Specializing Russian Doll Search *

Pedro Meseguer and Martì Sánchez

IIIA-CSIC, Campus UAB, 08193 Bellaterra, Spain
{pedro|marti}@iiia.csic.es

Abstract. Russian Doll Search (RDS) is a clever procedure to solve overconstrained problems. RDS solves a sequence of nested subproblems, where two consecutive subproblems differ in one variable only. We present the Specialized RDS (SRDS) algorithm, which solves the current subproblem for each value of the new variable with respect to the previous subproblem. The SRDS lower bound is superior to the RDS lower bound, which allows for a higher level of value pruning, although more work per node is required. Experimental results on random and real problems show that this extra work is often beneficial, providing substantial savings in the global computational effort.

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 and do not have any solution. In this situation, it is desirable to find the assignment that *best respects* the constraints under some preference criterion. Under this view, over-constrained CSPs are optimization problems for which *branch and bound* is a suitable solving strategy. The efficiency of branch and bound-based algorithms greatly depends on the lower bound 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.

An approach [4, 11, 7, 1, 8] for lower bound computation aggregates two main elements: (i) the global contribution of assigned variables, and (ii) the addition of individual contributions of unassigned variables. Another approach [10] keeps (i) but substitutes (ii) by a global contribution of unassigned variables. This is done by the Russian Doll Search (RDS) method, which requires n successive searches on nested subproblems to finally solve a problem of n variables.

Two consecutive subproblems differ in one variable only. RDS computes the cost of including that variable in the previously solved subproblem, getting the cost of the next subproblem. In this paper, we extend the RDS approach by computing the cost of including each value of that variable in the previously

* This work was supported by the IST Programme of the Commission of the European Union through the ECSPLAIN project (IST-1999-11969), and by the Spanish CICYT project TAP99-1086-C03-02.

solved subproblem. Given that this new approach performs RDS specialized per value, we call it *Specialized Russian Doll Search* (SRDS).

While RDS performs n searches, SRDS requires up to $n \times d$ independent searches (where d is the domain size) because it solves each subproblem for each feasible value of the new variable. This extra effort permits to compute a lower bound greater than (or equal to in the worst case) RDS lower bound, and allows the user to adjust good upper bounds when solving each subproblem. In addition, some searches can be skipped, under some conditions. Experimental results show that SRDS strategy is often superior to RDS, because its extra pruning causes substantial savings in global search effort.

In this paper we consider the Max-CSP model where all constraints are considered equally important, and the Weighted CSP model where constraints can be assigned different weights and the goal is to find the assignment that minimizes the accumulated weight of unsatisfied constraints. These two models are tested in the experimental results with the Random Problems and Celar Frequency Assignment Problems respectively. Our approach can also be applied to other frameworks.

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, analising the different lower bounds that can be calculated. Section 4 presents the SRDS algorithm. In Section 5 we discuss some experimental results. 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 [1]. An assignment of values to variables is complete if it includes every variable in X, otherwise it is partial. A *solution* for a CSP is a complete assignment satisfying every constraint. The problem is called over-constrained if such an assignment does not exist. It may be of interest to find a complete assignment that best respects all constraints [2, 9]. We consider the Max-CSP problem, for which the solution of an over-constrained CSP is a complete assignment satisfying the maximal number of constraints. 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

[1] We assume that for each pair of variables i, j there is only one constraint $R_{i,j}$

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. 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.

Partial forward checking (PFC) [4], is a branch and bound based algorithm whose lower bound has been improved by lookahead. 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). PFC lower bound is $dist(P) + \sum_{i \in F} \min_a(ic_{ia})$, where $dist(P)$ is the number of inconsistencies among past variables. This lower bound can be improved by adding inconsistencies among future variables in several ways,

- Directed arc inconsistency counts: constraints among future variables must be directed, either by a static variable ordering [11] or by a directed constraint graph [8]. Arc inconsistencies are recorded in the variable where the constraint points to, using directed arc inconsistency counts (DAC), which can be combined with IC to produce a new lower bound [11, 7, 8].
- Arc inconsistency counts: constraints among future variables are not directed. Arc inconsistencies are recorded using arc inconsistency counts (AC), a fraction of which can be combined with IC to produce a new lower bound [1].
- Russian doll search (RDS): at a point in search, the subproblem formed by future variables and constraints among them has been previously solved to optimality. The distance of the optimal solution of the subproblem is added to the PFC lower bound producing a new lower bound [10].

In this paper, we focus on RDS [10]. Its central idea is to replace one search by n successive searches on nested subproblems. Given a static variable ordering, the first subproblem involves the nth variable only, the ith subproblem involves all variables from the $(n-i+1)$th to the nth, and the nth subproblem involves all variables. Each subproblem is optimally solved using the RDS algorithm, following an increasing variable order: the first variable has the lowest number in the static ordering, and the last is always n. The central point in this technique is that, when solving the whole problem, subproblem solutions obtained before can be used in the lower bound in the following form,

$$LB(P, F) = dist(P) + \sum_{j \in F} min_b(ic_{jb}) + rds(F)$$

where $rds(F)$ is the distance of the optimal solution of the subproblem restricted to the variables in F. Let us suppose that we are solving the whole problem and

P involves the first $n-i$ variables. The set F involves from $n-i+1$ to n variables, that is, F is composed by the variables of the ith subproblem. The distance of the best solution found in that subproblem can be safely added to the contribution of P plus ic counters to form the lower bound, because each term counts different inconsistencies. This strategy is also used when solving the subproblem sequence. Solving the ith subproblem involves reusing all solutions from previously solved subproblems.

3 Specializing RDS

The motivation for specializing the RDS contribution for every value of the new variable, instead of a unique RDS contribution for that variable is as follows. In general, the cost of including the values of one variable in the previously solved subproblem is not necessarily homogeneous, that is, *good* values (with low cost) and *bad* values (with high cost) may be present in the variable domain. Standard RDS only takes into account the minimum cost of all values of the variable. Using this specialized contribution, SRDS is able to develop pruning conditions stronger than RDS ones.

In addition, for one future variable we can combine their ic and rds contributions for each value, and take the minimum in the lower bound.

The central idea of the algorithm is to replace the n successive searches on nested subproblems of RDS by $n \times d$ searches: one for including every value of every new variable. Given a static variable ordering, the $n-i+1$th subproblem considers the subset of variables $\{i, \ldots, n\}$. SRDS performs d independent searches for each value of variable i. The optimal cost of assigning value a to variable i in the $n-i+1$th subproblem is called rds_{ia}. After solving the $n-i+1$th subproblem for each value of variable i, counters recording rds_{ia} are updated and remain unchanged during the rest of the process. It is worth noting that,

$$rds(\{i, \ldots, n\}) = min_a rds_{ia}$$

3.1 A new lower bound

Let us consider the last iteration of SRDS, when the sets of past and future variables are $P = \{1, \ldots, i-1\}$ and $F = \{i, \ldots, n\}$. For every $k \in F$ and $a \in D_k$, rds_{ka} contains the optimal cost of solving the subproblem $n-k+1$ defined by the variables $(k, \ldots, n)$ with value a for variable k. From these new elements, a new family of lower bounds can be defined as follows,

$$\widehat{LB}^S(P, F, j) = dist(P) + min_a(ic_{ja} + rds_{ja}) + \sum_{k \in F, k \neq j} min_a ic_{ka} \quad j \in F$$

Property 1. $\widehat{LB}^S(P, F, j)$ is a safe lower bound for any variable $j \in F$.

Proof. The distance of any complete assignment including the partial assignment of P will include $dist(P)$. Given an arbitrary variable $j \in F$, $min_a(ic_{ja}+rds_{ja})$ is the minimum number of inconsistencies that j will have with any other variable of the problem when completing the current partial assignment, no matter which value is assigned to j. These inconsistencies are different from the ones recorded in $dist(P)$, so they can safely added. For any other variable $k \in F, k \neq j$, $\sum_{k \in F, k \neq j} min_a ic_{ka}$ is the minimum number of inconsistencies that all other future variables will have with past variables when completing the current partial assignment, no matter which value is assigned to them. This third term records different inconsistencies from the other two, $dist(P)$ and $min_a(ic_{ja} + rds_{ja})$, so they can be safely added. Only true inconsistencies are recorded, and no true inconsistency is recorded more than once. Therefore, $\widehat{LB}^S(P, F, j)$ is a safe lower bound. □

The SRDS lower bound, $LB^S(P, F)$, is the best lower bound of this family,

$$LB^S(P, F) = max_j \widehat{LB}^S(P, F, j)$$

Obviously, $LB^S(P, F)$ is a safe lower bound. In addition, it improves the standard lower bound of RDS.

Property 2. Using the same static variable ordering $LB^S(P, F) \geq LB(P, F)$

Proof.

$$\begin{aligned}
LB^S * (P, F) &= max_j \widehat{LB}^S * (P, F, j) \\
&\geq \widehat{LB}^S * (P, F, i) \\
&= dist(P) + min_a(ic_{ia} + rds_{ia}) + \sum_{k \in F, k \neq i} min_a ic_{ka} \\
&\geq dist(P) + min_a(ic_{ia}) + min_a(rds_{ia}) + \sum_{k \in F, k \neq i} min_a ic_{ka} \\
&= dist(P) + \sum_{k \in F} min_a ic_{ka} + min_a(rds_{ia}) \\
&= dist(P) + \sum_{k \in F} min_a ic_{ka} + rds(F) \\
&= LB(P, F)
\end{aligned}$$

Realizing that $min_a(rds_{ia})$ is referred to the first variable in F, it is clear that $min_a(rds_{ia}) = rds(F)$. □

Note that the expression,

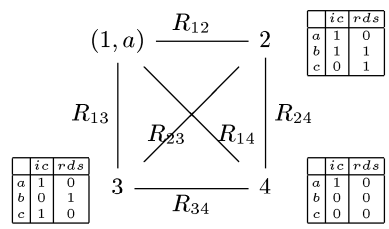
$$distance(P) + \sum_{k \in F} min_a(ic_{ka} + rds_{ka})$$

is not a safe lower bound because the same inconsistency may be counted more than once. In Figure 1 we provide an example of this.

$P = \{(1,a)\}, \quad F = \{2,3,4\}, \quad D_1 = D_2 = D_3 = D_4 = \{a,b,c\}$

$R_{12} = \{ac, bb, bc, cc\}, \quad R_{13} = \{ab, bb, bc, cb\}, \quad R_{14} = \{ab, ac, ba, cc\}$

$R_{23} = \{aa, ac, ba, cb\}, \quad R_{24} = \{aa, ab, ac, cb\}, \quad R_{34} = \{aa, ab, ac, cb\}$



$distance(P) + \sum_{k \in F} min_a(ic_{ka} + rds_{ka}) = 0 + 1 + 1 + 0 = 2$

$BestSolution = ((1,a)(2,c)(3,b)(4,b)), \quad distance(BestSolution) = 1$

Fig. 1. This example shows that rds_{ka} cannot be added over F without risk of repeating inconsistencies. After $(1,a)$, you count 2 inconsistencies, although only one exists (inconsistency on R_{34}).

3.2 Future Value Pruning

A value b of a future variable l can be pruned when the lower bound specialized for that value is greater than or equal to the current UB. For standard RDS, the specialized lower bound is,

$$LB_{lb}(P,F) = dist(P) + ic_{lb} + \sum_{k \in F, k \neq l} min_a ic_{ka} + rds(F)$$

In SRDS there is a family of lower bounds, $LB^S(P,F,j)$, which can be specialized for value b of future variable l as follows,

$$\widehat{LB}^S_{lb}(P,F,j) = \begin{cases} dist(P) + ic_{lb} + min_a(ic_{ja} + rds_{ja}) + \sum_{k \in F, k \neq j,l} min_a ic_{ka} & j \neq l \\ dist(P) + ic_{lb} + rds_{lb} + \sum_{k \in F, k \neq l} min_a ic_{ka} & j = l \end{cases}$$

The SRDS lower bound specialized for value b of future variable l is,

$$LB^S_{lb}(P,F) = max_j(\widehat{LB}^S_{lb}(P,F,j))$$

which is always better than the specialized lower bound of RDS.

Property 3. Using the same static variable ordering $LB^S_{lb}(P,F) \geq LB_{lb}(P,F)$.

Proof. Assuming that $F = \{i, \ldots, n\}$ and $l \neq i$, we have,

$$\begin{aligned}
LB^S_{lb}(P,F) &= max_j \widehat{LB}^S_{lb}(P,F,j) \\
&\geq \widehat{LB}^S_{lb}(P,F,i) \\
&= dist(P) + ic_{lb} + min_a(ic_{ia} + rds_{ia}) + \sum_{k \in F, k \neq i,l} min_a ic_{ka} \\
&\geq dist(P) + ic_{lb} + min_a ic_{ia} + min_a rds_{ia} + \sum_{k \in F, k \neq i,l} min_a ic_{ka} \\
&= dist(P) + ic_{lb} + \sum_{k \in F, k \neq l} min_a ic_{ka} + min_a rds_{ia} \\
&= LB_{lb}
\end{aligned}$$

If $l = i$, we have,

$$\begin{aligned}
LB^S_{lb}(P,F) &= max_j \widehat{LB}^S_{lb}(P,F,j) \\
&\geq \widehat{LB}^S_{lb}(P,F,i) \\
&= dist(P) + ic_{lb} + rds_{lb} + \sum_{k \in F, k \neq l} min_a ic_{ka} \\
&\geq dist(P) + ic_{lb} + \sum_{k \in F, k \neq l} min_a ic_{ka} + min_a rds_{la} \\
&= LB_{lb}
\end{aligned}$$

□

3.3 Initial Adjustment of UB

When solving subproblem $\{i, \ldots, n\}$ for every value of variable i, it is very important to start with a good UB, instead of taking $UB = \infty$ by default. In this way, pruning is optimized from the very beginning. Two initial adjustments of UB are considered, as follows,

1. When solving the subproblem composed by the variables $\{i+1, \ldots, n\}$, and a better solution is found, the current UB is decreased. Then, this new solution is extended to value a of variable i, and the distance of the extended solution is taken as a candidate UB for the subproblem to be solved with (i, a). This strategy is performed for all feasible a of variable i.
2. When solving the subproblem composed by the variables $\{i, \ldots, n\}$ with (i, a) and a better solution is found, the current UB is decreased. This new solution is modified, substituting a by b in variable i, if b is still unsolved in i. The distance of the modified solution is taken as a candidate UB for the subproblem to be solved with (i, b). This strategy is performed for all feasible b unsolved of variable i.

Candidates UB for subproblem with (i, a) are compared and the minimum is recorded as UB_{ia}. If $UB_{ia} = rds_{i+1,b}$, the resolution of the subproblem $\{i, \ldots, n\}$ with (i, a) is skipped, taking $rds_{ia} = UB_{ia}$.

4 The SRDS algorithm

The SRDS algorithm appears in Figure 2. The *specializedRDS* procedure is called with the problem variables and their domains, and it executes n times the PFC_SRDS procedure, following the RDS subproblem sequence. The PFC_SRDS procedure is called on a particular subproblem, with a set P of past variables, a set F of future variables and their current domains FD. If the set F is empty (line 1), it means that a solution better than the current one has been found, and this is recorded in Sol and UB (lines 2 and 3). Also the *adjustUB* procedure is executed, following the description given in Section 3.3. If the set F is not empty, the first variable i of F is extracted (line 6) and all its values are processed as follows. Considering a as the current value, if i is the first variable of the current subproblem the upper bound is initialized (line 8). If the pruning condition is not satisfied (line 10), the *lookahead* procedure is executed updating the *ic* counters (line 11). Again, the pruning condition is checked (line 12) and if it not satisfied, the *prune* function is executed on future domains (line 13), producing new future domains. If again the pruning condition is not satisfied (line 14), a recursive call is performed where the set P is incremented with (i, a) and new future domains are passed (line 15). When all possible combinations of values for F variables have been tried, the *rds* contribution for the current value of the first variable of the current subproblem is updated (line 16). Procedure *lookahead* and functions *prune* and LB^S_{jb} present no difficulty.

In line 12 of PFC_SRDS, a safe approximation of the lower bound LB^S is computed. Instead of computing the variable $j \in F$ that provides the highest contribution, $i + 1$ the first variable of F is taken. Something similar occurs in function LB^S_{jb}, where no complete maximization is performed on the set F. Instead, the best specialized lower bound for (j, b) is selected from two candidates, which differ in the variable that provides the *rds* contribution, $i + 1$ or j. These approximations have been done to reduce overhead without causing a serious decrement in the lower bound.

The fundamental difference between the RDS and the SRDS algorithms is as follows. When SRDS starts assigning a new value for the first variable of the subproblem, it reinitializes the upper bound (line 8). This is not performed by RDS, which reinitializes the upper bound for the first value only. In this way, RDS computes the minimum contribution of the whole subproblem, while SRDS specializes that contribution of each value of the first variable of the subproblem.

Removing line 8 of PFC_SRDS in Figure 2, we obtain an algorithm which is in between RDS and SRDS. We call it *limited SRDS*. It computes the *rds* contribution for each value of the first subproblem variable until it finds the minimum for that variable. This minimum will be stored as the *rds* contribution for the values which were not processed when that minimum was found. Limited SRDS requires the same search effort as RDS (both compute the minimum *rds* contribution of a subproblem), but SRDS lower bounds allow for a better usage of this information. $LB^S(P, F)$ combines *ic* and *rds* counters in one future variable, while $LB(P, F)$ always takes the *rds* contribution in isolation. Because of that, limited SRDS has a higher pruning capacity than RDS, with the same effort.

```
procedure specializedRDS({1,...,n}, {FD_1,...,FD_n})
1 for i from n downto 1 do
2   PFC_SRDS({}, {i,...,n}, {FD_i,...,FD_n});

procedure PFC_SRDS(P, F, FD)
1 if (F = ∅) then
2   Sol ← assignment(P);
3   UB ← distance(P);
4   adjustUB();
5 else
6   i ← PopAVariable(F);
7   for all a ∈ FD_i do
8     if P = ∅ then UB ← UB_ia^start;
9     newD ← distance(P) + ic_ia;
10    if (newD + rds_ia + Σ_{k∈F} min_b ic_kb < UB) then
11      lookahead(i, a, F, FD);
12      if (newD + min_b(ic_{i+1,b} + rds_{i+1,b}) + Σ_{k∈F,k≠i+1} min_b ic_kb < UB) then
13        newFD ← prune(F, FD);
14        if (newD + max_{j∈F}(min_b(ic_jb + rds_jb) + Σ_{k∈F,k≠j} min_b ic_kb) < UB) then
15          PFC_SRDS(P ∪ {(i, a)}, F, NewFD);
16    if P = ∅ then rds_ia ← UB

procedure lookahead(i, a, F, FD)
1 for all j ∈ F do
2   for all b ∈ FD_j do
3     if (inconsistent(i, a, j, b)) then ic_jb ← ic_jb + 1;

function prune(F, FD)
1 for all j ∈ F do
2   for all b ∈ FD_j do
3     if (LB^S_jb(newD, F) ≥ UB) then FD_j ← FD_j − {b};
4 return FD;

function LB^S_jb(newD, F)
1 lb_1 ← newD + ic_jb + Σ_{k∈F,k≠j,i+1} min_c ic_kc + min_c(ic_{i+1,c} + rds_{i+1,c});
2 lb_2 ← newD + ic_jb + rds_jb + Σ_{k∈F,k≠j} min_c ic_kc;
3 if (lb_1 > lb_2) return lb_1;
4 else return lb_2;
```

Fig. 2. Specialized Russian Doll Search algorithm

5 Experimental Results

Experimental results in [10] show that RDS performs better than Depth First Branch and Bound on high problem connectivity. This is observed also in SRDS

but in a sort of way that extremes are exaggerated: very connected problems give very good results and low connected problems (for example combined with high constraint tightness) can give very poor results. This is because the increase in the number of subproblems to solve up to $n * m$.

5.1 Random Problems

We give results on 6 classes of binary random problems. A binary random problem class is defined by the tuple $\langle n, m, p_1, p_2 \rangle$, where n is the number of variables, m is the number of values per variable, p_1 is the problem connectivity and p_2 is the constraint tightness. We tested the following six classes $\langle 10, 10, 1, p_2 \rangle$, $\langle 15, 5, 1, p_2 \rangle$, $\langle 15, 10, 50/105, p_2 \rangle$, $\langle 20, 5, 100/190, p_2 \rangle$, $\langle 25, 10, 37/300, p_2 \rangle$, $\langle 40, 5, 55/780, p_2 \rangle$, increasing the number of variables and decreasing connectivity. Results are presented in Figure 3, showing mean CPU time versus tightness. Each point is averaged over 50 executions. The graphics show the good behaviour of SRDS algorithm for complete graphs. For low connected problems the algorithm starts performing badly; this can be seen in the last random class $\langle 40, 5, 55/780, p_2 \rangle$, where SRDS performs as bad as RDS for high constraint tightness.

Random problems should not show the entire advantages of specializing RDS for every value. All values of a same variable have the same expected cost, this is because the homogeneity of the constraint tightness in the whole problem.

We have experimented also in the effect of adding values one by one to every variable of a highly connected problem. The results are very interesting because contradicting intuition, more values cause to increase the algorithm performance even if, theoretically, more subproblems are solved. This result can be visualized in Figure 4. This is only the case for high connected problems. For low connected problems solving every value to optimality can be computationally expensive. This will be explained in the next subsection for CELAR problems.

5.2 Frequency Assignment Problems

The Frequency Assignment Problem from CELAR [3] is a widely used binary overconstrained CSP benchmark. It consists of 11 instances to which different optimization criteria can be applied. We have centered our efforts in instance number 6 one of the hardest, which has 200 variables and the optimization criteria consists of minimizing the accumulated cost of the violated constraints. Constraints have violation costs that vary from 1 to 1000.

A simplification exists pointed out by Thomas Schiex for eliminating all the hard equality constraints of the problem and reducing the number of variables by two (because of the bijective property of the equality). From now we will assume this simplification.

Instance number 6 was first optimaly solved [5] in 32 days with a Sparc 5 workstation using a RDS technique. Five subinstances of the total instance number 6 were extracted, and the total accumulated cost of the subinstances was proven equal to best upperbound found at that moment for the global instance,

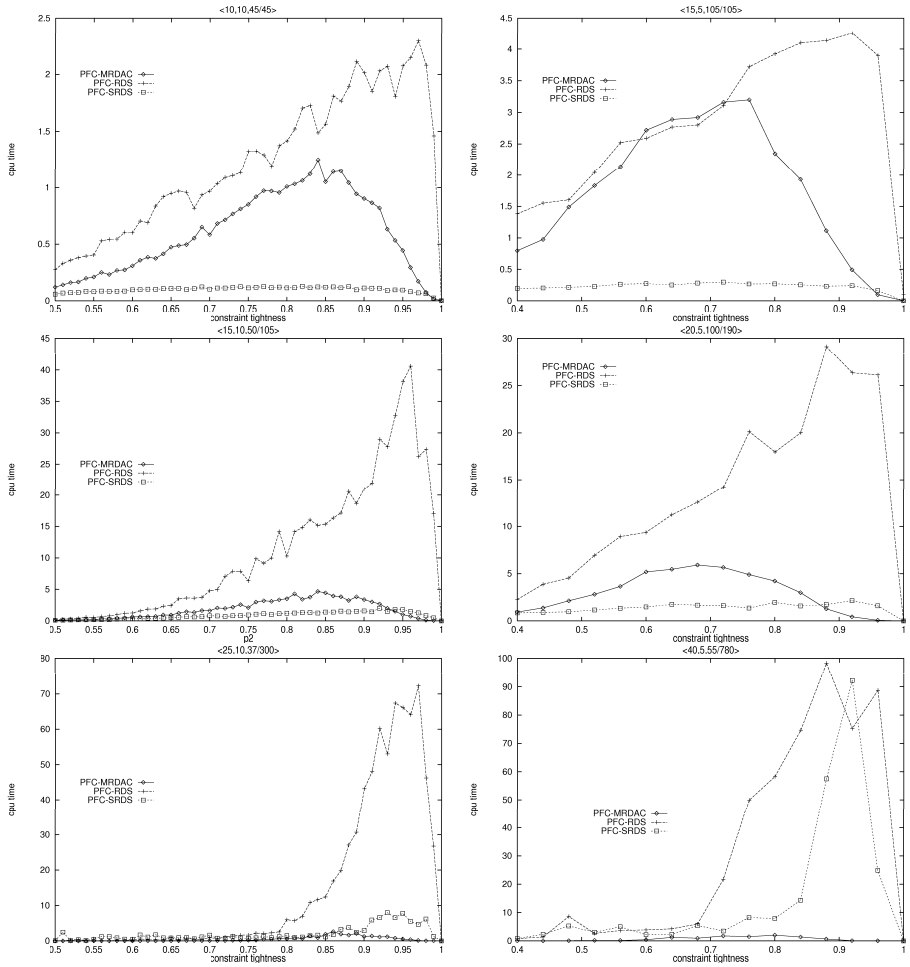


Fig. 3. Average CPU versus tightness for six classes of binary random problems.

so it was directly proven optimal. The subinstances all together induce a total cost of 3389 which is the optimal cost of instance 6.

Four subinstances are solved using directed arconsistency techniques in [8]. The CPU time given correspond to the time to prove optimality, that means the DAC algorithm is initialized with the optimal cost as upperbound.

Recently in [6] the whole instance 6 is solved to optimality in no more than 3 hours using graph decomposition techniques combined with a dynamic programing algorithm.

We have focussed solving the five subinstances and our results show a substantial decrement in CPU time, with respect to results reported in [5] and [8].

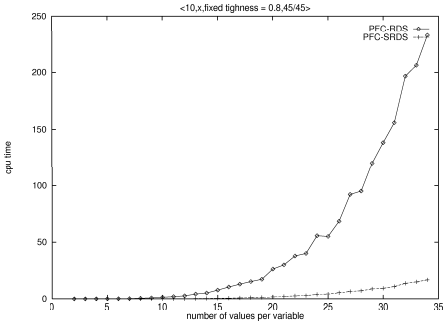


Fig. 4. Mean CPU time versus the number of values per variable on the $\langle 10, m, 1, 0.8\rangle$ binary random problem class (complete graph). As the number of values per variable increases, the RDS lower bound loses quality and becomes less efficient

Experiments show that the SRDS algorithm is very sensitive to problem connectivity. When problems became less connected our algorithm performance decreases. An alternative for this problem is to solve with SRDS the first n' variables, $n' < n$, while the $n - n'$ last variables are solved with Limited SRDS. A study of the effect of this parameter n' in the CELAR-6 subinstance 2 can be visualized in Figure 5.

Table 1 shows results for the 5 CELAR-6 subinstances. The subinstances subproblem is reached (this parameter is indicated in the column *limitedto*). For example, subinstance SUB_4 is solved using SRDS up to a 18 variables subproblem, for the following subproblems 19, 20, 21, and 22 variables, the limited version of SRDS is used instead. In Table 2 the detailed execution for solving subinstance SUB_4 is presented.

We have observed that CELAR-6 subinstances are a perfect example of poor homogeneity within the specialized lower bounds for every value rds_{ia} for the same variable i (in the same variable, values with high cost can coexist with consistent values with cost 0) and also it can happen that the mean value cost of a variable is high and the following (respect to the static order) is low; suggesting that the variable where to combine rds and ic that gives the best lower bound contribution is not necessary the previous variable in the static ordering. So, our algorithm could aport advantages in comparison with standard RDS.

We have also noticed the extreme sensitivity of the algorithm to the static order of SRDS resolution. In a particular subproblem of CELAR-6 involving only 8 variables, different static orders gave CPU solving time from 0.3 to 513 seconds. For RDS it is known that orders that give best results are those with minimum bandwidth, that is the maximum distance (in terms of number of variables) between two connected variables. This results need to be adapted for taking in account the cost associated to constraints.

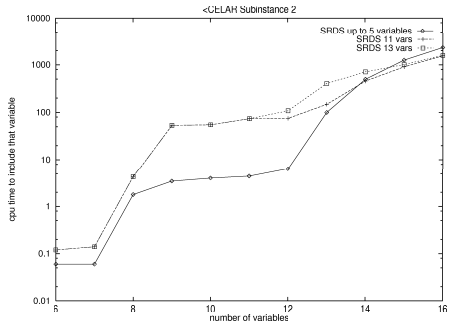


Fig. 5. CPU time for including each variable of subinstance 2 for different values of the limited version of SRDS, that is SRDS (to optimaly solve every value of a variable) is used up to different problem sizes. Plot in logarithmic scale.

instance	variables	connectivity	optimal cost	limited to	cpu time
SUB_0	16	0.47	159	all vars	$89s$
SUB_1	14	0.82	2669	10 vars	$711s$
SUB_2	16	0.74	2746	13 vars	$1573s$
SUB_3	18	0.69	3079	16 vars	$34671s$
SUB_4	22	0.56	3230	18 vars	$206314s$

Table 1. Results on CELAR-6 subinstances. The subinstances are solved using SRDS until a certain number of variables is reached (column 'limited to'). For the subproblems bigger than 'limited to' variables the limited version is used instead. CPU time corresponds to a Sun Ultra60 machine with SUN CC C++ compiler.

6 Conclusions

We have presented the SRDS algorithm that specializes the RDS approach solving each nested subproblem for each value of the new variable. This new approach requires more effort to solve completely each subproblem, but it provides a family of lower bounds, from which the best one is selected. This SRDS lower bound is superior to RDS lower bound. The main point in this new lower bound is the combination in one future variable of ic and rds contribution, taking the minimum of their sum, instead of adding both minimums as performed by RDS. The new lower bound can be specialized for future values, resulting in a value pruning condition better than the RDS one. Additionally, this allows for a better adjustment of initial upper bounds. We provided the SRDS algorithm and the approximations used to compute the lower bound in practice. This algorithm can be reduced to limited SRDS, an algorithm that requires the same search effort as RDS but offering a better performance. Empirical results show that this new approach provides substantial savings in computational effort.

For RDS-based approaches, the cost of solving a subproblem increases exponentially with the subproblem size. RDS is able to solve quickly small subprob-

number of variables	cpu time	optimal cost	maximum cost	mean cost
2	0.00	0	0	0.00
3	0.00	0	0	0.00
4	0.01	0	0	0.00
5	0.01	0	0	0.00
6	0.01	0	100	47.32
7	0.02	100	113	102.82
8	0.04	202	304	237.55
9	0.26	323	3611	2294.68
10	0.30	347	1457	619.56
11	9.10	350	664	469.05
12	19.69	435	1639	849.68
13	101.63	1115	1981	1362.00
14	337.39	1632	2731	1871.09
15	1072.77	2271	3435	2698.14
16	10702.25	2746	4246	3277.23
17	16012.65	2858	4081	3339.32
18	64744.13	3079	4293	3565.27
19	67852.85	3109	3109	3109.00
20	67852.85	3109	3109	3109.00
21	206314.41	3230	3420	3240.56
22	206314.41	3230	3230	3230.00

Table 2. Subproblem results of subinstance SUB_4 with the algorithm SRDS limited to 18 variables. Each subproblem is build from the previous one including a new variable. The columns are: number of variables of the subproblem, time for optimally solving the subproblem in seconds, optimal cost of the subproblem (that is $min_a rds_{ia}$ being i the variable included in this particular subproblem), maximum cost of all the values of the variable included (that is $max_a rds_{ia}$), and the mean of the costs of the values of the new variable.

lems, but it slows down dramatically when subproblems increase to reasonable size. SRDS performance is worse than RDS on small subproblems, but its superior lower bound allows for a better performance with increasing size. Above some size, is the limited SRDS algorithm the one that offers the best performance. More work, both theoretical and experimental, is needed to fully understand these empirical results. The final goal would be to capture the subproblem information that offers the best cost/benefit trade-off in global terms.

Acknowledgements

Authors thank anonymous reviewers for their constructive criticisms.

References

1. M. S. Affane and H. Bennaceur. A weighted arc consistency technique for Max-CSP. In *Proc. of the 13th ECAI*, 209–213, 1998.
2. S. Bistarelli, U. Montanari and F. Rossi. Constraint Solving over Semirings. In *Proc. of the 14th IJCAI*, 1995.
3. B.Cabon, S. De Givry, L.Lobjois, T.Schiex and J.P.Warners Radio Link Frequency Assignment *Constraints*, February 1999, Vol 4, Number 1.
4. E.C. Freuder and R.J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58:21–70, 1992.
5. S. Givry, G. Verfaillie and T. Schiex. Bounding the optimum of constraint optimization problems. In *Proc. of the 3th CP* 1997.
6. A.M. Koster, C. P. Hoesel and A. W. Kolen. Optimal solutions for a frequency assignment problem via tree-decomposition. In Lecture Notes in Computer Science, , 1665:338-349, 1999.
7. J. Larrosa and P. Meseguer. Exploiting the use of DAC in Max-CSP. In *Proc. of the 2th CP*, 308–322, 1996.
8. J. Larrosa, P. Meseguer, and T. Schiex. Maintaining reversible dac for max-csp. *Artificial Intelligence*, 107:149–163, 1999.
9. T. Schiex, H. Fargier and G. Verfaillie. Valued Constraint Satisfaction Problems: hard and easy problems. In *Proc. of the 14th IJCAI*, 631–637, 1995.
10. G. Verfaillie, M. Lemaître, and T. Schiex. Russian doll search. In *Proc. of the 13th AAAI*, 181–187, 1996.
11. R. Wallace. Directed arc consistency preprocessing. In M. Meyer, editor, *Selected papers from the ECAI-94 Workshop on Constraint Processing*, number 923 in LNCS, 121–137. Springer, Berlin, 1995.