# A decomposition strategy to reduce the search space of solutions of WCSP

Hachémi Bennaceur, Christophe Lecoutre, Olivier Roussel

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

**Abstract.** The objective of the Weighted Constraint Satisfaction Problem (WCSP) is to find an instantiation that minimizes the degree of violation of constraints 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 avoid exploring portions of the search space by exploiting some local conditions. The principle is to reason from the distance between the two best values in the domain of a variable. 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. This approach applies to both WCSP and Max-CSP.

## 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 minimizes the degree of violation of constraints. This is the Weighted Constraint Satisfaction Problem (WCSP). Alternatively, one can seek a complete instantiation which satisfies the greatest number of constraints (or equivalently, which minimizes the number of violated constraints). This is the Maximal Constraint Satisfaction Problem (Max-CSP) which is in fact a special case of WCSP.

During the last decade, many works have been carried out to solve these two problems. 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 [1–4], disjoint conflicting sets of constraints [5], or cost transfers between constraints [6–9].

Alternative approaches (usually) combine branch and bound search with dynamic programming or structure exploitation. On the one hand, Russian Doll Search [10, 11] and variable elimination [12] 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 [13–17] 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 [18], 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 both WCSP and Max-CSP (including the non-binary case) by exploiting the weights associated with each value of the problem. We show that it is possible to reason from the weights associated to the values of a variable 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 WCSP and Max-CSP the approach of [18]. 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 which applies to both WCSP and Max-CSP. Then, mainly in the Max-CSP context, we present two main exploitations of it: decomposition and pruning. At last, some experimental results on Max-CSP instances show the interest of the technique.

## 2 Background

In this article, we consider both the WCSP and Max-CSP problems. We briefly recall the terminology related to both problems.

### 2.1 WCSP

A WCSP instance $P$ is defined by a triplet $(\mathcal{X}, \mathcal{C}, k)$ of respectively variables, constraints and top cost.

$\mathcal{X}$ is a finite set of $n$ variables $\{X_1, X_2, \ldots, X_n\}$. Each variable $X$ must be assigned a value from its associated discrete domain $dom(X)$. An instantiation $I$ is the assignment of values to some variables of $\mathcal{X}$ and will be denoted $(X_i = v_i, X_j = v_j, \ldots)$

to indicate that $X_i$ is assigned value $v_i$, $X_j$ is assigned $v_j$ and so on. A complete instantiation is the assignment of a value to each variable of $\mathcal{X}$ and can be identified by the ordered list of values assigned to variables $(v_1, v_2, \ldots, v_n)$. In an instantiation $I$, the substitution of an assignment $X_i = v_i$ by the assignment $X_i = v'$ will be denoted $I_{X_i=v'}$. $var(I)$ denotes the set of variables which are assigned in $I$. The restriction of an instantiation to some variables in a set $S$ (projection on a set of variables) is denoted $I \downarrow S$.

$\mathcal{C}$ is a finite set of $e$ constraints $\{C_1, C_2, \ldots, C_e\}$. Each constraint $C$ involves an ordered subset $scp(C)$ of variables of $\mathcal{X}$, called its scope. $|scp(C)|$ is called the arity of $C$, and $C$ is binary if its arity is 2. A WCSP instance is binary if it only contains binary constraints. Two variables are neighbors iff they both belong to the scope of a constraint. The set of constraints adjacent to a variable $X$ is denoted $adj(X)$ and defined by $\{C_i | X \in scp(C_i)\}$. A constraint $C$ is a function which maps all possible instantiations of the variables in its scope to a non-negative integer which represents the cost of this instantiation. This cost can be an integer in the range $[0.. + \infty[$ or can be equal to $+\infty$. The cost of an instantiation $I$ for a constraint $C$ is defined by $C(I \downarrow scp(C))$. For the sake of simplicity, the projection operation will be considered implicit and $C(I \downarrow scp(C))$ will be simply denoted $C(I)$. The global cost of an instantiation is the sum over all constraints of the constraint costs: $cost(I) = \sum_i C_i(I)$. Costs can be assigned to variable values through unary constraints. For example, assigning a cost $w$ to a value $a$ of a variable $X$ is represented by a constraint $C$ of arity 1 with $scp(C) = \{X\}$ and such that $C(X = a) = w$.

The parameter $k$ is either a positive integer or $+\infty$. Instantiations with a cost greater than or equal to $k$ are not admissible and cannot be solution of the WCSP instance. Therefore when a constraint maps a tuple to a cost greater than or equal to $k$, this tuple is strictly forbidden and cannot appear in any solution.

A solution of a WCSP instance is a complete instantiation $I$ with a minimal cost (i.e. $\forall I', cost(I') \geq cost(I)$) and such that $cost(I) < k$.

Usually, the WCSP problem is defined by a valuation structure which restricts the cost of a constraint to the interval $[0..k]$ and introduces a specific addition operator which saturates at $k$ ($a \oplus b = min(k, a + b)$). The definition used in this paper is semantically equivalent and avoids the introduction of specific arithmetic operators.

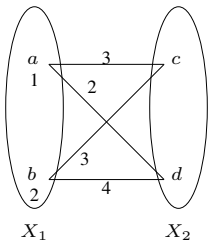


**Fig. 1.** Example of a WCSP instance

Figure 1 presents an example of a tiny WCSP instance where $dom(X_1) = \{a, b\}$, $dom(X_2) = \{c, d\}$ and the constraint $C_2$ between $X_1$ and $X_2$ has the following costs: $C_2(X_1 = a, X_2 = c) = 3$, $C_2(X_1 = a, X_2 = d) = 2$, $C_2(X_1 = b, X_2 = c) = 3$, $C_2(X_1 = b, X_2 = d) = 4$. Assigning $a$ to $X_1$ has a cost of 1, and assigning $b$ to $X_2$ has a cost of 2. This is represented by a constraint $C_1$ of arity 1 on $X_1$ such that $C_1(X_1 = a) = 1$ and $C_1(X_1 = b) = 2$.

### 2.2 Max-CSP

The Max-CSP problem is a special case of the WCSP problem where $k = +\infty$ and all constraints $C_i$ are functions mapping tuples to the set $\{0, 1\}$. When a constraint maps a tuple to 1, the tuple violates the constraint. Otherwise, a tuple which maps to 0 satisfies the constraint. Under these conditions, identifying a solution with the minimal cost amounts to finding a solution which minimizes the number of violated constraints (or alternatively which maximizes the number of satisfied constraints).

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 $(X, a)$. 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 $aic(X, a)$, 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 [18] developed in the context of binary CSP and the results of [19] obtained on Max-CSP.

**Definition 31** *Let $P$ be a WCSP instance and $X$ be a variable of $P$. The minimal cost of a constraint $C_i$ relative to $(X, a)$ denoted $mincost(C_i, X, a)$ is the minimum cost for $C_i$ of tuples involving $(X, a)$.*

$$mincost(C_i, X, a) = \min_{\substack{I \,:\, I \downarrow X = a, \\ var(I) = scp(C_i)}} C_i(I)$$

*We define the locally normalized constraint with respect to $(X, a)$ as $C_i^{(X,a)}(I) = C_i(I) - mincost(C_i, X, a)$. By definition, there exists at least one tuple $I$ such that $C_i^{(X,a)}(I) = 0$. $C_i^{(X,a)}(I)$ corresponds to the residual cost of the tuple $I$ of $C_i$ after enforcing soft arc consistency on the value $(X, a)$.*

*The cost* $cost(X, a)$ *of a value* $a \in dom(X)$ *is defined as the sum over the constraints* $C_i$ *involving* $X$ *of* $mincost(C_i, X, a)$.

$$cost(X, a) = \sum_{C_i \in adj(X)} mincost(C_i, X, a)$$

*A* best value *of* $X$ *is a value* $a \in dom(X)$ *such that* $cost(X, a)$ *is minimal, i.e.* $\forall c \in dom(X), cost(X, a) \leq cost(X, c)$. *A* second best value *of* $X$ *is a value* $b \in dom(X)$ *such that* $b \neq a$ *and* $\forall c \in dom(X) \setminus \{a, b\}, cost(X, b) \leq cost(X, c)$. *The* gap *of* $X$ *is defined as* $\delta = cost(X, b) - cost(X, a) + 1$.

The locally normalized constraint defined above directly corresponds to the notion of arc consistency in WCSP [20]. For a given constraint, when all tuples have a cost greater than or equal to $\alpha$, the cost $\alpha$ can be factorized and moved from the tuples to the variable values. $mincost(C_i, X, a)$ represents the cost that can be factorized from constraint $C_i$ under the assumption $X = a$. $cost(X, a)$ is the cost which is directly induced by assigning $a$ to $X$ and is a lower bound on the cost of any instantiation with $X = a$. Notice that $cost(X, a)$ includes the cost of unary constraints on $X$ (costs assigned to values of $X$). The only slight difference between our definition of $cost(X, a)$ and arc-consistency in WCSP is that we do not try to generate an arc-consistent network (which is not unique). The best value of a variable corresponds to the special value called the fully supported value after enforcing Existential Directional Arc Consistency (EDAC). Recall that EDAC requires that every variable must have at least one special value, that is a value with a null cost and it has a full support in every constraint [**?**].

On the example of Figure 1, $mincost(C_1, X_1, a) = 1$, $mincost(C_1, X_1, b) = 2$, $mincost(C_2, X_1, a) = min(3, 2) = 2$ and $mincost(C_2, X_1, b) = min(3, 4) = 3$. $cost(X_1, a) = 3$ and $cost(X_1, b) = 5$, hence $a$ is the best value of $X_1$ and $\delta = 3$.

**Theorem 31** *Let* $P$ *be a WCSP instance,* $X$ *be a variable of* $P$*,* $a$ *be a best value of* $X$*,* $\delta$ *be the gap of* $X$*. 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*

$$\sum_{C_i \in adj(X)} C_i^{(X,a)}(s^*_{X=a}) \geq \delta \tag{1}$$

*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 $s^*$ (necessarily, $v \neq a$). Let $s = s^*_{X=a}$. Because there's no optimal solution with $X = a$, $cost(s) > cost(s^*)$. Since $s$ and $s^*$ differ by only one variable, $cost(s)$ is equal to $cost(s^*)$ minus the contribution of $X = v$ and plus the contribution of $X = a$. This is expressed by $cost(s) = cost(s^*) - \sum_{C_i \in adj(X)} C_i(s^*) + \sum_{C_i \in adj(X)} C_i(s)$ (keep in mind that the costs associated to variable values are included in the sum since they are represented as unary constraints). Since $cost(s) > cost(s^*)$, this implies $cost(s^*) - \sum_{C_i \in adj(X)} C_i(s^*) + \sum_{C_i \in adj(X)} C_i(s) > cost(s^*)$

or equivalently $\sum_{C_i \in adj(X)} C_i(s) > \sum_{C_i \in adj(X)} C_i(s^*)$. Since in $s^*$, $X \neq a$, and $a$ is the best value of $X$, necessarily $\sum_{C_i \in adj(X)} C_i(s^*)$ is a cost greater than or equal to the cost of the second best value $b$. Hence, $\sum_{C_i \in adj(X)} C_i(s) > cost(X, b)$. We can introduce locally normalized constraints with $C_i(I) = mincost(C_i, X, a) + C_i^{(X,a)}(I)$. Now, $\sum_{C_i \in adj(X)} (mincost(C_i, X, a) + C_i^{(X,a)}(s)) > cost(X, b)$ which can be rewritten as $\sum_{C_i \in adj(X)} C_i^{(X,a)}(s) > cost(X, b) - cost(X, a)$ which is equivalent to $\sum_{C_i \in adj(X)} C_i^{(X,a)}(s) \geq cost(X, b) - cost(X, a) + 1 = \delta$. □

It should be noticed that the proof holds for any arity of constraint and that we do not rely on enforcing arc-consistency on the global network (which cannot be done in a unique way).
The second condition of the main theorem may be simplified when the WCSP satisfies the EDAC property since $mincost(C_i, X, a)$ is null for each $C_i \in adj(X)$ then the condition will be $\sum_{C_i \in adj(X)} C_i(s^*_{X=a}) \geq \delta$.

This theorem decomposes the search in two parts. The first part explores $X = a$ (the best value for variable $X$) and the second part explores $X \neq a$ and avoids the exploration of useless parts of the search-space. There are two different ways to split the search. A first way is to generate a decomposition of the instance into independent subproblems which uses the theorem to filter the variables domain in order to avoid useless search (see Section 4). Another way is to use the theorem as a pruning rule which enforces on the $X \neq a$ branch of the search the condition $\sum_{C_i \in adj(X)} C_i^{(X,a)}(s^*_{X=a}) \geq \delta$ (see Section 5).

In both cases, the key point is to use efficiently condition (1) to prune the search space. Because of the weights, enforcing this condition is slightly more difficult in the WCSP case than in the Max-CSP case.

### 3.1 The WCSP case

The condition $\sum_{C_i \in adj(X)} C_i^{(X,a)}(s^*_{X=a}) \geq \delta$ which appears in the theorem indicates that at any time in the search on the branch $X \neq a$, the sum of the (normalized) weights of tuples which are still considered in the search must be greater than or equal to $\delta$. This is very close to pseudo-Boolean constraints[1] and propagation techniques from the pseudo-Boolean field can be used.

The first technique is based on "counters". For each constraint $C_i$, we maintain the maximum (normalized) cost $m_i$ of tuples still considered in the search. As long as $\sum_i m_i \geq \delta$, the condition can be satisfied. As soon as $\sum_i m_i < \delta$, the condition is not satisfied any more which means that we're exploring a useless branch. Backtrack must occur. When $\sum_i m_i \geq \delta$ and $\exists k, (\sum_i m_i) - m_k < \delta$, we have to select for constraint $C_k$ the tuple with the maximal cost in order to satisfy the condition and avoid useless search.

The second technique uses some kind of watched literals to ensure that the condition $\sum_{C_i \in adj(X)} C_i^{(X,a)}(s^*_{X=a}) \geq \delta$ still applies. Instead of maintaining the maximum

[1] A pseudo-Boolean constraint is a constraint $\sum_i w_i.x_i \geq b$ where $w_i$ and $b$ are integers and $x_i$ are Booleans

(normalized) cost $m_i$ of tuples for all constraints $C_i$, $m_i$ is only maintained for a minimal number of constraints which is sufficient to ensure that $\sum_i m_i \geq \delta$. Unfortunately, the number of constraints to watch is not constant throughout the search and it's not clear if this technique is really more efficient than the counter based technique.

An alternative is to weaken the "pseudo-Boolean" condition into a cardinality constraint. This means that condition $\sum_{C_i \in adj(X)} C_i^{(X,a)}(s^*_{X=a}) \geq \delta$ is approximated by the condition "at least $n$ constraints among $adj(X)$ must have a cost greater than or equal to $m$" where $m, n$ are computed from the initial condition. Clearly, this approximation will not cut each useless branch but can be more efficient in some circumstances.

### 3.2 The Max-CSP case

In the Max-CSP context, each cost is either 0 or 1. Therefore, condition (1) is directly a cardinality constraint which is easier to enforce than general pseudo-Boolean constraints.

In this context, $cost(X, a) = aic(X, a)$ and condition (1) translates as "the number of constraints adjacent to $X$ which support $(X, a)$ and which are violated by $s^*_{X=a}$ is greater than or equal to $\delta$".

In the rest of the paper, we will present the approach on the easier Max-CSP case. Translation to the WCSP case is in general obvious by substituting the corresponding condition of the theorem.

## 4 The Decomposition Approach

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

**Definition 41** *Let $P$ be a Max-CSP instance, $X$ be a variable of $P$, $a$ be a best value of $X$, $\delta$ be the gap of $X$ and $C_1, \ldots, C_m$ be the $m$ constraints involving $X$ which support $(X, a)$. 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 neighbors of $X$ so that at least $\delta$ 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 31 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 $\delta$, 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 A Max-CSP Example

To illustrate the decomposition technique in the Max-CSP case, 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 best values. For example, $(X_1, 1)$ is one 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 |

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

In this case, for $X_1$ there is only one 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 in the Max-CSP case

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 $(X, a)$.

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 $\delta$ 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 $\delta$ 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 $\delta$ variables among the $m$ variables $Y_i$ have their domain reduced to $I_{Y_i}^{X,a}$ translates to the condition 'at least $\delta$ bits in the binary word representing the branch must be 0'. This condition is trivial to enforce in a binary branching.

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

| $Y_1$ | $Y_2$ | $Y_3$ | $Y_4$ |
|---|---|---|---|
| 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 $\delta = 3$

**Fig. 2.** List of branches to explore for $n = 4$ and different values of $\delta$

As an example, Figure 2 represents the branches that must be explored for two different values of $\delta$ 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 $\delta$ 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 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 [18] 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 [13–17]. A tree decomposition involves computing a pseudo-tree [21, 22] 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 [18], 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 in the Max-CSP case

Another way to exploit Theorem 31 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 [23], at each node $\nu$, a value $(X, a)$ is selected, and two branches are built from $\nu$: a left one labeled with the variable assignment $X = a$, and a right one labeled with the value refutation $X \neq a$. Considering the current instance at node $\nu$, let $a$, $\delta$ and $\{C_i\}$ be the best value of $X$, the 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 $\nu$. This constraint is violated as soon as it is no more possible to find in $\{C_i\}$, at least $\delta$ constraints which do not support anymore $(X, a)$. Of course, a constraint posted with respect to the right branch of node $\nu$ must be removed when the algorithm backtracks from $\nu$.

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 [24]), 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 $\delta$ constraints of $\{C_i\}$ can still be in a position of not supporting $(X, a)$, we can impose that these $\delta$ constraints do not support $(X, a)$, making then new inferences. For example, for a binary constraint of $\{C_i\}$, among the $\delta$ 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 Max-CSP 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 [2] 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 $dom/ddeg$ [25], 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 $dom * gap/ddeg$, involves the gap of the variables. More precisely, the ratio $dom/ddeg$ is multiplied by the gap in order to favor 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 behavior 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 [17]), $dimacs$ ($ssa$) and $composed$, and also to real-world 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 $74$

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

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.

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 both WCSP and Max-CSP the principle of inferred disjunctive constraints introduced in [18] for CSP. Using weights, 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 [18]) 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 on Max-CSP instances, 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.

## Acknowledgments

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

## References

1. Freuder, E., Wallace, R.: Partial constraint satisfaction. Artificial Intelligence **58**(1-3) (1992) 21–70
2. Larrosa, J., Meseguer, P., Schiex, T.: Maintaining reversible DAC for Max-CSP. Artificial Intelligence **107**(1) (1999) 149–163
3. Affane, M., Bennaceur, H.: A weighted arc-consistency technique for Max-CSP. In: Proceedings of ECAI'98. (1998) 209–213

| | | 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.

4. Larrosa, J., Meseguer, P.: Partition-Based lower bound for Max-CSP. Constraints **7** (2002) 407–419
5. Regin, J., Petit, T., Bessiere, C., Puget, J.: New lower bounds of constraint violations for over-constrained problems. In: Proceedings of CP'01. (2001) 332–345
6. Larrosa, J., Schiex, T.: In the quest of the best form of local consistency for Weighted CSP. In: Proceedings of IJCAI'03. (2003) 363–376
7. Cooper, M., Schiex, T.: Arc consistency for soft constraints. Artificial Intelligence **154**(1-2) (2004) 199–227
8. de Givry, S., Heras, F., Zytnicki, M., Larrosa, J.: Existential arc consistency: Getting closer to full arc consistency in weighted CSPs. In: Proceedings of IJCAI'05. (2005) 84–89
9. Cooper, M., de Givry, S., Schiex, T.: Optimal Soft Arc Consistency. In: Proceedings of IJCAI'07. (2007) 68–73
10. Verfaillie, G., Lemaitre, M., Schiex, T.: Russian doll search for solving constraint optimization problems. In: Proceedings of AAAI'96. (1996) 181–187
11. Meseguer, P., Sanchez, M.: Specializing russian doll search. In: Proceedings of CP'01. (2001) 464–478
12. Larrosa, J., Dechter, R.: Boosting search with variable elimination in constraint optimization and constraint satisfaction problems. Constraints **8**(3) (2003) 303–326
13. Jégou, P., Terrioux, C.: Hybrid backtracking bounded by tree-decomposition of constraint networks. Artificial Intelligence **146**(1) (2003) 43–75
14. Jégou, P., Terrioux, C.: Decomposition and Good Recording for Solving Max-CSPs. In: Proceedings of ECAI'04. (2004) 196–200
15. Marinescu, R., Dechter, R.: AND/OR Branch-and-Bound for Graphical Models. In: Proceedings of IJCAI'05. (2005) 224–229
16. Marinescu, R., Dechter, R.: Advances in AND/OR Branch-and-Bound Search for Constraint Optimization. In: Proceedings of the CP'05 Workshop on Preferences and Soft Constraints. (2005)
17. de Givry, S., Schiex, T., Verfaillie, G.: Exploiting Tree Decomposition and Soft Local Consistency In Weighted CSP. In: Proceedings of AAAI'06. (2006)
18. Freuder, E., Hubbe, P.: Using inferred disjunctive constraints to decompose constraint satisfaction problems. In: Proceedings of IJCAI'93. (1993) 254–261
19. Bennaceur, H., Lecoutre, C., Roussel, O.: A Decomposition Technique for Solving Max-CSP. In: 18th European Conference on Artificial Intelligence(ECAI'08), Patras, Greece (jul 2008) full paper to appear.
20. Larrosa, J.: Node and arc consistency in weighted csp. In: Eighteenth national conference on Artificial intelligence, Menlo Park, CA, USA, American Association for Artificial Intelligence (2002) 48–53
21. Freuder, E., Quinn, M.: Taking advantage of stable sets of variables in constraint satisfaction problems. In: Proceedings of IJCAI'85. (1985) 1076–1078
22. Bayardo, R., Miranker, D.: On the space-time trade-off in solving constraint satisfaction problems. In: Proceedings of IJCAI'95. (1995) 558–562
23. Hwang, J., Mitchell, D.: 2-way vs d-way branching for CSP. In: Proceedings of CP'05. (2005) 343–357
24. Boussemart, F., Hemery, F., Lecoutre, C., Sais, L.: Boosting systematic search by weighting constraints. In: Proceedings of ECAI'04. (2004) 146–150
25. Bessiere, C., Régin, J.: MAC and combined heuristics: two reasons to forsake FC (and CBJ?) on hard problems. In: Proceedings of CP'96. (1996) 61–75