A Decomposition Technique for Solving Max-CSP

Hachemi Bennaceur, Christophe Lecoutre, Olivier Roussel

A Decomposition Technique for Max-CSP

Hachémi Bennaceur, Christophe Lecoutre, Olivier Roussel[1]

Abstract. The objective of the Maximal Constraint Satisfaction Problem (Max-CSP) is to find an instantiation which minimizes the number of constraint violations in a constraint network. In this paper, inspired from the concept of inferred disjunctive constraints introduced by Freuder and Hubbe, we show that it is possible to exploit the arc-inconsistency counts, associated with each value of a network, in order to avoid exploring useless portions of the search space. The principle is to reason from the distance between the two best values in the domain of a variable, according to such counts. From this reasoning, we can build a decomposition technique which can be used throughout search in order to decompose the current problem into easier sub-problems. Interestingly, this approach does not depend on the structure of the constraint graph, as it is usually proposed. Alternatively, we can dynamically post hard constraints that can be used locally to prune the search space. The practical interest of our approach is illustrated, using this alternative, with an experimentation based on a classical branch and bound algorithm, namely PFC-MRDAC.

1 Introduction

The Constraint Satisfaction Problem (CSP) is the task of determining if a given constraint network is satisfiable or not, i.e. if it is possible to assign a value to all variables in order to satisfy all constraints. When no solution can be found, it may be interesting to identify a complete instantiation which satisfies the greatest number of constraints (or equivalently, which minimizes the number of violated constraints). This is called the Maximal Constraint Satisfaction Problem (Max-CSP).

During the last decade, many works have been carried out to solve this problem (and its direct extension, WCSP). The basic (complete) approach is to employ a branch and bound mechanism, traversing the search space in a depth-first manner while maintaining an upper bound, the best solution cost found so far, and a lower bound on the best possible extension of the current partial instantiation. When the lower bound is greater than or equal to the upper bound, backtracking (or filtering) occurs. Lower bound computations of constraint violations have been improved repeatedly, over the years, by exploiting inconsistency counts [7, 11, 1, 10], disjoint conflicting sets of constraints [13], or cost transfers between constraints [3, 4, 2].

Alternative approaches (usually) combine branch and bound search with dynamic programming or structure exploitation. On the one hand, Russian Doll Search [14] and variable elimination [9] can be considered as dynamic programming methods, whose principle is to solve successive sub-problems, one per variable of the initial problem. On the other hand, structural decomposition methods [8, 12, 5] exploit the structure of the problems in order to establish some conditions about possible decompositions. Such methods are based on tree decomposition, provide interesting theoretical time complexities which depend on the width of the decomposition (tree-width), and are becoming increasingly successful.

In [6], Freuder and Hubbe have proposed to exploit, for constraint satisfaction, the principle of inferred disjunctive constraints: given a satisfiable binary constraint network P, for any pair (X, a) where X is a variable of P and a a value in the domain of X, if there is no solution containing a for X, then there is a solution containing a value (for another variable) which is not compatible with (X, a). Using this principle, the authors show that it is possible to dynamically and iteratively decompose a problem.

In this paper, we generalize this approach to Max-CSP (including the non-binary case) by exploiting the arc-inconsistency counts, associated with each value of the problem. The arc-inconsistency count (aic for short) of a pair (X, a) corresponds to the number of constraints that do not support (X, a). The aic gap associated with the variable X is the absolute difference between the two lowest arc-inconsistency counts of values of X (plus 1). We show that it is possible to reason from the aic gap to obtain a condition under which we have the guarantee to obtain an optimal solution, while avoiding to explore some portions of the search space.

From this reasoning, we can build a decomposition technique which can be used throughout search to decompose the current problem into simpler sub-problems, generalizing for Max-CSP the approach of [6]. It is important to remark that unlike usual decomposition methods, this approach does not depend on the structure of the constraint graph, since the decomposition can always be applied, whatever the structure of the constraint graph is. Alternatively, we can dynamically post hard constraints that can be used locally to prune the search space. Depending on the implementation, these hard constraints can participate to constraint propagation, or just impose backtracking.

The paper is organized as follows. After some technical background, we introduce the central result of this paper. Then, we present two main exploitations of it: decomposition and pruning. After the presentation of some experimental results, we conclude.

2 Background

In this paper, we are dealing with the discrete CSP (Constraint Satisfaction Problem) framework. Each CSP instance P corresponds to a constraint network which is defined by a finite set of n variables $\{X_1, X_2, \ldots, X_n\}$ and a finite set of e constraints $\{C_1, C_2, \ldots, C_e\}$. Each variable X must be assigned a value from its associated discrete domain $dom(X)$, and each constraint C involves an ordered subset $scp(C)$ of variables of P, called its scope, and specifies the set $rel(C)$ of combinations of values allowed for

[1] Université Lille-Nord de France, Artois, F-62307 Lens – CRIL, F-62307 Lens – CNRS UMR 8188, F-62307 Lens – IUT de Lens – {bennaceur,lecoutre,roussel}@cril.univ-artois.fr

the variables of its scope. $|scp(C)|$ is called the arity of C, and C is binary if its arity is 2. A CSP instance is binary if it only contains binary constraints, and normalized if it does not contain two constraints with the same scope. Two variables are neighbours iff they both belong to the scope of a constraint. A complete instantiation is the assignment of a value to each variable. Let s denote a complete instantiation, $s(X, a)$ is the complete instantiation obtained from s by replacing the value assigned to X in s by a. A constraint C is violated (or unsatisfied) by a complete instantiation s iff the projection of s over $scp(C)$ does not belong to $rel(C)$. A solution is a complete instantiation that satisfies every constraint.

In some cases, the CSP instance may be over-constrained, and thus admits no such solution. We can then be interested in finding a complete instantiation that best respects the set of constraints. In this presentation, we consider the Max-CSP problem where the goal is to find an optimal solution, i.e. a complete instantiation satisfying as many constraints as possible. A Max-CSP instance is also represented by a constraint network.

Given a constraint C with $scp(C) = \{X_{i_1}, \ldots, X_{i_r}\}$, any tuple in $dom(X_{i_1}) \times \ldots \times dom(X_{i_r})$ is called a valid tuple on C. A value a for the variable X is often denoted by (X, a). A constraint C supports the value (X, a) (equivalently, a value (X, a) has a support on C) iff either $X \notin scp(C)$ or there exists a valid tuple on C which belongs to $rel(C)$ and which contains the value a for X. When any value is supported by a constraint, this constraint is said (generalized) arc-consistent. For the binary normalized case, we say that a variable Y supports the value (X, a) iff either no constraint involves both X and Y, or such a constraint supports (X, a). For a binary constraint C such that $scp(C) = \{X, Y\}$, a value (X, a) is compatible with a value (Y, b) iff (a, b) belongs to $rel(C)$. The arc-inconsistency count of a value (X, a), denoted by $aic(X, a)$, is the number of constraints (variables for the binary normalized case) which do not support (X, a).

3 Main Theorem

In this section, we present the main result of this paper, generalizing the approach [6] developed in the context of binary CSP.

Definition 3.1 *Let P be a Max-CSP instance and X be a variable of P. An* aic best value *of X is a value $a \in dom(X)$ such that $aic(X, a)$ is minimal, i.e. $\forall c \in dom(X), aic(X, a) \leq aic(X, c)$. An* aic second best value *of X is a value $b \in dom(X)$ such that $b \neq a$ and $\forall c \in dom(X) \setminus \{a, b\}, aic(X, b) \leq aic(X, c)$. The* aic gap *of X is defined as $\delta = aic(X, b) - aic(X, a) + 1$.*

Theorem 3.1 *Let P be a Max-CSP instance, X be a variable of P, a be an aic best value of X, δ be the aic gap of X and $C_1, \ldots, C_m$ be the m constraints involving X which support (X, a). There always exists an optimal solution s^* of P such that:*

- *either X is assigned the value a in s^*,*
- *or X is assigned a value different from a in s^*, and at least δ constraints among $C_1, \ldots, C_m$ are violated by $s^*(X, a)$.*

Proof: When P has an optimal solution where X is assigned a, the first condition is obviously satisfied and the theorem is verified. Otherwise if there is no optimal solution where $X = a$, let $s^* = (v_1, ..., v, ..., v_n)$ be an optimal solution of P, and let v be the value of X in s^*. Let C_X be the set of constraints of P involving the variable X (C_X is a superset of $\{C_1, \ldots, C_m\}$). Assume that s^* violates p constraints of C_X and $s^*(X, a)$ violates q constraints of C_X. Since there is no optimal solution with $X = a$, $s^*(X, a)$ necessarily violates more constraints of C_X than s^* and therefore $q > p$. Necessarily, we have $p \geq aic(X, v)$ and $q \geq aic(X, a)$ since arc-inconsistency counts computed wrt P represent lower bounds of aic counts obtained after assigning all variables of P. Therefore, $\exists t \geq 0, r \geq 0$ s.t. $p = aic(X, v) + r$ and $q = aic(X, a) + t$. Since $q > p$, $aic(X, a) + t > aic(X, v) + r$ or equivalently $t > aic(X, v) - aic(X, a) + r$. Since $v \neq a$, we have $aic(X, v) \geq aic(X, b)$ (b is an aic second best value of X) and therefore $t > aic(X, b) - aic(X, a) + r$. As $r \geq 0$ and $\delta = aic(X, b) - aic(X, a) + 1$, we obtain $t \geq \delta$. This means that at least δ constraints of P which support (X, a) involve variables whose values given by s^* are not compatible with (X, a). Therefore, the theorem is also verified. □

This theorem can be used in two different ways. It can be used to generate a decomposition of the Max-CSP instance or it can be exploited as a pruning rule.

4 The Decomposition Approach

The decomposition of a Max-CSP instance P around a variable X is defined as follows.

Definition 4.1 *Under the hypotheses and with the notations of Theorem 3.1, the decomposition of a Max-CSP instance P around the value a of variable X generates the sub-problems $P_0, P_1, \ldots P_k$ (with $k = \binom{m}{\delta}$) defined by:*

- *P_0 is derived from P by assigning a to variable X*
- *P_i (with $i \in 1..k$) is derived from P by removing a from the domain of X and restricting the assignments of neighbours of X so that at least δ of the constraints supporting (X, a) in P do not support (X, a) in P_i any more.*

These sub-problems may be solved independently and Theorem 3.1 guarantees that at least one of them contains an optimal solution of P. It should be noticed that this decomposition may prune some (equivalent) optimal solutions of P.

As described in the definition, the sub-problems are not disjoint which means that an assignment may be a solution of several sub-problems simultaneously. It is however easy to generate disjoint sub-problems as will be shown in section 4.2. With m denoting the number of constraints that support (X, a), this decomposition generates $1 + \binom{m}{\delta}$ sub-problems (when $\delta = 1$, this number is equal to $1 + m$ and is bounded by $n - aic(X, a)$ with n the number of variables). Although the number of sub-problems is exponential in δ, Section 4.3 proves that the search space of the different sub-problems $P_0, \ldots P_k$ is exponentially smaller than the search space of the initial problem P provided that we generate disjoint sub-problems. This means that the decomposition is always beneficial because, even if it may generate many sub-problems, they are always easier to solve globally than the initial problem.

4.1 Example

To illustrate the decomposition technique, let us consider the binary constraint network P built on $\{X_1, X_2, X_3\}$ and containing the constraints $\{C_{12}, C_{13}, C_{23}\}$. We have $dom(X_i) = \{1, 2, 3\}$ for $i \in 1..3$, and the constraints are defined by the following tables (allowed tuples):

$rel(C_{12})$

X_1	X_2
1	1
1	2
3	1

$rel(C_{13})$

X_1	X_3
1	1
1	2
2	1
2	3
3	2

$rel(C_{23})$

X_2	X_3
1	3
3	1

An optimal solution of this Max-CSP instance violates one constraint. For example, $X_1 = 1, X_2 = 1, X_3 = 2$ is an optimal solution which violates the constraint C_{23}. To perform the decomposition strategy, we have to select one variable and one of its aic best values. For example, $(X_1, 1)$ is one aic best value of X_1 since $aic(X_1, 1) = 0$, $aic(X_1, 2) = 1$ and $aic(X_1, 3) = 0$. Here, we have $\delta = 1$. The decomposition around $(X_1, 1)$ leads to the following independent sub-problems: P_0 is derived from P by assigning $X_1 = 1$. In P_0, $dom(X_1^0) = \{1\}, dom(X_2^0) = dom(X_3^0) = \{1, 2, 3\}$. P_1 is derived from P by asserting $X_1 \neq 1$ and restricting the domain of X_2 to the values incompatible with $(X_1, 1)$. In P_1, $dom(X_1^1) = \{2, 3\}, dom(X_2^1) = \{3\}, dom(X_3^1) = \{1, 2, 3\}$. P_2 is derived from P by asserting $X_1 \neq 1$ and restricting the domain[2] of X_2 to the values incompatible with $(X_1, 1)$ and restricting the domain of X_3 to the values compatible with $(X_1, 1)$. In P_2, $dom(X_1^2) = \{2, 3\}, dom(X_2^2) = \{1, 2\}$, $dom(X_3^2) = \{3\}$.

Notice that the sub-problem where $dom(X_1) = \{2, 3\}$, $dom(X_2) = \{1, 2\}$ and $dom(X_3) = \{1, 2\}$ is pruned and this sub-problem contains an optimal solution of the whole problem which is $X_1 = 3, X_2 = 1$ and $X_3 = 2$.

Now, let us modify slightly the initial problem. Assume that the value 3 of X_1 is incompatible with all values of X_3, then we have:

$rel(C_{13})$

X_1	X_3
1	1
1	2
2	1
2	3

In this case, for X_1 there is only one aic best value (since $aic(X_1, 1) = 0, aic(X_1, 2) = 1$ and $aic(X_1, 3) = 1$) and so $\delta = 2$. Thus, the decomposition leads only to two sub-problems P_0 and P_1. P_0 is unchanged and P_1 is obtained from P by asserting $X_1 \neq 1$ and restricting the domain of X_2, X_3 to the values incompatible with $(X_1, 1)$. In P_1, $dom(X_1^1) = \{2, 3\}, dom(X_2^1) = \{3\}, dom(X_3^1) = \{3\}$. In this case we have discarded the following two sub-problems: P_2 where $dom(X_1^2) = \{2, 3\}, dom(X_2^2) = \{3\}$ and $dom(X_3^2) = \{1, 2\}$, and P_3 where $dom(X_1^3) = \{2, 3\}, dom(X_2^3) = \{1, 2\}$ and $dom(X_3^3) = \{1, 2, 3\}$. The sub-problem P_3 contains one optimal solution of P: $X_1 = 3, X_2 = 1$ and $X_3 = 3$.

For the initial problem, the decomposition prunes 2^3 out of 3^3 possible complete instantiations while in the modified problem it prunes 16 (more than a half) of them.

4.2 Enumeration of Sub-problems

For the sake of simplicity, we now assume that constraints are binary and normalized (i.e. they all have different scopes) but the method is easy to generalize[3]. When constraints are binary, ensuring that a constraint C with $scp(C) = \{X, Y\}$ does not support (X, a) simply amounts to reducing the domain of Y to the values incompatible with (X, a).

[2] This restriction is enforced to obtain disjoint sub-problems, see 4.2

[3] This restriction just ensures that reducing the domain of a neighbour of X will affect only one constraint on X. Otherwise we have to take into account some variables more than once.

Enumerating all the sub-problems in the decomposition and ensuring that these problems are disjoint is as simple as enumerating the values of a binary counter under the constraint that at least δ of its bits must be 0.

Let $I_Y^{X=a}$ be the values of domain $dom(Y)$ which are incompatible with (X, a) and $C_Y^{X=a}$ be the values of $dom(Y)$ which are compatible with (X, a). By definition, $dom(Y) = I_Y^{X=a} \cup C_Y^{X=a}$ and $I_Y^{X=a} \cap C_Y^{X=a} = \varnothing$. Clearly, sub-domains I and C form a partition of each domain and this can be used to decompose the search in a systematic way. Exhaustive search on all values of a variable Y can be performed by first restricting the domain to $I_Y^{X=a}$ and then to $C_Y^{X=a}$. This is a binary branching. Since this can be done recursively, each branch can be represented by a binary word $b_{Y_1}, \ldots, b_{Y_m}$ where $b_{Y_i} = 0$ indicates that the domain of Y_i is restricted to $I_{Y_i}^{X=a}$ and $b_{Y_i} = 1$ indicates that the domain of Y_i is restricted to $C_{Y_i}^{X=a}$. Exhaustive search on all values of all variables Y will enumerate the 2^m binary words (from all 0 to all 1).

When $X = a$ is chosen for the decomposition of a problem P, the first sub-problem is P_0 where $X = a$ and the other sub-problems are the ones where $X \neq a$ and where δ variables among the m variables Y_i which support (X, a) have their domain reduced to $I_{Y_i}^{X=a}$. A simple solution to avoid any redundant or useless search is to use the binary branching scheme presented above. The restriction where δ variables among the m variables Y_i have their domain reduced to $I_{Y_i}^{X=a}$ translates to the condition 'at least δ bits in the binary word representing the branch must be 0'. This condition is trivial to enforce in a binary branching.

Y_1	Y_2	Y_3	Y_4
0	$*$	$*$	$*$
1	0	$*$	$*$
1	1	0	$*$
1	1	1	0

(a) search with $\delta = 1$

Y_1	Y_2	Y_3	Y_4
0	0	0	$*$
0	0	1	0
0	1	0	0
1	0	0	0

(b) search with $\delta = 3$

Figure 1. List of branches to explore for $n = 4$ and different values of δ

As an example, Figure 1 represents the branches that must be explored for two different values of δ and for $n = 4$ variables. For clarity, $*$ is used as a joker to represent any 0/1 value.

4.3 Some Complexity Results

Interestingly, this binary branching scheme allows to draw immediate complexity results. Assume that $Y_1, \ldots, Y_m$ are the variables which support (X, a) and that $Z_1, \ldots, Z_r$ are the other unassigned variables. Without applying the decomposition, an exhaustive search of the sub-problem where $X \neq a$ will have to explore the Cartesian product of the domains which amounts to $\prod_{i=1}^{m} |dom(Y_i)|. \prod_{i=1}^{r} |dom(Z_i)|$ complete instantiations. When the decomposition is used, at least δ variables Y must have their domain reduced to $I_Y^{X=a}$. This means that the number of complete instantiations which are not explored amounts to:

$$\sum_{S \in 2^{\{Y_i\}} \text{ with } card(S) < \delta} \prod_{Y_i \in S} |I_{Y_i}^{X=a}|. \prod_{Y_i \notin S} |C_{Y_i}^{X=a}|. \prod_{i=1}^{r} |dom(Z_i)|$$

As an illustration, if all $C_{Y_i}^{X=a}$ have the same size c and all $I_{Y_i}^{X=a}$ have the same size i, the number of pruned complete instantiations simplifies as:

$$\sum_{j \leq \delta} \binom{n}{j} i^j c^{m-j} \prod_{i=1}^{r} |dom(Z_i)|$$

When $\delta = 1$, the number of pruned complete instantiations is just $i.c^{m-1} \prod_{i=1}^{m} |dom(Z_i)|$. It roughly corresponds to the size of the so-called *consistent sub-problem* identified in [6] for the CSP case. In any case, the number of complete instantiations that are explored when the decomposition is applied is smaller than the initial number of complete instantiations to explore (by an exponential factor in the general case).

4.4 Related Work

Classical structural decomposition methods combine tree decomposition of graphs with branch and bound search [8, 12, 5]. A tree decomposition involves computing a pseudo-tree which covers the set of variables by clusters. Two clusters are adjacent in this tree if they share some variables. An important property of tree decomposition is that the sub-problems associated with clusters may be solved independently after assigning values to the shared variables. In practice, the efficiency of decomposition methods highly depends on the structure of the constraint graph.

The decomposition approach presented here, inspired from [6], proceeds differently from classical ones since the principle is to directly decompose the whole problem into independent sub-problems without computing any pseudo-tree or assigning any variable of the problem. Each sub-problem can be solved independently while in the same time, a portion of the search space of the whole problem is pruned. The downside of this method is that the number of generated sub-problems may be large. However, the decomposition does not rely on the structure of the constraint graph.

5 The Pruning Approach

Another way to exploit Theorem 3.1 is to interpret it as a pruning rule which can be integrated into any method based on tree search to solve the Max-CSP problem. Assuming here a tree search algorithm employing a binary branching scheme, at each node ν, a value (X, a) is selected, and two branches are built from ν: a left one labelled with the variable assignment $X = a$, and a right one labelled with the value refutation $X \neq a$. Considering the current instance at node ν, let a, δ and $\{C_i\}$ be the best aic value of X, the aic gap of X and the set of constraints supporting (X, a), respectively. As soon as the left branch has been explored, one can post a *hard* constraint $atLeastUnsatisfied(\delta, \{C_i\}, (X, a))$ before exploring the right branch of ν. This constraint is violated as soon as it is no more possible to find in $\{C_i\}$, at least δ constraints which do not support anymore (X, a). Of course, a constraint posted with respect to the right branch of node ν must be removed when the algorithm backtracks from ν.

These hard constraints, dynamically added to the instance, can be used to impose backtracking, and consequently, to avoid exploring useless portions of the search space. After each propagation phase, one can simply check that all currently posted hard constraints are still satisfied. If this is not the case, backtracking occurs. We will denote any tree search algorithm A, exploiting this approach, by A-PC (Pruning Constraints). Interestingly, except for some particular search heuristics (such as the ones based on constraint weighting), we have the guarantee that A-PC will always visit a tree which is included in the one built by A.

On the other hand, the additional hard constraints can also participate to constraint propagation. When for a constraint $atLeastUnsatisfied(\delta, \{C_i\}, (X, a))$, we can determine that at most δ constraints of $\{C_i\}$ can still be in a position of not supporting (X, a), we can impose that these δ constraints do not support (X, a), making then new inferences. For example, for a binary constraint of $\{C_i\}$, among the δ ones, involving X and another variable Y, any value of Y compatible with (X, a) can be removed. Here, we can imagine sophisticated mechanisms to manage propagation such as the use of lazy structures (e.g. watched literals).

Importantly, notice that this pruning approach can be integrated into many search algorithm solving the Max-CSP problem, including hybrid ones that combine tree decomposition with enumeration.

6 Experimental Results

In order to show the practical interest of the approach described in this paper, we have conducted an experimentation on a cluster of Xeon 3,0GHz with 1GiB under Linux using the benchmark suite used for the 2006 competition of Max-CSP solvers (see `http://www.cril.univ-artois.fr/CPAI06/`). We have used the classical branch and bound PFC-MRDAC algorithm [11] which maintains reversible directed arc-inconsistency counts in order to compute lower bounds at each node of the search tree, and have been interested in the impact of using the PC (Pruning Constraints) approach (see Section 5). We have used here the variant that just imposes backtracking, and have not still implemented the one that allows to make inferences. We have not still implemented the decomposition approach either.

Two variable ordering heuristics have been considered. The first one is $dom/ddeg$, usually considered for Max-CSP, which selects at each node the variable with the lowest ratio *domain size* on *dynamic degree*. The second one, denoted by $dom * gap/ddeg$, involves the aic gap of the variables. More precisely, the ratio $dom/ddeg$ is multiplied by the aic gap in order to favour variables for which there is a large gap between the best value and the following one. We believe that it may help quickly finding good solutions and, more specifically, increasing the efficiency of our approach. Finally, the value with the lowest aic is always selected. Notice that it can be seen as a refinement of the $ic + dac$ counters usually used to select values.

The protocol used for our experimentation is the following: for each instance, we start with an initial upper bound[4] set to infinity, and record the (cost of the) best solution found (and time-stamp it) within a given time limit (here, $1,500$ seconds). Even if this protocol prevents us from getting some useful results for some instances (for example, if the same best solution is found by the different algorithms after a few seconds), it benefits from being easily reproducible and exploitable, whether the optimum value is known or not.

First of all, recall that we have the guarantee that PFC-MRDAC-PC always visits a tree which is smaller than the one built by PFC-MRDAC. It makes our experimental comparisons easier. We can then make a first general observation about the results of our experimentation. The overhead of managing PC hard constraints is usually between 5% and 10% of the overall cpu time. Since on random instances, our approach permits to only save a limited number of nodes (as expected), we obtain a similar behaviour with PFC-MRDAC and PFC-MRDAC-PC. This is not shown here, due to lack of space. On the other hand, on structured instances, Table 1 presents the results on representative instances and clearly demonstrates the interest of our approach. These instances belong to academic and patterned series $maxclique$ ($brock$, $p - hat$,san), $kbtree$ (introduced in [5]), $dimacs$ (ssa) and $composed$, and also to real-world

[4] In the experimentation, Max-CSP was considered as the problem of minimizing the number of violated constraints.

series $celar$ ($scen$, $graph$) and $spot$. The ratio introduced in the table corresponds to the cpu of PFC-MRDAC divided by the cpu of PFC-MRDAC-PC. It is either an exact value (when both methods have found the same upper bound) or an approximate one (in this case, we use the time limit $1,500$ as a lower bound). For example, on instance $spot5-404$, we obtain 74 as upper bound with PFC-MRDAC and 73 with PFC-MRDAC-PC. Since, any node visited by PFC-MRDAC-PC is necessarily visited by PFC-MRDAC, we know that at least $1,500$ seconds are required by PFC-MRDAC to find the upper bound 73. We then obtain a speedup ratio which is greater than $1,500/99 = 15.1$. Remark that, as expected, the results are more impressive when using the heuristic $dom * gap/ddeg$ (more than two orders of magnitude on some instances) which besides, often allows us to find better upper bounds.

		PFC-MRDAC					
		dom/ddeg			dom*gap/ddeg		
		$\neg PC$	PC	$ratio$	$\neg PC$	PC	$ratio$
Academic and Patterned instances							
brock-200-1	ub	184	183		184	183	
	cpu	1,490	706	> 2.1	3	57	> 26.3
brock-200-2	ub	191	191		191	190	
	cpu	638	92	> 6.9	85	201	> 7.4
composed-25-1-2-1	ub	3	3		6	3	
	cpu	613	332	$= 1.8$	19	846	> 1.7
composed-25-1-25-1	ub	4	4		6	3	
	cpu	92	72	$= 1.2$	14	1,407	> 1
kbtree-9-2-3-5-20-01	ub	6	0		3	0	
	cpu	996	1,333	> 1.1	0	15	> 100
kbtree-9-2-3-5-30-01	ub	13	13		14	4	
	cpu	1,037	1,009	$= 1.0$	1,177	392	> 3
keller-4	ub	162	160		162	160	
	cpu	36	303	> 4.9	1	149	> 10.0
p-hat300-1	ub	293	293		293	293	
	cpu	396	76	$= 5.2$	1,481	224	$= 6.6$
p-hat500-1	ub	493	493		493	492	
	cpu	1,357	652	$= 2.0$	33	717	> 2.0
san-200-0.9-1	ub	174	173		157	155	
	cpu	1,425	1,287	> 1.1	0	888	> 1.6
sanr-200-0.7	ub	185	185		185	184	
	cpu	426	94	$= 4.5$	808	324	> 4.6
ssa-0432-003	ub	82	73		11	2	
	cpu	0	175	> 8.5	46	19	> 78.9
ssa-2670-130	ub	392	390		52	49	
	cpu	1	56	> 26.7	55	1,126	> 1.3
Real-world instances							
graph6	ub	342	341		366	365	
	cpu	216	935	> 1.6	7	406	> 1.0
graph8-f11	ub	161	159		160	160	
	cpu	5	1,299	> 1.1	644	56	$= 11.5$
graph11	ub	576	576		620	620	
	cpu	5	5	$= 1$	677	70	$= 9.6$
scen6	ub	269	269		211	211	
	cpu	69	27	$= 2.5$	20	14	$= 1.4$
scen10	ub	744	744		741	741	
	cpu	34	34	$= 1$	623	56	$= 11.1$
scen11-f12	ub	81	81		66	66	
	cpu	729	395	$= 1.8$	146	35	$= 4.1$
scenw-06-18	ub	215	214		133	131	
	cpu	8	934	> 1.6	231	442	> 3.3
scenw-06-24	ub	98	98		121	117	
	cpu	686	244	$= 2.8$	741	400	> 3.7
scenw-07	ub	353	353		525	524	
	cpu	1,239	471	$= 2.6$	25	8	> 187.5
spot5-28	ub	207	206		196	196	
	cpu	0	31	> 48.3	1	1	$= 1$
spot5-29	ub	52	51		49	48	
	cpu	25	305	> 4.9	807	29	> 51.7
spot5-42	ub	124	124		122	122	
	cpu	900	64	$= 14.0$	1,157	6	$= 192.8$
spot5-404	ub	74	73		76	73	
	cpu	85	99	> 15.1	0	331	> 4.5

Table 1. Best upper bound (ub, number of violated constraints) and cpu time (to reach it) obtained with PFC-MRDAC on structured instances, with (PC) and without ($\neg PC$) the Pruning Constraints method. The timeout was set to $1,500$ seconds per instance.

Finally, for a very limited number of these instances, we succeeded in finding an optimal value and proving optimality, given 20 hours of cpu time per instance. For example, for $brock$-200-2, optimality is proved when using PC in $13,394$ and $29,217$ seconds with dom/ddeg and $dom * gap/ddeg$ respectively, while optimality is not proved within $72,000$ seconds when PC is not employed. As another example, the instance $scenw$-06-24 is solved in $18,858$ seconds with PFC-MRDAC-PC-$dom * gap/ddeg$ and in $37,405$ seconds when PC is not used.

7 Conclusion

In this paper, we have generalized to Max-CSP the principle of inferred disjunctive constraints introduced in [6] for CSP. Using the so-called aic (arc-inconsistency count) gap, we have shown that it was possible to obtain a guarantee about the obtention of an optimal solution, while pruning some portions of the search space. Interestingly, this result can be exploited both in terms of decomposition (already addressed for CSP in [6]) and backtracking/filtering (by posting hard constraints). We have shown that our approach, grafted to a classical branch and bound algorithm, was really boosting search when solving structured instances. Indeed, using PFC-MRDAC, we have noticed a speedup that sometimes exceeds one order of magnitude with the heuristic $dom/ddeg$ and two orders of magnitude with the original $dom * gap/ddeg$.

We want to recall that dynamic programming and decomposition methods, which have recently received a lot of attention, still rely on branch and bound search. It means that all these methods may benefit from the approach developed in this paper. Finally, one perspective of this work is to extend it with respect to Weighted CSP and Valued CSP frameworks.

Acknowledgments

This paper has been supported by the IUT de lens, the CNRS and the ANR "Planevo" project n°JC05_41940.

REFERENCES

[1] M.S. Affane and H. Bennaceur, 'A weighted arc-consistency technique for Max-CSP', in *Proceedings of ECAI'98*, pp. 209–213, (1998).

[2] M.C. Cooper, S. de Givry, and T. Schiex, 'Optimal Soft Arc Consistency', in *Proceedings of IJCAI'07*, pp. 68–73, (2007).

[3] M.C. Cooper and T. Schiex, 'Arc consistency for soft constraints', *Artificial Intelligence*, **154**(1-2), 199–227, (2004).

[4] S. de Givry, F. Heras, M. Zytnicki, and J. Larrosa, 'Existential arc consistency: Getting closer to full arc consistency in weighted CSPs', in *Proceedings of IJCAI'05*, pp. 84–89, (2005).

[5] S. de Givry, T. Schiex, and G. Verfaillie, 'Exploiting Tree Decomposition and Soft Local Consistency In Weighted CSP', in *Proceedings of AAAI'06*, (2006).

[6] E.C. Freuder and P.D. Hubbe, 'Using inferred disjunctive constraints to decompose constraint satisfaction problems', in *Proceedings of IJCAI'93*, pp. 254–261, (1993).

[7] E.C. Freuder and R.J. Wallace, 'Partial constraint satisfaction', *Artificial Intelligence*, **58**(1-3), 21–70, (1992).

[8] P. Jégou and C. Terrioux, 'Hybrid backtracking bounded by tree-decomposition of constraint networks', *Artificial Intelligence*, **146**(1), 43–75, (2003).

[9] J. Larrosa and R. Dechter, 'Boosting search with variable elimination in constraint optimization and constraint satisfaction problems', *Constraints*, **8**(3), 303–326, (2003).

[10] J. Larrosa and P. Meseguer, 'Partition-Based lower bound for Max-CSP', *Constraints*, **7**, 407–419, (2002).

[11] J. Larrosa, P. Meseguer, and T. Schiex, 'Maintaining reversible DAC for Max-CSP', *Artificial Intelligence*, **107**(1), 149–163, (1999).

[12] R. Marinescu and R. Dechter, 'AND/OR Branch-and-Bound for Graphical Models', in *Proceedings of IJCAI'05*, pp. 224–229, (2005).

[13] J.C. Regin, T. Petit, C. Bessiere, and J.F. Puget, 'New lower bounds of constraint violations for over-constrained problems', in *Proceedings of CP'01*, pp. 332–345, (2001).

[14] G. Verfaillie, M. Lemaitre, and T. Schiex, 'Russian doll search for solving constraint optimization problems', in *Proceedings of AAAI'96*, pp. 181–187, (1996).