Russian Doll Search for Solving Constraint Optimization Problems

Gérard Verfaillie and **Michel Lemaître**
CERT/ONERA
2 av. Edouard Belin, BP 4025
31055 Toulouse Cedex, France
{verfaillie, lemaitre}@cert.fr

Thomas Schiex
INRA
Chemin de Borde Rouge, Auzeville, BP 27
31326 Castanet Tolosan Cedex, France
tschiex@toulouse.inra.fr

Abstract

If the *Constraint Satisfaction* framework has been extended to deal with *Constraint Optimization* problems, it appears that optimization is far more complex than satisfaction. One of the causes of the inefficiency of complete tree search methods, like *Depth First Branch and Bound*, lies in the poor quality of the lower bound on the global valuation of a partial assignment, even when using *Forward Checking* techniques. In this paper, we introduce the *Russian Doll Search* algorithm which replaces one search by n successive searches on nested subproblems (n being the number of problem variables), records the results of each search and uses them later, when solving larger subproblems, in order to improve the lower bound on the global valuation of any partial assignment. On small random problems and on large real scheduling problems, this algorithm yields surprisingly good results, which greatly improve as the problems get more constrained and the bandwidth of the used variable ordering diminishes.

The *Constraint Satisfaction* framework (*CSP*) is now widely used to represent and solve numerous *Artificial Intelligence* problems (planning, scheduling, diagnosis, design. . .). But it appears that most problems are combinations of *imperative requirements*, *user preferences* and *uncertainties*. They are not pure constraint *satisfaction* problems, but constraint *satisfaction* and *optimization* problems : some constraints have strictly to be met, the others have preferably to be met.

In order to take these aspects into account, several extensions of the classical framework have been proposed: *possibilistic*, *fuzzy*, *additive*, *probabilistic* . . . *CSP*. In (Bistarelli, Montanari, & Rossi 1995) and (Schiex, Fargier, & Verfaillie 1995), two general algebraic frameworks, that cover most of these extensions, have been proposed. We will use the second of these, the so-called *Valued Constraint Satisfaction Problems* framework (*VCSP*), as a basis for our work.

Depth First Branch and Bound, which aims at finding a provenly optimal solution, is the natural extension of the *Backtrack* algorithm in this framework. It is well known that the quality of the bound used by this kind of algorithm is crucial: the better the bound, the smaller the search tree. As shown in (Schiex, Fargier, & Verfaillie 1995), if *Backward* and *Forward Checking* are easily extended to provide bounds in the *VCSP* framework, it is much more difficult with *Arc Consistency*. Furthermore, if the bound provided by *Forward Checking* is better than the one provided by *Backward Checking*, it still remains too optimistic, especially at the beginning of the search, when few variables are assigned. As a consequence, the *Depth First Branch and Bound* algorithm may explore huge subtrees which do not contain any complete assignment better than the best found so far.

Experiments on small random *VCSPs* show that optimization is much more difficult than satisfaction: unlike in the classical *CSP* framework, complexity does not decrease beyond the frontier between consistent and inconsistent problems; worse, this frontier defines the beginning of a continuous and steep increase in complexity, as graph connectivity or constraint tightness increases. These results are confirmed by experiments on large real problems. Even when satisfaction can easily be decided, optimization on the same problems is often out of reach within a reasonable time.

The idea of the *Russian Doll Search* is to replace one search by n successive searches on nested subproblems, where n is the number of variables in the problem: given an ordering of the problem variables, the first subproblem involves the last variable only, the ith subproblem involves all the variables from the $n - i + 1$th variable to the last, and the nth subproblem is the whole problem. At first sight, replacing one search by n searches using the same *Depth First Branch and Bound* algorithm may seem counterproductive. But the key to the efficiency of this method lies in the recording, for each subproblem, of the quality of an optimal assignment, and the ability, when the same static variable ordering is used, to use this information to improve the bound provided by *Forward Checking*. This improvement, which is particularly noticeable at the beginning of the search, allows the search tree to be pruned much earlier. Each search being much shorter, the sum of the costs of the n searches is often better than the cost of a classical *Depth First Branch and Bound*, even if the latter can benefit from a dynamic variable ordering.

After an introduction to the *VCSP* framework, to the *Branch and Bound* algorithms, and to the various existing bounds, we describe the *Russian Doll* algorithm in detail. Then we consider some related algorithms and theoretical results. Finally, we present some experimental results obtained on small random problems and on large real scheduling problems.

Valued Constraint Satisfaction Problems

As a classical *CSP* is defined as a pair $P = (V, C)$, where V is a set of variables and C a set of constraints, a *VCSP* (Schiex, Fargier, & Verfaillie 1995) can be defined as a quadruple $P = (V, C, S, \varphi)$, where:

- V is a set of variables;
- C is a set of constraints;
- S is a valuation structure, which itself is a quintuple $(E, \succ, \perp, \top, \otimes)$, where E is a set, $\succ$ is a total order on E, $\perp$ is the minimum element in E, $\top$ is the maximum element in E and $\otimes$ is a binary closed operation on E, which satisfies the following properties: commutativity, associativity, monotonicity according to $\succ$, $\perp$ as identity and $\top$ as absorbing element;
- φ is an application from C to E.

The set E is used to define a gradual notion of constraint violation and inconsistency. The elements of E, so-called *valuations*, can be compared using the total order $\succ$ and combined using the operation $\otimes$. The minimum element $\perp$ is used to express constraint satisfaction and consistency, the maximum element $\top$ is used to express imperative constraint violation and complete inconsistency. The application φ assigns a valuation to each constraint. This valuation expresses the importance of the constraint (equal to $\top$ in case of imperative constraint).

Let A be a complete assignment and $C_{viol}(A)$ be the set of the constraints violated by A. The valuation $\varphi(A)$ of A is the combination of the valuations of all the constraints in $C_{viol}(A)$:

$$\varphi(A) = \bigotimes_{c \in C_{viol}(A)} \varphi(c)$$

The standard objective is then to find a complete assignment of minimum valuation. The valuation $\varphi(P)$ of a problem P is the valuation of such an assignment.

Let A be a partial assignment. Its global valuation is the minimum valuation obtainable by extending it on the unassigned variables or, in other words, the valuation of the problem P restricted by the partial assignment A. The valuation of a problem P is the global valuation of the empty assignment.

Specific frameworks, such as *classical CSP*, *possibilistic CSP*, *additive CSP*, *probabilistic CSP*, *lexicographic CSP*... can be produced by instantiating the valuation structure S. For example, in *additive CSP*, E is the set of the natural integers, enlarged with a special element $+\infty$, $\succ$ is the natural order $>$ on integers, $\perp = 0$, $\top = +\infty$ and $\otimes$ the usual addition $+$ ($>$ and $+$ being extended in order to take into account the special element $+\infty$). *Partial CSP*, studied in (Freuder & Wallace 1992), is a particular case of *additive CSP*, where all the constraints are non imperative and have the same valuation (equal to 1).

Branch and Bound methods

Backtrack is the usual algorithm to find a complete consistent assignment in the classical *CSP* framework. *Depth First Branch and Bound*, described in algorithm 1, is its natural extension to find a complete assignment of minimum valuation in the *VCSP* framework.

function DFBB(lb_{init}, ub_{init}, LOWER-BOUND)
return DFBBI(lb_{init}, ub_{init}, 0, LOWER-BOUND)

function DFBBI(lb_{init}, ub_{init}, *i*, LOWER-BOUND)
The problem to solve includes the variables in $]i..n]$. $i = 0$ with the usual DFBB algorithm. $0 \leq i \leq n-1$ with the RDS algorithm (see algorithm 2). An assignment A of the problem variables, of minimum valuation $\varphi(A)$, such that $lb_{init} \preceq \varphi(A) \prec ub_{init}$, is sought. If such an assignment exists, it is recorded and its valuation is returned; otherwise, **failure** is returned. For clarity's sake, it is assumed that all the variables have d values and that static variable and value orderings are used. LOWER-BOUND is a functional parameter which determines the way of computing a lower bound lb on the global valuation of a partial assignment. It is equal to LB_1 or LB_2 with the usual DFBB algorithm (see Equations 1 and 2) or to LB_3 with the RDS algorithm (see Equation 4). $]i..v]$ is the set of assigned variables. v is the current variable. ub is the valuation of the best assignment of the problem variables found so far.

$lb, ub, v, success \leftarrow \perp, ub_{init}, i,$ **false**

search-depth :
if $v = n$ {a better complete assignment has been found} **then**
    $success$, $ub \leftarrow$ **true**, lb {the upper bound ub is updated}
    record the current assignment A
    if $lb_{init} \prec ub$ **then go** search-width **else go** end-search
else
    $A[v \leftarrow v+1] \leftarrow 0$
    go search-width

search-width :
if $A[v] = d$ {all the domain values have been tried} **then**
    $v \leftarrow v - 1$ {a backtrack occurs}
    if *v*=*i* **then go** end-search **else go** search-width
else
    $A[v] \leftarrow A[v] + 1$
    $lb \leftarrow$LOWER-BOUND(A, i, v)
    if $lb \prec ub$ **then go** search-depth **else go** search-width

end-search :
if *success* **then return** ub **else return failure**

Algorithm 1: *Depth First Branch and Bound*

Let us assume that any complete assignment, whose valuation is higher than or equal to ub_{init}, is unacceptable and that it is known, for any reason, that there is no complete assignment, whose valuation is lower than lb_{init}. The problem is to find a complete assignment A, of minimum valuation $\varphi(A)$, such that $lb_{init} \preceq \varphi(A) \prec ub_{init}$. By default, $lb_{init} = \perp$ and $ub_{init} = \top$.

At any moment, the algorithm searches for a complete assignment whose valuation is lower than a current upper bound ub. This upper bound is initialized with the value ub_{init} and decreases during the search: each time a complete assignment, whose valuation is lower than ub, is found, its valuation is used as a new upper bound.

It maintains a lower bound lb on the global valuation of the current partial assignment A, and backtracks each time $ub \preceq lb$, since there then exists no complete extension of A whose valuation is lower than ub.

It ends when a complete assignment, whose valuation is equal to lb_{init}, is found or when no complete assignment whose valuation is lower than ub could be found.

There are three main ways of improving this algorithm: (1) to choose an adequate variable ordering to reduce the tree size or to rapidly increase the lower bound lb; (2) to choose an adequate value ordering to rapidly find good complete assignments and therefore decrease the upper bound ub; (3) to compute a high lower bound (function LOWER-BOUND) on the valuation of any partial assignment, to cut earlier unproductive branches. *Russian Doll Search* uses this last way to improve *Depth First Branch and Bound*.

Partial Assignment Valuation

Backward Checking is the simplest way of bounding the global valuation of a partial assignment A, by only taking into account the constraints which are assigned[1] by A. Let $C_{ass-viol}(A)$ be the set of constraints assigned and violated by A. We obtain the bound $\text{LB}_1(A)$, also called the local valuation of A:

$$\text{LB}_1(A) = \text{LB}_{bc}(A) = \bigotimes_{c \in C_{ass-viol}(A)} \varphi(c) \qquad (1)$$

Forward Checking is an attempt to improve this bound, by taking into account the constraints which are *quasi*-assigned by A *i.e.*, such that only one variable is unassigned. Let $V'(A)$ be the set of the variables which are not assigned by A. If v is an element of $V'(A)$, let $d(v)$ be its domain. If val is an element of $d(v)$, let $C_{qass-viol}(A, v, val)$ be the set of the constraints which are not completely assigned by A but assigned and violated by $A \cup \{(v, val)\}$. We obtain the bound $\text{LB}_2(A)$:

$$\begin{aligned} \text{LB}_2(A) &= \text{LB}_{bc}(A) \otimes \text{LB}_{fc}(A) \qquad (2)\\ \text{LB}_{fc}(A) &= \bigotimes_{v \in V'(A)} [\min_{val \in d(v)} \Delta\text{LB}_{bc}(A, v, val)] \\ \Delta\text{LB}_{bc}(A, v, val) &= \bigotimes_{c \in C_{qass-viol}(A,v,val)} \varphi(c) \end{aligned}$$

where $\Delta\text{LB}_{bc}(A, v, val)$ is the increase in local valuation which would result from the assignment of the value val to the variable v.

Since, for each A, $\text{LB}_1(A) \preceq \text{LB}_2(A)$, *Forward Checking* provides a better bound than *Backward Checking* and yields more pruning. Moreover, even when no backtrack is possible ($\text{LB}_2(A) \prec ub$), values can be safely removed from the domains of the unassigned variables. Let v be an unassigned variable and val be one of its values. This value can be removed if:

$$\begin{aligned} \text{LB}_{bc}(A) &\otimes \Delta\text{LB}_{bc}(A, v, val) \qquad (3)\\ &\otimes [\bigotimes_{v' \in V'(A)-\{v\}} [\min_{val' \in d(v')} \Delta\text{LB}_{bc}(A, v', val')]] \\ &\succeq ub \end{aligned}$$

But *Forward Checking* remains too optimistic, especially at the beginning of the search, since it does not take into account constraints between unassigned variables. In the *CSP* framework, *Arc Consistency* embedded in a *Backtrack* algorithm (Haralick & Elliot 1980; Sabin & Freuder 1994) does so. But, as it has been shown, in (Bistarelli, Montanari, & Rossi 1995) and (Schiex, Fargier, & Verfaillie 1995), extending it to the *VCSP* framework is difficult, at least with a strictly monotonic operation (*additive*, *lexicographic* or *probabilistic CSP*), since the corresponding problem becomes NP-complete. *Directed Arc Consistency Counts* (Wallace 1994) are a way of taking into account constraints between unassigned variables. *Russian Doll Search* can be seen as another simple, but powerful, way of doing so.

Russian Doll Search

The *Russian Doll Search*, described in algorithm 2, assumes a static variable ordering. It performs n successive searches on nested subproblems as russian dolls[2], where n is the number of variables in the problem. The first subproblem only involves the last variable, the ith subproblem involves all the variables from the $n-i+1$th variable and the nth subproblem is the whole problem. Each search uses the *Depth First Branch and Bound* algorithm described in Algorithm 1, with the same static variable ordering. The global result is obtained with the last search, but the optimal assignments of all the solved subproblems, along with their valuation, are systematically recorded in order to help future searches.

---

function RDS(ub_{init})
The problem to solve includes n variables. An assignment A of the problem variables, of minimum valuation $\varphi(A)$, such that $\varphi(A) \prec ub_{init}$, is sought. If such an assignment exists, its valuation is returned; otherwise, **failure** is returned. $rds[i]$ is the valuation of the subproblem limited to the variables in $]i..n]$.

$rds[n] \leftarrow \perp$
foreach i **from** $n-1$ **downto** 0 **do**
  $lb' \leftarrow rds[i+1]$
  $ub' \leftarrow$ UPPER-BOUND(ub_{init}, lb', i)
  $ub \leftarrow$ DFBBI(lb', ub', i,LB$_3$)
  if $ub \neq$ **failure then** $rds[i] \leftarrow ub$ **else return failure**
return ub

Algorithm 2: *Russian Doll Search*

First, each search can use the valuation of the previous subproblem to bound the valuation of the current one: let

---

[1] A constraint is assigned iff all its variables are assigned.

[2] We could also have called this method *Chinese Box Search*.

P' be the previous subproblem and P'' be the current one ($P' \subset P''$); let $\varphi(P')$ and $\varphi(P'')$ be their valuations and C' be the set of the constraints that appear in P'', but not in P', it is obvious that:

$$\varphi(P') \preceq \varphi(P'') \preceq \varphi(P') \otimes [\bigotimes_{c \in C'} \varphi(c)]$$

This inequation is used in the function UPPER-BOUND to give a better value of the initial upper-bound ub_{init}.

Each search can also use the optimal assignment produced by the previous search as a value heuristic for the current search: for each variable, try first the value it had in this previously optimal assignment.

Finally, and most importantly, since each search uses the same static order, it can use the valuations of all the previous subproblems to improve the bound on the global valuation of any partial assignment A. Let $P'(A)$ be the subproblem limited to the variables which are not assigned by A and $\varphi(P'(A))$ its previously recorded valuation[3]. We obtain the new bound $\text{LB}_3(A)$:

$$\begin{aligned} \text{LB}_3(A) &= \text{LB}_{bc}(A) \otimes \text{LB}_{fc}(A) \otimes \text{LB}_{rds}(A) \qquad (4) \\ \text{LB}_{rds}(A) &= \varphi(P'(A)) \end{aligned}$$

To be convinced that $\text{LB}_3(A)$ is a lower bound on the valuation of any partial assignment A, it is sufficient to observe that the subsets of constraints taken into account by each of its three components (LB_{bc}, LB_{fc} and LB_{rds}) are disjunct (provided there is no unary constraint) and that each component is itself a lower bound on the best valuation of the corresponding subset of constraints, for all the possible complete extensions of A.

In the particular case of binary *CSPs*, Figure 1 shows how constraints contribute to each component of the bound. In this case, we obtain a partition of the set of the problem constraints in three subsets. In the general case of n-ary *CSPs*, constraints which involve assigned and unassigned variables, but are not *quasi*-assigned, are still ignored by the bound LB_3.

Since, for each A, $\text{LB}_1(A) \preceq \text{LB}_2(A) \preceq \text{LB}_3(A)$, the bound provided by the *Russian Doll Search* is better than the bound provided by *Forward Checking* or *Backward Checking*. This improvement should be all the more dramatic as $P'(A)$ is large *i.e.*, at the beginning of the search, when the bound provided by *Forward Checking* or *Backward Checking* is very poor. Let us also note that, when no backtrack is possible, the component $\text{LB}_{rds}(A)$ can also be added to $\text{LB}_{bc}(A)$ in Equation 3 in order to remove more values from the domains of the unassigned variables.

Related algorithms and theoretical results

Let us note a similarity between the *Russian Doll Search* method and *Dynamic Programming*. Like *Dynamic Programming*, *Russian Doll Search* solves subproblems in order to solve the whole problem. But, whereas *Dynamic Programming* directly combines the results obtained on subproblems to get the result of the whole problem, *Russian Doll Search* only uses them as bounds during its search. Note that if no constraint is imposed on the constraint graph structure (such as a tree or hyper-tree structure), *Dynamic Programming* would need exponential space to solve *VCSPs*.

[3]In algorithms 1 and 2, A is the assignment of the variables in $]i..v]$, $P'(A)$ is the subproblem limited to the variables in $]v..n]$ and $\varphi(P'(A)) = rds[v]$.

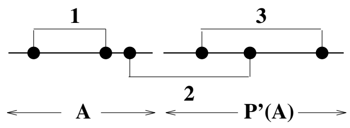

Figure 1: *Constraints taken into account by each component of the bound: (1) Backward Checking, (2) Forward Checking, (3) Russian Doll Search (previous search)*

Let us also note a similarity between the *Russian Doll Search* method and the *Depth First Iterative Deepening* algorithms (Korf 1985). As with *Depth First Iterative Deepening*, search is performed at increasing depth. But the main difference is the use, with the *Russian Doll Search*, of the results of the previous iterations in order to improve the lower bound on the global valuation of any partial assignment.

It is moreover easy to recover a traditional result of the *Iterative Deepening* algorithms: since, on a problem involving n variables and d values per variable, a *Depth First Branch and Bound* search examines in the worst case d^n terminal nodes, the n successive searches of the *Russian Doll Search* examine in the worst case $N = d + d^2 + \cdots + d^n$ terminal nodes; if $d > 1$, $0 < \frac{1}{d} < 1$ and we have:

$$N = d^n \cdot \sum_{i=0}^{n-1} (\frac{1}{d})^i < d^n \cdot \sum_{i=0}^{+\infty} (\frac{1}{d})^i = d^n \cdot \frac{1}{1 - \frac{1}{d}} = d^n \cdot \frac{d}{d-1}$$

The ratio between the number of examined terminal nodes, in the worst cases, by the *Russian Doll Search* and the usual *Depth First Branch and Bound* is then less than $\frac{d}{d-1}$, a number which is itself less than or equal to 2 and decreases as d increases.

In practice and as it can be observed in experiments on random and real problems (see the two next sections), the synergy between successive iterations of the *Russian Doll Search*, and mainly the improvement of the lower bound on the global valuation of a partial assignment, can rapidly offset this unfavourable ratio.

In order to better understand this phenomenon, let us consider two extreme cases: two partial binary *CSPs*, denoted by P_{loose} and P_{tight}, each of them involving n variables, d values per variable and $e = n(n-1)/2$ constraints (complete constraint graph); whereas each constraint of P_{loose} allows all the possible pairs of values, each constraint of P_{tight} allows none of them; the valuations of P_{loose} and P_{tight} are respectively equal to 0 and e.

With P_{loose}, the usual *Depth First Branch and Bound* only examines one terminal node, whereas the *Russian Doll Search* examines n. The ratio is equal to n on behalf of the former.

With P_{tight}, the usual *Depth First Branch and Bound* examines d^n terminal nodes with the lower bound LB_1 (*Backward Checking*) and still d^{n-1} with the lower bound LB_2 (*Forward Checking*). Since, in this case, LB_3 provides the *Russian Doll Search* with the exact global valuation e of each assignment of the first variable, the *Russian Doll Search* examines nd terminal nodes. The ratio is now equal to n/d^{n-2} on behalf of the latter.

These examples show that, if the *Russian Doll Search* may be more expensive in some cases, it may bring exponential savings in other cases.

Randomly Generated Problems

We first present results which have been obtained on small binary random *partial CSPs*. These problems have been randomly generated according to the model described in (Smith 1994), slightly modified in order to produce problems of limited bandwidth. Let us recall that the four parameters of this model are n (the number of variables), d (the domain size, equal for all the variables), c (the graph connectivity) and t (the constraint tightness, equal for all the constraints).

Let us also recall that the bandwidth of a graph (Zabih 1990) is the minimum bandwidth on all its possible vertex orderings and that the bandwidth of an ordering is the maximum distance, according to this ordering, between two connected vertices. Computing the bandwidth of a graph is an NP-complete problem.

The method we chose for generating *CSPs* of limited bandwidth b was as follows: given n, d, c, t and b, let O be a variable ordering; let S be the set of all the pairs of variables, which are separated by a distance lower than or equal to b according to O; randomly select $c{\cdot}n(n-1)/2$ pairs of variables in S and randomly generate a binary constraint of tigthness t for each selected pair.

This method guarantees that the bandwidth of the ordering O, and therefore the graph bandwidth, is lower than or equal to b. Let us note that not all the combinations of n, c and b are allowed: a small bandwidth implies a small connectivity since the following relation holds between n, c and b: $(n-b)(n-b-1) \leq n(n-1)(1-c)$.

We compare two algorithms:

- a *Depth First Branch and Bound* (*bb*), using the lower bound LB_2 on the global valuation of a partial assignment defined in Equation 2 and the following dynamic variable and value orderings:
  - given the generation variable ordering, among the variables of minimum current domain size, choose the first variable of maximum degree;
  - if A is the partial current assignment and v the variable to be assigned, choose the first value which minimizes $\Delta\text{LB}_{bc}(A, v, val)$ *i.e.*, the value which yields the smallest increase in local valuation of the current partial assignment;
- a *Russian Doll Search* (*rds*), using the lower bound LB_3 on the global valuation of a partial assignment defined in Equation 4, the static generation variable ordering and the following dynamic value ordering:
  - first choose the value the variable had in the optimal assignment found on the previous subproblem, then use the same heuristics as with the *Depth First Branch and Bound* algorithm.

These orderings are, in each case, the most efficient of those we experimented. Algorithms have been written in *Common Lisp* and tests have been performed with the *CMUCL* implementation on a *Sparc 5* workstation with $32Mb$ of memory.

In order to get a first global view of the relative efficiency of *bb* and *rds*, we carried out preliminary experiments with $n = 20$, $d = 5$, $c = 0.1, 0.3, 0.5, 0.7, 0.9$, $t = 0.1, 0.3, 0.5, 0.7, 0.9$, $b = 4, 10, 16$, and 20 randomly generated problems for each combination of c, t and b. In Figure 2, we point, using a plus sign, at the combinations of c, t and b for which *rds* is better than *bb*, in terms of the number of problems more rapidly solved.

$c\downarrow$	$t\rightarrow$ / $b\downarrow$	0.1	0.3	0.5	0.7	0.9
0.1	4	-	-	-	-	+
	10	-	-	-	-	-
	16	-	-	-	-	-
0.3	4	-	-	+	+	+
	10	-	-	-	-	+
	16	-	-	-	-	+
0.5	10	-	-	-	+	+
	16	-	-	-	+	+
0.7	10	-	-	+	+	+
	16	-	-	+	+	+
0.9	16	-	-	+	+	+

Figure 2: *A global view of the areas where the Russian Doll Search is more efficient than the usual Depth First Branch and Bound*

It appears that *rds* is more efficient than *bb* on the most inconsistent problems (great values of c and t), which are also the hardest to solve. It also appears that graph bandwidth is important, since, for a given combination of c and t, *rds* may be more efficient than *bb* with only a small value of b.

In order to judge the influence of the inconsistency degree of the problem on the relative efficiency of *bb* and *rds*, we carried out more systematic experiments with $n = 20$, $d = 5$, $c = 0.7$, $b = 10$, t varying from 0.1 to 0.9 in steps of 0.1, 100 randomly generated problems for each value of t and a time limit of 500 seconds per problem. Note that, with these problems, the frontier between consistency and inconsistency is about $t = 0.2$. Figure 3 shows, for both algorithms, the mean *cpu* time and the number of problems solved within the time limit.

It appears that *rds* gets more efficient as constraint tightness increases. Moreover, whereas the number of problems solved by *bb* suddenly diminishes as constraint tightness

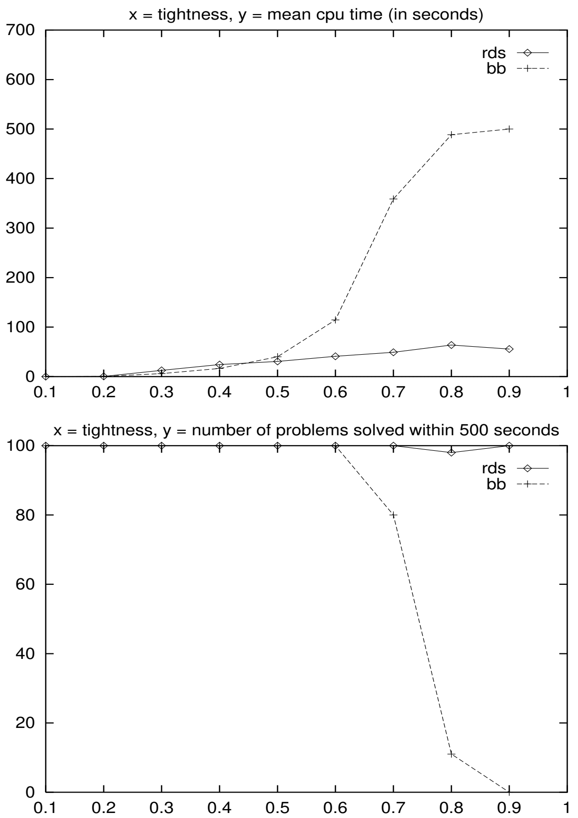

Figure 3: *Influence of constraint tightness*

increases, nearly all the problems generated are solved by *rds* within 500 seconds.

In order to assess the influence of the graph bandwidth on the relative efficiency of *bb* and *rds*, we performed other experiments with $n = 20$, $d = 5$, $c = t = 0.5$, b varying from 6 to 18 in steps of 2 and 100 randomly generated problems for each value of b. Figure 4 shows the mean *cpu* time for both algorithms. It appears that graph bandwidth is important, since *rds* becomes less efficient as it increases.

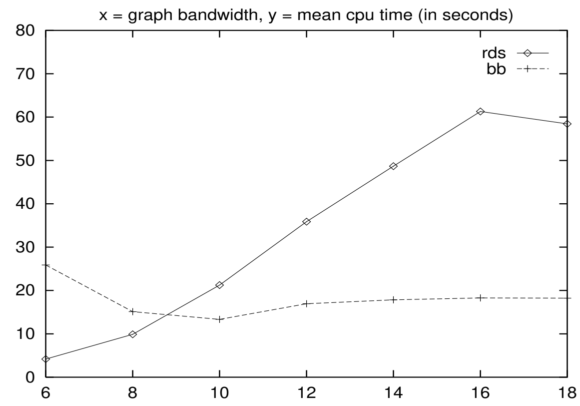

Figure 4: *Influence of graph bandwidth*

Earth Observation Satellite Scheduling Problems

We now present the results which have been obtained on large real scheduling problems and, more precisely, on daily management problems for an earth observation satellite, for which the idea of the *Russian Doll Search* was originally conceived (Agnèse *et al.* 1995). These problems can be roughly described as follows:

- given a set S of photographs which can be taken the next day from at least one of the three instruments, *w.r.t.* the satellite trajectory; given, for each photograph, a weight expressing its importance;
- given a set of imperative constraints: non overlapping and minimal transition time between two successive photographs on the same instrument, limitation on the instantaneous data flow through the satellite telemetry;
- find an admissible subset S' of S (imperative constraints met) which maximizes the sum of the weights of the photographs in S'.

They can be casted as *additive CSPs* by:

- associating a variable v with each photograph p; associating with v a domain d to express the different ways of achieving p and adding to d a special value, called *rejection* value, to express the possibility of not selecting p; then associating with v an unary constraint forbidding the *rejection* value, with a valuation equal to the weight of p;
- translating as imperative constraints (binary and ternary) the constraints of non overlapping and minimal transition time between two photographs on the same instrument, and of limitation on the instantaneous data flow;

On these problems, we compare:

- *bb* using the lower bound LB_2 on the global valuation of a partial assignment and the following variable and value orderings:
  - among the variables of maximum weight, choose the first according to the chronological photograph ordering; when only one value remains in a variable domain, immediately assign this variable this value (dynamic ordering);
  - last choose the *rejection* value in each domain (static ordering);
- *rds* using the lower bound LB_3 on the global valuation of a partial assignment, slightly modified in order to take into account unary constraints, and the following variable and value orderings:
  - choose the first variable according to the chronological photograph ordering, since the bandwidth of this ordering is naturally small in scheduling problems (static ordering);
  - first choose the value the variable had in the optimal assignment found on the previous subproblem; when this value is not the *rejection* value, last choose the *rejection* value (static ordering);

These orderings are, in each case, the most efficient we experimented. Figure 5 shows the results we obtained on eight typical problems (data can be downloaded from ftp://ftp.cert.fr/pub/lemaitre/LVCSP/Pbs/SPOT5/). A row is associated with each problem. The three following columns give the number n of variables, the number e of constraints and the bandwidth b of the chronological ordering for the largest independent subproblem[4]. The four last columns give the results of *bb* and *rds*, using a time limit of 1800 seconds per independent subproblem. For each algorithm, the first column shows the valuation of the best assignment found within the time limit and the second one the *cpu* time used to get this result. A $*$ sign points out provenly optimal valuations.

			bb		*rds*	
n	e	b	val	time	val	time
100	610	33	48	1800	49*	0.5
199	2032	62	3076	1800	3082*	14
300	4048	77	15078	1800	16102*	29
364	9744	150	21096	1800	22120*	86
105	403	30	8095	1800	9096*	2.5
240	2002	59	12088	1800	13100*	15
311	5421	103	12110	1800	15137*	55
348	8276	134	19104	1800	19125*	106

Figure 5: *Results on earth observation satellite scheduling problems*

It appears that, whereas *bb* solves none of the eight problems within the time limit, *rds* solves all of them. It also appears that it solves them all the more easily as the bandwidth of the chronological ordering it uses is small.

Conclusion

Although *Depth First Branch and Bound* tree search algorithms allow *Constraint Optimization* problems to be dealt with, extensions of *Backward* and *Forward Checking*, which do not take into account constraints between unassigned variables, provide the search with bounds, which are too optimistic, so long as it does not reach the depths of the tree. The practical result is an enormous waste of time and an inability to solve even small problems within a reasonable time.

The *Russian Doll Search* algorithm we have described in this paper is an extremely simple and yet powerful method of taking into account constraints between unassigned variables and then of providing the search with realistic bounds from the beginning. Although it must perform n searches, each of them using a static variable ordering, the *Russian Doll Search* may outperform the usual *Depth First Branch and Bound* algorithm, even if the latter uses the best dynamic variable orderings. It has been observed that this improvement is all the more dramatic as problems are more inconsistent and the bandwidth of the used variable ordering is small. This allows large, previously unsolvable, scheduling problems to be easily solved.

This algorithm certainly deserves further improvements, including the minimization of the bandwidth of the used variable ordering, the use of a *Constraint Satisfaction* algorithm as long as subproblems are consistent and of a mixed static and dynamic variable ordering (static at the beginning of the search, dynamic as soon as the subproblems defined by the unassigned variables are consistent and their valuations cannot improve the bound).

[4] Some problems can be decomposed in independent subproblems, which can be solved sequentially.

Acknowlegments

This work was done at CERT-ONERA and supported by the French Space Agency (CNES). We are indebted to Denis Blumstein and Jean Claude Agnèse, from CNES, who first conceived, implemented and experimented the *Russian Doll Search* method for earth observation satellite scheduling problems, and to Eric Bensana, from CERT-ONERA, who helped us to generalize it to the *Constraint Optimization* framework.

References

Agnèse, J.; Bataille, N.; Bensana, E.; Blumstein, D.; and Verfaillie, G. 1995. Exact and Approximate Methods for the Daily Management of an Earth Observation Satellite. In *Proc. of the 5th ESA Workshop on Artificial Intelligence and Knowledge Based Systems for Space*.

Bistarelli, S.; Montanari, U.; and Rossi, F. 1995. Constraint Solving over Semirings. In *Proc. of IJCAI-95*, 624–630.

Freuder, E., and Wallace, R. 1992. Partial Constraint Satisfaction. *Artificial Intelligence* 58:21–70.

Haralick, R., and Elliot, G. 1980. Increasing Tree Search Efficiency for Constraint Satisfaction Problems. *Artificial Intelligence* 14(3):263–313.

Korf, R. 1985. Depth-First Iterative Deepening: An Optimal Admissible Tree Search. *Artificial Intelligence* 27:97–109.

Sabin, D., and Freuder, E. 1994. Contradicting Conventional Wisdom in Constraint Satisfaction. In *Proc. of ECAI-94*, 125–129.

Schiex, T.; Fargier, H.; and Verfaillie, G. 1995. Valued Constraint Satisfaction Problems : Hard and Easy Problems. In *Proc. of IJCAI-95*, 631–637.

Smith, B. 1994. Phase Transition and the Mushy Region in Constraint Satisfaction Problems. In *Proc. of ECAI-94*, 100–104.

Wallace, R. 1994. Directed Arc Consistency Preprocessing. In *Constraint Processing (Lecture Notes in Computer Science 923)*. Springer. 121–137.

Zabih, R. 1990. Some Applications of Graph Bandwidth to Constraint Satisfaction Problems. In *Proc. of AAAI-90*, 46–51.