for each arc ∈ path do
(25)     cost = flow ∗ arc.cost;
(26)   return cost;
```

Fig. 6. Projection with *soft-alldifferent* global constraint.

```
(1)  procedure ProjectFromAtMostToUnary(global, v)
(2)  if global.v = v and D_self.contains(v) then
(3)    singletonCounter = 0; cost = 0;
(4)    for each x_i ∈ global.vars do
(5)      if D_{x_i}.contains(v) and D_{x_i}.size() = 1 then
(6)        singletonCounter = singletonCounter + 1;
(7)    if singletonCounter > global.k then
(8)      cost = singletonCounter − global.k;
(9)      if cost > global.projectedCost then
(10)       cost = temp;
(11)       cost = cost − global.projectedCost;
(12)       global.projectedCost = temp;
(13)       if global.vars[0] = self then C_self(v) = C_self(v) + cost;
```

Fig. 7. Projection with *soft-atmost*$[k, v]$ global constraint.

space reduction is larger). To maintain equivalence, the *soft-atmost* constraint stores this cost, that will be decremented from any future projection performed. Pseudocode appears in Figure 7.

5 Experimental Results

To evaluate the impact of including soft global constraints, we tested on several random DCOPs sets including *soft-alldifferent* and *soft-atmost* global constraints. The first set of experiments considers binary random DCOPs with 10 variables and domain size of 5. The number of binary cost functions is $n(n-1)/2*p_1$, where n is the number of variables and p_1 varies in the range $[0.2, 0.9]$ in steps of 0.1. Binary costs are selected from an uniform cost distribution. Two types of binary cost functions are used, cheap and

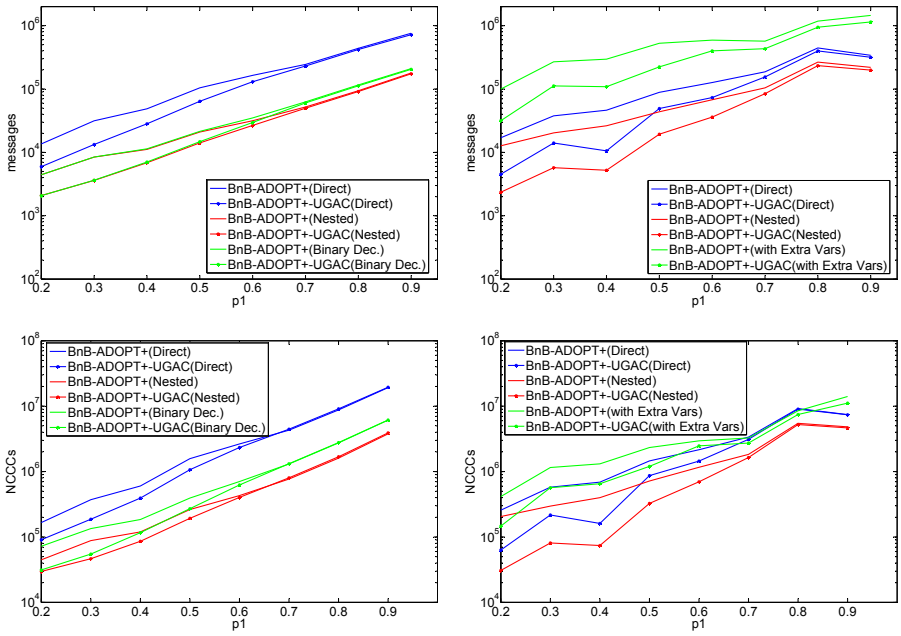

Fig. 8. Experimental results of random DCOPs including (*left*) *soft-alldifferent* global constraints with the violation measure μ_{dec}; (*right*) *soft-atmost* global constraints with the violation measure μ_{var}.

expensive. Cheap cost functions extract costs from the set $\{0, ..., 10\}$ while expensive ones extract costs from the set $\{0, ..., 1000\}$. The proportion of expensive cost functions is 1/4 of the total number of binary cost functions (this is done to introduce some variability among binary tuple costs [6]). In addition to binary constraints, global constraints are included. The first set of experiments includes 2 *soft-alldifferent*(T) global constraints in every instance, where T is a set of 5 randomly chosen variables. The violation measure is μ_{dec}. The second set of experiments includes 2 *soft-atmost[k,v]*(T) global constraints in every instance, where T is a set of 5 randomly chosen variables, k (number of repetitions) is randomly chose from the set $\{0, ..., 3\}$ and v (value) is randomly selected from the variable domain. The violation measure used is μ_{var}. To balance binary and global costs, the cost of the *soft-alldifferent* and *soft-atmost* constraints is calculated as the amount of the violation measure multiplied by 1000.

We tested the extended versions of BnB-ADOPT$^+$ and BnB-ADOPT$^+$-UGAC able to handle global constraints using a discrete event simulator. Computational effort is evaluated in terms of non-concurrent constraint checks (NCCCs) [11]. Network load is evaluated in terms of the number of messages exchanged. Execution considers the different models to incorporate soft global constraints. For *soft-atmost* with μ_{var} we tested direct and nested representation, and the decomposition with extra variables. For

soft-alldifferent with μ_{dec} we tested the direct and nested representations, jointly with its binary decomposition.

Specifically for UGAC enforcement, computational effort is measured as follows. For the sets including *soft-alldifferent* global constraints, we use a special propagator proposed in [9]. Every time a projection operation is required, instead of exhaustively looking at all tuples of the global constraint (which would increment the NCCC counter for every tuple), we compute the minimum flow of this graph. Minimum flow cost computation is based on the successive shortest path algorithm. We can think of every shortest path computation as a variable assignment of the global constraint.

We assess the computational effort of computing the shortest path algorithm as the number of nodes of the graph where this algorithm is executed (looks reasonable for small graphs, which is the case here). Each time the successive shortest path algorithm is executed, we add this number to the NCCC counter of the agent.

For the sets including the *soft-atmost* global constraints, every time the cost of the violation measure is computed as the number of singleton domains $\{v\}$ minus k, the NCCC counter is incremented.

Figure 8 (left) contains the results of the first experiment including *soft-alldifferent* with violation measure μ_{dec}. Figure 8 (right) contains the results of the second experiment including *soft-atmost* with violation measure μ_{var}.

From these results, we observe the following facts. First, for BnB-ADOPT$^+$, the nested representation offers the best performance both in terms of communication cost and computation effort (number of messages and NCCCs, observe the logarithmic scale), at substantial distance. In the direct representation, VALUE messages are sent to all children and the last pseudochild in the global constraint, whereas in the nested representation VALUE messages are sent to all children and all pseudochildren in the global constraint. However the early detection of dead-ends compensates by far the extra number of messages that should be sent in the nested representation.

Second, for BnB-ADOPT$^+$, the nested representation is substantially better (in terms of messages and NCCCs) than the binary decomposition (in *soft-alldifferent)* and than the decomposition with extra variables (in *soft-atmost*). If an agent changes value, it will send the same number of VALUE messages in the nested representation than in the clique of binary constraints (assuming that the binary decomposition is a clique, as happens with *soft-alldifferent*). However, in the nested representation receivers will evaluate larger constraints (with arity greater than 2), so they are more effective and as global effect this representation requires less messages than the binary decomposition. The decomposition with extra variables includes many new variables in the problem, causing many extra messages. These messages lead to more computational effort (more NCCCs).

Third, for all the representations and for most of the problems considered, UGAC maintenance always pays off (in terms of messages and NCCCs). In other words, BnB-ADOPT$^+$-UGAC consistently uses less computational and communication resources than BnB-ADOPT$^+$, no matter the used representation. Savings are substantial, specially for low and medium connectivities. Although UGAC causes to do more work each time a message is exchanged, the reduction in messages is so drastic that the overall effect is less computation (NCCC curves have the same shape as number of

messages). This important fact indicates the impact of this limited form of soft GAC maintenance in distributed constraint optimization.

A closer look to the *soft-alldifferent* results of nested representation and binary decomposition indicate that as connectivity increases, messages/NCCCs required by the binary decomposition grow higher than those of the nested representation. The number of value deletions using the nested representation is higher than using the binary decomposition (because pruning using global constraints is more powerful than using the binary decomposition); as consequence, the search space is slightly smaller when using the nested representation, and due to this, less messages are required for its complete exploration. Processing less messages causes to decrease the computational effort measured in NCCCs.

From these results, we conclude that the nested decomposition offers the best performance, at a substantial distance of the other considered representations. Decomposition with extra variables using virtual agents and the direct representation are models to avoid when representing contractible global constraints in distributed constraint optimization. Enforcing UGAC pays off, causing nice savings in all representations.

6 Conclusions

In this paper we have introduced the use of soft global constraints in distributed constraint optimization. We have proposed several ways to represent soft global constraints in a distributed constraint network, depending on soft global constraint properties. We extended the distributed search algorithm BnB-ADOPT$^+$ to support the inclusion of global constraints. We evaluated its performance with and without the UGAC consistency level (generalized arc consistency with unconditional deletions).

From this work, we can extract the following conclusions:

- the use of global constraints is necessary in distributed constraint optimization to extend DCOP expressivity,
- considering two global constraints as a proof of concept, we show,
  - if the added global constraint is contractible, the nested representation is the one that, at substantial distance from others, offers the best performance both in terms of communication cost (number of messages) and computational effort (NCCCs),
  - UGAC maintenance always pays off in terms of number of messages, causing also less NCCCs in a very substantial portion of the experiments.

As future work, we plan to consider the extension of this work to other DCOP solving algorithms, as well as extending the empirical evaluation to other global constraints.

References

1. D. Allouche, C. Bessiere, P. Boizumault, S. de Givry, P. Gutierrez, S. Loudni, J.P. Meétivier, and T. Schiex. Filtering decomposable global cost functions. In *Proc. AAAI-12*, 2012.
2. C. Bessiere, I. Brito, P. Gutierrez, and P. Meseguer. Global constraints in distributed constraint satisfaction. In *Proc. AAMAS-12*, 2012.
3. C. Bessiere, E. Hebrard, B. Hnich, Z. Kiziltan, and T. Walsh. Slide: A useful special case of the cardpath constraint. In *Proc. ECAI-08*, pages 475–479, 2008.
4. C. Bessiere and P. Van Hentenryck. To be or not to be ... a global constraint. In *Proc. CP-03*, pages 789–794, 2003.
5. I. Brito and P. Meseguer. Connecting ABT with arc consistency. In *Proc. CP-08*, pages 387–401, 2008.
6. P. Gutierrez and P. Meseguer. BnB-ADOPT$^+$ with several soft AC levels. In *Proc. ECAI-10*, pages 67–72, 2010.
7. P. Gutierrez and P. Meseguer. Saving redundant messages in BnB-ADOPT. In *Proc AAAI-10*, pages 1259–1260, 2010.
8. J. Larrosa and T. Schiex. In the quest of the best form of local consistency for weighted CSP. In *Proc. IJCAI-03*, pages 239–244, 2003.
9. J. H. M. Lee and K. L. Leung. Towards efficient consistency enforcement for global constraints in weighted constraint satisfaction. In *Proc. IJCAI-09*, pages 559–565, 2009.
10. M.J. Maher. Soggy constraints: Soft open global constraints. In *Proc. CP-09*, pages 584–591, 2009.
11. A. Meisels, E. Kaplansky, I. Razgon, and R. Zivan. *Comparing Performance of Distributed Constraint Processing Algorithms*, pages 86–93. 2002.
12. P. Meseguer, F. Rossi, and T. Schiex. *Soft Constraints*, chapter 9 of Handbook of Constraint Programming, pages 281–328. Elsevier, 2006.
13. P. J. Modi, W. M. Shen, M. Tambe, and M. Yokoo. Adopt: asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161(1–2):149–180, 2005.
14. T. Petit, J. C. Regin, and C. Bessiere. Specific filtering algorithms for over-constrained problems. In *Proc. CP-01*, pages 451–463, 2001.
15. W. J. van Hoeve, G. Pesant, and L. M. Rousseau. On global warming: flow-based soft global constraints. *Journal of Heuristics*, 12:347–373, 2006.
16. W. Yeoh, A. Felner, and S. Koenig. BnB-ADOPT: An asynchronous branch-and-bound DCOP algorithm. *JAIR*, 38:85–133, 2010.