Partition-Based Lower Bound for Max-CSP *

A. Javier Larrosa (larrosa@lsi.upc.es)
Dep. Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya, Barcelona, Spain.

B. Pedro Meseguer (pedro@iiia.csic.es)
Institut d'Investigació en Intel.ligència Artificial
IIIA-CSIC, Bellaterra, Spain.

Abstract. In this paper we address Max-CSP, a constraint optimization problem typically solved using a branch and bound scheme. It is well known that the efficiency of branch and bound greatly depends on the quality of the available lower bound. Previous approaches aggregate to the lower bound individual contributions of unassigned variables. In this paper, we augment this approach by adding global contributions of disjoint subsets of unassigned variables, which requires a partition of the set of unassigned variables. Using this idea, we introduce the *partition*-based lower bound. It improves previous approaches based on individual contributions in the sense that our method can be *added up* to previous bounds, possibly increasing their value. We demonstrate our method presenting two new algorithms, PFC-PRDAC and PFC-MPRDAC, which are the natural successors of PFC-RDAC and PFC-MRDAC augmented with our approach. We provide experimental evidence for the superiority of the new algorithms on random problems and real instances of weighted over-constrained problems.

Keywords: Constraint Optimization, Max-CSP, Branch and Bound

1. Introduction

When solving a Constraint Satisfaction Problem (CSP), one has to assign values to variables satisfying a set of constraints. In real applications it often happens that problems are over-constrained (namely, problems do not have any assignment that satisfies every constraint). In this situation, it is desirable to find the assignment that *best respects* the global set of constraints under some preference criterion. Under this view, CSPs are minimization problems for which *branch and bound* is a suitable solving strategy. The efficiency of branch and bound-based algorithms greatly depends on its lower bound, which is used to detect deadends and to avoid the exploration of large regions in the search space. This lower bound should be both as large and as cheap to compute as possible.

A previous approach [3, 10, 4, 5] for lower bound computation aggregates two main elements: (i) the global contribution of assigned

* Research supported by the Spanish CICYT project TAP1999-1086-C03-02/03.

variables, and (ii) individual contributions of unassigned variables. Another approach [9] keeps (i) but substitutes (ii) by a global contribution of unassigned variables. In this paper we extend the first approach in the following sense: In addition to aggregating individual contributions of unassigned variables, we also aggregate global contributions of disjoint subsets of unassigned variables. This requires a partition of the set of unassigned variables (thus the name of *partition*-based lower bound). Our approach can be put on top of any lower bound of the first approach. Interestingly, the new contributions possibly increase and never decrease the bound. Because of that, the partition-based lower bound dominates any other lower bound of the first approach.

For the sake of simplicity in the exposition, in this paper we restrict ourselves to Max-CSP [3] where all constraints are considered equally important and the goal is to find the assignment that minimizes the number of violations in an over-constrained problem. However, we want to emphasize that the idea can be easily adapted to more general frameworks such as *valued* CSPs [7] or *semiring-based* CSPs [1]. We present two algorithms based on our approach and experimentally show that they outperform state-of-the-art competitors. Our experiments include random over-constrained CSPs and real instances of weighted CSPs.

This paper is structured as follows. In Section 2 we introduce notation and briefly review previous approaches to Max-CSP lower bound. In Section 3 we introduce our approach, illustrated with examples. In Section 4 we analyze its complexity and we present two new algorithms. Experimental results of these algorithms on two different domains appear in Section 5. Finally, Section 6 contains some conclusions and directions of further work.

2. Preliminaries

A discrete *binary constraint satisfaction problem* (CSP) is defined by a finite set of *variables* $X = \{1, \ldots, n\}$, a set of finite *domains* $\{D_i\}_{i=1}^n$ and a set of *binary constraints* $\{R_{ij}\}$. Each variable i takes values in its corresponding domain D_i. A constraint R_{ij} is a subset of $D_i \times D_j$ which only contains the allowed value pairs for variables i, j. An assignment of values to variables is *complete* if it includes every variable in X, otherwise it is incomplete. A *solution* for a CSP is a complete assignment satisfying every constraint. If the problem is over-constrained, such an assignment does not exist, and it may be of interest to find a complete assignment that best respects all constraints [1, 7]. We consider the Max-CSP problem, for which the solution of an over-constrained CSP is a complete assignment violating the minimum number of constraints [3]. Formally, the solution is given by,

$$\mathrm{argmin}_{a_1,\ldots,a_n}\{\sum_{1\leq i<j\leq n} r_{ij}(a_i,a_j)\}, \quad a_k \in D_k,\ 1\leq k\leq n$$

where,

$$r_{ij}(a_i,a_j) = \begin{cases} 1 & \text{if } \exists R_{ij} \ \wedge (a_i,a_j)\notin R_{ij} \\ 0 & \text{otherwise} \end{cases}$$

The number of variables is n, the maximum cardinality of domains is d and the number of constraints is e. Letters $i, j, k\ldots$ denote variables, $a, b, c\ldots$ denote values, and a pair (i, a) denotes the value a of variable i.

Most exact algorithms for solving Max-CSP follow a *branch and bound* schema. These algorithms perform a depth-first traversal on the search tree defined by the problem, where internal nodes represent incomplete assignments and leaf nodes stand for complete ones. Assigned variables are called *past* (P), while unassigned variables are called *future* (F). The *distance* of a node is the number of constraints violated by its assignment. Namely,

$$distance(P) = \sum_{i,j\in P, i<j} r_{ij}(a_i,a_j)$$

where a_i and a_j are the values assigned to past variables i and j, respectively. At each node, branch and bound computes the *upper bound* (UB) as the distance of the best solution found so far (complete assignment with minimum distance in the explored part of the search tree), and the *lower bound* (LB) as an underestimation of the distance of any leaf node descendant from the current one. When $UB \leq LB$, we know that the current best solution cannot be improved below the current node. In that case, the algorithm prunes all its successors and performs backtracking.

In this paper, we focus on *partial forward checking* algorithm (PFC) [3], a branch and bound based algorithm whose lower bound has been improved repeatedly, as summarized in the sequel. When a variable is assigned, PFC performs lookahead on future variables and its effects are recorded in *inconsistency counts* (IC). The inconsistency count of value a of a future variable i, ic_{ia}, is the number of past variables inconsistent with (i, a). Namely,

$$ic_{ia} = \sum_{j\in P} r_{ij}(a, a_j)$$

where a_j is the value assigned to past variable j. PFC simplest lower bound is $distance(P) + \sum_{i\in F} \min_a(ic_{ia})$. This lower bound is improved including inconsistencies among future variables by the usage of *directed arc-inconsistency counts* (DAC) [10]. Given a static ordering in X, $1, 2, \ldots, n$, the directed arc-inconsistency count of value a of variable i, dac_{ia}, is the number of variables in X which are arc-inconsistent[1] with (i, a) and appear after i in the ordering. Formally,

$$dac_{ia} = \sum_{j>i} \min_{b \in D_j} \{r_{ij}(a, b)\}$$

A new lower bound is $distance(P) + \sum_{i \in F} \min_a(ic_{ia}) + \sum_{i \in F} \min_a(dac_{ia})$ [10], provided that branch and bound assignes variables following the static order. The second and third terms of this expression can be combined to form a better lower bound as $distance(P) + \sum_{i \in F} \min_a(ic_{ia} + dac_{ia})$ in the PFC-DAC algorithm [4].

A new approach uses *graph-based* DAC [5], relaxing the condition of a static ordering in X. Its only requirement is that constraints among future variables must be directed: if $i, j \in F$, R_{ij} is given a direction, for instance from j to i. Arc-inconsistencies of R_{ij} are recorded in the DAC of i. In this way, the same inconsistency cannot be recorded in the DAC of two different variables. Directed constraints among future variables induce a directed constraint graph G^F, where $\textsc{Nodes}(G^F) = F$ and $\textsc{Edges}(G^F) = \{(j, i) | i, j \in F, \text{ direction of } R_{ij} \text{ from } j \text{ to } i\}$. Given a directed constraint graph G^F, the graph-based DAC of value a of variable i, $dac_{ia}(G^F)$, is the number of predecesors of i in G^F which are arc-inconsistent with (i, a). Namely,

$$dac_{ia}(G^F) = \sum_{(j,i) \in \mathrm{Edges}(G^F)} \min_{b \in D_j} \{r_{ij}(a, b)\}$$

The *minimum number of inconsistencies* recorded by variable i, $MNI(i, G^F)$, is as follows,

$$MNI(i, G^F) = \min_{a \in D_i} (ic_{ia} + dac_{ia}(G^F))$$

and the graph-based lower bound (based on the directed graph G^F) is,

$$LB(P, F, G^F) = distance(P) + \sum_{i \in F} MNI(i, G^F)$$

This approach is complemented with the dynamic selection of G^F. Given that any G^F is suitable for DAC computation, a local optimization process looks for a good[2] G^F with respect to current IC values

[1] A variable j is arc-inconsistent with (i, a) when no value of j is compatible with (i, a).

[2] Finding the best subgraph is an NP-hard problem [6].

at each node. Since the only possible change in G^F is reversing the orientation of its edges (by reversing the direction of its constraints), this approach is called *reversible* DAC (RDAC). These features are included in the PFC-RDAC algorithm [5]. To allow for an efficient reversal of $dac_{ia}(G^F)$, the data structure $GivesDac$ is used. If $i, j \in X$, $a \in D_i$, $GivesDac$ is defined as follows,

$$GivesDac(i,a,j) = \begin{cases} true & \text{if } j \text{ is arc-inconsistent with} (i,a) \\ false & \text{otherwise} \end{cases}$$

An important feature of all PFC-based algorithms is their ability to detect and prune infeasible values of future variables, which further reduces the search space. The lower bound associated with value b of future variable j is obtained by substituting in the lower bound the contribution of variable j by the contribution of (j, b). For instance, with graph-based DAC, the lower bound associated with value b of future variable j, $LB(P, F, G^F)_{jb}$ is,

$$LB(P,F,G^F)_{jb} = distance(P) + ic_{jb} + dac_{jb}(G^F) + \sum_{i \in F,\, i \neq j} MNI(i, G^F)$$

When this lower bound reaches UB, value b can be pruned from D_j because it cannot belong to a solution including the current assignment and better than the current best solution.

In algorithm PFC-MRDAC [5], RDACs are updated (i.e., maintained) during search taking into account arc-inconsistencies produced by future value pruning.

3. Partition-Based Lower Bound

Let us suppose the following situation,

1. $i, j \in F$ are two mutually constrained variables, $(i, j) \in \textsc{Edges}(G^F)$,
2. (i, a) and (j, b) are the values with the minimum number of recorded inconsistencies in their respective domains (we assume a strict minimum, that is, any other value has a greater number of recorded inconsistencies),
3. (i, a) and (j, b) are incompatible, but i is not arc-inconsistent with (j, b).

The joint contribution of variables i and j to $LB(P, F, G^F)$ is,

$$MNI(i, G^F) + MNI(j, G^F) = ic_{ia} + dac_{ia}(G^F) + ic_{jb} + dac_{jb}(G^F)$$

which is, simply, the sum of its individual contributions of i and j. Clearly, $LB(P, F, G^F)$ only considers inconsistencies among future variables which are recorded in DAC counters. However, the minimum number of inconsistencies provided by the subset $\{i, j\}$ as a whole is,

$$ic_{ia} + dac_{ia}(G^F) + ic_{jb} + dac_{jb}(G^F) + 1$$

because the constraint R_{ij} is unsatisfied with values a and b. Since this inconsistency is not recorded in $dac_{ia}(G^F)$, it can be added. With other values different from a or b, R_{ij} may be satisfied, but since a and b are strict minima, any other value will increase the number of recorded inconsistencies in at least 1. Therefore, the above expression is the actual contribution of the subset $\{i, j\}$ to the lower bound. This is a simple example for the new lower bound approach presented in this paper: *to include inconsistencies among future variables that are not recorded in DAC counters and that will necessarily occur in the best (lowest cost) case.*

In the previous paragraph we grouped two variables. In general, this can be done with subsets of variables, which can be obtained from a partition among future variables. Let $\mathcal{P}(F) = \{\mathcal{F}_q\}$ be a partition of F ($\mathcal{F}_q \cap \mathcal{F}_{q'} = \emptyset$ if $q \neq q', \bigcup_q \mathcal{F}_q = F$). The minimum number of inconsistencies provided by a subset $\mathcal{F}_q = \{i_1, \ldots, i_p\}$, $MNI(\mathcal{F}_q, G^F)$[3], is defined as follows,

$$MNI(\mathcal{F}_q, G^F) = min_{a_1 \ldots a_p}\{\sum_{j=1}^{p}[ic_{i_j a_j} + dac_{i_j a_j}(G^F)] + cost'((i_1, a_1), \ldots, (i_p, a_p))\}$$

where value a_j corresponds to variable i_j. The expression $cost'((i_1, a_1), \ldots, (i_p, a_p))$ accounts for the number of constraints among variables in $\{i_1, \ldots, i_p\}$, which are violated by the values $a_1, \ldots, a_p$ and this violation is not recorded as a directed arc-inconsistency. It is defined as follows,

$$cost'((i_1, a_1), \ldots, (i_p, a_p)) = \sum_{i_j, i_k \in \{i_1, \ldots, i_p\}} r'_{i_j i_k}(a_j, a_k)$$

where, if $(i_j, i_k) \in \textsc{Edges}(G^F)$, $r'_{i_j i_k}(a_j, a_k)$ is,

$$r'_{i_j i_k}(a_j, a_k) = \begin{cases} 1 & \text{if} (a_j, a_k) \notin R_{i_j i_k} \wedge GivesDac(i_k, a_k, i_j) = false \\ 0 & \text{otherwise} \end{cases}$$

and 0 if $(i_j, i_k) \notin \textsc{Edges}(G^F)$.

[3] Abusing notation, we use the same term MNI to express the minimum number of inconsistencies recorded in a single variable or in a subset of variables.

The new lower bound[4], called *partition-based lower bound* because it depends on the selected partition $\mathcal{P}(F)$, is defined as follows,

$$LB(P, \mathcal{P}(F), G^F) = distance(P) + \sum_{\mathcal{F}_q \in \mathcal{P}(F)} MNI(\mathcal{F}_q, G^F)$$

An example of this new lower bound appears in Figure 3. It is easy to check that $LB(P, \mathcal{P}(F), G^F)$ has the following properties:

PROPERTY 1. *$LB(P, \mathcal{P}(F), G^F)$ is a lower bound.*

Proof. First, no inconsistency is recorded twice. ic_{ia} record inconsistencies between past and future variables, $dac_{ia}(G^F)$ and $cost'((i_1, a_1) \ldots, (i_p, a_p))$ record inconsistencies among future variables, and those recorded in $cost'((i_1, a_1) \ldots, (i_p, a_p))$ cannot be recoded in any $dac_{ia}(G^F)$. So these three quantities always record different inconsistencies and can be added. Second, the contribution $MNI(\mathcal{F}_q, G^F)$ of subset $\mathcal{F}_q$ is, by definition, the minimum number of inconsistencies that will occur in that subset, no matter which values are finally assigned. And third, no repeated inconsistency exists among different subsets. Therefore, $LB(P, \mathcal{P}(F), G^F)$ is a lower bound of the number of inconsistencies of any node descendant from the current node. □

PROPERTY 2. *$LB(P, \mathcal{P}(F), G^F) \geq LB(P, F, G^F)$.*

Proof. If every $cost'((i_1, a_1) \ldots, (i_p, a_p)) = 0$, then $LB(P, \mathcal{P}(F), G^F)$ is equal to $LB(P, F, G^F)$. If some $cost'((i_1, a_1) \ldots, (i_p, a_p)) > 0$, then $LB(P, \mathcal{P}(F), G^F)$ may be greater than or equal to $LB(P, F, G^F)$, but never lower. □

PROPERTY 3. *$MNI(\mathcal{F}_q, G^F) - \sum_{i_j \in \mathcal{F}_q} MNI(\{i_j\}, G^F) \leq \frac{p(p-1)}{2}$*

Proof. The maximum value of $cost'((i_1, a_1), \ldots, (i_p, a_p))$ is $\frac{p(p-1)}{2}$, when all variables in $\mathcal{F}_q$ are mutually constrained and every constraint contributes with one. It may happen that values achieving individual minima of $ic_{i_j a_j} + dac_{i_j a_j}(G^F)$ are the same that minimize the joint contribution of $\{i_1, \ldots, i_p\}$. □

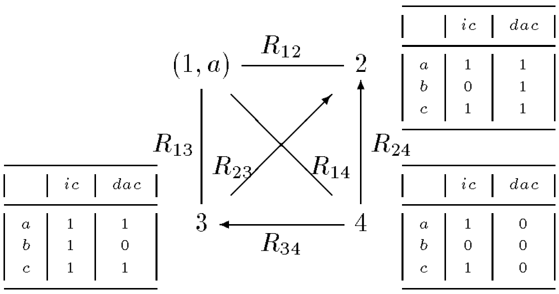
Regarding future value pruning, one can compute the corresponding partition-based lower bound of a future value (j, b) by removing the future variable j from the partition element where it belongs and adding

[4] We use the same term, LB, to denote the previous, graph-based, lower bound and this new, partition-based, lower bound. The second parameter makes the difference (F for the graph-based, $\mathcal{P}(F)$ for the partition-based).

$X = \{1, 2, 3, 4\} \quad D_1 = D_2 = D_3 = D_4 = \{a, b, c\}$

$R_{12} = \{(a, b), (b, a), (c, b)\} \quad R_{13} = \{(b, b), (c, c)\}$
$R_{14} = \{(a, b), (b, c), (c, a)\} \quad R_{23} = \{(a, b), (a, c), (c, b)\}$
$R_{24} = \{(a, b), (b, a), (b, c)\} \quad R_{34} = \{(b, a), (b, c)\}$

$P = \{(1, a)\},\ F = \{2, 3, 4\}$



$MNI(2, G^F) = 1,\ MNI(3, G^F) = 1,\ MNI(4, G^F) = 0;$
$LB(P, F, G^F) = 0 + 1 + 1 + 0 = 2$

$\mathcal{P}_1(F) = \{\{2, 3\}, \{4\}\},\ MNI(\{2, 3\}, G^F) = 2;$
$LB(P, \mathcal{P}_1(F), G^F) = 0 + 2 + 0 = 2$

$\mathcal{P}_2(F) = \{\{2\}, \{3, 4\}\},\ MNI(\{3, 4\}, G^F) = 2;$
$LB(P, \mathcal{P}_2(F), G^F) = 0 + 1 + 2 = 3$

$\mathcal{P}_3(F) = \{\{2, 3, 4\}\}, \quad MNI(\{2, 3, 4\}, G^F) = 3;$
$LB(P, \mathcal{P}_3(F), G^F) = 0 + 3 = 3$

Figure 1. A simple problem and the computation of the $LB(P, F, G^F)$ and $LB(P, \mathcal{P}(F), G^F)$, the later using three different partitions. Constraints among future variables are oriented, each constraint pointing to the variable which records its inconsistencies. The *ic* and *dac* counts of each future variable are shown.

the contribution of (j, b). If $\mathcal{F}_{q'}$ is the subset containing j, it takes the following form,

$$LB(P, \mathcal{P}(F), G^F)_{jb} = distance(P) + MNI(\mathcal{F}_{q'} - \{j\}, G^F) +$$
$$ic_{jb} + dac_{jb}(G^F) + \sum_{\mathcal{F}_q \in \mathcal{P}(F),\, q \neq q'} MNI(\mathcal{F}_q, G^F)$$

4. Practical Usage of Partition-Based Lower Bound

4.1. Computing Partition-Based Lower Bounds

The number of steps required to compute $LB(P, F, G^F)$ is in $O(nd)$ (looking for the minimum among d values of a variable, times at most n future variables). The number of steps required to compute $LB(P, \mathcal{P}(F), G^F)$ is in $O(\sum_{\mathcal{F}_q \in \mathcal{P}(F)} d^{|\mathcal{F}_q|})$ (looking for the minimum among $d^{|\mathcal{F}_q|}$ values of subset $\mathcal{F}_q$, summed over all subsets in $\mathcal{P}(F)$). Then, the new lower bound is exponentially more expensive to compute than the old one, and the exponent depends on the size of the largest subset in $\mathcal{P}(F)$. Although large subsets may cause large increments in the old lower bound (up to quadratic in the size of the largest subset, Property 3), the conditions for each new contribution are so demanding that large increments rarely occur. Therefore, we prefer partitions with small subsets, in which the cost of computing the new lower bound is not prohibitive. In addition, if we restrict to partitions with subsets not greater than two elements, we simplify the computation of $LB(P, \mathcal{P}(F), G^F)$, because the following property.

PROPERTY 4. *Let D_i^{min} and D_j^{min} be the values with the minimum number of recorded inconsistencies for variable i and j, respectively.*

$$\forall a \in D_i^{min},\ b \in D_j^{min},\ r'_{ij}(a,b) = 1 \ if\ and\ only\ if$$
$$MNI(\{i,j\}, G^F) = MNI(\{i\}, G^F) + MNI(\{j\}, G^F) + 1$$

The proof of this property is the generalization to non strict minima in the introductory example of Section 3. This result has an obvious practical interest. Instead of computing $MNI(\{i,j\})$ by evaluating all combinations of values of i and j, only combinations of those values with minimum recorded inconsistencies have to be considered.

4.2. PFC-PRDAC and PFC-MPRDAC

In this subsection, we describe the PFC-PRDAC and PFC-MPRDAC algorithms. They are new versions of PFC-RDAC and PFC-MRDAC algorithms [5], in which the partition-based lower bound has been introduced. For the sake of simplicity, we only give a high level description. The interested reader will find the source code used in our experiments at: `http://www.lsi.upc.es/~larrosa/PFC-MPRDAC`.

In PFC-RDAC each node inherits the directed graph from its parent. After propagating the current assignment, the algorithm goes through a process in which some graph edges are reversed for LB improvement. If

```
function PartitionBasedBound(P, F, G^F)
    NewLB ← LB(P, F, G^F);
    P(F) ← ⋃_{i∈F} {i};
    for each i, j ∈ F, s.t. R_ij
        if (∀a_i ∈ D_i^min, ∀a_j ∈ D_j^min, r'_ij(a_i, a_j) = 1)
            P(F) ← P(F) − {i} − {j} ∪ {i, j};
            NewLB ← NewLB + 1;
    if NewLB ≤ UB
        for each i ∈ F
            for each a ∈ D_i
                if LB(P, P(F), G^F)_ia ≥ UB
                    prune(i, a);
    return(NewLB);
```

Figure 2. Builds a partition of future variables, $\mathcal{P}(F)$, and returns its associated lower bound $LB(P, \mathcal{P}(F), G^F)$.

the new bound detects a deadend, the algorithm backtracks. Otherwise, search continues on the current node successors.

PFC-PRDAC takes over at this point. If LB with the optimised graph cannot produce a deadend by itself, PFC-PRDAC builds a partition on future variables aiming at further LB increment. This is done in the `PartitionBasedBound` function. This function constructs a partition $\mathcal{P}(F)$, computes its associated lower bound $LB(P, \mathcal{P}(F), G^F)$ and prunes future values accordingly. From a partition in which every future variable forms an independent element, it iterates over pairs of future connected variables. For each pair $(i, j) \in \textsc{Edges}(G^F)$, it checks if every value with minimum IC+DAC of variable i is inconsistent with every value with minimum IC+DAC of variable j, and this inconsistency is not recorded in DAC of variable j. If this condition is satisfied, the two variables are merged into a single partition element and the current lower bound is increased by one (Property 4). After this process, if no deadend is detected, all future values go through the corresponding pruning process. It is worth noting that during the partition construction, when two variables are merged, they are not further considered. This ensures that $\mathcal{P}(F)$ is formed by singletons and pairs, only. It is also worth mentioning that pairs of variables are merged upon a greedy criterion (immediate LB increment). Thus, the algorithm does not guarantee that the best partition (*i.e.* the partition producing the highest lower bound) is generated. `PartitionBasedBound` has cost $O(ed^2)$.

PFC-MPRDAC is exactly like PFC-PRDAC except in the following aspect. If `PartitionBasedBound` does not produce a deadend, DAC counters are updated considering values pruned up to this point in the current node. The new recorded inconsistencies may cause lower bound increments, which may render new values unfeasible that must also be propagated.

5. Experimental Results

In this Section we evaluate the practical use of our approach on the benchmarks already used in [5]. In the first experiment, we have tested the performance of our algorithms on over-constrained binary random CSP. A binary random CSP class is characterized by $\langle n, d, p_1, p_2 \rangle$ where n is the number of variables, d the number of values per variable, p_1 the graph *connectivity* defined as the ratio of existing constraints, and p_2 the constraint *tightness* defined as the ratio of forbidden value pairs. The constrained variables and the forbidden value pairs are randomly selected [8]. Using this model, we have experimented on the following problem classes:

1. $\langle 10, 10, 1, p_2 \rangle$,
2. $\langle 15, 5, 1, p_2 \rangle$,
3. $\langle 15, 10, 50/105, p_2 \rangle$,
4. $\langle 20, 5, 100/190, p_2 \rangle$,
5. $\langle 25, 10, 37/300, p_2 \rangle$,
6. $\langle 40, 5, 55/780, p_2 \rangle$.

Observe that (1) and (2) are highly connected problems, (3) and (4) are problems with medium connectivity, and (5) and (6) are sparse problems. For each problem class and each parameter setting, we generated samples of 100 instances.

Each problem is solved by four algorithms: PFC-RDAC and PFC-MRDAC as described in [5], PFC-PRDAC and PFC-MPRDAC as described in Section 4. All four algorithms use *domain size* divided by *forward degree* as dynamic variable ordering. Values are always selected by increasing *ic*+*dac*. All algorithms share code and data structures whenever it is possible. Experiments were performed using a Sun Sparc 2 workstation.

Figure 5 reports the mean number of visited nodes required for solving the six problem classes. It can be seen that PFC-PRDAC and PFC-MPRDAC visit significantly less nodes than PFC-RDAC and PFC-MRDAC. It can be observed that the partition-based lower bound produces greater gains as problems become sparser. Improvement ratios for PFC-PRDAC versus PFC-RDAC range from 2 for the most dense problems to 7 for the most sparse problems, at the left of the peak.

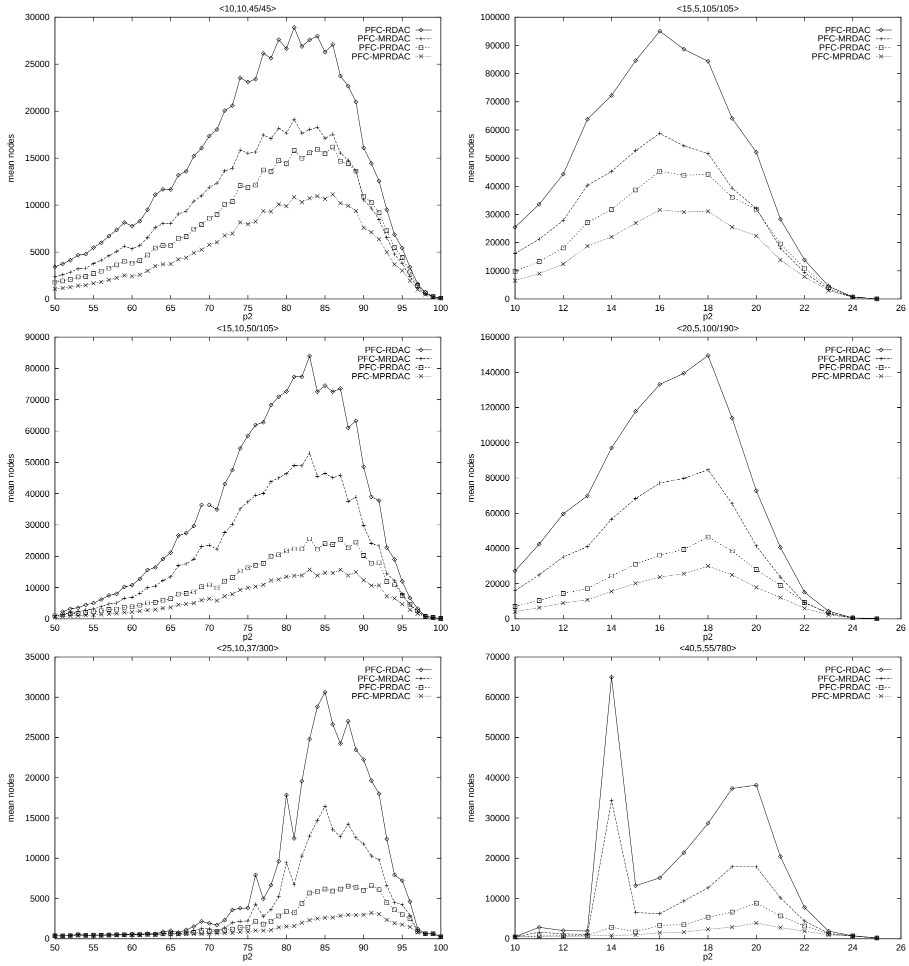


Figure 3. Average visited nodes versus tightness for six classes of binary random problems.

The highest average gain occurs at the $\langle 40, 5, 55/780, 14\rangle$ class where PFC-PRDAC visits 22 times less nodes than PFC-RDAC. A closer look to this class shows that there is a particular problem instance that is very hard for PFC-RDAC and PFC-MRDAC, but it turns out to be easy when these algorithms are enhanced with our approach. Comparing PFC-MPRDAC with PFC-MRDAC, a similar behaviour is observed although the dominance of PFC-MPRDAC is slightly smaller. Regarding PFC-PRDAC versus PFC-MPRDAC, it is clear that the latter dominates.

Figure 5 reports the mean cpu time required to solve the six problem classes. Cpu time results are in agreement with those of visited nodes. The same dominance behaviour with respect problem density is

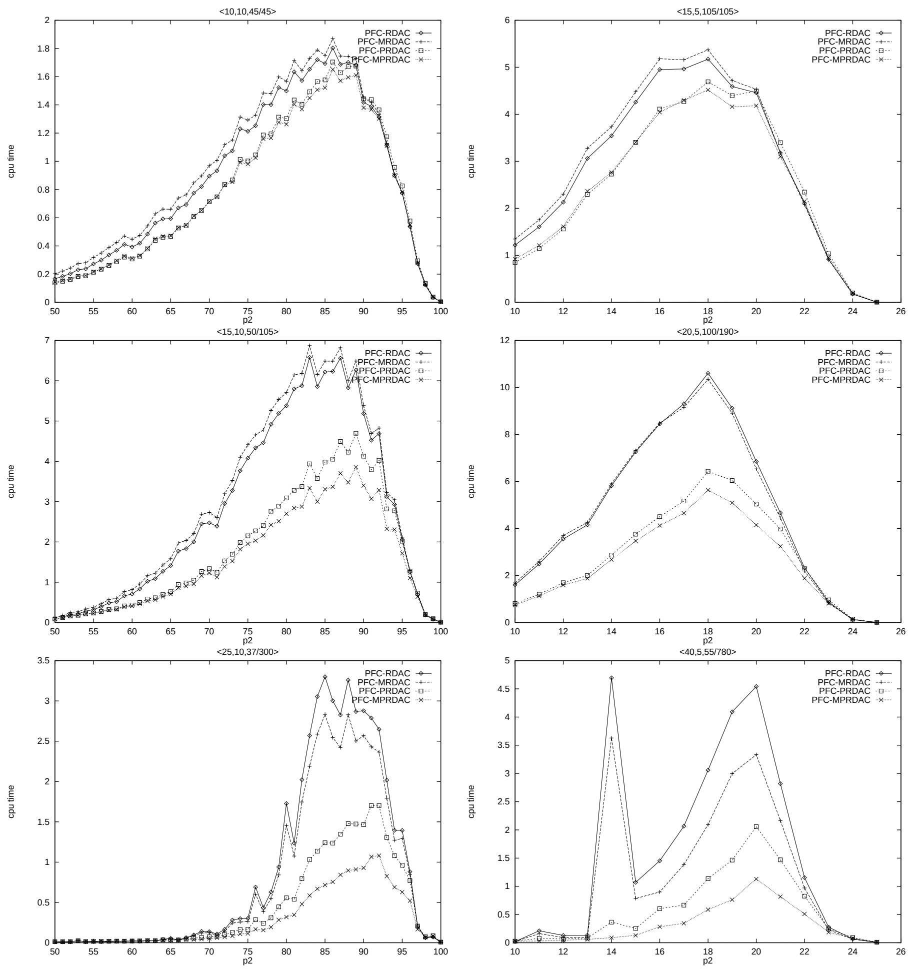


Figure 4. Average CPU versus tightness for six classes of binary random problems.

observed. In addition, it can be observed that the left-side of the complexity peak is the region where the new lower bound is most beneficial. Since computing the partition-based lower bound is time consuming, the dominance ratios are smaller. Typical improvement ratios for PFC-PRDAC versus PFC-RDAC range from 1.2 for dense problems to 3 for sparse problems. In the $\langle 40, 5, 55/780, 14\rangle$ class PFC-PRDAC is 13 times faster than PFC-RDAC. Regarding PFC-PRDAC versus PFC-MPRDAC, the overhead of propagating deletions hardly pays off in dense problem classes, where PFC-MPRDAC is slightly faster than PFC-PRDAC. On medium and sparse classes, PFC-MPRDAC is up to 1.6 times faster than PFC-PRDAC.

Our second experiment considers the Radio Link Frequency Assignment Problem (RLFAP). It is a communication problem where the goal is to assign frequencies to a set of radio links in such a way that all the links may operate together without noticeable interference [2]. Some RLFAP instances can be naturally cast as *weighted* binary Max-CSP where each forbidden tuple has an associated penalty cost. We have extended the use of partition-based lower bound to this framework and tested them on four publicly available RLFAP subinstances called CELAR6-SUB$_i$ ($i = 1, \ldots, 4$) in [2]. Each CELAR6-SUB$_i$ is a subinstance of CELAR6-SUB$_{i+1}$ and is therefore presumably easier to solve. As in the previous experiment, each instance is solved with the same four algorithms. They use the same variable ordering heuristic that was used in [5].

Table I. Cost of solving subinstances 1 to 4 of the CELAR6 problem.

	PFC-RDAC		PFC-PRDAC	
	nodes	cpu time	nodes	cpu time
SUB$_1$	$1.9\ 10^7$	$3.5\ 10^3$	$4.6\ 10^6$	$1.7\ 10^3$
SUB$_2$	$1.2\ 10^8$	$2.9\ 10^4$	$1.9\ 10^7$	$9.5\ 10^3$
SUB$_3$	$2.7\ 10^8$	$7.1\ 10^4$	$3.5\ 10^7$	$2.0\ 10^4$
SUB$_4$	$1.2\ 10^9$	$3.4\ 10^5$	$1.5\ 10^8$	$1.0\ 10^5$
	PFC-MRDAC		PFC-MPRDAC	
	nodes	cpu time	nodes	cpu time
SUB$_1$	$1.8\ 10^6$	$2.3\ 10^3$	$1.4\ 10^6$	$1.7\ 10^3$
SUB$_2$	$1.6\ 10^7$	$2.3\ 10^4$	$9.0\ 10^6$	$1.3\ 10^4$
SUB$_3$	$3.7\ 10^7$	$6.8\ 10^4$	$1.7\ 10^7$	$2.7\ 10^4$
SUB$_4$	$1.2\ 10^8$	$2.6\ 10^5$	$7.6\ 10^7$	$1.3\ 10^5$

Table I shows the cost required to prove optimality (*i.e.* the upper bound is initialized with the optimum cost). We report both number of visited nodes and cpu-time in seconds. We can see that the partition-based lower bound is clearly beneficial in this domain as well. Regarding visited nodes, PFC-MPRDAC is the best algorithm. Comparing it with PFC-MRDAC, it visits nearly half the nodes in the hardest instances. However, the overhead of maintaining DAC updated does not pay off in performance. In terms of cpu time, PFC-PRDAC is the algorithm of choice. It clearly outperforms PFC-MRDAC, which was the best algorithm for this domain reported in [5]. In SUB$_4$, the hardest instance, our approach causes a cpu speed up of 2.6 with respect previous results. It is worth noting that there is a saving of 1.8 CPU days for a single problem instance.

6. Conclusions

The presented approach shows once more that branch and bound efficiency greatly depends on the lower bound quality and that small increments in the lower bound cause large gains in search space reduction. This paper introduces the partition-based lower bound. It was developed in the Max-CSP context but it can be easily extended to to other constraint optimization frameworks. Our empirical work included both Max-CSP and weighted CSPs. We showed that although our approach increases the overhead of the algorithm, the additional work clearly pays off in terms of performance.

Our current implementation exploits partitions in a very limited way: only subset of size two that satisfy a very simple greedy condition are considered. The computation of more sophisticated partitions yielding higher increments in the lower bound remains as future work.

References

1. Bistarelli, S., U. Montanari, and F. Rossi: 1997, 'Semiring-Based Constraint Satisfaction and Optimization'. *Journal of the ACM* **44**(2), 201–236.
2. Cabon, B., S. de Givry, L. Lobjois, T. Schiex, and J. Warners: 1999, 'Radio Link Frequency Assignment'. *Constraints* **4**, 79–89.
3. Freuder, E. and R. Wallace: 1992, 'Partial Constraint Satisfaction'. *Artificial Intelligence* **58**, 21–70.
4. Larrosa, J. and P. Meseguer: 1996, 'Exploiting the use of DAC in Max-CSP'. In: *Proc. of the 2^{th} CP*. Boston (MA), pp. 308–322.
5. Larrosa, J., P. Meseguer, and T. Schiex: 1999, 'Maintaining Reversible DAC for Max-CSP'. *Artificial Intelligence* **107**(1), 149–163.
6. Schiex, T.: 1998, 'Maximizing the reversible DAC lower bound in Max-CSP is NP-hard'. Technical Report 1998/02, INRA.
7. Schiex, T., H. Fargier, and G. Verfaillie: 1995, 'Valued Constraint Satisfaction Problems: hard and easy problems'. In: *Proc. of the 14^{th} IJCAI*. Montréal, Canada, pp. 631–637.
8. Smith, B.: 1994, 'Phase transition and the mushy region in constraint satisfaction'. In: *Proceedings of the 11th European Conference on Artificial Intelligence*. pp. 100–104.
9. Verfaillie, G., M. Lemaître, and T. Schiex: 1996, 'Russian Doll Search'. In: *Proc. of the 13^{th}AAAI*. Portland, OR, pp. 181–187.
10. Wallace, R.: 1995, 'Directed Arc Consistency Preprocessing'. In: M. Meyer (ed.): *Selected papers from the ECAI-94 Workshop on Constraint Processing*, No. 923 in LNCS. Berlin: Springer, pp. 121–137.