

Range-Based Algorithm for Max-CSP

Thomas Petit, Jean-Charles Régin, Christian Bessiere

▶ **To cite this version:**

Thomas Petit, Jean-Charles Régin, Christian Bessiere. Range-Based Algorithm for Max-CSP. CP: Principles and Practice of Constraint Programming, Sep 2002, Ithaca, NY, United States. pp.280-294, 10.1007/3-540-46135-3_19 . lirmm-00268450

HAL Id: lirmm-00268450

https://hal-lirmm.ccsd.cnrs.fr/lirmm-00268450

Submitted on 4 Sep 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Range-Based Algorithm for Max-CSP

Thierry Petit[1/2], Jean-Charles Régin[1] and Christian Bessière[2]
{regin, tpetit}@ilog.fr, {bessiere, tpetit}@lirmm.fr

[1]ILOG, 1681, route des Dolines, 06560 Valbonne, FRANCE
[2]LIRMM (UMR 5506 CNRS), 161, rue Ada, 34392 Montpellier Cedex 5, FRANCE

Abstract. A Max-CSP consists of searching for a solution which minimizes the number of violated constraints. The best existing solving algorithm is PFC-MRDAC. It is based on the computation of a lower bound of the number of violations. This lower bound is obtained by evaluating the violations involved by each value of each domain. Unfortunately, some applications imply thousands of variables with huge domains. In scheduling, it arises that numerous activities have to be scheduled over several month with a unit of time of a few minutes. In this situation using PFC-MRDAC requires a large amount of memory which can prevent from using it. In this paper, we propose an algorithm called the Range-based Max-CSP Algorithm (RMA), based on the exploitation of bound-based filtering algorithms of constraints. This technique does not require to study each value of each domain: its complexity depends only on the number of variables and the number of constraints. No hypothesis is made on the constraints except that their filtering algorithms are related to the bounds of the involved variables, the general case of scheduling constraints. Then, when the only information available for a variable x w.r.t. a constraint C are the new bounds of $D(x)$ obtained by applying the filtering algorithm of C, the lower bounds of violations provided by PFC-MRDAC and RMA are identical.

1 Introduction

A problem is over-constrained when no assignment of values to variables satisfies all constraints. The simplest theoretical framework for over-constrained problems is the Maximal Constraint Satisfaction Problem (Max-CSP). A solution of a Max-CSP is a total assignment of values to variables that minimizes the number of constraint violations.

The best solving algorithm for Max-CSP, PFC-MRDAC [1], has been extended to the non binary case [4,5,3]. Moreover, in [4,5] it has been shown that a Max-CSP can be encoded through a global constraint. When we define a Max-CSP as a global constraint, then we use the PFC-MRDAC algorithm as the filtering algorithm of the constraint. This formulation is suited to real-life problems, because generally some parts correspond to Max-CSPs although the whole problem cannot be expressed as a Max-CSP. Moreover any search algorithm can be used. Therefore we selected it to describe the algorithms presented in this paper.

For sake of clarity, we consider first that a Max-CSP is solved by a Branch and Bound based search algorithm : successive assignments of values to variables are performed through a depth-first traversal of the search tree, where internal nodes represent incomplete assignments and leaf nodes stand for complete ones. The number of violations to minimize is expressed by an objective variable obj. For any given node, $UB = max(D(obj)) + 1$ corresponds to the number of violations of the best solution found so far. That is, the solution we would like to improve. $min(D(obj))$ is the number of violated constraints in C detected by the solver at the current step of the search. In fact, $min(D(obj))$ can be simply defined as the number of constraints C such that values have been assigned to all variables in the set $var(C)$ of the variables involved in C, and C is violated. This number is generally called the *distance*. If $distance > max(D(obj))$ then the current best solution cannot be improved below the current node. Thus it is not necessary to traverse the sub-tree rooted by the current node.

PFC-MRDAC improves that condition by computing a lower bound LB of the number of violations, equal to *distance* plus an under-estimation of the number of violations entailed by constraints involving some variables which have not yet been instantiated. The new condition of pursuit of the search is $LB \leq max(D(obj))$. When filtering, PFC-MRDAC combines generally LB with a lower bound local to each value, in order to remove this value if it cannot belong to a solution.

All these lower bounds are based on direct violations of constraints by values: they require to maintain for each value a of the domain of a variable x the number of constraints C such that a is not consistent with C. This principle is applicable to a wide number of problems, providing that domain sizes are not too big.

Unfortunately, some applications involve a big number of variables with huge domain sizes.

For instance, consider the problem of scheduling the construction of offices. Some activities, as painting or installing windows, have a short duration and require essentially human resources. They can be performed over a large period. They concern each office of each floor in each building, and they are linked to other activities by precedence rules. Given that the manager has to deal with a large number of workers which express preference constraints with respect to their week schedule, it is mandatory to have a time unit of at most one quarter of hour. If the total duration of the project is one year then for each activity we have 35040 possible start dates. If there exists thousands of activities then it is not realistic to maintain for each possible date of each activity a counter of inconsistencies. Moreover, since scheduling constraints generally propagate on bounds, redundant computations will be made when testing the consistency of each value of a domain with a constraint. Therefore, PFC-MRDAC is not really suited to this case.

In this paper, we propose a new algorithm called the Range-based Max-CSP Algorithm (RMA), which computes a lower bound by exploiting filtering

algorithms of constraints[1]. The principle is similar to PFC-MRDAC except concerning the computation of inconsistencies involved by each variable. Instead of evaluating each value of a domain $D(x)$, the RMA stores two entities per constraint C in which x is involved: the new minimum and maximum of $D(x)$ obtained when applying the filtering algorithm of C. For instance, for 100 variables with domains of 35040 values and 100 constraints, the RMA requires to maintain 200 minima and maxima per variable instead of 35040 in PFC-MRDAC. That is, instead of having a total of 3.5 millions of counters with PFC-MRDAC we have 20000 datas with the RMA.

We show that in the case of constraints which propagate only bounds of domains, the same LB is obtained by using PFC-MRDAC and the RMA.

We compare the complexity of a non incremental implementation with PFC-RDAC[2]: when domain sizes are big the amortized complexity of the RMA is better because it does not depends on the number of values of domains.

We provide a filtering algorithm based on the same principle, which propagate on bounds. We discuss about the incremental version of the RMA and about a filtering algorithm which performs holes in domains. We point out a promising generalization of our algorithm: for general problems, it is possible to take into account ranges of succesive values involving the same number of violations to compute the lower bounds.

2 Preliminaries

CSP A *constraint network* $\mathcal{N}$ is defined as a set of n *variables* $\mathcal{X} = \{x_1, \ldots, x_n\}$, a set of *domains* $\mathcal{D} = \{D(x_1), \ldots, D(x_n)\}$ where $D(x_i)$ is the finite set of possible *values* for variable x_i, and a set $\mathcal{C}$ of *constraints* between variables. A *constraint* C on the ordered set of variables $var(C) = (x_{i_1}, \ldots, x_{i_r})$ (also denoted by $C(x_{i_1}, \ldots, x_{i_r})$) is a subset $rel(C)$ of the Cartesian product $D(x_{i_1}) \times \cdots \times D(x_{i_r})$ that specifies the *allowed* combinations of values for the variables $x_{i_1}, \ldots, x_{i_r}$. $D(var(C)) = \cup_{x \in var(C)} D(x)$. An element of $D(x_{i_1}) \times \cdots \times D(x_{i_r})$ is called a *tuple on* $var(C)$. $|var(C)|$ is the *arity* of C. C is *binary* iff $|var(C)| = 2$. A value a for a variable x is denoted by (x, a). A tuple τ on $var(C)$ is *valid* if $\forall (x, a) \in \tau, a \in D(x)$. C is *consistent* iff there exists a tuple τ of $rel(C)$ which is valid. A value $a \in D(x)$ is *consistent with* C iff $x \notin var(C)$ or there exists a valid tuple τ of $rel(C)$ in which a is the value assigned to x. Given $Y \subseteq \mathcal{X}$, an *instantiation* I of Y is an assignment of values to variables Y such that $\forall x \in Y$, the value a assigned to x belongs to $D(x)$. Given $Y \subseteq \mathcal{X}$ and $C \in \mathcal{C}$ such that $var(C) \subseteq Y$, an instantiation I of Y *satisfies* a constraint C iff the projection of I on $var(C)$ belongs to $rel(C)$. If I does not satisfy C, then I *violates* C. The *Constraint Satisfaction Problem* (CSP) consists of finding an instantiation I of X such that $\forall C \in \mathcal{C}$, I satisfies C.

[1] An implementation of PFC-MRDAC based on filtering algorithms of constraints were proposed in [6], but this one requires also to maintain one counter by value.

[2] PFC-RDAC [2] is the non incremental version of PFC-MRDAC.

Over-Constrained problem When a CSP has no solution, we say that the problem is *over-constrained*. $\mathcal{C}_h \subseteq \mathcal{C}_s$ is the set of *hard* constraints, that is, the constraints that must necessarily be satisfied. $\mathcal{C}_s = \mathcal{C} \setminus \mathcal{C}_h$ is the set of *soft* constraints. Let I be an instantiation of X. If I is a solution of an over-constrained problem then $\forall C \in \mathcal{C}_h$, I satisfies C.

Max-CSP The Max-CSP is the problem where $\mathcal{C}_h = \emptyset$ and the goal is to find an assignment of values to variables that minimizes the number of violations in $\mathcal{C} = \mathcal{C}_s$.

Max-CSP as a global constraint Let $\mathcal{N} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ be a constraint network. Constraints in $\mathcal{C}$ can be encapsulated into a single constraint [4,5], called the Satisfiability Sum Constraint (ssc):

Definition 1 *Let $\mathcal{C} = \{C_i, i \in \{1, \ldots, m\}\}$ be a set of constraints, and $S[\mathcal{C}] = \{s_i, i \in \{1, \ldots, m\}\}$ be a set of variables and obj be a variable, such that a one-to-one mapping is defined between $\mathcal{C}$ and $S[\mathcal{C}]$. A* **Satisfiability Sum Constraint** *is the constraint $ssc(\mathcal{C}, S[\mathcal{C}], obj)$ defined by:*

$$[obj = \sum_{s_i=1}^{m} s_i] \wedge \bigwedge_{i=1}^{m} [(C_i \wedge (s_i = 0)) \vee (\neg C_i \wedge (s_i = 1))]$$

Notation 1 *Given a $ssc(\mathcal{C}, S[\mathcal{C}], obj)$, a variable x, $a \in D(x)$ and $\mathcal{K} \subseteq \mathcal{C}$:*
- *$max(D(obj))$ is the highest value of current domain of obj;*
- *$min(D(obj))$ is the lowest value of current domain of obj;*
- *$minObj(\mathcal{C}, S[\mathcal{C}])$ is the minimum value of obj consistent with $ssc(\mathcal{C}, S[\mathcal{C}], obj)$;*
- *$minObj((x, a), \mathcal{C}, S[\mathcal{C}])$ is equal to $minObj(\mathcal{C}, S[\mathcal{C}])$ when $x = a$;*
- *$S[\mathcal{K}]$ is the subset of $S[\mathcal{C}]$ equals to the projection of variables $S[\mathcal{C}]$ on $\mathcal{K}$;*
- *$X(\mathcal{C})$ is the union of $X(C_i), C_i \in \mathcal{C}$.*

The variables $S[\mathcal{C}]$ are used in order to express which constraints of $\mathcal{C}$ must be violated or satisfied: value 0 assigned to $s \in S[\mathcal{C}]$ expresses that its corresponding constraint C is satisfied, whereas 1 expresses that C is violated. Throughout this formulation, a solution of a Max-CSP is an assignment that satisfies the ssc with the minimal possible value of obj. Comparing a lower bound of the objective to $max(D(obj))$ of a Max-CSP leads to a necessary consistency condition of the ssc. Domain reduction algorithms for the Max-CSP correspond to specific filtering algorithms of the ssc.

3 Generalized version of PFC-MRDAC

PFC-MRDAC [1] is considered as the best algorithm for solving binary Max-CSPs. This section is a summary of the generalization of this algorithm to non-binary constraints [4,5,3]. Basically, PFC-MRDAC is based on counters of violations involved by each value of each domain :

Definition 2 *Let x be a variable, a be a value of $D(x)$, $\mathcal{C}$ be a set of constraints,* $\#inc((x,a),\mathcal{C}) = |\{C \in \mathcal{C} \text{ s.t. } (x,a) \text{ is not consistent with } C\}|$.

In a solver, this information can be obtained by applying independently the specific filtering algorithms of the constraints in which x is involved while the domain of x is reduced to the value a we study.

3.1 Necessary condition of consistency

From the definition of $minObj(\mathcal{C}, S[\mathcal{C}])$ we have:

Property 1 *If $minObj(\mathcal{C}, S[\mathcal{C}]) > max(D(obj))$ then $ssc(\mathcal{C}, S[\mathcal{C}], obj)$ is not consistent.*

A lower bound of $minObj(\mathcal{C}, S[\mathcal{C}])$ provides a necessary condition of consistency of a *ssc*. A possible way for computing it is to perform a sum of independent lower bounds of violations, one per variable. For each variable a lower bound can be defined by:

Definition 3 *Given a variable x and a constraint set $\mathcal{K}$,* $\#inc(x,\mathcal{K}) = min_{a \in D(x)}(\#inc((x,a),\mathcal{K}))$.

The sum of these minima with $\mathcal{K} = \mathcal{C}$ cannot lead to a lower bound of the total number of violations, because some constraints can be taken into account more than once. In this case, the lower bound can be overestimated, and an inconsistency could be detected while the *ssc* is consistent.

In the binary case, the constraint graph[3] is used in order to guarantee this independence [1]. Each edge is oriented and for each variable x only the constraints out-going x are taken into account.

This idea can be generalized to the non binary case, by associating with each constraint C one and only one variable x involved in the constraint [4,5]: C is then taken into account only for computing the $\#inc$ counter of x. Therefore, the constraints are partitioned according to the variables that are associated with:

Definition 4 *Given a set of constraints $\mathcal{C}$, a* **var-partition** *of $\mathcal{C}$ is a partition $\mathcal{P}(\mathcal{C}) = \{P(x_1), ..., P(x_k)\}$ of $\mathcal{C}$ in $|X(\mathcal{C})|$ sets such that $\forall P(x_i) \in \mathcal{P}(\mathcal{C}) : \forall C \in P(x_i), x_i \in X(C)$.*

Given a var-partition $\mathcal{P}(\mathcal{C})$, the sum of all $\#inc(x_i, P(x_i))$ is a lower bound of the total number of violations, because all sets belonging to $\mathcal{P}(\mathcal{C})$ are disjoint; thus we obtain the following lower bound:

$$LB_{\mathcal{P}(\mathcal{C})} = \sum_{x_i \in X(\mathcal{C})} \#inc(x_i, P(x_i))$$

Property 2 $\forall \mathcal{P}(\mathcal{C}) = \{P(x_1), ..., P(x_k)\}$, $LB_{\mathcal{P}(\mathcal{C})} \leq minObj(\mathcal{C}, S[\mathcal{C}])$.

[3] the graph where vertices are variables and edges are binary constraints between pairs of variables.

The necessary condition of consistency of a *ssc* is deduced from this Property:

Corollary 1 $\forall \mathcal{P}(\mathcal{C}) = \{P(x_1), ..., P(x_k)\}$, *If* $LB_{\mathcal{P}(\mathcal{C})} > max(D(obj))$ *then* $ssc(\mathcal{C}, S[\mathcal{C}], obj)$ *is not consistent.*

3.2 Filtering algorithm

PFC-MRDAC [1] and its extension to the non binary case [4,5,3] include a look-ahead procedure used to reduce domains of variables that have not yet an assigned value. From definition of $minObj((x,a), \mathcal{C}, S[\mathcal{C}])$ we have the following theorem:

Theorem 1 $\forall x \in X(\mathcal{C}), \forall a \in D(x)$*: if* $minObj((x,a), \mathcal{C}, S[\mathcal{C}]) > max(D(obj))$ *then* (x,a) *is not consistent with ssc* $(\mathcal{C}, S[\mathcal{C}], obj)$.

Any lower bound of $minObj((x,a), \mathcal{C}, S[\mathcal{C}])$ can be used to check the consistency of (x,a). An obvious lower bound is $\#inc((x,a), \mathcal{C})$:

Property 3 $\#inc((x,a), \mathcal{C}) \leq minObj((x,a), \mathcal{C}, S[\mathcal{C}])$

From this Property and theorem 1, we obtain a first filtering algorithm. It can be improved by including the lower bound of Property 2. This idea was introduced by Larrosa et al. [1], for binary constraints networks.

It can be applied in the general case [4,5,3]. In order to do so, we suggest to split $\mathcal{C}$ into two disjoint sets $P(x)$ and $\mathcal{C} - P(x)$, where $P(x)$ is the subset of constraints associated with x in a var-partition $\mathcal{P}(\mathcal{X})$ of $\mathcal{C}$. Consider the following corollary of Theorem 1:

Corollary 2 *Let* $\mathcal{P}(\mathcal{X})$ *be a var-partition of* $\mathcal{C}$, x *a variable and* $a \in D(x)$, *if* $minObj((x,a), P(x), S[P(x)]) + minObj((x,a), \mathcal{C} - P(x), S[\mathcal{C} - P(x)]) > max(D(obj))$ *then* (x,a) *is not consistent with* $ssc(\mathcal{C}, S[\mathcal{C}], obj)$.

Proof: e.g., [5].

Note that $minObj(\mathcal{C} - P(x), S[P(x)]) \leq minObj((x,a), \mathcal{C} - P(x), S[P(x)])$. From this remark and Properties 2 and 3 the following theorem can be deduced:

Theorem 2 $\forall \mathcal{P}(\mathcal{C})$ *a var-partition of* $\mathcal{C}, \forall x \in X(\mathcal{C}), \forall a \in D(x)$, *if* $\#inc((x,a), P(x)) + LB_{\mathcal{P}(\mathcal{C} - P(x))} > max(D(obj))$ *then* a *can be removed from* $D(x)$.

4 The Range-based Max-CSP Algorithm

4.1 Principle

PFC-MRDAC algorithm requires to maintain one counter $\#inc$ $((x,a), P(x))$ for each value a of each domain $D(x)$. The computation of these counters is sufficient to efficiently compute the consistency of a satisfiability sum constraint (see Corollary 1) and to apply the filtering algorithm associated with it (see

Theorem 2). Therefore, the main issue of the PFC-MRDAC algorithm is the computation of these counters, and if this coumputation is speed-up then the algorithm is improved.

We propose a new algorithm called the Range-based Max-CSP Algorithm (RMA) that does not require one counter per value although the principle is very similar to the non binary PFC-MRDAC: for each variable x we consider a set $P(x)$ of a var-partition of constraints $\mathcal{P}(\mathcal{C})$, in order to compute a lower bound $\#inc(x, P(x))$ of the violations involved by x.

The difference is the way we compute $\#inc(x, P(x))$. In the RMA, there is no need to maintain one counter of violation $\#inc((x, a), P(x))$ for each value a in $D(x)$ as it is the case in PFC-MRDAC.

The idea exploited in our algorithm is based on the following definition:

Definition 5 *Let $I \subseteq D(x)$ be a range of consecutive values. If $\forall a \in I$, $\forall b \in I$, $\#inc((x, a), P(x)) = \#inc((x, b), P(x))$ then I is homogeneously inconsistent.*

If any value in a range I violates the same number of constraints of $P(x)$ then we can consider globally the range to evaluate the number of violations, instead of studying the values one by one:

Property 4 *Let I be an homogeneously inconsistent range. The number of constraints in $P(x)$ violated if $D(x) = I$ is $\#inc(I, P(x)) = \#inc((x, a), P(x)), a \in I$.*

The proof of this property is obvious (from definition 5).

It is possible to take into account only a set of homogeneously inconsistent ranges to compute $\#inc(x, P(x))$, provided that each value of $D(x)$ belongs to one range. More formally, we have:

Definition 6 *A set $\mathcal{I}(P(x)) = \{I_1, ..., I_m\}$ of homogeneously inconsistent ranges such that $\forall a \in D(x), \exists I \in \mathcal{I}(P(x))$ and $a \in I$ is called a set of homogeneously inconsistent ranges which covers $D(x)$.*

Property 5 *Let $\mathcal{I}(P(x))$ be a set of homogeneously inconsistent ranges which covers $D(x)$. Then we have: $\#inc(x, P(x)) = min_{I \in \mathcal{I}(P(x))} \#inc(I, P(x))$.*

proof: from Property 4

Thus, if we are able to identify such a set $\mathcal{I}(P(x))$ with $|\mathcal{I}(P(x))| < |D(x)|$ then it is possible to improve PFC-MRDAC in two ways:

1. the number of counters required to compute $\#inc(x, P(x))$ is smaller,
2. $\#inc(x, P(x))$ can be computed faster.

The first point is quite important in practice. Indeed, in some problems the number of variables and the size of the domains make methods requiring for each value of each variable some additional data, as $\#inc((x, a), P(x))$, unusable in

pratice. For instance, this is the case of almost all real world scheduling applications. Moreover, in such problems, the filtering algorithms associated with constraints are range-based filtering algorithms: they reduce only the bounds of the domain. They do not create any "holes" in domains.

Such a range-based approach is interesting if some issues can be efficiently solved:

- the number of ranges is small,
- for each range I, $\#inc(I, P(x))$ can be efficiently computed.

In the remaining of this section, we will consider that only range-based filtering algorithms are used. Under this condition, we will prove that:

- the size of $\mathcal{I}(P(x))$ is at most $2 * |P(x)| + 1$,
- all $\#inc(I, P(x))$ can be computed in $O(|P(x)|)$ provided that we have computed $\mathcal{I}(P(x))$.

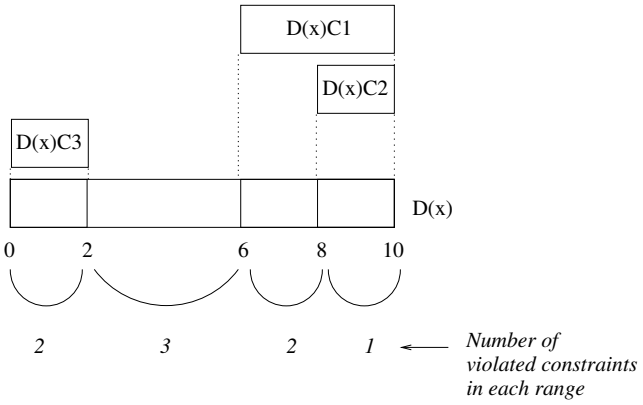
4.2 Computation of $\#inc$ counters

The principle is to consider the bounds obtained by applying independently each filtering algorithm of constraints in $P(x)$ on $D(x)$:

Notation 2 *Let $\mathcal{P}(\mathcal{C})$ be a var-partition, x a variable, $P(x) \in \mathcal{P}(\mathcal{C})$ and $C \in P(x)$: $D(x)_C$ is the domain obtained by applying the filtering algorithm of C on $D(x)$.*

For each constraint C we consider the minimum $min(D(x)_C)$ and the maximum $max(D(x)_C)$ of $D(x)_C$. By ordering all these minima and maxima from the lower to the greater, it is possible to divide $D(x)$ in different ranges. Each of them corresponds to a certain number of violations. The following figure illustrates an example where $P(x)$ is the set of constraints $\{C_1, C_2, C_3\}$, which involve four variables x, y, z, t such that $D(x) = D(y) = D(z) = D(t) = [0, 10]$:

- $C_1 = [x - y > 5]$ that leads to $min(D(x)_{C_1}) = 6$, $max(D(x)_{C_1}) = 10$,
- $C_2 = [x - z > 7]$ that leads to $min(D(x)_{C_2}) = 8$, $max(D(x)_{C_2}) = 10$,
- $C_3 = [t - x > 7]$ that leads to $min(D(x)_{C_3}) = 0$, $max(D(x)_{C_3}) = 2$.



For this example the homogeneously inconsistent ranges are $[0,3), [3,6), [6,8), [8,11)$ and $\#inc(x, P(x)) = \#inc(x, P(x)) = min_{I \in \mathcal{I}(P(x))} \#inc(I, P(x)) = min(\#inc([0,3), P(x)) = 2, \#inc([3,6), P(x)) = 3, \#inc([6,8), P(x)) = 2, \#inc([8,11), P(x)) = 1) = 1$.

Now, we present more formally, this idea:

Definition 7 *Let $B(P(x))$ be the set of bounds sorted in ascending order, defined by the bounds of $D(x)$ and the bounds of domains $D(x)_C$ for all constraints C in $P(x)$. We will denote by $\mathcal{I}(B(P(x)))$ the set of ranges such that each range is defined by a pair of two consecutive elements of B. The k^{th} range of $\mathcal{I}(B(P(x)))$ is denoted by $I_k = [p, q)$, where p is the minimum of I_k and $q - 1$ is the maximum.*

Property 6 *The following properties hold:*

1. *the maximal possible number of ranges in $\mathcal{I}(B(P(x)))$ is $2 * |P(x)| + 1$*
2. *$\forall a \in D(x)\ \exists I \in \mathcal{I}(B(P(x)))$ such that $a \in I$*
3. *$\forall I \in \mathcal{I}(B(P(x)))\forall C \in P(x) : I \subseteq D_C(x)$ or $I \cap D_C(x) = \emptyset$*
4. *$\forall I \in \mathcal{I}(B(P(x)))$, I is homogeneously inconsistent.*
5. *$\mathcal{I}(B(P(x)))$ is a set of homogeneously inconsistent ranges which covers $D(x)$.*

proof:

1. $B(P(x))$ contains at most $2 * |P(x)| + 2$ values, thus the maximum number of ranges is $2 * |P(x)| + 1$.
2. By construction of $\mathcal{I}(B(P(x)))$
3. Suppose that $I \cap D_C(x) = J$ with $J \neq \emptyset$ and $J \subset I$. Then, one bound of J is equal to a bound of $D_C(x)$ and is not equal to a bound of I. So, I contains a bound of $D_C(x)$ which is not one of its bounds. This is not possible by construction of $\mathcal{I}(B(P(x)))$.
4. By 3 $\forall C \in P(x)$ and $\forall a, b \in I : a \cap D_C(x) = b \cap D_C(x)$. Therefore, $\forall a \in I$ $\#inc((x, a), P(x)) = \#inc(I, P(x))$.
5. Immediate from 2 and 4.

The problem is then to compute the value $\#inc(I, P(x))$ for each $I \in P(x))$. It is not possible to compute *independently* each $\#inc(I, P(x))$ in $O(1)$, but we will show that all $\#inc(I, P(x))$ can be computed in $O(|P(x)|)$. Therefore, since $|\mathcal{I}(B(P(x)))| \leq 2 * |P(x)| + 1$, the amortized complexity for an $\#inc(I, P(x))$ is $O(1)$.

The following property helps us to compute all $\#inc(I, P(x))$ in $O(|P(x)|)$.

Property 7 $\#inc(I, B(P(x))) = |P(x)| - |\{C / D_C(x) \cap I = I\}|$

proof: $\#inc(I, B(P(x))) = |\{C / D_C(x) \cap I = \emptyset\}|$ and by Property 6.3 $|\{C / D_C(x) \cap I = \emptyset\}| + |\{C / D_C(x) \cap I = I\}| = |P(x)|$

The ranges $\mathcal{I}(B(P(x))$ are traversed w.r.t. the ascending order, applying for each $I_K = [a, b)$ the following rules to determine how many constraints are satisfied:

- if a is a minimum of a domain $D(x)_C$ then we enter in a new $D(x)_C$ and the range I_k satisfies one constraint more than the previous one, I_{k-1}.
- if $a-1$ is a maximum of a domain $D(x)_C$: we close a domain $D(x)_C$ and I_k satisfies one constraint less than I_{k-1}.

This idea has to be refined, given that a value in $D(x)$ may correspond to minima or maxima of several $D(x)_{C_i}$. Therefore we count the number of minima of domains equal to a, and maxima of domains equal to $a-1$.

Notation 3 *Given $a \in D(x)$.*

- *$nMin(a, B(P(x)))$ denotes the number of domains $D(x)_C \in B(P(x))$ such that $min(D(x)_C) = a$,*
- *$nMax(a, B(P(x)))$ denotes the number of domains $D(x)_C \in B(P(x))$ such that $max(D(x)_C) = a$.*

Proposition 1 *Given $I_k = [p, q)$. The number of satisfied constraint $\#sat(I_k, P(x))$ of a range $I_k \in \mathcal{I}(B(P(x)))$ is defined recursively by :*

$$\#sat(I_k, P(x)) = \#sat(I_{k-1}, P(x)) + nMin(p, B(P(x))) - nMax(p-1, B(P(x)))$$

with $\#sat(I_{-1}, P(x)) = 0$.

proof: By induction. I_0 is equal to $[min(D(x)), o)$, because all $D(x)_C \subseteq D(x)$. Thus, $\#sat(I_0, P(x)) = \#sat(I_{-1}, P(x)) + nMin(min(D(x)), B(P(x))) - nMax(min(D(x)) - 1, B(P(x))) = nMin(min(D(x)), B(P(x)))$. Therefore the property holds for I_0. Suppose that it is true for I_{k-1}, and $I_k = [p, q)$. By definition of $nMin$ and $nMax$ and from Property 6.3 we have $\#sat(I_k, P(x)) = \#sat(I_{k-1}, P(x)) + nMin(p, B(P(x))) - nMax(p-1, B(P(x)))$

From this proposition and Property 7 the following property holds:

Property 8 *Given $I = [p, q)$. Then, $\#inc(I, P(x)) = |P(x)| - \#sat(I, P(x))$.*

Consider again the previous example in order to illustrate this principle:

Interval I_k	$\#sat(I_k, P(x))$	$\#inc(I_k, P(x))$
$I_1 = [0, 3)$	0 + 1 - 0 = 1	2
$I_2 = [3, 6)$	1 + 0 - 1 = 0	3
$I_3 = [6, 8)$	0 + 1 - 0 = 1	2
$I_4 = [8, 11)$	1 + 1 - 0 = 2	1

We deduce $\#inc(x, P(x)) = min_{I \in P(x)}(\#inc(I_k, P(x)) = 1$.

Algorithm 1 is a possible implementation of the computation of $\#inc$ counters.

Algorithm 1: ComputeMinInc

```
Data   : x, P(x), B(P(x)), I(B(P(x)))
Result : #inc(x, P(x))
#sat ← 0;
#inc(x, P(x)) ← |P(x)|;
k ← 1;
while k ≠ |I(B(P(x)))| do
    let I_k = [p, q) be the k^th range of I(B(P(x)));
    #sat ← #sat + nMin(p, B(P(x))) − nMax(p − 1, B(P(x)));
    if |P(x) − #sat| < #inc(x, P(x)) then
        #inc(x, P(x)) ← |P(x) − #sat|;
        k ← k + 1
    end
end
return #inc(x, P(x));
```

4.3 Consistency of a ssc

The $LB_{\mathcal{P}(\mathcal{C})}$ is computed by performing a sum of all $\#inc(x, P(x))$, following a schema similar to PFC-MRDAC. Note that the LB provided by PFC-RDAC and RMA are the same when filtering algorithms reduce only the bounds of the domains.

Compared with PFC-MRDAC, the computation of the lower bound $LB_{\mathcal{P}(\mathcal{C})}$ of Property 2 remains the same. Thus, we have: $LB_{\mathcal{P}(\mathcal{C})} = \sum_{x_i \in X(\mathcal{C})} \#inc(x_i, P(x_i))$. The difference is the computation of all $\#inc(x_i, P(x_i))$.

Algorithm Given $\mathcal{I}(B(P(x)))$, we propose an implementation of the algorithm used to compute $LB_{\mathcal{P}(\mathcal{C})}$:

Algorithm 2: ComputeLB

```
Data   : P(C), X, D
Result : LB
LB ← 0;
foreach x ∈ X do
    LB ← LB + ComputeMinInc(x, D(x), P(x));
end
return LB;
```

Complexity In order to compare the complexity of checking the consistency of a ssc with RMA and with PFC-RDAC (the non incremental version of PFC-MRDAC [2]), we insist on the fact that the principle is the same except for computing the $\#inc(x, P(x))$. Consequently, we compare the procedure ComputeMinInc with the method used in PFC-RDAC for computing $\#inc(x, P(x))$:

Notation 4 *We denote by :*

- *d the maximal domain size: $max_{x \in \mathcal{X}}(|D(x)|)$,*
- *p the maximal size of a set in the var-partition: $max_{P(x) \in \mathcal{P}(\mathcal{C})}(|P(x)|)$,*
- *f the maximal complexity of a filtering algorithm of a constraint C applied on a domain $D(x)$ (useful either to know if a value $a \in D(x)$ is consistent with a constraint C, in this case we call the algorithm with $D(x) = \{a\}$, or to compute $D(x)_C$).*

In PFC-RDAC, a counter $\#inc((x, P(x), a))$ of violations involved by each value a of $D(x)$ is computed. Thus, for each $a \in D(x)$ the complexity is $O(f * |P(x)|)$; and then we iterate over all such counters to determine the minimal one, $\#inc(x, P(x))$. Finally the complexity is $O(|D(x)| * f * |P(x)|)$.

In RMA, the filtering algorithm of each constraint is called only once. That is, $|P(x)| * f$. We sort the $|P(x)| * 2 + 2$ bounds to compute $\mathcal{I}(B(P(x)))$: $O(|P(x)| * log(|P(x)|))$; and we iterate on them to determine $\#inc(x, P(x))$. Finally the complexity is $O(|P(x)| * log(|P(x)|) + f * |P(x)|)$.

To obtain LB, these computations will be performed at most n times (n is the number of variables). Since $\sum_{x_i \in X(\mathcal{C})}(|P(x_i)|) = |\mathcal{C}|$ we have:

	PFC-RDAC	RMA
$\#inc(x, P(x))$	$O(\|D(x)\| * f * \|P(x)\|)$	$O(\|P(x)\| * log(\|P(x)\|) + f * \|P(x)\|)$
$LB_{\mathcal{P}(\mathcal{C})}$	$O(\|\mathcal{C}\| * d * f)$	$O(\|\mathcal{C}\| * (log(p) + f))$
$LB_{\mathcal{P}(\mathcal{C})}$ when f is negligible	$O(\|\mathcal{C}\| * d)$	$O(\|\mathcal{C}\| * log(p))$

Note that $p \leq |\mathcal{C}|$ (if the partition is uniform we have even $p = |\mathcal{C}|/n$). RMA is interesting when the number of values in domains is important w.r.t. the number of constraints. In other terms, in some cases using PFC-MRDAC is almost forbidden because domain sizes are too big, and then the RMA is a good alternative.

4.4 Filtering Algorithm

In this section we propose to apply the principle described in previous sections to filter domains.

It is important to note that if only range based filtering algorithms are associated with the constraints then it is useless to introduce a filtering algorithms which is able to create "holes" in the domains, because this "holes" will not be used by the other algorithms. However, this can have an interest to the search heuristic which can be based on the number of remaining values in the domains

of the variables. Therefore, we first propose an algorithm which reduce only the bounds.

Following the schema of PFC-MRDAC (see Theorem 2), we will take into account the lower bound corresponding to all sets in the var-partition $\mathcal{P}(\mathcal{C})$ except $P(x)$: $LB_{\mathcal{P}(\mathcal{C}-P(x))}$. As in PFC-MRDAC, viability of values will be evaluated according to $max(D(obj))$:

Theorem 3 *Let $I \in \mathcal{I}(B(P(x)))$. If $\#inc(I, P(x)) + LB_{\mathcal{P}(\mathcal{C}-P(x))} > max(D(obj))$ then all values in I can be removed from $D(x)$.*

proof: I is homogeneously independent thus any value in I violated the same number of constraints. From theorem 2 the theorem holds.

The algorithm is based on this theorem: starting from $I = [min(D(x)), q)$, we remove successive ranges while the condition of Theorem 3 is satisfied; and similarly, from the maximum of $D(x)$: starting from $I = [p, max(D(x)) + 1)$, we remove successive ranges while the condition of Theorem 3 is satisfied.

This algorithm can be slightly modified in order to obtain further domain reductions, that is, to create "holes" in the domain: for each range, Theorem 3 is applied and values are accordingly modified.

These algorithms have the same worst case complexity, and the gain in regards to PFC-MRDAC is similar as for the consistency checking.

Once again, if the filtering algorithms associated with soft constraints are range based algorithms, the second algorithm and PFC-MRDAC will lead to the same domain reduction.

4.5 Incrementality

Until now, we considered that all the computations are done from scratch. In this section, we propose to study the incremental behavior of RMA.

After some modifications of domains of variables, the filtering algorithm associated with one constraint $C \in P(x)$ may reduce the domain of x in a different way than the previous one. In this case, some operations have to be done in order to accordingly update $\#inc(x, P(x))$.

Let us study in detail what append when a constraint leads to a new domain reduction of D(x). Suppose that the previous was $[p, q]$ and the new is $[p', q']$. Necessarily we have $p' \geq p$ and $q' \leq q$. Consider the current set $\mathcal{I}(B(P(x)))$. This set has to be updated.

We will consider the modification due to p'. If $p' = p$ then there is no modification thus we will study only $p' > p$.

Let $I_k = [p, r) \in \mathcal{I}(B(P(x)))$ and $I_l = [s, q + 1) \in \mathcal{I}(B(P(x)))$ Two cases must be studied: $p' \leq r$ and $p' > r$.

• $p' \leq r$: If $nMin(p, B(P(x))) > 1$ then I_k is splitted into two sets: $I_{k_1} = [p, p')$ and $I_{k_2} = [p', r)$ with $\#sat(I_{k_1}) = \#sat(I_k) - 1$ and $\#sat(I_{k_2}) = \#sat(I_k)$. Thus $\#inc(x, P(x))$ can be immediatly updated.
If $nMin(p, B(P(x))) = 1$ then the ranges $I_{k-1} = [o, p)$ and I_k are modified as

follows: $I_{k-1} = [o, p')$ with $\#sat(I_{k-1}) \leftarrow \#sat(I_{k-1}) - 1$ and $I_k = [p', r)$ with $\#sat(I_k) \leftarrow \#sat(I_k)$. Thus $\#inc(x, P(x))$ can be immediatly updated.
• $p' > r$: Consider the range I_l which contains p'. If $nMin(p, B(P(x))) > 1$ then I_k is not changed but $\#sat(I_k) \leftarrow \#sat(I_k) - 1$ and the set I_l is modified as in the previous case. Thus, $\#inc(x, P(x))$ can be immediatly updated.
If $nMin(p, B(P(x))) = 1$ then the ranges $I_{k-1} = [o, p)$ and I_k are merged into one range $[o, r)$ with $\#sat([o, r)) = \#sat(I_k) - 1$ and the set I_l is modified as in the previous case. Thus, $\#inc(x, P(x))$ can be immediatly updated.

The same kind of reasoning can be applied for q' and q.

The complexity of this method depends on the time needed to find the range containing a specific value. If each constraint C of $P(x)$ contains the range starting by $D(x)_C$ and the range ending by $D(x)_C$, then at most $|D(x)|$ operations are required to find the searched range. However, the maximal number of ranges is in $O(|P(x)|)$. In the worst case, and for only one constraint: $O(min(|P(x)|, |D(x)|))$ tests are needed, but for k constraints a more sophisticated algorithm should be designed in order to reduce the global complexity.

Therefore, $\#inc(x, P(x))$ can be efficiently updated in $O(min(|P(x)|, |D(x)|))$ when the filtering algorithm associated with one constraint of $P(x)$ lead to a new domain. This complexity is certainly not optimal and could be improved.

5 Discussion

The method we propose can be used when:

1. domains are not range
2. filtering algorithm associated with constraints are not range based algorithm

5.1 Domains are not range

We can either consider that domains are ranges even if they are not, or directly deal with such domains.

In the first case, this means that we consider a relaxation of the domains and, then, RMA is still valid but it is no longer equivalent to PFC-MRDAC.

In the second case, we consider that a domain is a union of ranges. RMA can be applied, but the complexity of this algorithm will change because each range of a domain has to be taken into account. Thus, the number of ranges in $\mathcal{I}(B(P(x)))$ is accordingly changed and also the complexity of the algorithm. If a domain contains r ranges then the new complexity when only range based filtering algorithms are used becomes:$O(|P(x)+r| * log(|P(x)+r|) + f * |P(x)+r|)$. Therefore, this method is better than PFC-MRDAC if $log(|P(x)| + r) \leq O(|D(x)|)$.

5.2 General filtering algorithms

The application of a filtering algorithm for a constraint may lead to domains which are not ranges. That is, a domain will no longer correspond to one range, but to several ranges. Then, two possibilities can be studied: the "holes" in the domain are ignored or the algorithm will deal with the set of ranges corresponding to each domain. In the first case, RMA is still valid but it is no longer equivalent to PFC-MRDAC. In the second case, RMA is equivalent to PFC-MRDAC, but the complexity is changed. In fact, the number of ranges corresponding to a domain have to be taken into account. The complexity becomes: $O((|P(x)| + \sum r_i) * log(|P(x)| + \sum r_i) + f * (|P(x)| + \sum r_i))$ where r_i is equal to the number of ranges required to represent the domain of x when the filtering algorithm associated with $C_i \in P(x)$ is applied. Roughtly, this method is better than PFC-MRDAC if $log(|P(x)| + \sum r_i) \leq O(|D(x)|)$.

6 Conclusion

In this paper, we propose an alternative to PFC-MRDAC called the Range-based Max-CSP Algorithm (RMA), based on the exploitation of bound-based filtering algorithms of constraints. This technique does not require to study each value of each domain: its complexity depends only on the number of variables and the number of constraints. When the only informations available for a variable x w.r.t. a constraint C are the values of bounds of $D(x)$ when the filtering algorithm of C is applied, the lower bounds provided by PFC-MRDAC and RMA are identical and RMA outperforms PFC-MRDAC algorithm.

Some variations of RMA are also studied in order to deal with general cases: domains which are not ranges, filtering algorithms associated with constraints which are not range based algorithms and so on... For each case we show the advantages and the drawbacks of our new method.

References

1. J. Larrosa, P. Meseguer, and T. Schiex. Maintaining reversible DAC for Max-CSP. *Artificial Intelligence*, 107:149–163, 1999.
2. J. Larrosa, P. Meseguer, T. Schiex, and G. Verfaillie. Reversible DAC and other improvements for solving Max-CSP. *Proceedings AAAI*, pages 347–352, 1998.
3. P. Meseguer, J. Larrosa, and M. Sanchez. Lower bounds for non-binary constraint optimization problems. *Proceedings CP*, pages 317–331, 2001.
4. J.-C. Régin, T. Petit, C. Bessière, and J.-F. Puget. An original constraint based approach for solving over constrained prolems. *Proceedings CP*, pages 543–548, 2000.
5. J.-C. Régin, T. Petit, C. Bessière, and J.-F. Puget. New lower bounds of constraint violations for over constrained prolems. *Proceedings CP*, pages 332–345, 2001.
6. J.-C. Régin, J.-F. Puget, and T. Petit. Representation of soft constraints by hard constraints. *Proceedings JFPLC'02*, 2002.

View publication stats