Dead-End Elimination for Weighted CSP

Simon de Givry[1,2] ⋆, Steve Prestwich[2], and Barry O'Sullivan[2]

[1] MIA-T, UR 875, INRA, F-31320 Castanet Tolosan, France
[2] Cork Constraint Computation Centre, University College Cork, Ireland
degivry@toulouse.inra.fr,{s.prestwich,b.osullivan}@cs.ucc.ie

Abstract. Soft neighborhood substitutability (SNS) is a powerful technique to automatically detect and prune dominated solutions in combinatorial optimization. Recently, it has been shown in [26] that enforcing partial SNS (PSNS^r) during search can be worthwhile in the context of Weighted Constraint Satisfaction Problems (WCSP). However, for some problems, especially with large domains, PSNS^r is still too costly to enforce due to its worst-case time complexity in $O(ned^4)$ for binary WCSP. We present a simplified dominance breaking constraint, called *restricted dead-end elimination* (DEE^r), the worst-case time complexity of which is in $O(ned^2)$. Dead-end elimination was introduced in the context of computational biology as a preprocessing technique to reduce the search space [13, 14, 16, 17, 28, 30]. Our restriction involves testing only one pair of values per variable instead of all the pairs, with the possibility to prune several values at the same time. We further improve the original dead-end elimination criterion, keeping the same time and space complexity as DEE^r. Our results show that maintaining DEE^r during a depth-first branch and bound (DFBB) search is often faster than maintaining PSNS^r and always faster than or similar to DFBB alone.

Keywords: combinatorial optimization, dominance rule, weighted constraint satisfaction problem, soft neighborhood substitutability

1 Introduction

Pruning by dominance in the context of combinatorial optimization involves reducing the solution space of a problem by adding new constraints to it [19]. We study dominance rules that reduce the domains of variables based on optimality considerations (in relation to the optimization of an objective function). The idea is to automatically detect values in the domain of a variable that are *dominated* by another *dominant* value of the domain such that any solution using the dominant value instead of the dominated ones has a better score. Various dominance rules have been studied recently by the Constraint Programming community [5, 6, 26]. In particular, *soft neighborhood substitutability* (SNS) [3, 26] allows us to detect dominated values in polynomial time under specific conditions for Weighted Constraint Satisfaction Problems (WCSP). In a different community, similar dominance rules and others, called *dead-end elimination (DEE) criteria*,

⋆ This work has been partly funded by the "Agence nationale de la Recherche", reference ANR-10-BLA-0214 and the European Union, reference FP7 *ePolicy* 288147.

have been studied for many years in the context of *computational protein design* [1, 13, 14, 16, 17, 28, 30]. However to the best of our knowledge, these criteria have never been used during search, possibly due to their high computational cost. Following the work done in [26] showing the interest of maintaining such dominance rule during search, we propose a faster pruning by dominance algorithm combining SNS and DEE in a partial and optimistic way.

2 Weighted Constraint Satisfaction Problems

A Weighted Constraint Satisfaction Problem (WCSP) P is a triplet $P = (X, F, k)$ where X is a set of n variables and F a set of e cost functions. Each variable $x \in X$ has a finite domain, $domain(x)$, of values that can be assigned to it. The maximum domain size is denoted by d. For a set of variables $S \subseteq X$, $l(S)$ denotes the set of all labelings of S, *i.e.,* the Cartesian product of the domain of the variables in S. For a given tuple of values t, $t[S]$ denotes the projection of t over S. A cost function $f_S \in F$, with scope $S \subseteq X$, is a function $f_S : l(S) \mapsto [0, k]$ where k is a maximum integer cost used for forbidden assignments. A cost function over one (resp. zero) variable is called a *unary* (resp. *nullary*, *i.e.,* a constant cost payed by any assignment) cost function, denoted either by $f_{\{x\}}$ or f_x (resp. by $f_\varnothing$). We denote by $\Gamma(x)$ the set of cost functions on variable x, *i.e.,* $\Gamma(x) = \{f_S \in F | \{x\} \subseteq S\}$.

The Weighted Constraint Satisfaction Problem consists in finding a complete assignment t minimizing the combined (*sum*) cost function $\sum_{f_S \in F} f_S(t[S])$. This optimization problem has an associated NP-complete decision problem.

Enforcing a given local consistency property on a problem P involves transforming $P = (X, F, k)$ into a problem $P' = (X, F', k)$ that is equivalent to P (all complete assignments keep the same cost) and that satisfies the considered local consistency property. This enforcing may increase $f_\varnothing$ and provide an improved lower bound on the optimal cost. It is achieved using *Equivalence Preserving Transformations* (EPTs) that move costs between different scopes [8–12, 21, 22, 24, 31]. In particular, node consistency [21] (NC) satisfies $\forall x \in X, \min_{a \in domain(x)} f_x(a) = 0, \forall a \in domain(x), f_\varnothing + f_x(a) < k$. Soft arc consistency (AC*) [21, 31] satisfies NC and $\forall f_S \in F, \forall x \in S, \forall a \in domain(x)$, $\min_{t \in l(S \setminus \{x\})} f_S(t \cup \{(x, a)\}) = 0$.

3 Dead-End Elimination

The original dead-end elimination criterion is [14]:

$$\sum_{f_S \in \Gamma(x)} \max_{t \in l(S \setminus \{x\})} f_S(t \cup \{(x, a)\}) \leq \sum_{f_S \in \Gamma(x)} \min_{t \in l(S \setminus \{x\})} f_S(t \cup \{(x, b)\}). \quad (1)$$

This condition implies that value b can be safely removed from the domain of x since the total cost of all the cost functions on x taking their best assignment with x assigned b is still worse than that produced by their worst assignment with x assigned a. This condition was further improved in [17]:

$$\sum_{f_S \in \Gamma(x)} \max_{t \in l(S \setminus \{x\})} f_S(t \cup \{(x,a)\}) - f_S(t \cup \{(x,b)\}) \leq 0. \quad (2)$$

where the best and worst-cases are replaced by the worst difference in costs for any labeling of the remaining variables in the scope of each cost function. It is easy to see that this condition is always stronger than the previous one.

More recently, the authors in [26] reformulated Equation 2 in the specific context of WCSP with bounded cost addition $a \oplus b = \min(k, a+b)$ proving that the reformulated criterion[3] is equivalent to soft neighborhood substitutability when $\Gamma(x)$ is separable (*i.e.,* $\forall f_S, f_{S'} \in \Gamma(x) \times \Gamma(x), S \cap S' = \{x\}$) and $\alpha < k$. In practice, testing Eq. 2 or its reformulation will prune the same values.

They also noticed that if the problem is soft arc consistent then the worst-cost differences are always positive. Equation 1 can be further simplified thanks to soft AC because all the best-case terms are precisely equal to zero:

$$\sum_{f_S \in \Gamma(x)} \max_{t \in l(S \setminus \{x\})} f_S(t \cup \{(x,a)\}) \leq f_x(b). \quad (3)$$

We propose a stronger condition than Eq. 2 or Eq. 3 by discarding forbidden partial assignments with x assigned b when computing the worst-cost difference:

$$\sum_{f_S \in \Gamma(x)} \max_{t \in l(S \setminus \{x\}) \text{ st. } C(f_S, t \cup \{(x,b)\}) < k} f_S(t \cup \{(x,a)\}) - f_S(t \cup \{(x,b)\}) \leq 0. \quad (4)$$

where $C(f_S, t) = f_s(t) + \sum_{y \in S, |S|>1} f_y(t[y]) + f_\varnothing$. This new condition is equivalent to Eq. 2 except that some tuples have been discarded from the max operation. These discarded tuples t are forbidden partial assignments when x is assigned b because the sum of the associated cost function $f_S(t \cup \{(x,b)\})$ plus, if $|S| > 1$, all the unary costs on the variables in S assigned by $t \cup \{(x,b)\}$ plus the current lower bound $f_\varnothing$ is greater than or equal to the current upper bound k. Such tuples t do not need to be considered by the max operation because $t \cup \{(x,b)\}$ does not belong to any optimal solution, whereas $t \cup \{(x,a)\}$ can be.

For CSP (*i.e.,* $k = 1$), Eq. 2 and Eq. 4 are both equivalent to neighborhood substitutability [15]. For Max-SAT, Eq. 3 and Eq. 2 are equivalent if the problem is soft AC, and correspond to the *Dominating 1-clause rule* [29]. In the general case, Eq. 4 is stronger[4] (more domain values can be pruned) than Eq. 2, which is stronger than Eq. 3. More complex dominance criteria have been defined in the context of protein design (*e.g.,* a value being dominated by a set of values instead of a single one, see [30] for an overview), but they all incur higher computational costs. In the next section, we recall how to enforce Eq. 2 in WCSP, as originally

[3] They replace the maximum of cost differences $\alpha - \beta$ by the opposite of the minimum of cost pairs (β, α), ordered by the relation $(\beta, \alpha) \leq (\beta', \alpha') \equiv \beta - \alpha < \beta' - \alpha' \vee (\beta - \alpha = \beta' - \alpha' \wedge \alpha < \alpha')$. Equation 2 becomes $\sum_{f_S \in \Gamma(x) \cup f_x} \min_{t \in l(S \setminus \{x\})} (f_S(t \cup \{(x,b)\}), f_S(t \cup \{(x,a)\})) \geq 0$ where $(\beta, \alpha) \geq 0$ if $\beta \geq \alpha$.

[4] The definition of soft AC on fair VCSPs [12] makes Eq. 4 and Eq. 2 equivalent.

shown in [26]. Then, in Section 5, we present a modified version to partially enforce the two conditions, Eq. 4 and 3, with a lower time complexity.

4 Enforcing Soft Neighborhood Substitutability

Assuming a soft arc consistent WCSP (see *e.g.,* W-AC*2001 algorithm in [24]), enforcing partial[5] soft neighborhood substitutability (PSNS^r) is described by Algorithm 1. For each variable x, all the pairs of values $(a, b) \in domain(x) \times domain(x)$ with $a < b$ are checked by the function DominanceCheck to see if b is dominated by a or, if not, vice versa (line 3). At most one dominated value is added to the value removal queue Δ at each inner loop iteration (line 2). Removing dominated values (line 4) can make the problem arc inconsistent, requiring us to enforce soft arc consistency again. We successively enforce soft AC and PSNS^r until no value removals are made by both enforcing algorithms.

Function DominanceCheck$(x, a \to b)$ computes the sum of worst-cost differences as defined by Equation 2 and returns a non-empty set containing value b if Eq. 2 is true, meaning that b is dominated by value a. It exploits early breaks as soon as Eq. 2 can be falsified (lines 5 and 6). Worst-cost differences are computed by the function getDifference$(f_s, x, a \to b)$ applied to every cost function related to x. Worst-cost differences are always positive (line 7) due to soft AC.

The worst-case time complexity of getDifference is $O(d^{r-1})$ for WCSP with maximum arity r. DominanceCheck is $O(qd^{r-1})$ where $q = |\Gamma(x)|$. Thus, the time complexity of one iteration of Algorithm 1 (PSNS^r) is $O(nd^2qd^{r-1} + nd) = O(ed^{r+1})$ where $e = nq$. Interleaving PSNS^r and soft AC until a fixed point is reached is done at most nd times, resulting in a worst-case time complexity of PSNS^r in $O(ned^{r+2})$. Its space complexity is $O(nd^2)$ when using *residues* [26].

In the following, we always consider PSNS^r using the better condition given by Equation 4 instead of Eq. 2. This does not change the previous complexities.

5 Enforcing Partial SNS and Dead-End Elimination

In order to reduce the time (and space) complexity of pruning by dominance, we test only one pair of values per variable. The new algorithm is described in Algorithm 2. We select the pair $(a, b) \in domain(x) \times domain(x)$ in an optimistic way such that a is associated with the minimum unary cost and b to the maximum unary cost (lines 8 and 9). Because arc consistency also implies node consistency, we always have $f_x(a) = 0$.[6] When all the unary costs (including the maximum) are null (line 10), we select as b the maximum domain value (or its minimum if this value is already used by a). By doing so, we should favor more pruning on max-closed or submodular subproblems[7].

[5] Enforcing complete soft neighborhood substitutability is co-NP hard as soon as $k \neq +\infty$ (*i.e.,* no restriction on α in the reformulated Equation 2).

[6] In fact, we set the value a to the *unary support* offered by NC [21] or EDAC [22].

[7] Assuming a problem with two variables x and y having the same domain and a single submodular cost function $f(x, y) = 0$ if $x \leq y$ else $x - y$ or a single max-closed constraint $x < y$, then DEE^r assigns $x = min(domain(x))$ and $y = max(domain(y))$.

Algorithm 1: Enforce PSNS^r [26]

```
  Procedure PSNS^r(P: AC* consistent WCSP)
    Δ := ∅ ;
1   foreach x ∈ variables(P) do
2     foreach (a, b) ∈ domain(x) × domain(x) such that a < b do
        R := DominanceCheck(x, a → b) ;
3       if R = ∅ then R := DominanceCheck(x, b → a) ;
        Δ := Δ ∪ R ;
4   foreach (x, a) ∈ Δ do  remove (x, a) from domain(x) ;

  /* Check if value a dominates value b */
  Function DominanceCheck(x, a → b): set of dominated values
5   if f_x(a) > f_x(b) then return ∅ ;
    δ_{a→b} := f_x(a) ;
    foreach f_s ∈ F such that {x} ⊂ S do
      δ := getDifference(f_s, x, a → b) ;
      δ_{a→b} := δ_{a→b} + δ ;
6     if δ_{a→b} > f_x(b) then return ∅ ;
    return {(x, b)} /* δ_{a→b} ≤ f_x(b) */ ;

  /* Compute largest difference in costs when using a instead of b */
  Function getDifference(f_s, x, a → b): cost
7   δ_{a→b} := 0 ;
    foreach t ∈ l(S \ {x}) do
      δ_{a→b} := max(δ_{a→b}, f_s(t ∪ {(x, a)}) − f_s(t ∪ {(x, b)})) ;
    return δ_{a→b} ;
```

Instead of checking the new Equation 4 for the pair (a, b) alone, we also check Eq. 3 for all the pairs (a, u) such that $u \in domain(x) \setminus \{a\}$. This is done in the function MultipleDominanceCheck (lines 16 and 17). This function computes at the same time the sum of maximum costs ub_a for value a (lines 12 and 13) and the sum of worst-cost differences $\delta_{a\to b}$ for the pair (a, b). The new function getDifference-Maximum$(f_s, x, a \to b)$ now returns the worst-cost difference, discarding forbidden assignments with $t \cup \{(x, b)\}$ (line 18), as suggested by Eq. 4, and also the maximum cost in f_S for x assigned a. By construction of the two criteria, we have $\delta_{a\to b} \le ub_a$, so the stopping condition is unchanged at line 14. When the maximum cost of a value is null for all its cost functions, we can directly remove all the other values in the domain avoiding any extra work (line 15). Finally, if the selected pair (a, b) *prunes* b, then a new pair is checked.

Notice that DEE^r is equivalent to PSNS^r on problems with Boolean variables, such as Weighted Max-SAT. For problems with non-Boolean domains, DEE^r is still able to detect and prune several values per variable. Clearly, its time (resp. space) complexity is $O(ned^r)$ (resp. $O(n)$ using only one residue per variable), reducing by a factor d^2 the time and space complexity compared to PSNS^r.

6 Experimental Results

We implemented PSNS^r and DEE^r in `toulbar2`[8]. All methods use residues and variable queues with timestamps as in [26]. PSNS^r uses MultipleDominanceCheck

[8] C++ solver version 0.9.6 `mulcyber.toulouse.inra.fr/projects/toulbar2/`

Algorithm 2: Enforce DEE^r

Procedure DEE^r(P: AC* consistent WCSP)
$\Delta := \emptyset$;
foreach $x \in variables(P)$ **do**
8 $a := \arg\min_{u \in domain(x)} f_x(u)$;
9 $b := \arg\max_{u \in domain(x)} f_x(u)$;
10 **if** $a = b$ /* $\forall u \in domain(x), f_x(u) = 0$ */ **then**
if $a = max(domain(x))$ **then**
$b := min(domain(x))$;
else
$b := max(domain(x))$;
$R :=$ MultipleDominanceCheck$(x, a \rightarrow b)$;
11 **if** $R = \emptyset$ **then** $R :=$ MultipleDominanceCheck$(x, b \rightarrow a)$;
$\Delta := \Delta \cup R$;
foreach $(x, a) \in \Delta$ **do** remove (x, a) from $domain(x)$;

/* Check if value a dominates value b and possibly other values */
Function MultipleDominanceCheck$(x, a \rightarrow b)$: set of dominated values
if $f_x(a) > f_x(b)$ **then return** $\emptyset$;
$\delta_{a \rightarrow b} := f_x(a)$;
12 $ub_a := f_x(a)$;
foreach $f_s \in F$ such that $\{x\} \subset S$ **do**
$(\delta, ub) :=$ getDifference-Maximum$(f_s, x, a \rightarrow b)$;
$\delta_{a \rightarrow b} := \delta_{a \rightarrow b} + \delta$;
13 $ub_a := ub_a + ub$;
14 **if** $\delta_{a \rightarrow b} > f_x(b)$ **then return** $\emptyset$;
15 **if** $ub_a = 0$ **then return** $\{(x, u) | u \in domain(x)\} \setminus \{(x, a)\}$;
$R := \{(x, b)\}$ /* $\delta_{a \rightarrow b} \leq f_x(b)$ */ ;
16 **foreach** $u \in domain(x)$ such that $u \neq a$ **do**
17 **if** $(f_x(u) \geq ub_a)$ **then** $R := R \cup \{(x, u)\}$;
return R ;

/* Compute largest cost difference and maximum cost for value */
Function getDifference-Maximum$(f_s, x, a \rightarrow b)$: pair of costs
$\delta_{a \rightarrow b} := 0$;
$ub_a := 0$;
foreach $t \in l(S \setminus \{x\})$ **do**
18 **if** $f_s(t \cup \{(x, b)\}) + f_\varnothing + f_x(b) + \sum_{y \in S \setminus \{x\}} f_y(t[y]) < k$ **then**
$\delta_{a \rightarrow b} := \max(\delta_{a \rightarrow b}, f_s(t \cup \{(x, a)\}) - f_s(t \cup \{(x, b)\}))$;
$ub_a := \max(ub_a, f_s(t \cup \{(x, a)\}))$;
return $(\delta_{a \rightarrow b}, ub_a)$ /* $\delta_{a \rightarrow b} \leq ub_a$ */ ;

and getDifference-Maximum instead of DominanceCheck and getDifference. MultipleDominanceCheck prunes the dominated values directly instead of queuing them into R. It speeds-up further dominance checks without assuming soft AC anymore during the process (soft AC being restored at the next iteration until a fixed point is reached for AC and SNS/DEE). We compared PSNS^r and DEE^r on a collection of binary WCSP benchmarks (`http://costfunction.org`) (except for *spot5* using ternary cost functions). The *celar* [4] ($n \leq 458, d \leq 44$) and *computational protein design* [1] ($n \leq 55, d \leq 148$) have been selected as they offer good opportunities for neighborhood substitutability, at least in preprocessing as shown in [14, 20]. We added Max SAT *combinatorial auctions* using the CATS generator [27] with 60 goods and a varied number of bids from 70 to 200 (100 to 230 for *regions*) [23]. Other benchmarks were selected by [26] and include: *DIMACS graph coloring* (minimizing edge violations) ($n \leq 450, d \leq 9$), *optimal* *planning* [7] ($n \leq 1433, d \leq 51$), *spot5* ($n \leq 1057, d = 4$) [2], and *uncapacitated warehouse location* [22] ($n \leq 1100, d \leq 300$). Experiments were performed on a cluster of AMD Opteron 2.3 GHz under Linux.

In Table 1, we compared a Depth First Branch and Bound algorithm using EDAC [22] alone (*EDAC* column), EDAC and DEE^r (*EDAC+DEE*^{r}), EDAC and PSNS^r in preprocessing only (*EDAC+PSNS*$^r_{pre}$), EDAC and PSNS^r in preprocessing and DEE^r during search (*EDAC+PSNS*$^r_{pre}$*+DEE*^{r}), EDAC and PSNS^r (*EDAC+PSNS*^{r}), and no initial upper bound for all. For each benchmark, we report the number of instances, and for each method, the number of instances optimally solved in less than 1,200 seconds. In parentheses, average CPU time over the solved instances (in seconds), average number of nodes, and average number of value removals per search node are reported where appropriate. First, we used a static lexicographic variable ordering and a binary branching scheme (`toulbar2` options *-nopre -svo -d:*). DEE^r solved always a greater or equal number of instances compared to EDAC alone, and it performed better than PSNS^r on *celar*, *planning*, *protein*, and *warehouse* benchmarks, all having large domains. We also give the results, when available, in terms of the number of solved instances by PSNS^r over the total number of instances solved by at least one method as reported in [26], showing the good performance of our approach. They used the same settings except a cluster of Xeon 3.0 GHz and *max degree* static variable ordering (only identical to our lexicographic ordering for *warehouse*). In addition, we solved the *celar7-sub1* instance with the same *max degree* ordering: *EDAC+DEE*^{r} solved in (7.7 seconds, 57,584 nodes, 0.96 removals per node), and *EDAC+PSNS*^{r} in (69.5, 39,346, 7.2), or (86.4, 70,896, 6) as reported in [26]. Secondly, we used a dynamic variable ordering combining Weighted Degree with Last Conflict [25] and an initial Limited Discrepancy Search (LDS) phase [18] with a maximum discrepancy of 2 (option *-l=2*, except for *protein* using also *-sortd -d:* as in [1]). This greatly improved the results for all the methods and benchmarks except for *warehouse* where LDS slowed down the methods. DEE^r remained the best method in terms of the number of solved instances; PSNS^r in preprocessing and DEE^r during search being a good alternative, especially on the *protein* benchmark. We compared a subset of our results with the last Max SAT 2012 evaluation (`http://maxsat.ia.udl.cat:81/12`). With roughly the same computation time limit (20 min. with 2.3 GHz instead of 30 min. with AMD Opteron 1.5 GHz), for *auction/paths* and *auction/scheduling*, DEE^r solved 85+82 instances among 170, being in 3rd position among 11 Max SAT solvers.

7 Conclusion

We have presented a lightweight algorithm for automatically exploiting a dead-end elimination dominance criterion for WCSPs. Experimental results show that it can lead to significant reductions in search space and run-time on several benchmarks. In future work, we plan to study such dominance criteria applied during search in integer linear programming.

Acknowledgements. We thank the Genotoul Bioinformatic platform for the cluster and Seydou Traoré, Isabelle André, and Sophie Barbe for the protein instances.

Table 1. For each method, number of instances optimally solved in less than 1,200 seconds, and in parentheses, average CPU time (in seconds) over the solved instances, average number of search nodes, and average number of value removals per node where appropriate.

	#inst.	EDAC	EDAC+DEE^r	EDAC+PSNS$^r_{pre}$	EDAC+PSNS$^r_{pre}$+DEE^r	EDAC+PSNS^r	[26]
Depth First Branch and Bound with static variable ordering							
celar	46	**24 (180.6, 954K)**	24 (187.2, 877K, 0.80)	24 (188.4, 945K)	24 (187.7, 877K, 0.77)	17 (168.0, 100K, 8.33)	12/16
coloring	40	19 (47.7, 2.4M)	19 (45.7, 2.2M, 0.08)	19 (46.9, 2.3M)	19 (45.6, 2.2M, 0.08)	**20 (103.5, 3.7M, 0.96)**	8/8
planning	76	68 (9.8, 39K)	75 (7.2, 32K, 4.46)	69 (18.3, 127K)	**75 (6.9, 32K, 4.46)**	75 (10.5, 31K, 5.27)	27/27
protein	12	9 (34.4, 70K)	9 (30.9, 42K, 1.50)	9 (26.0, 50K)	**9 (25.7, 40K, 1.32)**	9 (139.0, 31K, 4.37)	
spot5	24	4 (0.1, 68)	7 (93.7, 2.7M, 0.42)	6 (172.7, 3.7M)	7 (93.2, 2.7M, 0.39)	**7 (87.0, 2.5M, 0.42)**	3/3
warehouse	55	**46 (55.6, 709)**	46 (66.1, 542, 34.34)	46 (61.3, 688)	46 (58.6, 542, 34.73)	45 (56.3, 429, 75.00)	29/34
auction/paths	420	138 (225.4, 5.9M)	**148 (212.8, 5.2M, 0.06)**	138 (223.5, 5.9M)	148 (213.7, 5.2M, 0.06)	148 (214.0, 5.2M, 0.06)	
auct./regions	420	364 (137.5, 3.3M)	404 (98.1, 1.9M, 0.03)	373 (131.2, 3.2M)	403 (94.3, 1.9M, 0.03)	**405 (100.2, 2M, 0.03)**	
a./scheduling	420	**392 (115.3, 2.3M)**	392 (118.4, 2.3M, 0.00)	392 (113.3, 2.3M)	391 (115.6, 2.2M, 0.00)	390 (114.1, 2.2M, 0.00)	
total	1513	1064	**1124**	1076	1122	1116	
Depth First Branch and Bound with dynamic variable ordering and initial LDS with maximum discrepancy of 2							
celar	46	40 (22.7, 45K)	40 (24.5, 43K, 1.90)	**40 (19.8, 40K)**	40 (24.9, 38K, 1.80)	38 (114.0, 25K, 10.64)	
coloring	40	23 (6.6, 167K)	24 (39.4, 484K, 0.86)	23 (6.7, 167K)	24 (38.9, 484K, 0.86)	**24 (9.1, 162K, 1.19)**	
planning	76	76 (1.3, 1.5K)	76 (1.2, 1.4K, 3.05)	**76 (0.8, 1.1K)**	76 (1.3, 1.5K, 3.02)	76 (1.2, 1.3K, 3.34)	
protein	12	9 (10.1, 7.7K)	9 (10.5, 8K, 1.77)	9 (9.0, 10K)	**9 (8.5, 8K, 1.33)**	9 (55.0, 11K, 5.67)	
spot5	24	8 (21.7, 669K)	8 (14.1, 418K, 0.13)	8 (27.1, 841K)	8 (16.2, 483K, 0.14)	**8 (12.3, 350K, 0.19)**	
warehouse	55	**45 (67.1, 957)**	43 (30.7, 630, 17.87)	45 (70.8, 949)	43 (30.2, 618, 18.65)	42 (8.7, 411, 31.45)	
auction/paths	420	345 (139.0, 2.5M)	**356 (137.4, 2.4M, 0.16)**	346 (138.5, 2.5M)	356 (137.6, 2.4M, 0.16)	355 (139.0, 2.4M, 0.16)	
auct./regions	420	**420 (2.5, 27K)**	**420 (2.5, 27K, 0.03)**	**420 (2.5, 27K)**	**420 (2.5, 27K, 0.03)**	**420 (2.5, 27K, 0.03)**	
a./scheduling	420	**413 (54.8, 1.5M)**	413 (57.8, 1.5M, 0.00)	413 (55.5, 1.5M)	413 (57.8, 1.5M, 0.00)	413 (57.8, 1.5M, 0.00)	
total	1513	1379	**1389**	1380	**1389**	1385	

References

1. Allouche, D., Traoré, S., André, I., de Givry, S., Katsirelos, G., Barbe, S., Schiex, T.: Computational protein design as a cost function network optimization problem. In: Proc. of CP-12. Quebec City, Canada (2012)
2. Bensana, E., Lemaître, M., Verfaillie, G.: Earth observation satellite management. Constraints 4(3), 293–299 (1999)
3. Bistarelli, S., Faltings, B., Neagu, N.: Interchangeability in Soft CSPs. In: International Workshop on Constraint Solving and Constraint Logic Programming. pp. 31–46 (2002)
4. Cabon, B., de Givry, S., Lobjois, L., Schiex, T., Warners, J.: Radio link frequency assignment. Constraints Journal 4, 79–89 (1999)
5. Chu, G., Banda, M., Stuckey, P.: Exploiting subproblem dominance in constraint programming. Constraints 17(1), 1–38 (2012)
6. Chu, G., Stuckey, P.: A generic method for identifying and exploiting dominance relations. In: Proc. of CP'12. pp. 6–22. Quebec, Canada (2012)
7. Cooper, M., Cussat-Blanc, S., Roquemaurel, M.D., Régnier, P.: Soft arc consistency applied to optimal planning. In: Proc. of CP'06. pp. 680–684. No. 4204 in LNCS, Springer, Nantes, France (2006)
8. Cooper, M., de Givry, S., Sanchez, M., Schiex, T., Zytnicki, M., Werner, T.: Soft arc consistency revisited. Artificial Intelligence 174, 449–478 (2010)
9. Cooper, M., de Givry, S., Sanchez, M., Schiex, T., Zytnicki, M.: Virtual arc consistency for weighted CSP. In: Proc. of AAAI-08. Chicago, IL (2008)
10. Cooper, M.C.: High-order consistency in Valued Constraint Satisfaction. Constraints 10, 283–305 (2005)
11. Cooper, M.C., de Givry, S., Schiex, T.: Optimal soft arc consistency. In: Proc. of IJCAI'2007. pp. 68–73. Hyderabad, India (Jan 2007)
12. Cooper, M.C., Schiex, T.: Arc consistency for soft constraints. Artificial Intelligence 154(1-2), 199–227 (2004)
13. Dahiyat, B., Mayo, S.: Protein design automation. Protein Science 5(5), 895–903 (1996)
14. Desmet, J., Maeyer, M., Hazes, B., Lasters, I.: The dead-end elimination theorem and its use in protein side-chain positioning. Nature 356(6369), 539–542 (1992)
15. Freuder, E.C.: Eliminating interchangeable values in constraint satisfaction problems. In: Proc. of AAAI'91. pp. 227–233. Anaheim, CA (1991)
16. Georgiev, I., Lilien, R., Donald, B.: Improved pruning algorithms and divide-and-conquer strategies for dead-end elimination, with application to protein design. Bioinformatics 22(14), e174–e183 (2006)
17. Goldstein, R.: Efficient rotamer elimination applied to protein side-chains and related spin glasses. Biophysical Journal 66(5), 1335–1340 (1994)
18. Harvey, W.D., Ginsberg, M.L.: Limited discrepency search. In: Proc. of the 14^{th} IJCAI. Montréal, Canada (1995)
19. Jouglet, A., Carlier, J.: Dominance rules in combinatorial optimization problems. European Journal of Operational Research 212(3), 433–444 (2011)
20. Koster, A.M.C.A.: Frequency assignment: Models and Algorithms. Ph.D. thesis, University of Maastricht, The Netherlands (Nov 1999), available at www.zib.de/koster/thesis.html.
21. Larrosa, J.: On arc and node consistency in weighted CSP. In: Proc. AAAI'02. pp. 48–53. Edmondton, (CA) (2002)

22. Larrosa, J., de Givry, S., Heras, F., Zytnicki, M.: Existential arc consistency: getting closer to full arc consistency in weighted CSPs. In: Proc. of the 19^{th} IJCAI. pp. 84–89. Edinburgh, Scotland (Aug 2005)
23. Larrosa, J., Heras, F., de Givry, S.: A logical approach to efficient max-sat solving. Artif. Intell. 172(2-3), 204–233 (2008)
24. Larrosa, J., Schiex, T.: Solving weighted CSP by maintaining arc consistency. Artif. Intell. 159(1-2), 1–26 (2004)
25. Lecoutre, C., Saïs, L., Tabary, S., Vidal, V.: Reasoning from last conflict(s) in constraint programming. Artificial Intelligence 173, 1592,1614 (2009)
26. Lecoutre, C., Roussel, O., Dehani, D.: WCSP Integration of Soft Neighborhood Substitutability. In: Proc. of CP'12. pp. 406–421. Quebec, Canada (2012)
27. Leyton-Brown, K., Pearson, M., Shoham, Y.: Towards a Universal Test Suite for Combinatorial Auction Algorithms. ACM E-Commerce pp. 66–76 (2000)
28. Looger, L., Hellinga, H.: Generalized dead-end elimination algorithms make large-scale protein side-chain structure prediction tractable: implications for protein design and structural genomics. Journal of molecular biology 307(1), 429–445 (2001)
29. Niedermeier, R., Rossmanith, P.: New upper bounds for maximum satisfiability. J. Algorithms 36(1), 63–88 (2000)
30. Pierce, N., Spriet, J., Desmet, J., Mayo, S.: Conformational splitting: A more powerful criterion for dead-end elimination. Journal of computational chemistry 21(11), 999–1009 (2000)
31. Schiex, T.: Arc consistency for soft constraints. In: Principles and Practice of Constraint Programming - CP 2000. LNCS, vol. 1894, pp. 411–424. Singapore (Sep 2000)