

MARS: a hybrid scheme based on Resolution Search and Constraint Programming for Constraint Satisfaction Problems.

Mireille Palpant¹, Christian Artigues², Cristian Oliva¹

¹Departamento de Ingeniera Industrial, Edmundo Larenas 215, Concepción, Chile

²Université de Toulouse, LAAS-CNRS, 7 avenue du Colonel Roche, 31077
Toulouse Cedex 4, France

mireille.palpant@orange.fr, artigues@laas.fr, oliva@udec.cl

Abstract

In this paper, we present a solution approach based on Chvátal's Resolution Search [2] to solve combinatorial optimization problems. Resolution Search constitutes an alternative to classical enumeration methods and possesses strong connections to nogood recording approaches, and in particular dynamic backtracking, though it is designed to deal with binary linear programming problems. Accordingly, we suggest to hybridize the procedure using constraint programming techniques, in order to apply it to Constraint Satisfaction Problems (CSP). Furthermore, the introduction of such techniques allows some specific improvements that speed up the process and let many outcomes for further research.

In order to evaluate the interest of the proposed method, we use it to tackle some particular graph coloring instances, known as the Queens- n^2 problem. The experimental results obtained so far prove the validity of the approach and compete with state-of-the-art complete solution methods.

1 Introduction

In 1997, Chvátal [2] proposes a new solution method, called Resolution Search, dedicated to binary linear programming problems. Unlike classi-

cal implicit enumeration procedures, this approach is based on an original exploration of the search space. Indeed, like intelligent backtracking approaches, such as among others dynamic backtracking [5], it involves learning to identify and memorize fails occurring all along the search. More precisely, whenever a fail is detected at a given node of the search tree, the (partial) instantiation conducting to the fail is memorized as a nogood in order to (hopefully) discard the whole subtree below the incriminated node for the rest of the search process. Resolution Search allows to manage these nogoods so to limit both memory space and computational time and indicates how to continue the search after a failure, while preserving completeness.

To our knowledge, this procedure was only applied once, by Demassey et al. [3], to the widely studied resource-constrained project scheduling problem. The results they obtain by strictly applying the procedure seem to prove its effectiveness towards a classical enumeration procedure but still, it remains widely dedicated to binary linear problems. Thus, our goal is twofold: first, to apply the approach to CSPs; and secondly, to confirm its validity through experimental results. In order to meet those two requirements, the original procedure has been integrated with constraint programming techniques and applied to the $Queens_{n^2}$ problem.

The paper is organized as follows: in Section 2, Resolution Search principles are introduced along with some definitions and notations that will be used throughout the paper, afterwards we present our new hybrid scheme. Section 3 is devoted to the $Queens_{n^2}$ problem, including preliminary results obtained by a solution approach based on the previously described method. Finally, Section 4 presents additional improvements conjuncted to extensive experimental results and clues for further research.

2 An hybrid scheme based on Resolution Search to solve CSPs

2.1 Resolution Search for combinatorial optimization

As previously said, Resolution Search constitutes an alternative to classical implicit enumeration methods, and more precisely branch-and-bound. Indeed, it is more likely to first explore the deep levels of the search tree before exploring its top. More precisely, the method exploits failure information to exclude subspaces of increasing size until excluding the entire search space.

This mechanism is based on the ground resolution refutation principle [11], which aims to generate a sequence of clauses leading to optimality proof.

Resolution Search starts by involving a classical enumeration process, in a descent move all along the tree, until a terminal node is reached, i.e. a node whose subtree does not contain any improving solution. At this point, a backtrack phase is initiated in order to find an ascendant node that still satisfies the failure condition. Since Resolution Search is dedicated to combinatorial optimization, this "waning phase" has to take into account the optimization criterion and is not as trivial as in intelligent backtracking processes. Thus, all decisions taken during the descent phase are reversed one by one, starting from the last but one. If a node appears to be non-terminal, the associated decision is re-established and the process goes on with the preceding decision. At the end of the backtracking move, the clause corresponding to the minimal terminal node is recorded and the descent or "waxing phase" is initiated again from another point of the search space. Both phases alternate until the root of the search tree is proven to be terminal. At this point, the problem is solved, and the best solution encountered so far is optimal.

It is worth noting that, due to specific backtrack and restart mechanisms, the search space is more likely represented by a graph rather than a classical tree (i.e., during the waning phase, the nodes are not explored along the descent branch as the last decision is kept). This specificity hopefully leads the method to be less dependent to branching rules.

In order to avoid reconsidering bad regions of the search space in the future, one could decide to memorize all the encountered terminal nodes. This may induce an exponential memory usage and a loss of computational time so to ensure that the search does not move towards a previously explored region. Resolution Search, on the contrary, is based on the use of a family of clauses whose properties allow (hopefully) to avoid these two drawbacks. Indeed, the heart of the procedure lies in the way this family is managed: its specific path-like structure enables clauses aggregation as well as a quick determination of the new starting point of the search.

2.2 Nogood management

Let us consider the following minimization problem:

$$P : \min f(X), X = (x_1, \dots, x_n), AX \geq b, X \in \{0, 1\}^n \quad (1)$$

A partial instantiation u of the variables of this problem is given by a vector

$$u = (u_1, \dots, u_n) \in \{0, 1, *\}^n \quad (2)$$

where u_i corresponds to the value assigned to variable x_i with $u_i = *$ if x_i is unassigned. u is associated to a value $oracle(u_1, \dots, u_n)$ equivalent to the optimal value of the relaxed problem.

Resolution Search works on a family $F = \{C^1, C^2, \dots, C^M\}$ of clauses, i.e. partial instantiations u whose associated search space is removed from further exploration. To simplify the notations, we express u under the form of a set of literals:

$$\{x_i | i \in I_0\} \cup \{\bar{x}_i | i \in I_1\} \quad (3)$$

where x_i and \bar{x}_i correspond respectively to indices $i \in I_0$ such that $u_i = 0$ and $i \in I_1$ such that $u_i = 1$. For example, partial instantiation $(1, *, *, 1, *, 0)$ will be represented by $\bar{x}_1 \bar{x}_4 x_6$. The empty clause $(*, \dots, *)$, denoted \emptyset , corresponds to the root of the tree.

To guide the search towards a region not included in any of the subspaces induced by the clauses of F , Resolution Search exploits the notion of extension, which defines a partial order relation over $\{0, 1, *\}^n$, denoted \sqsubseteq : $u \sqsubseteq v$ if the search space associated to v is included on the one associated to u (v constitutes an extension of u). For example, $\bar{x}_1 \bar{x}_4 x_6 x_7$ constitutes an extension of $\bar{x}_1 \bar{x}_4 x_6$. Resolution Search considers partial instantiations that are not extensions of any of the clauses in F . Consequently, at least one literal of each clause has to be reversed in the partial instantiations that are to be examined in the future.

Thus, Resolution Search associates to clauses $\{C^1, C^2, \dots, C^M\}$ of family F the literals l^1, l^2, \dots, l^M that have to be reversed in the continuation of the search. Furthermore, it maintains a particular structure, called path-like, to F by supplying to the following properties:

$$l^i \in C^j \iff i = j \quad (4)$$

$$\bar{l}^i \in C^j \implies j > i \quad (5)$$

$$\forall l, \bar{l} \in C^i, \bar{l} \in C^j : l = \bar{l}^i \text{ or } l = \bar{l}^j \quad (6)$$

Thanks to these properties, we can associate a clause u_F to family F , by defining the new starting point of the descent phase, such that:

$$u_F = (\bigcup_{k=1}^M (C_k \setminus \{l^k\} \cup \bar{l}^k)) \quad (7)$$

It is easy to verify that u_F is well-defined, i.e. that it does not exist any literal l such that $l \in u_F$ et $\bar{l} \in u_F$. Moreover, and trivially, u_F is not an extension of any clause C^k of F (since $\bar{l}^k \in u_F$ and $l^k \in C^k$).

In addition, the descent phase ends when encountering a nogood u' that constitutes an extension of u_F . Since minimal nogood u is deduced from u' by unassigning some variables, u' is also an extension of u . As $u \sqsubseteq u'$ and $u_F \sqsubseteq u'$, the following property is verified:

$$l \in u_F \Rightarrow \bar{l} \notin u \quad (8)$$

This observation assumes that the path-like structure of F is maintained while updating it with nogood u .

The way F is updated depends on the nature of nogood u . In the case u is obtained after a descent phase, it is simply added to F and it is easy to verify that it exists a literal $l \in u \setminus u_F$ that we can associate to u (since at least one literal has been added during the descent phase and that the last decision is kept). This operation trivially maintains the path-like structure of F . On the contrary, if u has not been obtained after a descent phase, that means, if u_F itself induces a failure, then $u \sqsubseteq u_F$ and it is impossible to determine $l \in u \setminus u_F$. F is also updated by a more complicated process.

Basically, if we consider two clauses u and v , differing from each other by only one literal x , for example $\bar{x}_1\bar{x}_4x_6$ and $\bar{x}_1\bar{x}_4\bar{x}_6$, then those two clauses are said in clash and we can define $C = u \nabla v$ as resolvent of u and v , such that:

$$C = u \setminus \{x\} \cup v \setminus \{\bar{x}\} \quad (9)$$

It is clear that, if u and v are the two only children of C , and if their associated search space is about to be removed from further exploration, then those two clauses can advantageously be replaced by C itself. Next, if all the children of the root of the tree are terminal, the root itself is proven to be a terminal node and the best solution encountered so far is optimal. This principle is extended in Resolution Search to reduce F with u when the latter covers part of the search space induced by a clause from F . This iterative process, involving series of resolvents, allows to save memory space at a low computational time. Moreover, although the excluded search space region delimited by the new family may not cover entirely the previous one (i.e. the procedure may visit previously explored solutions), it is strictly more extended and yields completeness.

For a more formal description of convergence proof and management of the nogood family, we refer to [2].

2.3 Example

In order to illustrate the Resolution Search principle, we apply it to a simple minimization problem containing three variables, x_1, x_2, x_3 . Let us now suppose that the lower bound value is equal to 0 if $x_1 = 1$ and $x_3 = 0$ or one of these two variables is unassigned, 1 otherwise. Finally, we consider that branching over variables follows the lexicographic order while branching over values first tries value 0, then 1.

The search begins from the root until computing the complete solution $(0, 0, 0)$. From this point, the new incumbent solution is recorded, fixing an upper bound value of 1, and the waning phase is initiated. It begins by unassigning x_2 $(0, *, 0)$. The lower bound still being equal to 1, the search goes on, unassigning x_1 $(*, *, 0)$, obtaining a new lower bound equal to 0. Nogood x_1x_3 is then recorded and the process is iterated by inverting a decision, for example $(0, *, 1)$. Since this node is terminal (the lower bound is equal to 1), associated nogood $x_1\bar{x}_3$ is used to update F : the two clauses are replaced by their resolvent $x_1x_3 \nabla x_1\bar{x}_3 = x_1$. The search continues from $(1, *, *)$ until new best solution $(1, 0, 0)$ is reached, updating then the upper bound value to 0. Consequently all nodes are now terminal, since it is impossible to compute any partial solution with a lower bound value lesser than 0. We then skip the end of the execution since F will successively been reduced by series of resolvents until it becomes empty.

This approach is very similar to intelligent backtracking processes, and in particular dynamic backtracking [5]. The main difference is that nogoods are not just kept for their relevance to the current partial instantiation, but are directly used to reorient the search (i.e. the next partial instantiation is build according to these nogoods). A simple example is as follows: considering x_1x_2 , the nogood extracted from partial instantiation $(0, 0, *, 1, 0, 0, 1)$ after a first descent phase, with x_2 the last assigned variable, dynamic backtracking goes on by considering $(0, 1, *, 1, 0, 0, 1)$ while Resolution Search restarts the search from $(0, 1, *, *, *, *, *)$. It thus enables a mobility even bigger in the search of solutions.

2.4 Hybridizing Resolution Search with constraint programming

In this section we describe precisely the Resolution Search-based method we propose to solve CSPs. Note also that optimization problems can be

reduced to CSPs. The main idea is to exploit Resolution Search guiding of the search within a backtracking framework. On one hand, some problems seem to be solved more effectively by constraint programming rather than linear programming and it might then be prejudicious to solve them by strictly applying Resolution Search procedure. On the other hand, Resolution Search involves an original and seductive methodology that seems to be efficient compared with classical enumeration schemes, not only regarding execution times but also the number of nodes expanded during the search [4].

Let us consider the following CSP:

$$CSP : (X, D, C), X = (x_1, \dots, x_n), x_i \in D_i \forall i \quad (10)$$

where C denotes the set of constraints of CSP .

As explained before, the search will be performed within a constraint programming framework involving consistency algorithms to cut the search space. Consequently, the algorithm maintains a set C_F of the assignment constraints throughout the whole process.

Associating to every variable x_i a set of binary variables y_{ij} , we can easily verify the following assertion:

$$\{(x_i \neq v) : i \in I_0\} \cup \{(x_i = v) : i \in I_1\} \iff \{y_{iv} : i \in I_0\} \cup \{\bar{y}_{iv} : i \in I_1\} \quad (11)$$

This property allows us to switch between the constraint and linear programming formulations involved during the distinct phases of the algorithm, whose general procedure is given in Figure 1.

At each decision point, a variable x_i is chosen among the non-fixed variables and assigned to a value v of its domain D_i , following ad-hoc strategies that have to be specified according to the considered problem. C_F is then updated with assignment constraint $(x_i = v)$ and the new information is propagated. This consists in checking whether or not the new partial instantiation is consistent within the initial constraints of the problem by reducing the domain of non-instantiated variables. The process is performed by a call to function $\text{Propagate}(C_F, D, C)$ which returns FALSE if an inconsistency is detected, TRUE otherwise. In the first case, minimal nogood C_{min} in case of the failure is determined and converted to clause u which is used to update F as explained in Subsection 2.2. Clause u_F and associated set C_F , constituting the new starting point of the search, are then generated. The whole process is iterated until F becomes empty or a complete solution is found.

Begin

1. $F = \emptyset$; $C_F = \emptyset$;
2. **While** Propagate(C_F, D, C)=TRUE et $\exists i \mid x_i = *$
3. Chose $x_i = *$ and assign it to a value $v \in D_i$
4. $C_F = C_F \cup \{(x_i = v)\}$
5. **If** all variables are assigned
6. Return C_F
7. **Else**
8. Determine nogood $C_{min} \subseteq C_F$ and associated clause u
9. Update F with u
10. **If** $F = \emptyset$ **Then**
11. Return \emptyset
12. **Else**
13. Determine u_F and C_F
14. Goto 2.

End

Figure 1: *General algorithm of the proposed method*

As a first advantage, the proposed method enables an automatic identification of failure reasons. Indeed, Resolution Search takes into account the optimization criterion, which complicates the detection of the minimal nogood. On the contrary, constraint programming, since it always maintains local consistency, allows to immediately identify the partial instantiation in case of fail. Indeed, a failure occurs whenever a constraint is violated by the current partial instantiation. Trivially, it is sufficient to check out the variables invoked in this constraint to identify the associated nogood. In our method, this mechanism replaces advantageously the waning phase, which may hopefully lead to a speed up of the global process.

A second advantage is that this hybrid scheme allows to solve pure CSPs with dedicated tools that are likely to tackle them efficiently, in particular constraint propagation techniques.

Note that this hybridization is closely linked to the experiments that have been successfully carried out to associate intelligent backtracking techniques with constraint propagation such as [7].

3 Solving the Queens $_n^2$ problem with Resolution Search

In this section we propose an innovating approach since it deals with the application of Resolution Search to a CSP. Our choice concerns the Queens $_n^2$ problem that can be expressed in the simplest way with the help of constraint programming.

In Subsection 3.1, we briefly describe this problem as well as the used model and make a short review of exact approaches encountered in the literature. Next, we detail all the specific points of the approach in Subsection 3.2. Finally, some experimental results are shown in Subsection 3.3.

3.1 The Queens $_n^2$ problem

Considering a $n \times n$ chess board game, we can define a graph $G = (V, E)$, namely the queen graph, composed of n^2 vertices, each corresponding to a case of the board, and whose arcs represent the queen move rules: two vertices are connected by an arc if the cases they represent are located in the same row, column or diagonal.

The coloring problem consists on assigning a color to each vertice assuming that two adjacent vertices are assigned different colors. The objective studied in this paper deals with the determination of the chromatic number of a queen graph of size n^2 , and more precisely, the question if it is equal to n . Obviously, as such a graph possesses maximal cliques of size n , the problem lies in finding a valid n -coloration for this graph.

In our study, we consider the simplest model that consists in determining the vector $C = \{c_1, \dots, c_{n^2}\}$ of the colors assigned to the vertices of the graph, under the following constraints:

$$c_i \in \{1, \dots, n\} \quad \forall i \in V \quad (12)$$

$$c_i \neq c_j \quad \forall (i, j) \in E \quad (13)$$

While coloring problems are intensively studied, few references deal with that particular case. We refer to [6] for a complete study over coloring problems and methods involved to tackle them.

Regarding complete solution methods for the Queens $_n^2$ problem, several approaches are encountered. Caramia and Dell'Olmo [1] propose a complex algorithm which iteratively intends to extend the coloring of a maximum

clique. As for Mehrotra and Trick [8], and later Vasquez [12], they work on the independent set formulation of the problem. The first authors present a column generation approach involving an efficient algorithm to solve the problem arising in the column generation process. Vasquez begins by generating all the possible n -sized independent sets with the help of a depth-first search invoking forward checking before searching for a complete covering of the chess board with an implicit enumeration method associated with constraint propagation rules. It is worth noting that an incomplete version of the latter approach, involving specific symmetry rules that remove part of the global search space, has obtained the best results ever on this problem, solving all instances up to $n = 28$, excepted for $n = 27$. Nevertheless, best exact methods fail so far to solve instances for $n > 14$.

3.2 Specification of Resolution Search for the Queens- n^2 problem

The general algorithm begins by fixing one of the main diagonal and performing adequate constraint propagation before entering the process described in Subsection 2.4. At each decision point, a vertice i not fixed yet is selected and assigned to a color c_i of its domain D_i . As a general strategy, the process intends to assign the colors one by one to the distinct vertices: it first tries to fix all first-colored queens, then second-colored queens, . . . , and so on until all colors have been filled up. More precisely, at each decision point, we select a vertice whose domain contains the corresponding color and perform the assignment. In case several vertices are candidate, the tie is broken by selecting the first vertice in the lexicographic order.

Apart from that explicit branching scheme, another branching mechanism is involved implicitly by the Resolution Search component. Indeed, selecting the literal associated to a clause entering the nogood family amounts to guiding the search along the reverse decision. After a descent phase, the last assigned variable is memorized, since it is directly in case of the failure (and consequently appears in the nogood) and satisfy to condition (6). In the other case (after a reduction phase), the first literal in the lexicographic order satisfying the required conditions is chosen.

Another key point of the method concerns the consistency algorithm employed to propagate new decisions. This works on the maximal cliques of the graph restricted to rows, columns and diagonals, and consists in a "light"

version of AllDiff algorithm [10]. Since the scope of this paper deals with Resolution Search hybridization using constraint programming techniques, we don't give here the details of this algorithm. We refer to [9] for further information.

3.3 Experimental results

In this section, we present a first set of experimental results. The method has been assessed upon Queens- n^2 instances, n varying from 5 to 14. The code is written in C++ and experimentations have been conducted on a 2.4 GHz PC equipped with 1 Mo RAM.

In order to demonstrate the validity of our new hybrid scheme (RS), we first compared the obtained results with those of a classical backtracking method (CP). In order to ensure a fair comparison, all branching schemes and constraint propagation algorithms employed are strictly the same for the two approaches. In both cases, the process ends when reaching an execution time of one hour.

We report in Table 1 the number of expanded nodes and the computational times (in seconds) obtained by the two approaches over the set of instances for n from 5 to 11 (instances such that $n \leq 5$ are solved at the root of the tree, those for $n \geq 12$ are left unsolved by the two methods).

n	# nodes		CPU	
	(RS)	(CP)	(RS)	(CP)
5	1	1	0	0
6	10	4	0	0
7	17	24	0	0
8	633	15333	0	1
9	43646	14482595	14	576
10	2097498	-	1158	-
11	1386527	-	1115	-

Table 1: *First results for $n \leq 11$*

It is clear that the hybrid procedure performs better than the classical one. Indeed, excepted for instance Queens-6², both criterion values are much smaller for RS than for CP. Moreover, CP fails to solve instances such that

$n \geq 10$. This preliminary results seem to indicate the usefulness of letting Resolution Search conduct the search and confirm the conclusions stated by Demassez et al. [4].

In order to make a more valuable comparison, we also compared our procedure with some state-of-the-art exact methods. Then, we confront the obtained results with those of Mehrotra and Trick [8], Caramia and Dell’Ormo [1] and finally Vasquez [12].

Table 2 shows comparative results in terms of computational times (in seconds in case no other indication is reported) of all above approaches on instances for n varying from 5 to 14, since none of the presented methods is able to solve instances such that $n \geq 15$.

n	(RS)	[8]	[1]	[12]
5	0	0	0	0
6	0	1	0	0
7	0	4	0	0
8	0	19	0	0
9	14	515	0	0
10	1158	-	-	0
11	1115	-	-	1
12	-	-	-	6963
13	-	-	-	168 hours
14	-	-	-	131 hours

Table 2: *Comparative results for n from 5 to 14*

Though our results are far beyond those of Vasquez, they are really encouraging if we refer to the two other methods. Here, excepted for instance Queens. n^9 , the computational times are in favor of RS. Moreover, unlike these approaches, our procedure was able to solve Queens. 10^2 and Queens. 11^2 .

In addition to these encouraging results, handling Resolution Search within a constraint programming framework allows a major improvement of the procedure, whose details are given in the following section.

4 Extending the Resolution Search addition mechanism: MARS

In this section, we propose a new method, called MARS (for Multiple Additions Resolution Search), that exploits constraint programming advantages to tune adequately Resolution Search process.

Indeed, the use of constraint programming techniques allows us to incorporate many additional modules in the general framework. We then present a major improvement that speeds up the global process. The aim is to exploit more accurately the consistency information so to include additional nogoods to family F all along the descent phase.

The principle behind this improvement is detailed in Subsection 4.1. We then present in Subsection 4.2 the experimental results we obtain using this new version of the algorithm. Finally, some clues to further research that exploit the constraint programming aspect of the method are highlighted.

4.1 Including additional nogoods to F

At a given decision point, several possibilities are available to extend the current partial solution and continue the search. From all of these, only one choice is made, w.r.t. suitable selection rules, discarding all others possibilities, some of which that could lead to inconsistency. Our objective is to exploit all those alternate inconsistent branching possibilities to update family F while performing the descent process, following the mechanism detailed in Figure 2.

Once a branching variable x_i and its assignment value v_{elig} are selected, all other possible assignments are considered so to add to family F all the nogoods corresponding to choices that lead to inconsistent instantiations. In Step 7., if both complementary choices ($x_i = v$) and ($x_i \neq v$) induce failure, then the node immediately ascendent is itself terminal and F can be updated with associated nogood u . In the case the assignment ($x_i = v$) only leads to inconsistency, the minimal partial set C_{min} is computed as usual and its associated nogood u added to F . The process is then iterated until all values of domain D_i have been considered. At the end of the loop (if the process has not been exited at Step 7.), the descent phase goes on normally by branching on ($x_i = v_{elig}$).

Since the procedure only discards inconsistent instantiations, this tech-

Begin

1. Select branching variable x_i ;
 2. **For** all value $v \in D_i, v \neq v_{elig}$
 3. $C_F = C_F \cup \{(x_i = v)\}$
 4. **If** Propagate(C_F, D, C)=FALSE
 5. $C_F = C_F - \{(x_i = v)\} \cup \{(x_i \neq v)\}$
 6. **If** Propagate(C_F, D, C)=FALSE
 7. Update F with nogood u corresponding to $C_F = C_F - \{(x_i \neq v)\}$
 8. Return
 9. **Else**
 10. Update F with nogood u corresponding to $C_{min} \subseteq C_F$
 11. **Else**
 12. $C_F = C_F - \{(x_i = v)\}$
 13. Branch on $(x_i = v_{elig})$
 14. **End**
-

Figure 2: *Update of F at a given decision point*

nique does not remove regions of the search space that are likely to contain the optimal solution and allows then to maintain completeness. Moreover, this mechanism keeps the path-like structure of F and avoids recomputing each time a new starting point of the search, as the latter can be continued from the current node while respecting Resolution Search assertions.

At first sight, this technique presents many advantages, as several bad branching choices are discarded once in a row. First, the nogoods are generated all along the descent phase, without need of calculating each time the new starting point of the search. Secondly, these multiple additions are likely to activate a quicker closing of ascendant nodes. All of these characteristics could then lead to a global speed up of the process. In order to confront those intuitive assessments with reality, we applied the methodology to the Queens- n^2 problem. The obtained results are discussed in the following subsection.

4.2 Experimental results

The new procedure is obtained by adding the described module to the previous implementation. As before, the method has been assessed on instances

with n varying from 5 to 14.

In order to judge the efficiency of the proposed improvement, Table 3 presents the comparative results of the two versions of the method in terms of expanded nodes and computational times on instances such that $n \geq 5$ and $n \leq 11$.

n	# nodes		CPU	
	(MARS)	(RS)	(MARS)	(RS)
5	1	1	0	0
6	2	10	0	0
7	15	17	0	0
8	567	633	0	0
9	40406	43646	23	14
10	1997871	2097498	2049	1158
11	1336845	1386527	2106	1115

Table 3: *Results of the new version of the method for $n \leq 11$*

A spotting point is that MARS performs better than the original version of RS regarding the number of nodes expanded, while requiring much more computational times as the problem size increases slightly. A simple explanation probably lies in the fact that the operations realized on family F become more costly: for example, when the maximal family size is equal to 37 with the original version of the method, it reaches the value 78 in the new implementation! Moreover, it is coherent to think that some of these extra-additions only contribute to slow down the process, as the information they provide may be irrelevant, as for nodes located very deep in the tree. It appears indeed more interesting to keep clauses as small as possible in F . This observation leads us to consider a new improvement that performs these multiple additions only upon a given depth. Thus, a new parameter d is used to specify the depth from which the first version of the method is applied strictly. Figure 3 shows the evolution of both the number of expanded nodes and the computational time function to the value of d on instance Queens_9².

It is clear that the two curves vary in a reverse mode: as the number of expanded nodes decreases, the computational time increases. Moreover, the variations themselves seem to follow the same behavior. Indeed, to the limits, both curves behave in an asymptotic manner while varying exponentially in-between.

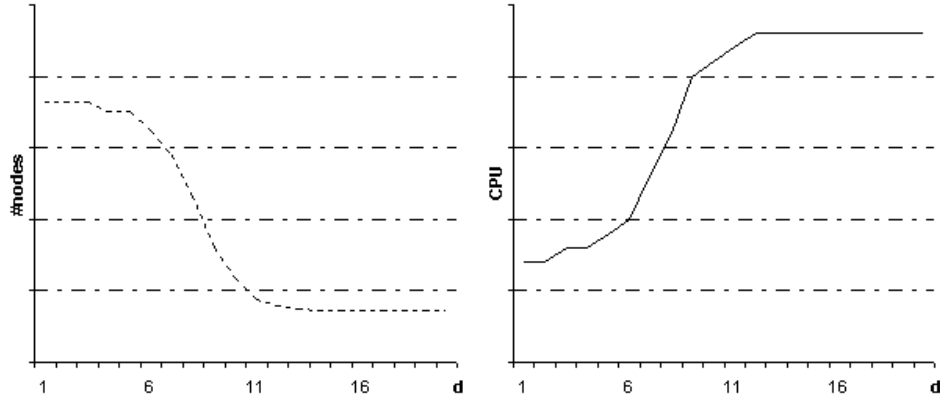


Figure 3: *Evolution of the number of expanded nodes and the computational times according to depth parameter d*

Two observations come out of those graphics: a small value of d does not provide a great improvement considering the number of expanded nodes, and it remains true for big values of d . The fact possesses a realistic justification since few additional clauses are likely to be added on top of the tree (as there may be too few fixed variables to induce inconsistency), as well as in the bottom of the tree (as most of the variables may possess very small domains, thanks to effective constraint propagation). Thus, it seems more interesting to apply the multiple additions methodology at depths ranging from an upper value d_{max} to a lower value d_{min} .

Empirically fixing d_{min} to 0 if $n < 11$, 1 otherwise, and $d_{max} = d_{min} + 5$, Table 4 displays the final results obtained by applying this methodology.

Though the method still fails to solve problems such that $n \geq 12$, it appears to be more effective regarding both criteria, particularly computational times, since they are almost divided by a factor of 2. However, in order to solve problems of larger size, the request to a heuristic procedure seem to be necessary. Constraint programming precisely offers various possibilities.

Indeed, in backtracking procedures, a guiding heuristic may be used to lead the search towards most promising regions. In particular, this heuristic allows to rank all the branching possibilities. The main idea from the restricted candidate lists is inspired from that principle. Thus, the methodology consists of removing from further search branches unlikely to contain good solutions, w.r.t. a discriminating or heuristic criterion. Generaliz-

n	# nodes		CPU	
	(MARS)	(RS)	(MARS)	(RS)
5	1	1	0	0
6	2	10	0	0
7	7	17	0	0
8	578	633	0	0
9	43498	43646	9	14
10	2096239	2097498	686	1158
11	1386524	1386527	547	1115

Table 4: *Results of the final version of the method for $n \leq 11$*

ing MARS' multiple additions idea, family F can easily be updated with the clauses corresponding to non promising partial instantiations, excluding then assignment values for variables that are likely to lead to non-improving solutions.

Another possibility of hybridization with heuristic components consists of delegating part of the search process to a (meta)heuristic method, in a scheme that could be seen as a Depth-Bounded Resolution Search, in the same spirit as the work presented in [13].

The approach can be summed up as follows: a parameter d determines the depth upon which MARS is applied as-if. Once this depth is reached, a heuristic process is then initiated in order to explore the sub-tree located under the current partial instantiation. To this end, many methods can be employed, all depending from the desired time/quality trade-off. Thus, these could range from the simplest greedy algorithm to the most sophisticated metaheuristic. In the latter however, it is necessary to conveniently restrict the search to the adequate sub-tree.

Once the sub-tree explored in a more or less partial way, the clause corresponding to the initiating partial instantiation is added to the nogood family and the process goes on normally.

An interesting feature of the approach lies on the suitable tuning of parameter d , since it influences directly the quality/time ratio and brings then a great flexibility in use. Designing such flexible methods constitutes a major goal of our research.

Acknowledgements

The authors would like to thank Sophie Demasse, whose previous work in the field highlighted the path, and Philippe Michelon and Michel Vasquez for their friendly support.

References

- [1] M. Caramia and P. Dell’Olmo. Constraint propagation in graph coloring. *Journal of Heuristics*, 8(1):83–107, 2002.
- [2] V. Chvátal. Resolution search. *Discrete Applied Mathematics*, 73:81–99, 1997.
- [3] S. Demasse. *Méthodes hybrides de programmation par contraintes et programmation linéaire pour le problème d’ordonnement de projet à contraintes de ressources*. PhD thesis, Université d’Avignon, 2003.
- [4] S. Demasse, C. Artigues, and P. Michelon. An application of resolution search to the remsp. In *17th European Conference on Combinatorial Optimization ECCO’04*, Beyrouth, Lebanon, june 2004.
- [5] M. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993.
- [6] J. P. Hamiez. *Coloration de graphes et planification de rencontres sportives : heuristiques, algorithmes et analyses*. PhD thesis, Université d’Angers, 2002.
- [7] N. Jussien, R. Debruyne, and P. Boizumault. Maintaining arc-consistency within dynamic backtracking. In *Principles and Practice of Constraint Programming (CP 2000)*, number 1894 in Lecture Notes in Computer Science, pages 249–261. Springer-Verlag, 2000.
- [8] A. Mehrotra and M. A. Trick. A column generation approach for graph coloring. *INFORMS Journal of Computing*, 8(4):344–354, 1996.
- [9] M. Palpant. *Recherche exacte et approchée en optimisation combinatoire : schémas d’intégration et applications*. PhD thesis, Université d’Avignon, 2005.

-
- [10] J.-C. Regin. A filtering algorithm for constraints of difference in csps. In *12th National Conference on Artificial Intelligence (AAAI-94)*, pages 362–367, 1994.
- [11] J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12:23–41, 1965.
- [12] M. Vasquez. New results on the queens- n^2 graph coloring problem. *Journal of Heuristics*, 10(4):407–413, 2004.
- [13] T. Walsh. Depth-bounded discrepancy search. In *Fifteenth International Joint Conference on Artificial Intelligence*, pages 1388–1395. Morgan Kaufmann, 1997.