Domain-Splitting Generalized Nogoods from Restarts[1]

Luís Baptista[α,β] (student) and Francisco Azevedo[α] (supervisor)

[α]CENTRIA, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, Portugal
[β]Instituto Politécnico de Portalegre, Portugal
lmtbaptista@gmail.com, fa@di.fct.unl.pt

Abstract. The use of restarts techniques associated with learning nogoods in solving Constraint Satisfaction Problems (CSPs) is starting to be considered of major importance for backtrack search algorithms. Recent developments show how to learn nogoods from restarts and that those nogoods are essential when using restarts. Using a backtracking search algorithm, with 2-way branching, generalized nogoods are learned from the last branch of the search tree, immediately before the restart occurs. In this paper we further generalized the learned nogoods but now using domain-splitting branching and set branching. We believe that the use of restarts and learning of domain-splitting generalized nogoods will improve backtrack search algorithms for certain classes of problems.

Keywords: constraint, restarts, learning, nogoods, domain-splitting, branching

1 Introduction

Constraint Satisfaction Problems (CSPs) are a well-known case of NP-complete problems [1]. They have extensive application in areas such as scheduling, configuration, timetabling, resources allocation, combinatorial mathematics, games and puzzles, and many other fields of computer science and engineering.

The impressive progress in propositional satisfiability problems (SAT) has been achieved using restarts and nogoods recording. SAT and CSP share many solving techniques [2]. As noted in [3] the interest of the CSP community in restarts and nogood recording is growing.

In this paper we extend the work of Lecoutre on nogood recording from restarts [4], that use a 2-way branching scheme. We present a new form of nogoods recorded from restarts, when a domain-splitting branching scheme is used. We call them domain-splitting generalized nogoods. This is a theoretical improvement on nogoods from restarts. We show that our proposed nogoods have potentially more pruning power. We also show that our nogoods can be used when a set branching scheme is used. This is an important issue since recently set branching has been shown to be an important technique.

[1] PhD beginning date: September 2009; PhD conclusion date: 2013

The rest of the paper is organized as follows. Section 2 gives background about constraint satisfaction problems, the search algorithm used and about restarts and learning. Section 3 presents the related work. In section 4 we present the state of the art on recording nogoods from restarts and in section 4 we present our contribution. Finally in section 6 we present conclusions and future work.

2 Background

2.1 Constraint Satisfaction Problem

A Constraint Satisfaction Problem (CSP) consists of a set of variables, each with a domain of values, and a set of constraints on a subset of these variables.

Based on [1, 5], we define more formally a CSP. Consider a set of variables $X = \{x_1,\ldots,x_n\}$ with respective domains $D = \{d_1,\ldots,d_n\}$ associated with them. Each variable x_i ranges over the domain d_i, not empty, of possible values. Now consider a set of constraints $C = \{c_1,\ldots,c_m\}$ over the variables X. Each constraint c_i involves a subset X' of X, stating the possible values combinations of the variables in X'. If the cardinality of X' is 1 we say that the constraint is unary, and if the cardinality is 2 we say that the constraint is binary. Hence, a CSP is a set X of variables with respective domains D, together with a set C of constraints.

Now we must define for the CSP what a solution is. A problem state is defined by an assignment of value to some (or all) variables. As an example, consider $\{x_i = v_i, x_j = v_j\}$, where v_k is one value of the domain d_k assigned to variable x_k, for $1 \le k \le n$. An assignment is said to be complete if every variable of the problem has a value (are instantiated). An assignment that satisfies every constraint (does not violate constraints) is said to be consistent, otherwise it is said to be inconsistent. So, a complete and consistent assignment is a solution to the CSP. A problem is satisfiable if at least one solution exists. More formally, if there exists at least one element from the set $d_1 \times \ldots \times d_n$ which is a consistent assignment. A problem is unsatisfiable if it does not have solution. Formally, in this case, all elements from the set $d_1 \times \ldots \times d_n$ are inconsistent assignments.

In this paper we will consider a CSP with finite domains (CP(FD)).

A propositional satisfiability problem (SAT) is a particular case of a CSP where the variables are Boolean, and the constraints are defined by propositional logic expressed in conjunctive normal form.

2.2 Search Algorithm

Search algorithms for solving a CSP can be complete or incomplete. Complete algorithms will find a solution, if one exists. If a CSP does not have a solution complete algorithms can be used to prove it. Backtrack search is an example of a complete algorithm. Incomplete algorithms may not be able to prove that a CSP does not have a solution, but may be effective at finding a solution if one exists. Local

search is an example of an incomplete algorithm. In this paper we will use a complete backtrack search algorithm.

A backtrack search algorithm performs a depth-first search. At each node a uninstantiated variable is selected based on a variable selection heuristic. The branches out of the node correspond to instantiating the variable with a possible value (or a set of values) from the domain, based on a value selection heuristics. The constraints are used to check whether the assignments are consistent.

At each node of the search tree, an important technique, known as constraint propagation, is used to improve efficiency by maintaining local consistency. This technique can remove, during the search, inconsistent values from the domains of the variables and therefore prune the search tree. Also notice that the usually very important heuristics for variable ordering may depend on the outcomes of the constraint propagation mechanism. The variable selection heuristic based on the fail-first principle is an example. At each node of the search tree this heuristic chooses the variable with the smallest domain size. This is a dynamic heuristic, since the constraint propagation mechanism removes inconsistent values from the domain, which will influence the next variable selection.

At each node of the search different branching schemes could be used. Two traditional and widely used branching schemes are the d-way and 2-way. In the first one, at each node, branches are created, one branch for each of the possible values of the domain of the variable associated with the node. Branches correspond to assignments of values to variables. In the 2-way branching scheme two branches are created out of each node. In this scheme a value v_k, is selected from the domain d_k of a variable x_k. The left branch corresponds to the assignment of the value to the variable, and the right branch is the refutation of that value. This can be viewed as adding the constraint $x_k = v_k$ to the problem, the left branch; or, if this fails, adding the constraint $x_k \neq v_k$ to the problem, the right branch. An important difference in these two schemes is that in d-way branching the algorithm has to branch again on the same variables until the values of the domain are exhausted. In 2-way branching, when a value assignment fails, the algorithm can choose to branch on any other unassigned variable.

Another branching scheme is domain splitting [6]. This scheme splits the domain of the variable into two sets, typically based on the lexicographic order of the values. Two branches are crated out of each node, one for each set. In each branch the other set of values is removed from the domain of the variable. The algorithm evolves by reducing the domains of the variables. An assignment occurs when the domain size of a variable is reduced to one. Note that this scheme results in a much deeper search tree. This is useful for optimization problems and when the domains sizes of the variables are very large.

Set Branching refers to any branching scheme that split the domain values in different sets, based on some similarity criterion [7]. The algorithm then branches on those sets. Note that 2-way and domains splitting branching schemes can be viewed as a particular case of set branching.

2.3 Restarts and Learning

A complete backtrack search algorithm is randomized by introducing a fixed amount of randomness in the branching heuristic [8]. Randomization is a key aspect of restart strategies. The utilization of randomization results in different sub-trees being searched each time the search algorithm is restarted.

For many combinatorial problems, different executions of a randomized backtrack search algorithm, on the same instance, can result in extremely different runtimes. This large variability in the runtime of the complete search procedures can be explained by the phenomena of heavy-tail distribution [8-10]. The heavy-tail distribution is characterized by long tails, as we can see in figure 1, for the 8-queens problem. The curve gives the cumulative fraction of successful runs as a function of the number of backtracks [10].

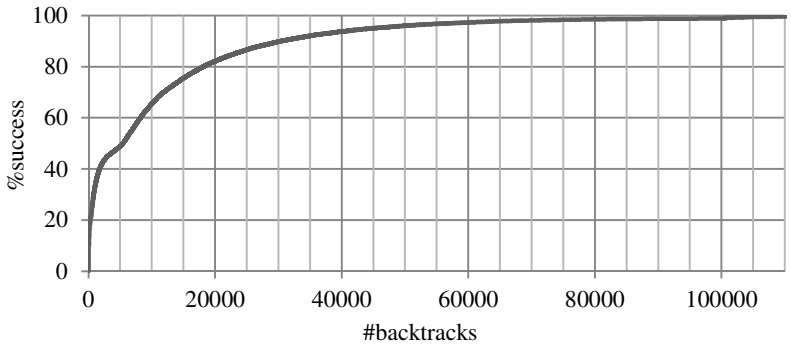

Fig. 1. Heavy-tail distribution example (8-queens)

A randomized complete search algorithm is repeatedly run (restart), each time limiting the maximum number of backtracks to a cutoff value. In practice a good cutoff values eliminates the heavy-tail phenomena, but unfortunately such a value has to be found empirically [8]. If restarts are used with a fixed cutoff value, the resulting algorithm is not complete. A solution to this problem is to implement a policy for increasing the cutoff value [11]. A simple policy is to increment by a constant the cutoff value after each restart. The resulting algorithm is complete, and thus able to prove unsatisfiability [12].

However, the incremental cutoff policy still exhibits a key drawback, because paths in the search tree can be visited more than once. This was addressed for SAT problems in [13] and for CSP in [4, 14], where nogoods are recorded, from the last branch of the search tree before the restart. Those recorded nogoods guarantee that the already visited search space will not be searched again.

3 Related Work

Learning in the context of SAT algorithms is known as conflict clause recording and in the context of CSP algorithms is known as nogood recording.

Nogood recording was introduced in [15], where a nogood is recorded when a conflict occurs during a backtrack search algorithm. Those recorded nogoods were used to avoid exploration of useless parts of the search tree.

Contrary to CSP, leaning is an important feature of SAT solver algorithms. Important progress in SAT solvers was due to the use of restarts, conflict clause recording [12, 16] and the use of very efficient data structures [16].

Standard nogoods correspond to variable assignments, but recently, a generalization of standard nogoods, that also uses value refutations, has been proposed by [17, 18]. They show that this generalized nogood allows learning more useful nogoods from global constraints. This is an important point since state of the art CSP solvers rely on heavy propagators for global constraints. The use of generalized nogoods significantly improves the runtime of the CSP algorithms. It is also important to notice that this generalized nogoods is much like clause recording in SAT solvers, and they show how to adapt other SAT techniques.

Recently the use of standard nogood and restarts in the context of CSP algorithms was studied [4, 14]. They record a set of nogoods after each restart (at the end of each run). Those nogoods are computed from the last branch of the search tree. So, the already visited tree is guaranteed not to be visited again. This approach is similar to one used for SAT, where clauses are recorded, from the last branch of the search tree before the restart (search signature) [13]. Recorded nogoods are considered as a unique global constraint with an efficient propagator. This propagator uses the 2-literal watching technique introduced for SAT. Experimental results show the effectiveness of this approach.

A hybrid CP(FD) solver that combines modeling and search of CP(FD) with learning and restarts of SAT solvers is proposed in [19]. The resulting solver is able to tackle problems that are beyond the scope of CP(FD) and SAT. They conclude that the combination of CP(FD) search with nogoods can be extremely powerful.

In SAT solvers the interplay of learning and restarts has proven to be extremely important for the success of the solvers. In the context of CP(FD) algorithms we are starting to study and understand this relation. But promising results show that learning is starting to be an interesting research area. As noted in [3] the impressive progress in SAT, unlike CSP, has been achieved using restarts and nogood recording (plus efficient lazy data structures). And this is starting to stimulate the interest of the CSP community in restarts and nogood recording.

4 Nogoods in 2-way Branching

In this section we explain the nogoods presented in [4]. Consider a search tree built by a backtracking search algorithm with a 2-way branching scheme. In figure 2 we can see an example of such a tree just before the restart occurs. The left branch is called

the positive decision and corresponds to an assignment. The right branch is called the negative decision and corresponds to a value refutation. A path in the search tree can be seen as a sequence of positive and negative decision.

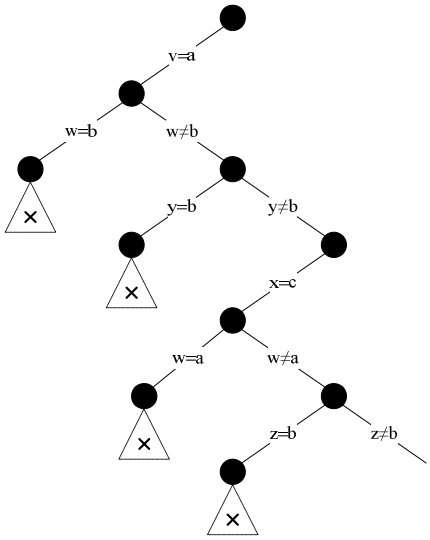

Fig. 2. Partial search tree before the restart, with 2-way branching.

Given a sequence of decision (positive and negative), an nld-subsequence (negative last decision subsequence) is a subsequence ending with a negative decision. As an example consider the sequence of decisions before the restart (the last branch of the search tree),

$$\langle v{=}a,\ w{\neq}b,\ y{\neq}b,\ x{=}c,\ w{\neq}a,\ z{\neq}b\rangle \tag{1}$$

The nld-subsequences that can be extracted from (1), are

$$\langle v{=}a,\ w{\neq}b\rangle \tag{2}$$

$$\langle v{=}a,\ w{\neq}b,\ y{\neq}b\rangle \tag{3}$$

$$\langle v{=}a,\ w{\neq}b,\ y{\neq}b,\ x{=}c,\ w{\neq}a\rangle \tag{4}$$

$$\langle v{=}a,\ w{\neq}b,\ y{\neq}b,\ x{=}c,\ w{\neq}a,\ z{\neq}b\rangle \tag{5}$$

A set of decisions is a nogood if they make the problem unsatisfiable. So, for any branch of the search tree a nogood can be extracted from each negative decision (nld-subsequence). Consider an nld-subsequence $\langle d_1, d_2, \ldots, d_i\rangle$, the set $\{d_1, d_2, \ldots, \neg d_i\}$ is a nogood (nld-nogood). The nld-nogoods that can be extracted from (1), one for each nld-subsequence (2-5) are,

$$\{v{=}a,\ w{=}b\} \tag{6}$$

$$\{v=a, w\neq b, y=b\} \tag{7}$$

$$\{v=a, w\neq b, y\neq b, x=c, w=a\} \tag{8}$$

$$\{v=a, w\neq b, y\neq b, x=c, w\neq a, z=b\} \tag{9}$$

As noted in [4] the nld-nogood corresponds to the definition of generalized nogoods [18], because it contains both positive and negative decisions. They also show that nld-nogoods can be reduced in size, considering only positive decisions. Consider an nld-subsequence $\langle d_1, d_2, \ldots, d_i\rangle$ and $Pos(\langle d_1, d_2, \ldots, d_i\rangle)$ denoting the set of positive decisions of the nld-subsequence, then the set $Pos(\langle d_1, d_2, \ldots, d_i\rangle) \cup \{\neg d_i\}$ is a nogood (reduced nld-nogood).

$$\{v=a, w=b\} \tag{10}$$

$$\{v=a, y=b\} \tag{11}$$

$$\{v=a, x=c, w=a\} \tag{12}$$

$$\{v=a, x=c, z=b\} \tag{13}$$

The advantages of using a reduced nld-nogood are more pruning power andreduction in space complexity. Also notice that the set of reduced nld-nogoods is equivalent to its original set of nld-nogoods. It consists of a more compact and efficient representation of nogoods.

5 Domain-Splitting Generalized Nogoods

In this section we give a generalization of the work presented in [4] about nogood recording from restarts. Adapting the same concepts we define a different kind of nogoods from restarts that use domain splitting instead of assignment. This work is a theoretical contribution for learning nogoods from restarts, in the context of backtrack search algorithms with a domain splitting branching scheme. Recall that we use finite domain.

Consider a search tree built by a backtracking search algorithm with a domain splitting branching scheme. As for the 2-way branching scheme this is also a binary tree. But now the domain is split lexicographically in one of the values. In figure 3 we can see an example of such a tree before the restart occurs (a, b, and c are integers). The left branch is called the positive decision and corresponds to constraining the variable to the left part of the domain (the less than or equal). The right branch is called the negative decision and corresponds to constraining the variable to the right part of the domain (the greater than). The right branch can be seen as negating the decision made in the left branch. A path in the search tree can be seen as a sequence of positive and negative decision.

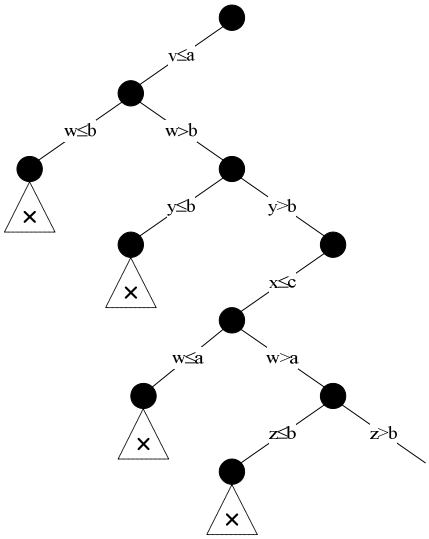

Fig. 3. Partial search tree before the restart, with domain-splitting branching.

Consider the sequence of decisions before the restart (the last branch of the search tree),

$$\langle v{\leq}a,\ w{>}b,\ y{>}b,\ x{\leq}c,\ w{>}a,\ z{>}b\rangle \tag{14}$$

In a similar way we can extract nld-subsequences from (14),

$$\langle v{\leq}a,\ w{>}b\rangle \tag{15}$$

$$\langle v{\leq}a,\ w{>}b,\ y{>}b\rangle \tag{16}$$

$$\langle v{\leq}a,\ w{>}b,\ y{>}b,\ x{\leq}c,\ w{>}a\rangle \tag{17}$$

$$\langle v{\leq}a,\ w{>}b,\ y{>}b,\ x{\leq}c,\ w{>}a,\ z{>}b\rangle \tag{18}$$

And the equivalent to nld-nogoods, that we call domain-splitting nogoods (ds-nogoods),

$$\{v{\leq}a,\ w{\leq}b\} \tag{19}$$

$$\{v{\leq}a,\ w{>}b,\ y{\leq}b\} \tag{20}$$

$$\{v{\leq}a,\ w{>}b,\ y{>}b,\ x{\leq}c,\ w{\leq}a\} \tag{21}$$

$$\{v{\leq}a,\ w{>}b,\ y{>}b,\ x{\leq}c,\ w{>}a,\ z{\leq}b\} \tag{22}$$

In the context of ds-nogoods a decision node splits the domain in two sets. In the context of nld-nogoods a decision node can be seen also as splitting the domain in two sets: in the positive decision branch the set has only one value, because of the assignment; and in the negative branch the set has the other values, because of the

refutation of the value. In this sense we can say that ds-nogoods are more powerful than nld-nogoods, because they use a more compact representation, since one positive decision can represent more than one value.

Similarly to reduced nld-nogoods, we can also have reduced ds-nogoods, considering only positive decision,

$$\{v \leq a,\ w \leq b\} \tag{23}$$

$$\{v \leq a,\ y \leq b\} \tag{24}$$

$$\{v \leq a,\ x \leq c,\ w \leq a\} \tag{25}$$

$$\{v \leq a,\ x \leq c,\ z \leq b\} \tag{26}$$

Again we can say that reduced ds-nogoods have potentially more pruning power than reduced nld-nogoods. Because they use a more compact representation, since one positive decision can represent more than one value.

5.1 Simplifying ds-nogoods

By construction, a CSP nogood does not contain two opposite decisions, e.g., $x \leq a$ and $x > a$. But a ds-nogood can have more than one decision on the same variable. As an example consider the decision over variable w in the ds-nogood (22), $w > b$ and $w > a$. It is easy to see, from the search tree in figure 3, that the decision $w > a$ subsumes $w > b$; because decision $w > a$ is made after $w > b$ we know that $a > b$. So for each ds-nogood (including the reduced version) a subsume procedure must be applied to remove unimportant decisions and thus simplify the nogood. For each variable, a ds-nogood only has to keep its last negative decision (if any) and its last positive decision (if any). In the case of reduced ds-nogoods, only the last (positive) decision has to be kept (for each of its variables). Thus, a great compaction can be obtained with such ds-nogoods.

5.2 Generalizing to dsg-nogoods

But ds-nogoods suffer from a possible key drawback, domain-splitting branching uses lexicographic order, which limits the expressive power of the search and consequently the learned nogoods. Namely, values are split in two sets lexicographically. It would be better for the sake of the flexibility of the search if the values could be split in any order.

We now assume a broad definition of domain-splitting branching scheme, which splits the domain D in two disjoint sets, s1 and s2 (i.e. $D = s1 \cup s2$), not necessarily in lexicographic order. The positive decision considers set s1 as the domain, in the left branch. If this fails the negative decision considers set s2 as the domain, in the right branch. More formally, for a variable x, the left branch corresponds to adding the constraint $x \in s1$, and the right branch to adding the constraint $x \in s2$ (the negative decision is $x \notin s1$ which is the same as $x \in s2$). The construction of the nogoods applies

trivially to this more generic case, and we call these dsg-nogoods (domain-splitting generalized nogood) and reduced dsg-nogoods.

Note that (reduced) nld-nogoods are a particular case of (reduced) dsg-nogoods, since we can simulate 2-way branching with this broad definition of domain-splitting. The assignment branch corresponds to a set with the value of the assignment. The refutation branch corresponds to a set with the remaining values of the domain.

Recent developments have shown the importance of backtracking search algorithms using set branching schemes [7, 20]. The use of restarts and dsg-nogoods (and the reduced version) in those algorithms is direct if we only have two sets. But even if we have more than two sets, we can use a 2-way style set branching [7], where the sets are tried in a series of binary choices. The positive decision considers one of the sets as the domain, if this fails the negative decision considers the removal of that set from the domain. The use of dsg-nogoods is thus direct.

6 Conclusions and Future Work

The utilization of restarts with nogoods recording in backtrack search algorithms for solving constraint satisfaction problems is starting to be considered of great importance. We present an extension of Lecoutre's work on recording nogoods from restarts, the so called nld-nogoods. This paper is a theoretical contribution. We generalized the nld-nogoods to work in the context of backtracking search algorithms with domain-splitting and set branching schemes. Additionally, we gave evidences that our proposed nogoods have potentially more pruning power. We call these new nogoods domain-splitting generalized nogoods (dsg-nogoods).

We believe that the use of restarts and nogoods in a backtracking search algorithm with domain-splitting branching scheme will boost the performance of the algorithm. The problems tackled by domain-splitting are mostly optimization problems and especially with bigger domain sizes. Recent successful applications of set branching to optimization problems [20] makes us believe that adding restarts and nogoods will improve the algorithm performance.

In the near future we expect to empirically evaluate the use of dsg-nogoods in the context of restarts. But this work is included in a wider research project, whose aim is to study the use of restarts in constraint programming with finite domains. We will evaluate the interplay of different techniques associated with restarts, namely, nogoods, search restart strategies, randomization and heuristics.

Acknowledgments. This work is supported by the Portuguese "Fundação para a Ciência e a Tecnologia" (SFRH/PROTEC/49859/2009).

References

1. Apt, K.R.: Principles of constraint programming. Cambridge University Press (2003).
2. Bordeaux, L., Hamadi, Y., Zhang, L.: Propositional Satisfiability and Constraint Programming: A comparative survey. ACM Comput. Surv. 38, 12 (2006).

3. Lecoutre, C.: Constraint Networks: Techniques and Algorithms. Wiley-ISTE (2009).
4. Lecoutre, C., Sais, L., Tabary, S., Vidal, V.: Nogood recording from restarts. Proceedings of the 20th international joint conference on Artifical intelligence. pp. 131-136. Morgan Kaufmann Publishers Inc., Hyderabad, India (2007).
5. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice Hall (2002).
6. Dincbas, M., Hentenryck, P.V., Simonis, H., Aggoun, A., Graf, T., Berthier, F.: The Constraint Logic Programming Language CHIP. FGCS'88. pp. 693-702 (1988).
7. Balafoutis, T., Paparrizou, A., Stergiou, K.: Experimental Evaluation of Branching Schemes for the CSP. 1009.0407. (2010).
8. Gomes, C.P., Selman, B., Kautz, H.: Boosting combinatorial search through randomization. Proceedings of the fifteenth national conference on Artificial intelligence. pp. 431-437. American Association for Artificial Intelligence, Madison, Wisconsin, United States (1998).
9. Gomes, C., Selman, B., Crato, N.: Heavy-Tailed Distributions in Combinatorial Search. Principles and Practices of Constraint Programming. 121-135 (1997).
10. Gomes, C.P., Selman, B., Crato, N., Kautz, H.: Heavy-Tailed Phenomena in Satisfiability and Constraint Satisfaction Problems. Journal of Automated Reasoning. 24, 67-100 (2000).
11. Walsh, T.: Search in a Small World. Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence. pp. 1172-1177. Morgan Kaufmann Publishers Inc. (1999).
12. Baptista, L., Silva, J.P.M.: Using Randomization and Learning to Solve Hard Real-World Instances of Satisfiability. Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming. pp. 489-494. Springer-Verlag (2000).
13. Baptista, L., Lynce, I., Marques-Silva, J.: Complete Search Restart Strategies for Satisfiability. IJCAI Workshop on Stochastic Search Algorithms (IJCAI-SSA). (2001).
14. Lecoutre, C., Saïs, L., Tabary, S., Vidal, V.: Recording and Minimizing Nogoods from Restarts. Journal on Satisfiability, Boolean Modeling and Computation. 1, 147–167 (2007).
15. Dechter, R.: Enhancement Schemes for Constraint Processing: Backjumping, Learning, and Cutset Decomposition. Artif. Intell. 41, 273-312 (1990).
16. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient SAT solver. Proceedings of the 38th annual Design Automation Conference. pp. 530-535. ACM, Las Vegas, Nevada, United States (2001).
17. Katsirelos, G., Bacchus, F.: Unrestricted Nogood Recording in CSP Search. CP. pp. 873-877 (2003).
18. Katsirelos, G., Bacchus, F.: Generalized NoGoods in CSPs. AAAI. pp. 390-396 (2005).
19. Feydy, T., Stuckey, P.J.: Lazy clause generation reengineered. Proceedings of the 15th international conference on Principles and practice of constraint programming. pp. 352-366. Springer-Verlag, Lisbon, Portugal (2009).
20. Kitching, M., Bacchus, F.: Set Branching in Constraint Optimization. Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI-2009). pp. 532-537 (2009).