Adding Flexibility to Russian Doll Search

Margarita Razgon and Gregory M. Provan
Department of Computer Science, University College Cork, Ireland
{m.razgon|g.provan}@cs.ucc.ie

Abstract

*The Weighted Constraint Satisfaction Problem (*WCSP*) is a popular formalism for encoding instances of hard optimization problems. The common approach to solving* WCSP*s is Branch-and-Bound (*B&B*), whose efficiency strongly depends on the method of computing a lower bound (LB) associated with the current node of the search tree. Two of the most important approaches for computing LB include (1) using local inconsistency counts, such as Maintaining Directed Arc-Consistency (*MDAC*), and (2) Russian Doll Search (*RDS*).*

In this paper we present two B&B*-based algorithms. The first algorithm extends* RDS*. The second algorithm combines* RDS *and* MDAC *in an adaptive manner. We empirically demonstrate that the* WCSP *solver combining the above two algorithms outperforms both* RDS *and* MDAC*, over all the problem domains and instances we studied. To the best of our knowledge this is the first attempt to combine these two methodologies of computing LB for a* B&B*-based algorithm.*

1 Introduction

The Weighted Constraint Satisfaction Problem (WCSP) is a popular formalism for encoding instances of hard optimization problems, whose applications include resource allocation, bioinformatics, probabilistic reasoning [8]. One of the most popular approaches to solving WCSPs is Branch-and-Bound (B&B). The efficiency of a B&B algorithm strongly depends on the method of computing a lower bound associated with the current node of the search tree.

There are two main approaches to computing lower bounds. The first approach is based on counting local inconsistencies of the given constraint network, e.g. [17, 6, 7, 9, 10, 4]. The other approach is based on Russian Doll Search (RDS) [16, 11, 12, 2, 14], where instead of solving one WCSP, the algorithm solves a sequence of nested subproblems, each including one more variable than the previous subproblem, until the whole problem is solved. Processing a given subproblem in RDS is designed so that the set of variables unassigned by the current partial solution always constitutes the set of variables of an already solved subproblem. The optimal solution weight of this subproblem is taken into account by the procedure computing the lower bound.

The RDS algorithm is somewhat inflexible due to the fact that, at each iteration, only one particular solved subproblem (namely, the one that includes all the unassigned variables) contributes to the lower bound evaluation. The papers [11, 12] present versions of RDS having greater flexibility, achieved by solving subproblems obtained by restricting domains of *unassigned* variables to single values and selecting the "best" of them during the lower bound evaluation process. However the set of variables of the subproblem selected for the lower bound computation is the same as for RDS (i.e. the set of all *unassigned* variables).

In this paper we present two modifications of RDS that are more flexible in this sense. The first algorithm, called Partially Assigned Big Doll Search (PABDS), considers the already solved subproblems, some variables of which are *assigned* by the current partial solution, and selects the one that provides the best lower bound. From the point of view of "pure" RDS, such subproblems are obsolete because they were used on previous iterations and failed to cause backtracking. In PABDS we show that these subproblems can be reused due to the fact that some of their variables are *assigned*, which potentially increases the lower bound on the weight of violated constraints.

The second modification of RDS considered in this paper combines RDS with counting local inconsistencies. Besides the subproblem including all the unassigned variables used by RDS, the proposed algorithm considers also smaller subproblems involving only part of the unassigned variables. For the unassigned variables that do not belong to the considered subproblems, the proposed algorithm counts the number of local inconsistencies. This increases the lower bound and may be especially helpful when the filtering component of the algorithm has removed many values from the domains of unassigned variables. The counting of local inconsistencies is performed according to the method of

Maintaining Directed Arc-Consistency (MDAC) [6, 7], and hence the resulting algorithm is called RDS-MDAC. We can say that RDS-MDAC is a hybrid of RDS and MDAC. To the best of our knowledge this is the first attempt to combine these two approaches of computing lower bounds.

We evaluate the proposed methods empirically using randomly generated MAX-CSPs and Earth Observation Satellite Scheduling Problems (SPOT5) [1]. In particular, we compare PABDS and RDS-MDAC-PABDS (a hybrid of RDS-MDAC and PABDS) with RDS and MDAC. The results of our experiments show that RDS-MDAC-PABDS outperforms both RDS and MDAC on the tested domains. Hence combing the RDS and local inconsistency counts is a promising approach to lower bound evaluation.

The algorithm of [9] is the most closely related to our approach. As in our case, the lower bound in [9] is evaluated by combination of local inconsistency counts and a more global estimation on disjoint sets of unassigned variables. However [9] does not use the RDS-based measure, but develops another method of global evaluation. It is worth noticing that this method is not competing with ours, but rather complements it. In particular, the RDS-MDAC algorithm described in the present paper can be enhanced with the partition-based lower bounds presented in [9].

The remainder of the paper is organized as follows. In Section 2 we give some essential background and terminology required for this paper. In Section 3 we describe algorithms PABDS and RDS-MDAC. In Section 4 we report experiments performed on binary random MAX-CSPs and SPOT5 problems. Finally, in Section 5 we draw the conclusion and outline possible directions of further investigation.

2 Background

In this paper we restrict our attention to *binary Constraint Networks* (CNs). Let Z be a CN with the set $\{x_1, \ldots, x_n\}$ of variables. We denote by $D(x_i)$ the domain of values of x_i. An *assignment* of Z is a pair (x_i, v_i) such that $v_i \in D(x_i)$. The constraints of Z are represented by the set of *conflicting pairs* of values (or *conflicts*) of the given CN. A set of assignments, at most one for each variable, is a *partial solution* of Z. A partial solution that assigns all the variables of Z is a *solution* of Z. In the *Weighted Constraint Satisfaction Problem* (WCSP) conflicts are assigned with importance weights, the *weight of a solution* is the sum of weights of violated conflicts. The task of WCSP is to find the *optimal* solution of Z, i.e. one that has the smallest weight.

Branch-and-Bound (B&B) is a widely used approach to WCSP solving. At every moment B&B maintains the *current partial solution* P. The *upper bound* UB of B&B is the smallest weight of a solution considered since the beginning of the execution till the current moment. The goal of B&B is to obtain a solution containing P and whose weight is smaller than UB. The *lower bound* LB is a number such that there is no solution having weight less than LB and containing P. If $LB \geq UB$ then B&B immediately *backtracks*. The efficiency of B&B strongly depends on the method of computing LB: the greater the value of LB, the more chances that B&B would backtrack earlier. In the rest of the section we overview methods of computing LB, assuming that P has the weight F.

The most obvious value of LB is F. A more sophisticated way of computing LB is employed by *Partial Forward Checking* (PFC) [5]. Let v be a variable *unassigned* by P. Let $val \in D(v)$. Denote by $Conf(v, val, P)$ the sum of weights of conflicts of (v, val) with the values of P. Let $MinConf(v, P) = min_{val \in D(v)} Conf(v, val, P)$. Denote by $Connect$ the sum of $MinConf(v, P)$ over all variables v unassigned by P. Then $LB = F + Connect$.

This LB is further improved by *Maintaining Directed Arc-Consistency* (MDAC) [17, 6, 7]. This technique of computing LB is based on the notion of *arc-inconsistency*. A variable w is *arc-inconsistent* with respect to a value (v, val) if each value of $D(w)$ conflicts with (v, val). Let Ord be a linear order over all variables unassigned by P. For an unassigned variable v and $val \in D(v)$, let $dac_{v,val}$ be the number of variables that precede v by Ord and arc-inconsistent with respect to (v, val). Let $DM(v, val, P) = Conf(v, val, P) + dac_{v,val}$, $MinDM(v, P) = min_{val \in D(v)} DM(v, val, P)$, and $SumDM$ be the sum of $MinDM(v, P)$ over all the unassigned variables (the abbreviation DM stands for $DAC\ Measure$). Then $LB = F + SumDM$. The order Ord may be static, computed once at the beginning (as in the algorithm PFC-DAC [17, 6]), or some pairs of variables may be dynamically reversed in order to tight the lower bound (as in the algorithm PFC-MRDAC [7]). Originally the lower bounds of PFC-DAC and PFC-MRDAC were proposed for MAX-CSP (the weights of all conflicts equal 1). To ensure their validity for WCSP, it is essential that the weight of each conflict is ≥ 1.

The pruning power of the lower bounds of PFC and MDAC can be enhanced by the use of a *filtering procedure*. One possible filtering procedure works as follows. In addition to computing LB with respect to P, it computes LB with respect to $P \cup \{(v, val)\}$ for each unassigned variable v and each $val \in D(v)$. If for a particular value (v, val) the corresponding LB exceeds UB then val is *discarded* from $D(v)$. With the use of a filtering procedure, $D(v)$ refers not to the *initial* domain of v but rather to the *current* domain of v containing only *feasible* values, i.e. the ones that have not been discarded in the iteration corresponding to the current node of the search tree or its ancestor.

The last algorithm considered in this section is *Russian Dolls Search* (RDS) [16]. To explain how LB is computed

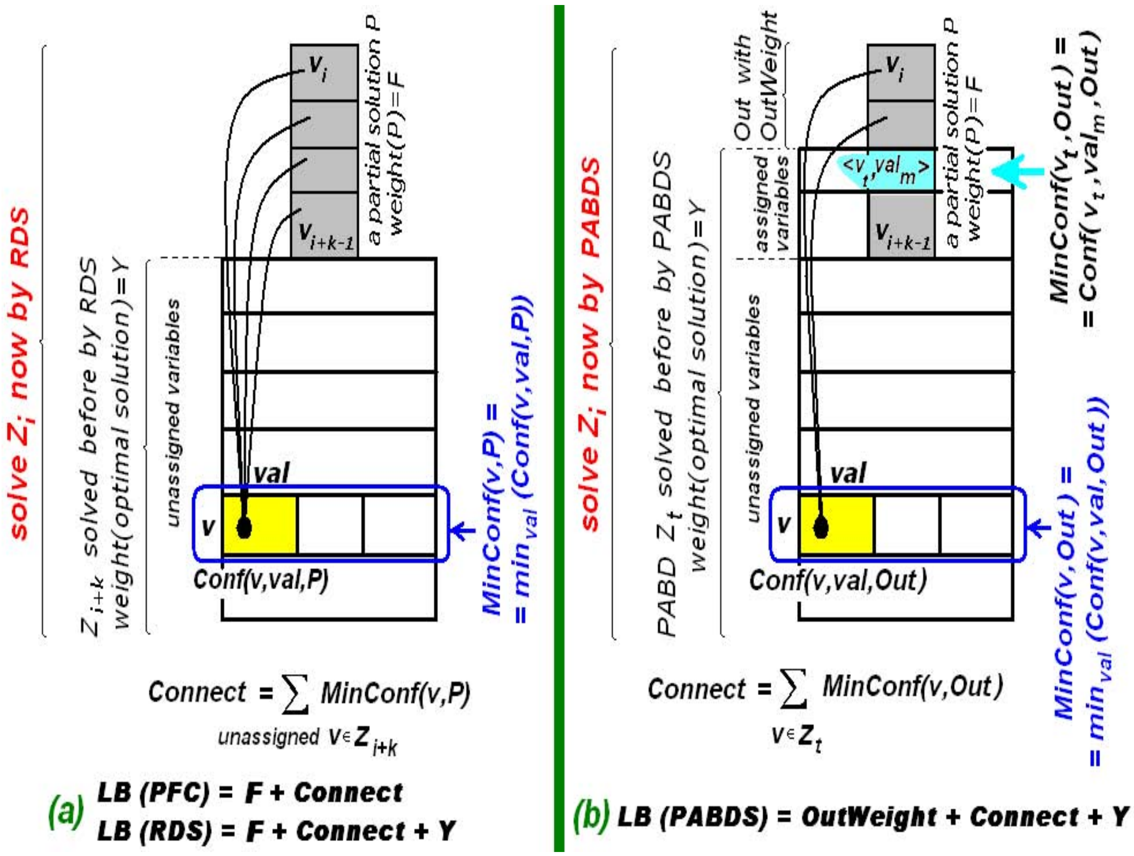

Figure 1. Solving Z_i and computing LB by: (a) RDS (on the left), (b) PABDS (on the right).

by RDS, consider again PFC. Let Z be a CN on the set of variables $\{v_1, \ldots, v_n\}$, assume that P assigns variables $v_1, \ldots, v_{i-1}$, and denote the projection of Z to $\{v_i, \ldots, v_n\}$ by Z_i. Let Y be the weight of an optimal solution of Z_i. Then the weight of any solution containing P is at least $LB = F + Connect + Y$. This LB is larger then the lower bound $F + Connect$ used by PFC. Thus PFC can perform better if in each iteration it has an oracle which tells the value of Y. RDS achieves that by fixing an order $v_1, \ldots, v_n$ and solving by PFC the WCSP first for Z_n, then for $Z_{n-1}, Z_{n-2}, \ldots$, and finally, for Z_1. For each iteration RDS "knows" the weight of an optimal solution of an projection of Z to *unassigned* variables. The computing of LB by RDS is schematically illustrated in Figure 1 (a) on the left, where we assume that the *current* WCSP being solved is Z_i, P assigns first k variables $v_i, \ldots, v_{i+k-1}$, and Y is the weight of an optimal solution of Z_{i+k} solved before.

The information obtained as a result of solving $Z_{i+1}, \ldots, Z_n$ can be also used for obtaining a good *initial* value of UB for Z_i. In particular, let T be an optimal solution of Z_{i+1} obtained on the previous run of the RDS algorithm. For each (v_i, val), the algorithm computes $ConfWeight(v_i, val, T)$. Let $T' = T \cup \{(v_i, val)\}$ such that $ConfWeight(v_i, val, T)$ is smallest possible. The weight of T' serves as the initial value of UB for Z_i.

3 Algorithms

3.1 Partially Assigned Big Doll Search (PABDS)

We propose a modification of RDS, which, in addition to LB computed by RDS, computes a number of other lower bounds with the hope that one of them would be larger than LB and increase the chances of earlier backtracking. We call the algorithm Partially Assigned Big Doll Search (PABDS).

Consider the search on Z_i, assuming that $Z_{i+1}, \ldots, Z_n$ have been already solved by PABDS (illustrated in Figure 1 (b) on the right). Let UB be the current upper bound on the weight of an optimal solution known at the considered moment. Assume that the current partial solution P assigns first k variables $v_i, \ldots, v_{i+k-1}$. Then a Partially Assigned Big Doll (PABD) is a subproblem which includes all *unassigned* variables $v_{i+k}, \ldots, v_n$ and a part of P, i.e. some *assigned* variables from the range $\{v_i, \ldots, v_{i+k-1}\}$. We can consider k such PABD-s consecutively increasing the size of the subproblem by adding one assigned variable, and estimate LB in each case. While RDS solves Z_i having all variables of considered subproblems *unassigned* in the current iteration, PABDS solves Z_i having some variables of considered subproblems *assigned* by P. While at each moment of pruning RDS examines only one subproblem of size equal to the number of unassigned variables, PABDS analyzes k subproblems consecutively, what increases the chances of backtracking at an earlier stage.

The value of LB is computed as follows. Let Z_t be some PABD on variables $v_t, \dots, v_{i+k-1}, v_{i+k}, \dots, v_n$ (when $i \le t \le i+k-1$). Let Y be the weight of an optimal solution of Z_t obtained before. We denote by Out the part of P that does not belong to Z_t, and by $OutWeight$ the weight of Out. We define $Conf(v, val, Out)$ as the sum of weights of conflicts of (v, val) with the values of Out. We define $MinConf(v, Out) = min_{val \in D(v)} Conf(v, val, Out)$, if the variable v is unassigned. However, if some variable v is assigned by value val_m in P then only this value val_m is considered, that is, $MinConf(v, Out) = Conf(v, val_m, Out)$. Finally, $Connect$ is the sum of $MinConf(v, Out)$ over all variables in Z_t.

We define $LB = OutWeight + Connect + Y$. It can be shown that LB is a lower bound on a weight of a full solution containing P. If $LB \ge UB$, it is safe to backtrack immediately.

Notice that for each value of t, RDS has already computed a lower bound at the moment when Out was the current partial solution and, since Out has been extended to P, that lower bound failed to cause backtracking. Hence, on the first glance, it may seem strange that we hope to backtrack at the current iteration by recomputing this lower bound. But the point is that the lower bound computed by PABDS is *not the same* lower bound that was computed by RDS. In particular, the value of $MinConf(v, Out)$ for an assigned variable v may be much larger than the same value on the time when v was unassigned. In addition, the lower bound computed by PABDS may be larger than the lower bound computed by RDS, because the weight Y of an optimal solution of Z_t may be larger than the weight of an optimal solution of Z_{i+k}. The combination of these two factors would produce a larger lower bound which would help us to backtrack earlier.

The PABDS algorithm applies *filtering* only for computing RDS lower bounds, i.e given the current partial solution P, we compute the RDS lower bound for each $P \cup \{v, val\}$ where val is a feasible value of an unassigned variable v (as explained in Section 2). If none of these lower bounds causes backtracking, the PABDS lower bounds are computed only for P. We do not apply filtering for the PABDS lower bounds because according to our empirical studies, filtering only slightly reduces the number of nodes of the search tree while considerably increases the runtime.

3.2 Combination of Russian Doll Search and Maintaining Directed Arc-Consistency (RDS-MDAC)

We propose an algorithm RDS-MDAC that combines RDS and MDAC (described in Section 2) in order to increase the value of LB and allow earlier pruning. For this algorithm we assume that weights of all conflicts are ≥ 1.

The idea is to apply RDS-MDAC after RDS was applied and failed to prune in the current iteration. Consider the search on Z_i, assuming that $Z_{i+1}, \dots, Z_n$ have been already solved by RDS-MDAC. Assume that the current partial solution P assigns first k variables $v_i, \dots, v_{i+k-1}$ and let F be the weight of P. Let Small Russian Doll (SRD) be a subproblem which includes only part of the unassigned variables $v_{i+k}, \dots, v_n$, leaving at least one variable outside. We consider $n - (i+k)$ such SRD-s, from Z_{i+k+1} to Z_n, consecutively decreasing the size of the subproblem by removing one unassigned variable.

For each SRD Z_t ($i+k+1 \le t \le n$), the value of LB is computed as follows. Let Y be the weight of an optimal solution of Z_t (obtained before). Let v be an unassigned variable and let $val \in D(v)$. Analogously to the case of RDS, we define $Conf(v, val, P)$ as the sum of weights of conflicts of (v, val) with the values of P, and $MinConf(v, P) = min_{val \in D(v)} Conf(v, val, P)$. We denote by $Connect$ the sum of $MinConf(v, P)$ over all variables in Z_t. On unassigned variables $v_{i+k}, \dots, v_{t-1}$ (which are not included in Z_t) we apply MDAC. Let Ord be a linear order over unassigned variables, where the variables of Z_t are placed before the rest of variables. For an unassigned variable v and $val \in D(v)$, let $dac_{v,val}$ be the number of *all* unassigned variables that precede v by Ord and arc-inconsistent with respect to (v, val), *including* variables involved in Z_t. We define $DM(v, val, P) = Conf(v, val, P) + dac_{v,val}$ and $MinDM(v, P) = min_{val \in D(v)} DM(v, val, P)$. Then we compute $SumDM$ as the sum of $MinDM(v, P)$ over unassigned variables $v_{i+k}, \dots, v_{t-1}$. We define $LB = F + Y + Connect + SumDM$.

Theorem 1 *$LB = F + Y + Connect + SumDM$ is a lower bound on a weight of a solution containing P.*

Proof. Let P^* be an full optimal solution containing P. We can describe P^* as $P^* = P + P_1 + P_2$, where P assigns k variables $v_i, \dots, v_{i+k-1}$, P_1 assigns variables $v_{i+k}, \dots, v_{t-1}$, and P_2 assigns variables $v_t, \dots, v_n$ (when $i+k+1 \le t \le n$). Let $w(P^*)$ be the weight of P^*, $w(P) = F$ be the weight of P, $w(P_2)$ be the weight of P_2, and $w_{pred}(v, val)$ be the weight of conflicts of assignment

$(v, val) \in P_1$ with predecessors of v in Ord. Then

$$w(P^*) = w(P) + w(P_2) + \sum_{(v,val)\in P_2} Conf(v, val, P) +$$

$$+ (\sum_{(v,val)\in P_1} Conf(v, val, P) + \sum_{(v,val)\in P_1} w_{pred}(v, val))$$

By definition of Y we get $w(P_2) \geq Y$.
By definition of $Connect$: $\sum_{(v,val)\in P_2} Conf(v, val, P) \geq$

$$\geq \sum_{v \text{ assigned by } P_2} MinConf(v, P) = Connect.$$

By definition of $SumDM$:

$$\sum_{(v,val)\in P_1} Conf(v, val, P) + \sum_{(v,val)\in P_1} w_{pred}(v, val) =$$
$$= \sum_{(v,val)\in P_1} (Conf(v, val, P) + w_{pred}(v, val)) \geq$$
$$\geq \sum_{(v,val)\in P_1} (Conf(v, val, P) + dac_{v,val}) =$$
$$= \sum_{(v,val)\in P_1} DM(v, val, P) \geq \sum_{v \text{ assigned by } P_1} MinDM(v, P) =$$
$$= SumDM.$$

The first inequality in the above calculations is based on the following argumentation. The value of $dac_{v,val}$ is a lower bound on the *number* of conflicts of (v, val) with the assignments of the predecessors of v in Ord. Since the weight of each conflict is at least 1, the sum of conflict weights cannot be smaller than the number of these conflicts, i.e. $w_{pred}(v, val) \geq dac_{v,val}$. Combining the above argumentation, we get $w(P^*) \geq F + Y + Connect + SumDM$. ■

Similarly to PABDS, and for the same reasons, RDS-MDAC applies filtering only for computing RDS lower bounds.

Why is the value of LB computed by RDS-MDAC larger then the value of LB computed by either RDS or MDAC? First of all, observe that RDS computes an optimal solution based on *complete* domains of the variables, and that the optimal solution based on the *current* domains of variables can be much larger. In this context it is desirable to estimate a lower bound more precisely. Measuring local inconsistency for a subset of unassigned variables may provide such an opportunity. But why not apply MDAC to the whole subset of unassigned variables? The answer is that, for a subset of variables, it may be still beneficial to compute the optimal solution by RDS, because none of the variables of this subset is arc-inconsistent with its predecessors. RDS-MDAC combines, in the adaptive manner, two ways of guessing a lower bound, and our hope is that it succeeds to catch a way of a good combination of these two measures.

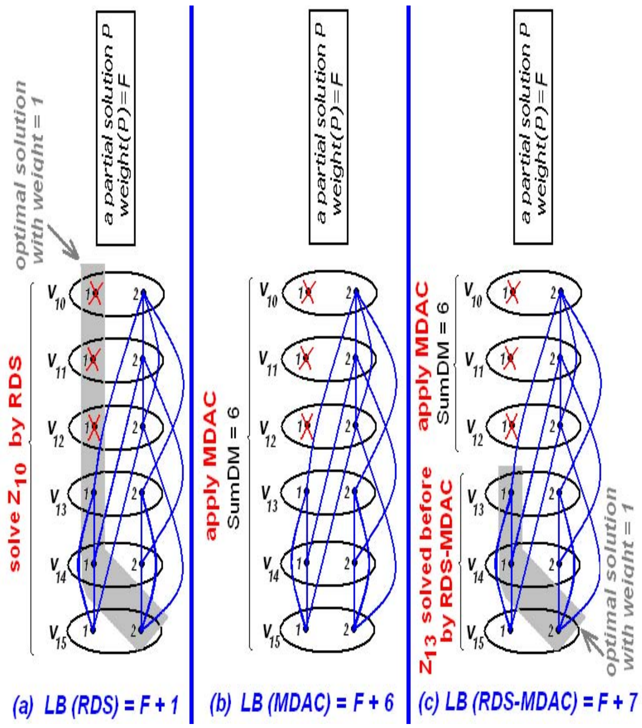

Figure 2. Computing LB by: (a) RDS, (b) MDAC, (c) RDS-MDAC.

In order to illustrate the above informal argumentation, consider the example shown in Figure 2. Set Z_{10} to consist of variables $v_{10}, \ldots, v_{15}$. We assume that the values $(v_{10}, 1), (v_{11}, 1), (v_{12}, 1)$ are removed from the domains of their variables due to their heavy conflicts with the current partial solution P. We assume also that P has no conflicts with the remaining values of $v_{10}, \ldots, v_{15}$, and all the conflicts shown in the figure have weight 1. Let us compute LB for the optimal extension of P.

Using RDS (Figure 2 (a)) we get an optimal solution $\{(v_{10}, 1), (v_{11}, 1), (v_{12}, 1), (v_{13}, 1), (v_{14}, 1), (v_{15}, 2)\}$ of Z_{10} having weight 1. Hence RDS provides $LB = F + 1$.

Applying MDAC on variables $v_{10}, \ldots, v_{15}$ (Figure 2 (b)), we assume that v_i is a predecessor of v_j whenever $i > j$. Then $dac_{v_{10},2} = 3, dac_{v_{11},2} = 2, dac_{v_{12},2} = 1$, and the measure supplied by the values of the rest of variables is 0. Hence $SumDM = 6$, and MDAC provides $LB = F + 6$.

Consider now RDS-MDAC (Figure 2 (c)). The largest LB is obtained when Z_t includes the variables v_{13}, v_{14}, v_{15}. An optimal solution $\{(v_{13}, 1), (v_{14}, 1), (v_{15}, 2)\}$ of Z_{13} obtained before by RDS-MDAC has weight 1. For the rest of variables we get $SumDM = 6$, arguing as in the previous paragraph. Thus RDS-MDAC provides $LB = F + 7$ which *larger* than the values of LB provided by RDS and MDAC taken alone.

4 Experiments

We present experiments to demonstrate the utility of our algorithms proposed in Section 3. We consider four different settings: RDS, PABDS, RDS-MDAC-PABDS, and MDAC. RDS-MDAC-PABDS is a combination of RDS-MDAC and PABDS applied in each pruning iteration after RDS failed to prune in this iteration. The directed arc-inconsistency counts were computed according to the RDS ordering. In other words, if $v_1, \ldots, v_n$ is the ordering of our variables such that the sequence of subproblems solved by RDS is $Z_n, \ldots, Z_1$ then, for computing of directed arc-inconsistency, v_i is a predecessor of v_j whenever $i > j$. The implementation of MDAC was the algorithm PFC-DAC [6, 7], where we used the same static ordering for computing $dac_{v,val}$ as in RDS-MDAC-PABDS. The reason why we chose PFC-DAC and not PFC-MRDAC is that we wanted to check how the hybrid of RDS and PFC-DAC performs with respect to each of these parts. In order to do this, it is important that all the algorithms follow the same order of variable instantiation. It is worth noting that we do not provide the results for RDS-MDAC because RDS-MDAC-PABDS gives strictly better results than RDS-MDAC.

The problem domains studied were binary random MAX-CSPs and Earth Observation Satellite Scheduling Problems (SPOT5), as in [16]. We measured the search efforts in terms of the number of backtracks and CPU-time. For every tuple of parameters of a tested instance, we report results as the average of 50 instances. The proposed algorithms were implemented using Microsoft Visual C++ 6.0, and the experiments were performed under Microsoft Windows XP 2002 on a 2GHz Pentium processor using 1GB RAM.

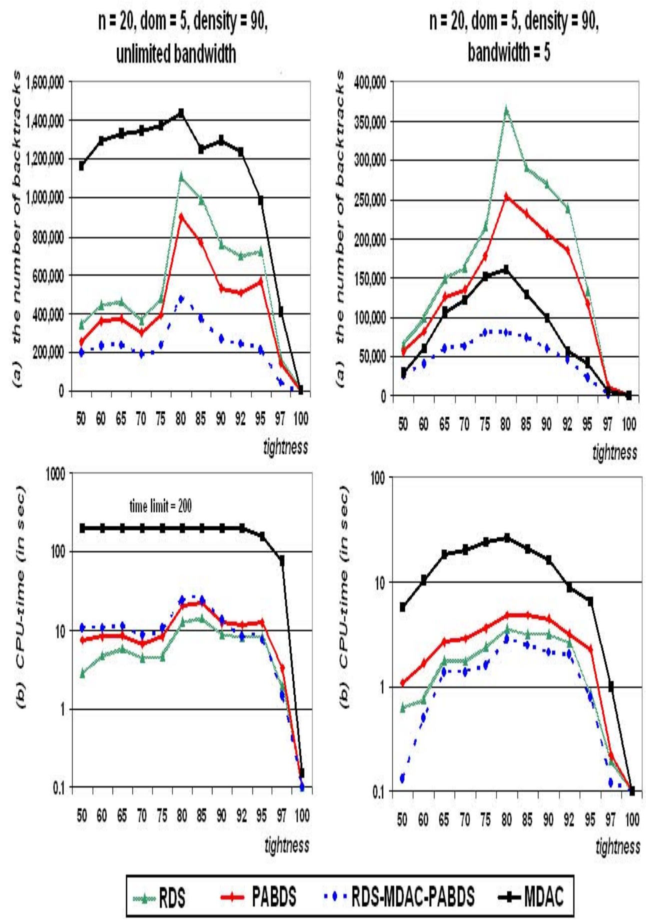

Figure 3. Binary Random MAX-CSPs: density=90, unlimited bandwidth (on the left), bandwidth=5 (on the right).

4.1 Binary Random MAX-CSPs

We generated CNs given the following four parameters: the number of variables n, the domain size dom, density $p1$, and tightness $p2$ [13]. Then we slightly modified the generator in order to produce problems of limited bandwidth [16].

In particular, given a graph G with n vertices, an ordering h is a one-to-one map from the vertices of G to the set $\{1, ..., n\}$. The bandwidth of a vertex v under an ordering h is the maximum value of $|h(v) - h(w)|$ over all vertices w connected to v. The bandwidth of a graph under an ordering is the maximum bandwidth of any vertex, and the bandwidth of a graph is its minimum bandwidth under any ordering. We generated CSPs of limited bandwidth b as follows. Given n, dom, $p1$, $p2$, b, and an ordering h. Let K be the set of all the pairs (v, w) of variables, such that $|h(v) - h(w)| \leq b$. We randomly selected pairs of constrained variables from K and conflicts between these variables. This method guarantees that the bandwidth of the ordering h, and therefore the graph bandwidth, is lower than or equal to b. Let us note that a small bandwidth b implies a maximum vertex's degree $2b$ in a constraint graph, and hence, a small connectivity in the constraint graph.

For all the tested algorithms we ordered the variables in decreasing order of their degrees in the constraint graph. For the RDS-based algorithms we used the static value ordering heuristic, first choosing the value that the variable had in the optimal assignment found on the previous subproblem [16], then choosing the first available value. For MDAC we chose the first available value.

We examined three sets of instances of unlimited bandwidth b by fixing the former three parameters (n, dom, density $p1 \in [90, 50, 10]$), and varying the tightness $p2$ over the range $[50, 100]$, to get problems of all possible difficulties [13]. The range of parameter $p2$ was chosen so that it includes the phase transition region. A time limit was 200 seconds per every tuple of parameters. The presented

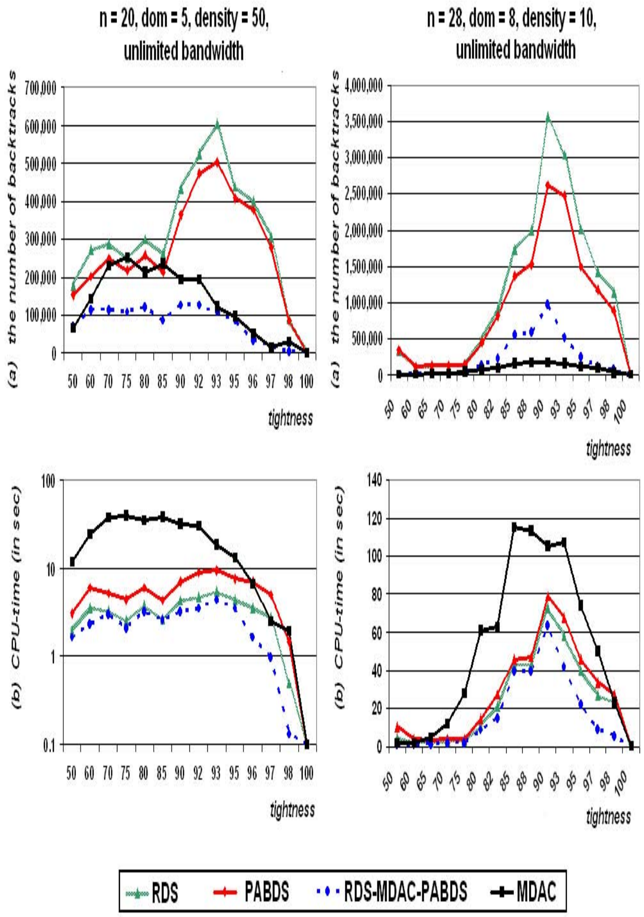

Figure 4. Binary Random MAX-CSP**s: density=50 (on the left), density=10 (on the right).**

three set of instances are: $\langle n = 20, dom = 5, p1 = 90\rangle$ in Figure 3 on the left, $\langle n = 20, dom = 5, p1 = 50\rangle$ and $\langle n = 28, dom = 8, p1 = 10\rangle$ in Figure 4. The set of instances of limited bandwidth b $\langle n = 20, dom = 5, p1 = 90, b = 5\rangle$ is given in Figure 3 on the right. The tightness $p2$ is presented along the horizontal axis, the actual number of backtracks and the CPU-time are presented on the vertical axis. The empirical results drawn from figures are as follows.

PABDS *outperforms* RDS in the number of backtracks, achieving a reduction of up to 42% for all the considered values of density and over the whole range of tightness for unlimited and limited bandwidth; however, it does not reduce the CPU-time.

RDS-MDAC-PABDS *outperforms* both RDS and PABDS in the number of backtracks for all the considered values of density and over the whole range of tightness. In some cases the rate of improvement by RDS-MDAC-PABDS over RDS

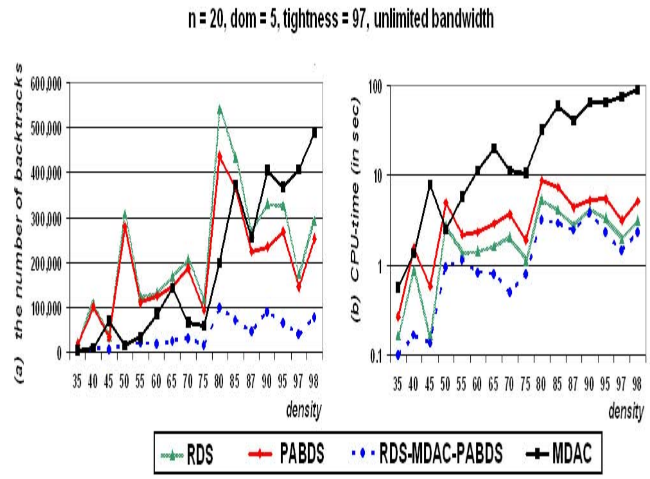

Figure 5. Binary Random MAX-CSP**s: tightness=97.**

is 4.4 times for density=90, 12 times for density=50, 16 times for density=10, when bandwidth is unlimited. That is, the rate of improvement in the number of backtracks by RDS-MDAC-PABDS over RDS is increased with decreasing of density. The obvious reason for that is the integration of MDAC into RDS-MDAC-PABDS. In particular, in the case of unlimited bandwidth, for density=90 MDAC has the largest number of backtracks; for density=50 MDAC beats RDS and PABDS in the number of backtracks in the most cases, giving up for RDS-MDAC-PABDS; for density=10 MDAC takes the least number of backtracks. That is, the reduction in the number of backtracks of MDAC compared with all other algorithms causes better performance of RDS-MDAC-PABDS.

The improvement in the CPU-time by RDS-MDAC-PABDS compared with RDS in case of unlimited bandwidth is as follows: for density=90 it is achieved for high values of tightness starting from 97; for density=50 and density=10 it performs over the whole range of tightness achieving up to $2-3$ times reduction in some cases. Note that the CPU-time consumed by MDAC is the highest for all types of density. Why does RDS-MDAC-PABDS improve the runtime of RDS? The explanation is that the number of variables, for which $SumDM$ is computed, is increased one by one, and frequently backtracking is initiated *before* $SumDM$ is computed for a considerably large number of variables, which prevents the MDAC component of RDS-MDAC-PABDS from incurring time penalties. To summarize, in case of unlimited bandwidth RDS-MDAC-PABDS *outperforms* all other algorithms (RDS, PABDS, and MDAC) in terms of the CPU-time as follows: in the case of high values of density for high values of tightness, and in the case of middle and low values of density over the whole range of tightness.

RDS-MDAC-PABDS achieves the highest rate of improvement in search efforts, given unlimited bandwidth and high values of tightness $p2$ over the whole range of density $p1$, outperforming all other algorithms. This conclusion is drawn from Figures 3 and 4 and shown separately in Figure 5 for $\langle n = 20, dom = 5, p2 = 97\rangle$, when density $p1$ is varied over the range $[35, 98]$.

MDAC performs better on problems with limited bandwidth (due to their lower connectivity), causing better performance of RDS-MDAC-PABDS. The more we limit the bandwidth b, the better rate of improvement in search efforts we achieve. In Figure 3 on the right we show the results for $\langle n = 20, dom = 5, p1 = 90, b = 5\rangle$. In this set MDAC performs better compared to other algorithms than in the case of unlimited bandwidth (Figure 3 on the left). The rate of improvement in the number of backtracks by RDS-MDAC-PABDS over RDS is up to 9 times. The improvement in the CPU-time by RDS-MDAC-PABDS over RDS is performed over the whole range of tightness, the reduction is $30 - 53\%$. Limiting the bandwidth in cases of middle and low values of density also allows us to obtain better results for RDS-MDAC-PABDS.

4.2 Earth Observation Satellite Scheduling Problems (SPOT5)

Daily management problems for an earth observation satellite (SPOT5) are large real scheduling problems, for which the idea of the Russian Doll Search was originally conceived and applied [16, 1]. The detailed description of SPOT5 problems as CSPs with valued variables is given in [1]. Let us just recall that given a set of photographs S (variables) having weights denoting importance (weights of variables), each photograph p is taken by different ways (domain $D(v)$ of corresponding variable v), there is the possibility of not selecting p (a special *rejection* value is added to $D(v)$), and a set of imperative constraints (binary and ternary) is defined. The task is to find an admissible subset S' of S (imperative constraints met) which maximizes the sum of the weights of the photographs in S'. That is, we find a partial solution that satisfies all the imperative constraints, whose weight (sum of weights of assigned variables) is maximum.

For all the tested algorithms we used the static variable ordering heuristic choosing the first variable according to the chronological photograph ordering, since the bandwidth of this ordering is naturally small in scheduling problems [16]. For the RDS-based algorithms we used the static value ordering heuristic, first choosing the value that the variable had in the optimal assignment found on the previous subproblem, then choosing the first available value [16]. For MDAC we chose the first available value.

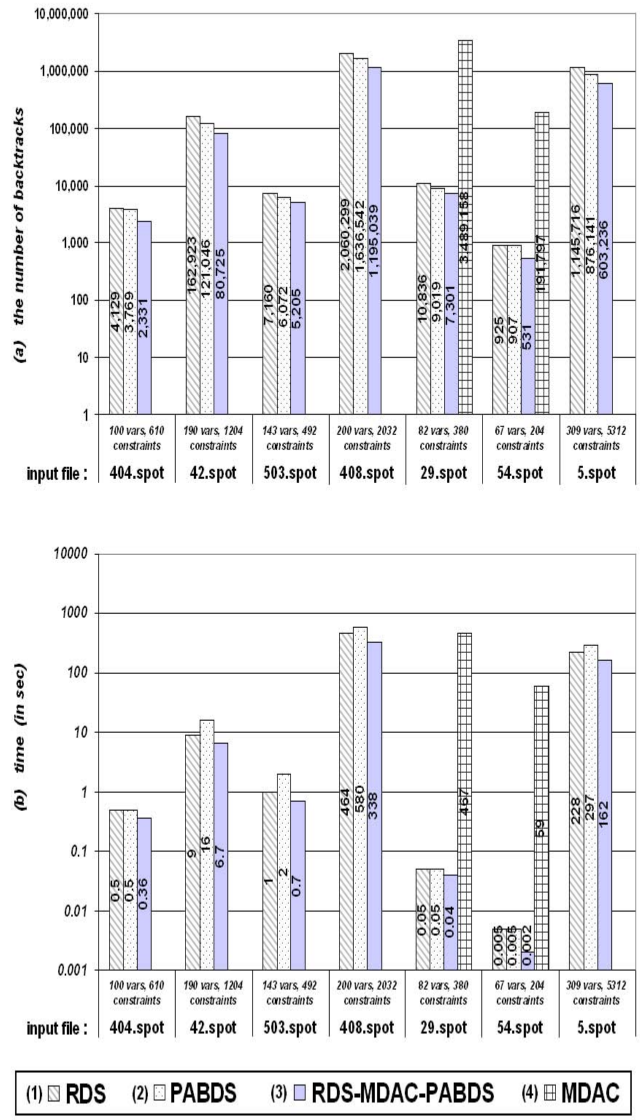

Figure 6. Experiments for SPOT5 problems.

We ran our experiments on the SPOT5 instances downloaded from the URL: *ftp://ftp.cert.fr/pub/lemaitre/LVCSP/Pbs/SPOT5/*. We translated ternary constraints in the given instances into binary ones using a standard transformation [15]. Because of the distribution of the possible images along each satellite resolution, some instances can be decomposed into independent sub-instances (no constraint between them), which can be solved separately. Figure 6 shows results for the largest independent sub-instances, when a time limit was 1800 seconds per this sub-instance. As MDAC does not solve most of the instances within the time limit, we omit its graphic description in that cases. The name of the SPOT5 instance, the number of variables and the constraints in the instance are presented along the horizontal axis. The actual number of backtracks and the CPU-time required to solve the instances are shown on the vertical axis.

The empirical results drawn from Figure 6 are as follows. PABDS *outperforms* RDS in the number of backtracks, achieving a reduction of up to 34%; however, it does not reduce the CPU-time, as in case of binary random MAX-CSPs. RDS-MDAC-PABDS *outperforms* all other algorithms (RDS, PABDS, and MDAC) in the number of backtracks and in the CPU-time. The rate of improvement by RDS-MDAC-PABDS over RDS in number of backtracks is 2 times in some cases. The reduction in CPU-time achieved by RDS-MDAC-PABDS over RDS is up to 40%. That is, RDS-MDAC-PABDS *outperforms* all algorithms over all studied instances. The savings in search efforts are smaller than for binary random MAX-CSPs, but the improvements are nonetheless clear.

5 Conclusion

In this paper we presented two new B&B-based algorithms for solving the WCSP problem. These algorithms can be considered as modifications of the well-known RDS algorithm. In particular, when computing the lower bound for the current partial solution, the proposed algorithms decide which one of the already solved subproblems would make the best contribution to the lower bound computation. The first proposed algorithm, PABDS, selects only the "big" subproblems which include variables assigned by the current partial solution. The second proposed algorithm, RDS-MDAC, looks into the opposite direction and considers the "small" subproblems, i.e. those that include only unassigned variables. The contribution of the unassigned variables that do not belong to the selected subproblem is evaluated based on directed arc-inconsistency counts according to the MDAC method. The empirical evaluation shows that RDS-MDAC-PABDS, the hybrid of RDS-MDAC and PABDS, outperforms both RDS and MDAC, and hence evidences that combining local inconsistency counts and the RDS-based measure is a promising approach for the lower bound evaluation.

An interesting direction of future research is to combine RDS with methods of soft arc-consistency [3, 4] that transform the given CN into an equivalent one having tighter efficiently computable lower bounds. Such a combination would potentially have a better performance than the combination of RDS with MDAC. However, it is not trivial to design such a hybrid algorithm because the CN transformation may affect the lower bound provided by RDS.

References

[1] E. Bensana, M. Lemaître, and G. Verfaillie. Earth Observation Satellite Management. *Constraints*, 4(3):293–299, 1999.

[2] K. C. K. Cheng and R. H. C. Yap. Search Space Reduction and Russian Doll Search. In *AAAI*, pages 179–184, 2007.

[3] M. C. Cooper and T. Schiex Arc consistency for soft constraints. *Artif. Intell.*, 154(1-2):199–227, 2004.

[4] S. de Givry, F. Heras, M. Zytnicki, and J. Larrosa. Existential arc consistency: Getting closer to full arc consistency in weighted CSPs. In *IJCAI*, pages 84–89, 2005.

[5] E. C. Freuder and R. J. Wallace. Partial Constraint Satisfaction. *Artif. Intell.*, 58(1-3):21–70, 1992.

[6] J. Larrosa and P. Meseguer. Exploiting the Use of DAC in MAX-CSP. In *CP*, pages 308–322, 1996.

[7] J. Larrosa, P. Meseguer, and T. Schiex. Maintaining Reversible DAC for Max-CSP. *Artif. Intell.*, 107(1):149–163, 1999.

[8] J. Larrosa and T. Schiex. In the quest of the best form of local consistency for Weighted CSP. In *IJCAI*, pages 239–244, 2003.

[9] Javier Larrosa and Pedro Meseguer. Partition-Based Lower Bound for Max-CSP. *Constraints*, 7(3-4):407–419, 2002.

[10] Javier Larrosa and Thomas Schiex. Solving weighted CSP by maintaining arc consistency. *Artif. Intell.*, 159(1-2):1–26, 2004.

[11] P. Meseguer and M. Sánchez. Specializing Russian Doll Search. In *CP*, pages 464–478, 2001.

[12] P. Meseguer, M. Sánchez, and G. Verfaillie. Opportunistic Specialization in Russian Doll Search. In *CP*, pages 264–279, 2002.

[13] P. Prosser. An empirical study of phase transitions in Binary Constraint Satisfaction Problems. *Artif. Intell.*, 81(1-2):81–109, 1996.

[14] E. Rollon and J. Larrosa. Multi-Objective Russian Doll Search. In *AAAI*, pages 249–254, 2007.

[15] E. P. K. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.

[16] G. Verfaillie, M. Lemaître, and T. Schiex. Russian Doll Search for Solving Constraint Optimization Problems. In *AAAI/IAAI, Vol. 1*, pages 181–187, 1996.

[17] R. J. Wallace. Directed Arc Consistency Preprocessing. In *Constraint Processing, Selected Papers*, pages 121–137, 1995.