Lower Bounds for Non-binary Constraint Optimization Problems *

Pedro Meseguer[1], Javier Larrosa[2], and Martì Sánchez[1]

[1] IIIA-CSIC, Campus UAB, 08193 Bellaterra, Spain
{pedro|marti}@iiia.csic.es
[2] Dep. LSI, UPC, Jordi Girona Salgado, 1-3, 08034 Barcelona, Spain
larrosa@lsi.upc.es

Abstract. The necessity of non-binary constraint satisfaction algorithms is increasing because many real problems are inherently non-binary. Considering overconstrained problems (and Partial Forward Checking as the solving algorithm), we analyze several lower bounds proposed in the binary case, extending them for the non-binary case. We show that techniques initially developed in the context of reversible DAC can be applied in the general case, to deal with constraints of any arity. We discuss some of the issues raised for non-binary lower bounds, and we study their computational complexity. We provide experimental results of the use of the new lower bounds on overconstrained random problems, including constraints with different weights.

1 Introduction

In the context of constraint satisfaction, increasing attention has been devoted to soft constraints in the last years. A constraint is *soft* when it can be violated by some solution, without making such solution unacceptable. A constraint is *hard* when it has to be satisfied by every solution. Soft constraints are used to express user preferences, which should be satisfied if possible but not necessarily, enhancing greatly the expressiveness of constraint programming. The inclusion of soft constraints has extended the CSP schema, which considers hard constraints only, into the VCSP [11] and Semiring CSP [3] schemas, able to model overconstrained problems for which a solution is the complete assignment that satisfies all hard constraints and best respects soft ones.

In parallel to these theoretical advances, new algorithms have appeared, able to solve with increasing efficiency different types of overconstrained CSPs. For simplicity reasons these algorithms assume binary constraints. However, beyond the theoretical equivalence between binary and non-binary formulations (see [8] for its applicability to overconstrained CSPs), nowadays is widely recognized the interest of solving directly non-binary constraints for real problems. As the CSP

* This work was supported by the IST Programme of the Commission of the European Union through the ECSPLAIN project (IST-1999-11969), and by the Spanish CICYT project TAP99-1086-C03.

experience has shown, real problems are inherently non-binary and solving their natural formulation often causes significant improvements.

In this paper, we aim to bridge the gap between state-of-the-art algorithms and solving requirements of real problems, extending previously developed algorithms for binary soft constraints into the non-binary case. This is done for constrained optimization problems, a class of problems that includes those over-constrained CSPs that use the addition of costs of no satisfied constraints as the objective function to be minimized when searching for a solution. In particular, this class includes the Max-CSP problem, for which all constraints are soft with the same weight. Results presented here can be easily adapted to other kinds of problems including soft constraints.

The structure of the paper is as follows. In Section 2 we present some basic concepts. In Section 3 we study five binary lower bound formulations. In Section 4 we develop the corresponding non-binary lower bounds, analyzing their usage in the Partial Forward Checking algorithm. In Section 5 we provide experimental results using these lower bounds on overconstrained random 5-ary problems. Finally, Section 6 contains the conclusions and directions for further research.

2 Preliminaries

2.1 CSP and COP

A finite *constraint satisfaction problem* (CSP) is defined by a triple $(\mathcal{X}, \mathcal{D}, \mathcal{C})$,

- $\mathcal{X} = \{x_1, \ldots, x_n\}$ is a set of n variables;
- $\mathcal{D} = \{D_0(x_1), \ldots, D_0(x_n)\}$ is a collection of finite domains; $D_0(x_i)$ is the initial set of possible values for x_i, while $D(x_i)$ is the current set of possible values for x_i;
- $\mathcal{C}$ is a set of constraints among variables. A constraint c_i on the ordered set of variables $var(c_i) = (x_{i_1}, \ldots, x_{i_{r(i)}})$, called the scope of c_i, specifies the relation $rel_0(c_i)$ of the *allowed* combinations of values for the variables in $var(c_i)$. An element of $rel_0(c_i)$ is a tuple $(v_{i_1}, \ldots, v_{i_{r(i)}})$, $v_{i_k} \in D_0(x_{i_k})$. During search, valid tuples are reduced to $rel(c_i) \subseteq rel_0(c_i)$ formed by non-pruned values. An element of $rel(c_i)$ is a tuple $(v_{i_1}, \ldots, v_{i_{r(i)}})$, $v_{i_k} \in D(x_{i_k})$[1].

An assignment of values to variables is complete if it involves all variables, otherwise is partial. A *solution* is a complete assignment satisfying all constraints. Solving a CSP is an NP-*complete* problem.

The scope size of a constraint is its *arity*. The arity of a problem is the maximum arity of its constraints. In the sequel, n, d, m and r will denote the number of variables, the largest domain size, the number of constraints and the problem arity. Letters $i, j, k \ldots$ denote variable indexes (x_i is referred as variable i), $a, b, \ldots$ denote values, and a pair (i, a) denotes the value a of variable i.

[1] In this paper, we assume that constraints are expressed extensionally through relations. Constraints can also be expressed intentionally, by a mathematical formula or a procedure.

The join of two relations $rel_0(c_i)$ and $rel_0(c_j)$, denoted by $rel_0(c_i) \bowtie rel_0(c_j)$, is the set of tuples over $var(c_i) \cup var(c_j)$ satisfying the two constraints c_i and c_j. Accordingly, the join of two tuples is their union if they *match* in their common variables or the empty tuple otherwise. The subset of variables involved in tuple t is $var(t)$. The projection of a tuple t on a particular variable i is denoted by $t[i]$. The projection of set $\mathcal{X}$ over a subset Y is denoted by $\mathcal{X}[Y]$.

A finite *constraint optimization problem* (COP) is defined by a triple $(\mathcal{X}, \mathcal{D}, \mathcal{C})$, where $\mathcal{X}$ and $\mathcal{D}$ are as in the CSP case, and $\mathcal{C}$ is a set of *cost functions* $\mathcal{C} = \{f_1, \ldots, f_m\}$ which denote *preferences* among tuples. A cost function f_i is,

$$f_i : \prod_{x_j \in var(f_i)} D_0(x_j) \rightarrow R^+$$

where R^+ is the set of non-negative real numbers. Low values of $f_i(t)$ mean high preferences for tuple t, while high values of $f_i(t)$ mean low preferences for t. The extreme values for cost functions, namely 0 and ∞, represent the most and the least preferred tuples, respectively[2]. Tuple t is consistent with constraint f_i iif $f_i(t) = 0$; otherwise it is inconsistent. The objective function, also called the *global cost function*, is the sum of all individuals cost functions,

$$F^*(\mathcal{X}) = \sum_{i=1}^{m} f_i(\mathcal{X}[var(f_i)])$$

The *solution* is the complete assignment that minimizes $F^*(\mathcal{X})$. Solving a COP is an NP-*hard* problem.

Problems with soft constraints can naturally be formulated as COPs. Observe that, without loss of generality, hard constraints can also be expressed in this model as functions returning two values: 0 for allowed tuples and ∞ for disallowed ones. Also, the Max-CSP problem is formulated as a COP using constraints returning two values: 0 for allowed tuples and 1 for disallowed ones.

2.2 Solving COP

Partial Forward Checking (PFC) is a depth-first search algorithm used to solve COPs. It follows a *Branch and Bound* (BB) schema. At any point in search, P is the set of assigned or *past* variables, while F is the set of unassigned or *future* ones. The current partial assignment is the tuple τ over the set P. Regarding constraints, $\mathcal{C}_\mathcal{P}$ is the set of constraints having only past variables in their scope, $\mathcal{C}_\mathcal{F}$ is the set of constraints having only future variables in their scope, and $\mathcal{C}_{\mathcal{PF}}$ is set of constraints having past and future variables in their scope.

BB traverses the search tree defined by the problem and it keeps the cost of the best solution found so far (complete assignment with minimum global cost in the explored part of the search tree). This cost is an *upper bound* (UB) of

[2] Alternatively, you could say that if $f_i(t) = 0$, tuple t satisfies completely the ith constraint, and if $f_i(t) = \infty$, tuple t violates completely the ith constraint.

the problem solution. At each node, BB computes an underestimation of the global cost of any leaf node descendant from the current one. This value is a *lower bound* (LB) of the best solution that can be found as long as the current search path is maintained. When $UB \leq LB$, the current best solution cannot be improved below the current node, so the current branch can be pruned.

Upon this schema, PFC performs lookahead at each node, assessing the impact of the current assignment into the set F. Lookahead allows to improve the quality of LB computed by BB. In addition, lookahead enables the computation of LB_{ia}, a specialization of LB for value a of future variable i. If $UB \leq LB_{ia}$, value a can be removed because it will not be present in any solution better than the current one. Removed values have to be restored when PFC backtracks above the node where they were eliminated.

3 Binary Case

For binary Max-CSP, the following types of inconsistency counters have been used when computing lower bounds.

- *Distance* [4]:

$$distance(\tau, \mathcal{C_P}) = card\{c \in \mathcal{C_P} | \tau \bowtie rel_0(c) = \emptyset\}$$

- *Inconsistency counts* [4]:

$$ic_{ia} = card\{c \in \mathcal{C_{PF}} | i \in var(c) \cap F, \tau \bowtie (i,a) \bowtie rel_0(c) = \emptyset\}$$

- *Directed arc-inconsistency counts* [16]: (static variable ordering)

$$dac_{ia} = card\{c \in \mathcal{C_F} | var(c) = \{i,j\}, i < j, (i,a) \bowtie rel_0(c) = \emptyset\}$$

- *Reversible DAC* [7]: (G^F is a directed graph on $\mathcal{C_F}$)

$$dac_{ia}(G^F) = card\{c \in \mathcal{C_F} | var(c) = \{i,j\}, (j,i) \in \textsc{Edges}(G^F), (i,a) \bowtie rel_0(c) = \emptyset\}$$

- *Arc-inconsistency counts* [1]:

$$ac_{ia} = card\{c \in \mathcal{C_F} | i \in var(c), (i,a) \bowtie rel_0(c) = \emptyset\}$$

- *Russian Doll Search* [15]:

$$RDS(F, \mathcal{C_F}) = min_t\{distance(t, \mathcal{C_F})\} \qquad t \in \prod_{j \in F} D_0(x_j)$$

Max-CSP assumes that every tuple unsatisfying any constraint has the same cost. This is a simplification that does not hold in general. Many real problems include constraints which can be unsatisfied with different costs. These problems can be formulated in terms of COPs, where constraints are represented by cost functions. We define the *initial cost* of value (i, a) in constraint f as,

$$icost(i, a, f) = min_{t[i]=a} f(t) \qquad t \in \prod_{j \in var(f)} D_0(x_j)$$

When search proceeds domains change. Every past variable has its domain reduced to its assigned value. Future domains may be reduced because value pruning. We define the *current cost* of value (i, a) in constraint f as,

$$ccost(i, a, f) = min_{t[i]=a} f(t) \qquad t \in \prod_{j \in var(f)} D(x_j)$$

Using these notions, previous counters can be easily generalized for COPs as,

- *Distance*:
$$distance(\tau, \mathcal{C}_{\mathcal{P}}) = \sum_{f \in \mathcal{C}_{\mathcal{P}}} f(\tau[var(f)])$$
- *Inconsistency counts*:
$$ic_{ia} = \sum_{f \in \mathcal{C}_{\mathcal{PF}}} ccost(i, a, f)$$
- *Directed arc-inconsistency counts*: (static variable ordering)
$$dac_{ia} = \sum_{f \in \mathcal{C}_{\mathcal{F}}, var(f)=\{i,j\}, i<j} icost(i, a, f)$$
- *Reversible DAC*: (G^F is a directed graph on $\mathcal{C}_{\mathcal{F}}$)
$$dac_{ia}(G^F) = \sum_{f \in \mathcal{C}_{\mathcal{F}}, var(f)=\{i,j\}, (j,i) \in \text{EDGES}(G^F)} icost(i, a, f)$$
- *Arc-inconsistency counts*:
$$ac_{ia} = \sum_{f \in \mathcal{C}_{\mathcal{F}}} icost(i, a, f)$$
- *Russian Doll Search*:
$$RDS(F, \mathcal{C}_{\mathcal{F}}) = min_t\{distance(t, \mathcal{C}_{\mathcal{F}})\} \qquad t \in \prod_{j \in F} D_0(x_j)$$

These counters no longer record numbers of inconsistencies, but they aggregate costs caused by inconsistencies in COPs. We maintain their names as in the Max-CSP case for homogeneity reasons. From these elements, five lower bounds have been proposed. They appear in Figure 1. Lower bounds LB_2 and LB_3 require a static variable ordering.

Counters dac_{ia}, $dac_{ia}(G^F)$ and ac_{ia} are defined in terms of *icost*. They can equally be defined in terms of *ccost*. In that case, the word *maintained* is added to their names, to emphasize that those counters are maintained updated during search, taking into account value deletions in future domains. This requires the use of an arc-consistency algorithm. This approach was followed in [7].

$$LB_1(P,F) = distance(\tau, \mathcal{C}_\mathcal{P}) + \sum_{i \in F} \min_a (ic_{ia}) \qquad [4]$$

$$LB_2(P,F) = distance(\tau, \mathcal{C}_\mathcal{P}) + \sum_{i \in F} \min_a (ic_{ia} + dac_{ia}) \qquad [16,6]$$

$$LB_3(P,F) = distance(\tau, \mathcal{C}_\mathcal{P}) + \sum_{i \in F} \min_a (ic_{ia}) + RDS(F, \mathcal{C}_\mathcal{F}) \qquad [15]$$

$$LB_4(P,F) = distance(\tau, \mathcal{C}_\mathcal{P}) + \sum_{i \in F} \min_a (ic_{ia} + \frac{1}{2} ac_{ia}) \qquad [1]$$

$$LB_5(P,F,G^F) = distance(\tau, \mathcal{C}_\mathcal{P}) + \sum_{i \in F} \min_a (ic_{ia} + dac_{ia}(G^F)) \qquad [7]$$

Fig. 1. Five lower bounds for binary COPs.

4 Non-binary Case

4.1 Lower Bounds

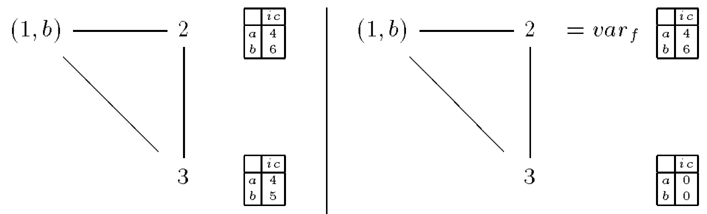
In the binary case, all constraints in $\mathcal{C}_{\mathcal{PF}}$ have one future variable in their scope. This is no longer true in the non-binary case: a constraint $f \in \mathcal{C}_{\mathcal{PF}}$ may have more than one future variable in its scope. If all extensions of τ are inconsistent with f and this is recorded in all its future variables, the lower bound cannot be added over the set F because the cost of the same inconsistency could be counted as many times as future variables are involved in the constraint. To prevent this, costs of inconsistencies of f have to be recorded in one of its future variables only. The same problem appears when considering a constraint $f' \in \mathcal{C}_\mathcal{F}$: inconsistencies of f' have to be recorded in one of its future variables, in order to allow for a safe lower bound computation as the addition of contributions of future variables. Therefore, for each $f \in \mathcal{C}_{\mathcal{PF}} \cup \mathcal{C}_\mathcal{F}$ we select one of its future variables as the only variable recording the costs of f inconsistencies. This variable, denoted as var_f, may change during the solving process among the future variables of the constraint. An example of this problem appears in Figure 2. In the Max-CSP context, this idea was presented in [9] (Section 4.2), at the ECAI-00 workshop *Modelling and Solving Problems with Constraints*. A similar approach was presented in [10] (Section 4.1), as a poster at the CP-00 conference.

This problem occurs for ic_{ia} and dac_{ia} counters. From a graph point of view, the constraint hypergraph formed by $\mathcal{C}_{\mathcal{PF}} \cup \mathcal{C}_\mathcal{F}$ has to be directed, in the sense that each hyperedge points towards one of the nodes that it connects. The hyperedge representing constraint f points towards the node representing variable var_f. Denoting by G^{PF} the directed hypergraph formed by $\mathcal{C}_{\mathcal{PF}}$, and by G^F the directed hypergraph formed by $\mathcal{C}_\mathcal{F}$, we generalize binary ic_{ia} and dac_{ia} counters as follows.

$\mathcal{X} = \{1,2,3\} \quad D_0(1) = D_0(2) = D_0(3) = \{a,b\} \quad \mathcal{C} = \{f\} \; var(f) = \{1,2,3\}$

1	2	3	f
a	a	a	0
a	a	b	1
a	b	a	2
a	b	b	3
b	a	a	4
b	a	b	5
b	b	a	6
b	b	b	7

$P = \{(1,b)\}, \; F = \{2,3\}$



$LB_1(P,F) = 0 + 4 + 4 = 8$ | $LB_1(P,F) = 0 + 4 + 0 = 4$

Fig. 2. A simple problem composed of three variables and one ternary constraint f. After assigning b to variable 1, all possible extensions of the current tuple are inconsistent with f. Recording the cost of these inconsistencies in every future variable of f causes to repeat the same cost in the computation of the lower bound, what renders it unsafe (left). The cost of inconsistencies is recorded in one future variable of f, var_f, causing a safe lower bound computation (right).

– *Inconsistency counts*: (G^{PF} is a directed hypergraph on $\mathcal{C}_{\mathcal{PF}}$)

$$ic_{ia}(G^{PF}) = \sum_{f \in \mathcal{C}_{\mathcal{PF}}, i = var_f} ccost(i,a,f)$$

– *Reversible DAC*: (G^F is a directed hypergraph on $\mathcal{C}_{\mathcal{F}}$)

$$dac_{ia}(G^F) = \sum_{f \in \mathcal{C}_{\mathcal{F}}, i = var_f} icost(i,a,f)$$

We maintain the IC and DAC names for pedagogical reasons, to keep a parallelism with the binary case. Observe, however, that in the non-binary case both counters record costs of directed inconsistencies, otherwise the cost of the same inconsistency could be recorded more than once. Keeping the meaning of $distance(\tau, \mathcal{C}_{\mathcal{P}})$, ac_{ia} and $RDS(F, \mathcal{C}_{\mathcal{F}})$ as in the binary case, we present the lower bounds for the non-binary COPs in Fig. 3. They correspond to the binary ones of Section 3 by using the same index, but for LB_2, which is now subsumed by LB_5. $LB_3(P, F, G^{PF})$ requires a static variable ordering.

As in the binary case, $dac_{ia}(G^F)$ are defined in terms of *icost*, but they can be defined in terms of *ccost*. In that case, $dac_{ia}(G^F)$ are maintained updated

$$LB_1(P,F,G^{PF}) = distance(\tau,\mathcal{C}_\mathcal{P}) + \sum_{i\in F}\min_a(ic_{ia}(G^{PF}))$$

$$LB_3(P,F,G^{PF}) = distance(\tau,\mathcal{C}_\mathcal{P}) + \sum_{i\in F}\min_a(ic_{ia}(G^{PF})) + RDS(F,\mathcal{C}_\mathcal{F})$$

$$LB_4(P,F,G^{PF}) = distance(\tau,\mathcal{C}_\mathcal{P}) + \sum_{i\in F}\min_a(ic_{ia}(G^{PF}) + \frac{1}{r}ac_{ia})$$

$$LB_5(P,F,G^{PF},G^F) = distance(\tau,\mathcal{C}_\mathcal{P}) + \sum_{i\in F}\min_a(ic_{ia}(G^{PF}) + dac_{ia}(G^F))$$

Fig. 3. Four lower bounds for non-binary COPs.

during search, taking into account value deletions. Any arc consistency strategy can be used for this purpose.

4.2 Partial Forward Checking

A PFC algorithm to be used with the proposed non-binary lower bounds is presented in Figure 4, where a generic lower bound is computed by the function `LB`. It follows the PFC algorithm of [7]. First, it checks if no more variables exists, then updates $BestS$ and $BestD$ (lines 1,2,3), the best solution and distance respectively. Otherwise, it selects a future variable i (line 5) and iterates on its feasible values (line 6). It assigns value a to variable i, it computes the new distance $newD$ (line 7), and it checks if the lower bound has reached the upper bound (line 8). If not, the lookahead is performed, returning new future domains $newFD$ (line 9). If no empty domain has been produced, a greedy optimization procedure is executed, redirecting the hypergraphs G^{PF} (for LB_1, LB_3, LB_4, LB_5) and G^F (for LB_5) in order to increase the lower bound (getting the optimum redirection of hypergraphs is a NP-hard problem [12], so we redirect hyperedges aiming at a *good* contribution to the lower bound). If the new lower bound does not reach the upper bound (line 12), it tries to remove value b in future variable j using the delete procedure (line 13). This is done redirecting the hypergraphs G^{PF} (for LB_1, LB_3, LB_4, LB_5) and G^F (for LB_5) in order to get the maximum contribution to LB_{jb}. If no empty domain has been produced (line 14), the process goes on with the recursive call (line 15).

This algorithm does not perform any maintenance of counters as values are pruned (which would imply redefining counters substituting *icost* by *ccost*). The inclusion of counter maintenance into the non-binary case is conceptually direct, although it can be computationally expensive.

```
procedure PFC(S, d, F, FD, G^PF, G^F)
if (F = ∅)
  BestD ← d;
  BestS ← S;
else
  i ← PopAVariable(F)
  for each (a ∈ FD_i)
    newD ← Distance(S ∪ {(i, a)});
    if (LB < BestD)
      newFD ← Lookahead(i, a, F, FD);
      if (¬ WipeOut(newFD))
        newG^PF, newG^F ← GreedyOpt(G^PF, G^F);
        if (LB < BestD)
          newFD ← Delete(F, newFD, newG^PF, newG^F);
          if (¬WipeOut(newFD))
            PFC(S ∪ {(i, a)}, newD, F, newFD, newG^PF, newG^F)
```

Fig. 4. Partial Forward Checking.

4.3 Complexity Analysis

In this subsection we discuss the time complexity of computing the proposed lower bounds. We begin with the complexity of each counter,

- $distance(\tau, \mathcal{C}_{\mathcal{P}})$ requires to check whether the current assignment is consistent with every constraint in $\mathcal{C}_{\mathcal{P}}$. Considering a consistency check a constant time operation, $distance(\tau, \mathcal{C}_{\mathcal{P}})$ is time $O(e)$.

- $ic_{ia}(G^{PF})$ requires to explore constraints in $\mathcal{C}_{\mathcal{PF}}$ having $i = var_c$ (there are at most e of these constraints). For them, one has to compute if $\tau \bowtie (i, a) \bowtie rel(c)$ is empty. There are at most $r-2$ *free* variables among which to search for the right values, consequently it can be done with at most $exp(r-2)$ consistency checks. Thus, the cost of computing ic_{ia} is $O(e \times exp(r-2))$.

- $dac_{ia}(G^F)$ is also $O(e \times exp(r-1))$. The exponential part (the individual contribution of each constraint) can be done prior search and recorded in an internal data structure.

- ac_{ia} is basically equivalent to compute dac_{ia} and has the same complexity.

- $RDS(F, \mathcal{C}_{\mathcal{F}})$ amounts solving $n-1$ problems with size (i.e. number of variables) $1, 2, \ldots, n-1$. Clearly, this is time $O(exp(n-1))$ and can be done as a pre-process, prior search.

Taking into account the previous complexities, is easy to see the complexity of computing each of the lower bounds considered in this paper. We differentiate

LB	pre-processing complexity	per node complexity
1	$O(1)$	$O(n \times d \times e \times exp(r-2))$
3	$O(exp(n-1))$	$O(n \times d \times e \times exp(r-2))$
4	$O(n \times d \times e \times exp(r-1))$	$O(n \times d \times e \times exp(r-2))$
5	$O(n \times d \times e \times exp(r-1))$	$O(n \times d \times e \times exp(r-2))$

Table 1. Complexity of computing the lower bounds of Figure 3

between the work that has to be done at each node (and therefore, repeated a number of times exponential in the problem size) and the work that can be done prior search (and therefore, only one time). For the sake of clarity and easy comparison, these complexities are depicted in Table 1.

4.4 Limited Versions

The complexity per node of all lower bounds is exponential in the problem arity. As a consequence, they may be of no practical use in problems having large arity constraints. With a closer look, we see that complexity is exponential in $r-2$, while $r-1$ is the number of future variables that a constraint may have in its scope. If we limit the constraints to be processed to those having at most k future variables in their scope [3], we assure that the cost of the processing will be below some complexity threshold, having less accurate inconsistency counters. Controlled by parameter k, the new counters are defined as follows,

- *k-inconsistency counts*: (G^{PF} is a directed hypergraph on $\mathcal{C}_{\mathcal{PF}}$)

$$ic_{ia}(G^{PF}) = \sum_{f \in \mathcal{C}_{\mathcal{PF}}, |var_f \cap F| \leq k, i = var_f} ccost(i, a, f)$$

- *k-reversible DAC*: (G^F is a directed hypergraph on $\mathcal{C}_{\mathcal{F}}$)

$$dac_{ia}(G^F) = \sum_{f \in \mathcal{C}_{\mathcal{F}}, |var_f| \leq k, i = var_f} icost(i, a, f)$$

- *k-arc-inconsistency counts*:

$$ac_{ia} = \sum_{f \in \mathcal{C}_{\mathcal{F}}, |var_f| \leq k} icost(i, a, f)$$

Clearly, their time complexity is bounded by $O(exp(k-1))$ and the increment of k produces the detection of more inconsistencies. Thus, parameter k controls the trade-off between overhead and accuracy.

[3] The idea of only processing those constraints with at most k future variables was presented in [2], in the context of non-binary forward checking.

5 Experimental Results

We have evaluated the performance of our PFC algorithm using the proposed lower bounds on random COPs of arity 5. A random COP class of arity 5 is characterized by $\langle n, d, p_1, p_2, w_{min}, w_{max} \rangle$ where n is the number of variables, d the number of values per variable, p_1 the graph *connectivity* defined as the ratio of existing constraints over the total number of possible constraints $\binom{n}{5}$, and p_2 the constraint *tightness* defined as the ratio of value 5-tuples that are inconsistent with the constraint over the total number of possible 5-tuples d^5. The constrained variables and the inconsistent value tuples are randomly selected, as in the random CSP case [14]. Given a tuple t inconsistent with constraint f, the value of $f(t)$ is randomly selected from the integer interval $[w_{min}, w_{max}]$.

Using this model, we have experimented on the following problem classes: $\langle 10, 5, \frac{25}{252}, p2, 1, 1 \rangle$ and $\langle 10, 5, \frac{25}{252}, p2, 1, 100 \rangle$. The first class represents Max-CSP problems, because all inconsistent tuples have the same cost. The second class represents Weighted CSP, and the costs of inconsistent tuples are randomly distributed between 1 and 100. Connectivity is low because of the high constraint arity. Tightness varies from 0.6 to 1, to get overconstrained instances.

Each problem is solved by PFC using different lower bounds (including limited versions). When using LB_3, PFC uses a static variable ordering defined heuristically to decrease bandwidth. Otherwise, PFC uses *domain size* divided by *forward degree* as dynamic variable ordering. Values are selected randomly. The computation of different lower bounds shares code and data structures whenever it is possible. Experiments were performed on a Sun Ultra 60. Each point is averaged over 50 executions.

The first experiment aims at evaluating the performance of redirecting the directed hypergraph G^{PF} to increment the lower bound. This technique was introduced for binary DAC in [7]. Using the simplest lower bound, LB_1, problems are solved by PFC without optimizing the lower bound [4] (that is, removing lines 11, 12 and 13 of Figure 4) and by standard PFC. Results appear in Figure 5. We can see that optimizing the lower bound always implies a decrement in the number of visited nodes. For Max-CSP problems, this causes a decrement in CPU time as well. For Weighed CSP, the mean CPU time for both executions is practically equal: the optimization overhead compensates the optimized savings.

The second experiment tries to assess the quality of the different lower bounds. In random problems with constraints of arity 5, the probability of a value to be arc inconsistent is extremely low. Because of that, we do not consider LB_4 and LB_5, since the *dac* and *ac* counters will be practically zero for all instances (except those with p_2 very close to 1). In Figure 6 we provide results

[4] The hypergraph G^{PF} is randomly selected at the beginning. It not redirected for optimizing LB_1, but the following form of redirection is allowed. If the algorithm selects var_f as the next variable and constraint f has other future variables in its scope, one of these variables is randomly selected as new var_f. Otherwise, the contribution of constraint f in terms of ic would be lost.

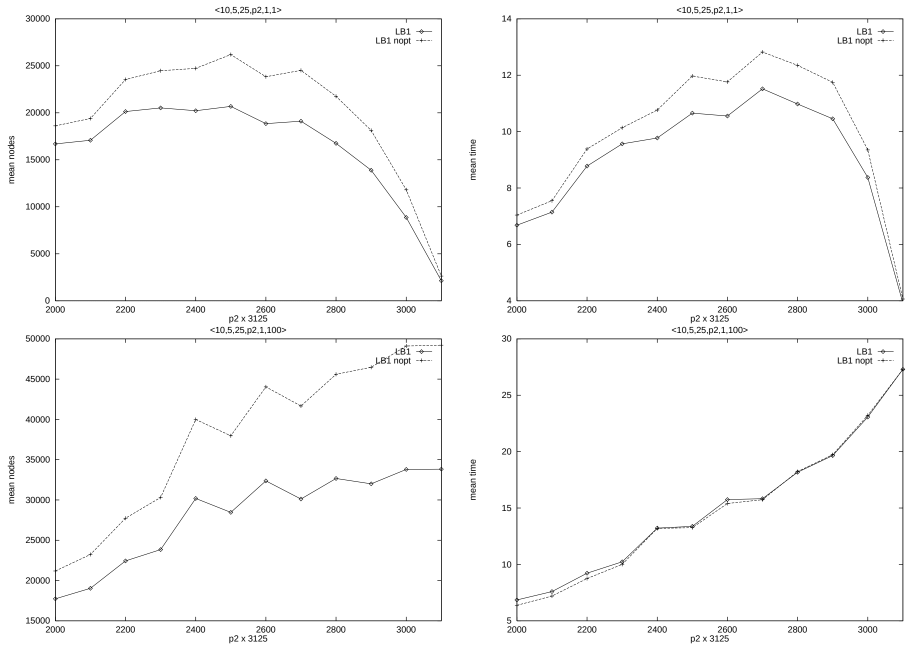


Fig. 5. Mean visited nodes and CPU time required by PFC using LB_1 with and without lower bound optimization, for the two problem classes tested.

for $LB1$, LB_3 and their limited versions for $k = 3, 2, 1$, that is, constraints in $\mathcal{C}_{\mathcal{PF}}$ are only propagated when they have up to 3, 2 and 1 future variables in their scope, respectively. We observe that LB_1 dominates LB_3 in both visited nodes and CPU time, for the two problem classes tested. This also happens for their limited versions. For Weighted CSP, there is no point in using LB_1 or LB_3 with $k > 2$ (there is no reduction in visited nodes when increasing k above 2).

Considering LB_1 in both problem classes, the mean number of visited nodes increases with p_2 faster for $k = 1$ than for $k = 2$. Given that the work per node is lower for $k = 1$ than for $k = 2$, there is a threshold in the number of visited nodes such that $k = 1$ dominates below it, and $k = 2$ dominates above. For Max-CSP, that threshold (around 40000 nodes) occurs at medium p_2, so $k = 2$ is the best trade-off between overhead and accuracy for the considered problems. For Weighted CSP, that threshold (around 90000 nodes) occurs at very high p_2, so $k = 1$ is the best trade-off for them. A similar analysis can be applied to LB_3.

The third experiment considers random problems with constraints of different arities. We generated instances of 10 variables and 5 values per variable, with 24 constraints distributed as follows: 6 constraints of arity 2, 6 of arity 3, 6 of arity 4, and 6 of arity 5. Constrained variables were selected randomly. All constraints shared the same tightness. As in the previous experiments, we considered two types of problems: Max-CSP, for which all inconsistent tuples have the same

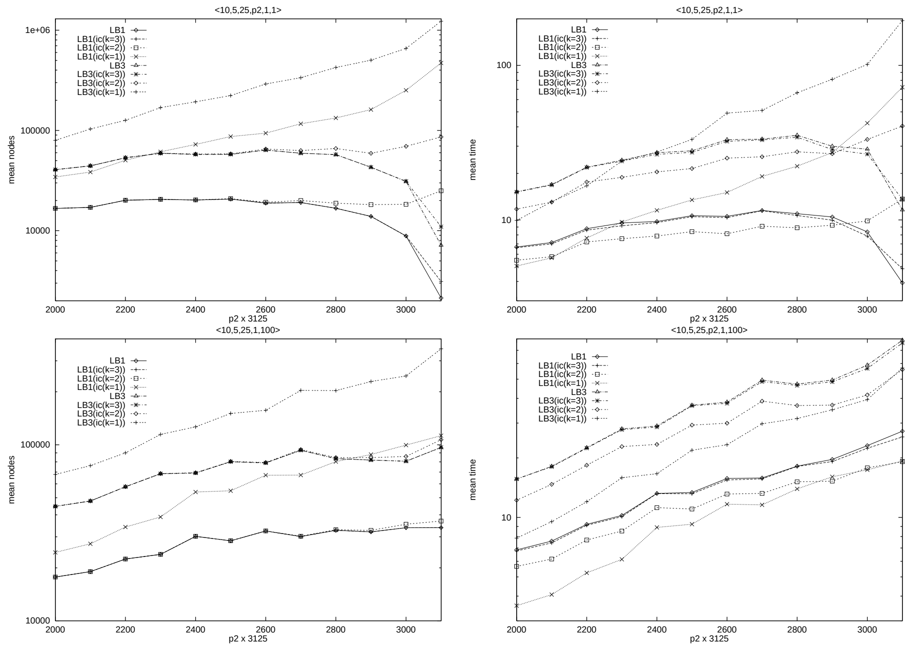


Fig. 6. Mean visited nodes and CPU time required by PFC using LB_1, LB_3 and their limited versions, for the two problem classes tested.

cost, and Weighted CSP, for which the cost of inconsistent tuples is randomly distributed between 1 and 100. We experimented with several limited versions of LB_1 and LB_5, with different k for ic and dac propagation. The best combinations are presented in Figure 7. We observed that the best limited versions of LB_5 are as follows: for Max-CSP, $k = 2$ for both ic and dac; for Weighted CSP, $k = 1$ for ic ($p_2 \leq 0.8$) and $k = 2$ for ic ($p_2 > 0.8$); $k = 2$ for dac. This is in agreement with the previous discussion of limited versions for LB_1, since dac contribution to LB_5 is limited to binary constraints (higher arities cause a very low probability for $dac > 0$ in random instances), playing a secondary role with respect to ic counters.

Therefore, according to these experimental results coming from random instances, the lower bounds of choice for non-binary PFC with low connectivity and medium to high tightness are the limited versions of LB_1 and LB_5. The main contribution to the lower bound comes from ic counters, for which a limited amount of propagation $k = 1, 2$ offers the best trade-off between the savings caused by propagation and its overhead. dac counters play a secondary role, limited to $k = 2$. By no means these experimental results prevent other bounds like LB_3 to be applicable to specific problem types. We also consider that LB_5 is applicable to real problems including arc-inconsistent values for some constraints.

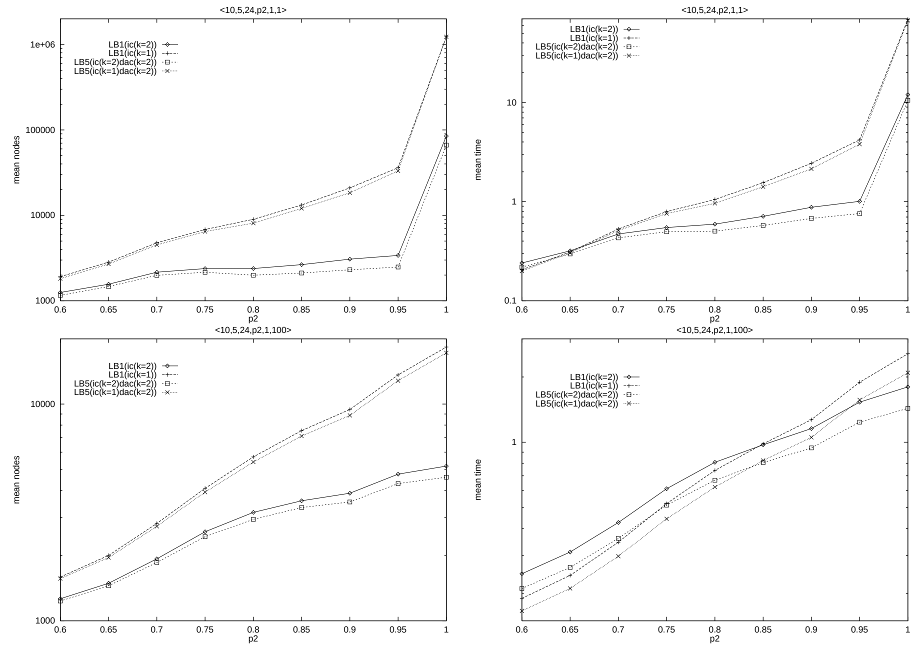


Fig. 7. Mean visited nodes and CPU time for two problem classes with constraints of different arities.

6 Conclusions and Further Work

In this paper, we have generalized binary lower bounds for the PFC algorithm developed for Max-CSP to COP, for which inconsistent tuples may have different costs. We have extended these lower bounds into the non-binary case. We have generalized the idea of directing the constraints (introduced in [7] for binary constraints between future variables) to any constraint with future variables in its scope. This means that there exists a variable per constraint where the cost of its inconsistencies is recorded. Also, we applied the idea of lower bound optimization by local search. Given the high complexity of lower bound computation, we presented the limited versions, where propagation is restricted to those constraints with a number of future variables in their scope below some limit. Experimental results on random instances show that these limited versions offer the best performance in terms of search effort.

More work, both practical and theoretical, is needed to fully understand the best way to solve COPs. On the practical side, the modelling and solving aspects of COPs present a number of issues to answer in the near future, in order to consolidate constraint technology on this type of problems. A more complete empirical evaluation is needed, including real problem instances. On the theoretical side, the development of new lower bounds based on specific local consistencies for soft constraints [13] will avoid the current limitation of counters which do not accumulate propagated inconsistencies. Also, the exploitation of the constraint graph topology [5] could speed up the resolution of COPs.

Acknowledgements

Authors thank anonymous reviewers for their constructive criticisms.

References

1. M. S. Affane and H. Bennaceur. A weighted arc consistency technique for Max-CSP. In *Proc. of the 13th ECAI*, 209–213, 1998.
2. C. Bessiere, P. Meseguer, E.C. Freuder, and J. Larrosa. On forward checking for non-binary constraint satisfaction. In *Non-binary constraints Workshop, IJCAI-99*, 1999.
3. S. Bistarelli, U. Montanari and F. Rossi. Constraint Solving over Semirings. In *Proc. of the 14th IJCAI*, 1995.
4. E.C. Freuder and R.J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58:21–70, 1992.
5. J. Larrosa. Boosting search with variable elimination. In *Proc. of the 6th CP*, 291–305, 2000.
6. J. Larrosa and P. Meseguer. Exploiting the use of DAC in Max-CSP. In *Proc. of the 2th CP*, 308–322, 1996.
7. J. Larrosa, P. Meseguer, and T. Schiex. Maintaining reversible dac for max-csp. *Artificial Intelligence*, 107:149–163, 1999.
8. J. Larrosa and R. Dechter. On the dual representation of non-binary semiring-based CSPs. In *Modelling and Solving Soft Constraints Workshop, CP-00*, 2000.
9. P. Meseguer. Lower bounds for non-binary Max-CSP. In *Modelling and Solving Constraint Problems Workshop, ECAI-00*, August 2000.
10. J.C. Régin, T. Petit, C. Bessière and J.F. Puget. An original constraint based approach for solving over constrained problems. In *Proc. of the 6th CP*, 543–548, September 2000.
11. T. Schiex, H. Fargier and G. Verfaillie. Valued Constraint Satisfaction Problems: hard and easy problems. In *Proc. of the 14th IJCAI*, 631–637, 1995.
12. T. Schiex. Maximizing the reversible DAC lower bound in Max-CSP is NP-hard. INRA Tec. Report 1998/02.
13. T. Schiex. Arc consistency for soft constraints In *Proc. of the 6th CP*, 411–424, 2000.
14. B. Smith. Phase transition and the mushy region in constraint satisfaction problems. In *Proc. of the 11th ECAI*, 100–104, 1994.
15. G. Verfaillie, M. Lemaître, and T. Schiex. Russian doll search. In *Proc. of the 13thAAAI*, 181–187, 1996.
16. R. Wallace. Directed arc consistency preprocessing. In M. Meyer, editor, *Selected papers from the ECAI-94 Workshop on Constraint Processing*, number 923 in LNCS, 121–137. Springer, Berlin, 1995.