Explanation-Based Weighted Degree

Emmanuel Hébrard, Mohamed Siala

Explanation-Based Weighted Degree

Emmanuel Hebrard[1] and Mohamed Siala[2⋆]

[1] LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France
hebrard@laas.fr
[2] Insight Centre for Data Analytics
Department of Computer Science, University College Cork, Ireland
mohamed.siala@insight-centre.org

Abstract. The weighted degree heuristic is among the state of the art generic variable ordering strategies in constraint programming. However, it was often observed that when using large arity constraints, its efficiency deteriorates significantly since it loses its ability to discriminate variables. A possible answer to this drawback is to weight a conflict set rather than the entire scope of a failed constraint.
We implemented this method for three common global constraints (AllDifferent, Linear Inequality and Element) and evaluate it on instances from the MiniZinc Challenge. We observe that even with simple explanations, this method outperforms the standard Weighted Degree heuristic.

1 Introduction

When solving a constraint satisfaction problem with a backtracking algorithm, the choice of the next variable to branch on is fundamental. Several heuristics have been designed to make this choice, often relying on the concept of fail firstness: "*To succeed, try first where you are most likely to fail*" [4].

The Weighted Degree heuristic (*wdeg*) [1] is still among the state of the art for general variable ordering search heuristics in constraint programming. Its principle is to keep track of how many times each constraint has failed in the past, with the assumption that constraints that were often responsible for a fail will most likely continue this trend. It is very simple and remarkably robust, which often makes it the heuristic of choice when no dedicated heuristic exists. Depending on the application domain, other generic heuristics may be more efficient, and it would be difficult to make a strong claim that one dominates the other outside of a restricted data set. Empirical evidence from the MiniZinc Challenge and from the CSP Solver Competition,[3] however, show that *wdeg* is among the best generic heuristics for classical CSP solvers (see [16] for an analysis of the reasons of its efficiency). It is also interesting to observe that SAT and clause-learning solvers often use Variable State Independent Decaying Sum ($VSIDS$) [10], which has many similarities with *wdeg*, whilst taking advantage of the conflicts computed by these algorithms.

⋆ Mohamed Siala is supported by SFI under Grant Number SFI/12/RC/2289.
[3] http://www.minizinc.org/challenge.html, http://www.cril.univ-artois.fr/CSC09

One drawback of Weighted Degree, however, is that the weight attributed to constraints of large arity has a weak informative value. In the extreme case, a failure on a constraint involving all variables of the problem does not give any useful information to guide search.

We empirically evaluated a relatively straightforward solution to this issue. We propose to weight the variables involved in a minimal conflict set rather than distributing it equally among all constrained variables. Such conflict sets can be computed in the same way as *explanations* within a clause learning CSP solver.

2 Background

A constraint network is a triplet $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ where $\mathcal{X}$ is a n-tuple of variables $\langle x_1, \ldots, x_n \rangle$, $\mathcal{D}$ is a mapping from variables to finite sets of integers, and $\mathcal{C}$ is a set of constraints. A constraint C is a pair $(\mathcal{X}(C), \mathcal{R}(C))$, where $\mathcal{X}(C)$ is tuple of variables and $\mathcal{R}(C) \in \mathbb{Z}^k$. The assignment of the k-tuple of values σ to the k-tuple of variables X satisfies the constraint C iff $\exists \tau \in \mathcal{R}(C)$ such that $\forall i, j \ X[i] = \mathcal{X}(C)[j] \implies \sigma[i] = \tau[j]$. A solution of a constraint network $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ is an assignment of a n-tuple σ to $\mathcal{X}$ satisfying all constraints in $\mathcal{C}$ with $\sigma[i] \in \mathcal{D}(x_i)$ for all $1 \leq i \leq n$. The constraint satisfaction problem (CSP) consists in deciding whether a constraint network has a solution.

Algorithm 1: SOLVE($P = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$)

1 **if** *not consistent(P)* **then** return `False`;
2 **if** $\forall x \in \mathcal{X}, \ |\mathcal{D}(x)| = 1$ **then** return `True`;
3 select $x \in \{x_i \in \mathcal{X}, |\mathcal{D}(x_i)| > 1\}$ and $v \in \mathcal{D}(x)$;
4 return Solve($P|_{x=v}$) or Solve($P|_{x \neq v}$);

We consider the class of *backtracking* algorithms, which can be defined recursively as shown in Algorithm 1. In Line 1, a certain property of consistency of the network is checked, through constraint propagation. This process yields either some domain reductions or, crucially, a *failure*. In the latter case, the last constraint that was processed before a failure is called the *culprit*. Line 2 detects that a solution was found. In Line 3, a heuristic choice of variable and value is made to branch on in Line 4, where $P|_{x=v}$ (resp. $P|_{x \neq v}$) denotes the constraint network equal to the current P except for $\mathcal{D}(x) = \{v\}$ (resp. $\mathcal{D}(x) = \mathcal{D}(x) \setminus \{v\}$).

3 General Purpose Heuristics

We first briefly survey the existing general-purpose variable ordering heuristics against which we compare the proposed improvement of Weighted Degree.

Impact-Based Search (IBS) [11] selects the variable with highest expected *impact*. The impact $I(x = a)$ of a decision $x = a$ corresponds to the reduction in

potential search space resulting from this decision. It is defined as $1 - S_P/S_{P|_{x=a}}$ where S_P is the size of the Cartesian product of the domains of P after constraint propagation, and $P|_{x=a}$ denotes the problem P with $\mathcal{D}(x)$ restricted to $\{a\}$. This value is updated on subsequent visits of the same decision. Moreover, in order to favor recent probes, the update is biased by a parameter α: $I^a(x = a) \leftarrow ((\alpha - 1)I^b(x = a) + I^p(x = a))/\alpha$, where I^p, I^b and I^a refer to, respectively, the last recorded impact, the impact stored before, and after the decision. The preferred variable minimizes $\sum_{a \in \mathcal{D}(x)}(1 - I(x = a))$, the sum of the complement-to-one of each value's impact, i.e., the variable with fewest options, and of highest impact. This idea also defines a branching strategy since the value with lowest impact can be seen as more likely to lead to a solution. Therefore, once a variable x has been chosen, the value a with the lowest $I(x = a)$ is tried first.

Activity-Based Search (ABS) [9] selects the variable whose domain was most often reduced during propagation. It maintains a mapping A from variables to reals. After Line 1 in Algorithm 1, for every variable x whose domain has been somehow reduced during the process, the value of $A(x)$ is incremented. In order to favor recent activity, for every variable x, $A(x)$ is multiplied by factor $0 \leq \gamma \leq 1$ before[4]. The variable x with lowest ratio $|\mathcal{D}(x)|/A(x)$ is selected first. Similarly to IBS, one can use the activity to select the value to branch on. The activity of a decision is defined as the number of variables whose domain was reduced as its consequence. It is updated as in IBS, using the same bias α.

The Last Conflict heuristic (LC) [7] relies on a completely different concept. Once both branches ($P|_{x=a}$ and $P|_{x \neq a}$) have failed, the variable x of this choice point is always preferred, until it is successfully assigned, that is, a branch $P|_{x=b}$ (with possibly $b = a$) is tried and the following propagation step succeeds. Indeed, a non-trivial subset of decisions forming a nogood must necessarily contain x. Moreover, if there is a nogood that does not contain the previous decision (or the next, etc.) then this procedure will automatically "backjump" over it. Often, no such variable exists, for instance when the search procedure dives through left branches and therefore a default selection heuristic is required.

Conflict Ordering Search (COS) [2] also tries to focus on variables that failed recently. Here, for every failure (Line 1, Algorithm 1 returning `False`), the variable selected in the parent recursion is stamped by the total number of failures encountered so far. The variable with the highest stamp is selected first. If there are several variables with equal stamp (this can only be 0, i.e., variables which never caused a failure), then a default heuristic is used instead.

Last, Weighted Degree ($wdeg$) [1] maintains a mapping w from constraints to reals. For every failure with culprit constraint C, a decay factor γ is applied exactly as in ABS and $w(C)$ is incremented. The weight $wdeg(x_i)$ of a variable x_i is the sum of the weight of its active neighboring constraints, i.e., those constraining at least another distinct unassigned variable: $wdeg(x_i) = \sum_{C \in \mathcal{C}_i} w(C)$ where $\mathcal{C}_i = \{C \mid C \in \mathcal{C} \wedge x_i \in \mathcal{X}(C) \wedge \exists x_j \neq x_i \in \mathcal{X}(C) \text{ s.t. } |\mathcal{D}(x_j)| > 1\}$. The variable x with lowest ratio $|\mathcal{D}(x)|/wdeg(x)$ is selected first.

[4] In practice, the increment value is divided by γ, and A is scaled down when needed.

4 Explanation-Based Weight

We propose to adapt the weighting function of *wdeg* to take into account the fact that not every variables in the scope a constraint triggering a failure may be involved in the conflict.

Consider for instance the constraint $\sum_{i=1}^{n} x_i \leq k$ where the initial domains are all in $\{0, 1\}$. When a failure is triggered, the weight of every variable is incremented. However, variables whose domain is equal to $\{1\}$ are sole responsible for this failure. It would be more accurate to increment only their weight.

When a failure occurs for a constraint C with scope $\langle x_1, \ldots, x_k \rangle$, it means that under the current domain $\mathcal{D}$, there is no tuple satisfying C in the cartesian product $\mathcal{D}(x_1)\times, \ldots, \times\mathcal{D}(x_k)$. It follows that there is no solution for the CSP under the current domain. Any subset $E = \{i_1, \ldots, i_m\}$ of $\{1, \ldots, k\}$ such that no tuple in $\mathcal{D}(x_{i_1})\times, \ldots, \times\mathcal{D}(x_{i_m})$ satisfies C is an *explanation* of the failure of C under the domain $\mathcal{D}$. The set $\{1, \ldots, k\}$ is the trivial explanation, however explanations that are strict subsets represent a valuable information and have been used to develop highly successful search algorithms. The goal in these algorithms (e.g., Nogood Recording [12] and CDCL [13, 14, 6]) is to use such explanations to derive a nogood, which also entails the failure under the current domain when added to the CSP, however without falsifying individually any constraint. Another goal is to make this nogood as weak as possible since we can add the negation of the nogood as an implied constraint. For this reason, as well as other practical reasons, a more restrictive language of literals is used to represent explanations and nogoods (typically $x = a, x \neq a, x \leq a$ and $x > a$).

Our purpose is simpler and more modest: we simply aim at computing an explanation E as small as possible, and do not care about the unary domain constraints. Indeed we use this explanation only to weight the variables whose indices are in E. We keep a weight function for variables instead of constraints. When a constraint C triggers a failure, we apply a decay factor γ and by default we increment the weight $w(x)$ of every variable $x \in \mathcal{X}(C)$. However, for a small set of constraints (AllDifferent, Element, and Linear Inequality) for which we have implemented an algorithm for computing an explanation, we update the weight only of the variables involved in the explanation.

There is one difference with *wdeg* concerning inactive constraints, i.e., with at most one variable currently unassigned. The weight of an inactive constraint does not contribute to the selection of its last variable. We decided to ignore this, and we count the weight of active and inactive constraint alike as it did not appear to be critical. However, one can implement this easily by keeping, for each constraint C and each variable $x \in \mathcal{X}(C)$, the quantity of weight $w(C, x)$ due to C. When a constraint becomes inactive and its last unassigned variable is x, then $w(C, x)$ is subtracted from $w(x)$, and added back upon backtrack.

Notice that since we use these conflicts for informing the heuristic only, they do not actually need to be minimal if, for instance, extracting a minimal conflict is computationally hard for the constraint. Similarly, the explanation does not even need to be correct. The three very straightforward procedures that we implemented and described below produce explanations that are correct but not

necessarily minimal. We chose to use fast and easy to implement explanations and found that it was sufficient to improve the behavior of *wdeg*:

AllDifferent($\langle x_1, \ldots, x_k \rangle$) $\Leftrightarrow \forall 1 \leq i < j \leq k,\ x_i \neq x_j$, where $\langle x_1, \ldots, x_k \rangle$ is a tuple of integer variables.

We used in our experiment two propagators for that constraint. First, with highest priority, arc consistency is achieved on binary inequalities. If a failure is obtained when processing the inequality $x_i \neq x_j$, it means that both variables are ground and equal, hence we use the conflict set $\{x_i, x_j\}$.

Then, with lower priority, we use the bounds consistency propagation algorithm described in [8]. This algorithm fails if it finds a Hall interval, i.e., a set of at least $b - a + 2$ variables whose domains are included in $\{a, \ldots, b\}$. When this happens we simply use this set of variables as conflict.

Linear Inequality($\langle x_1, \ldots, x_k \rangle, \langle a_1, \ldots, a_k \rangle, b$) $\Leftrightarrow \sum_{i=1}^{k} a_i x_i \leq b$, where b is an integer, $\langle x_1, \ldots, x_k \rangle$ a tuple of integer variables and $\langle a_1, \ldots, a_k \rangle$ of integers.

The constraint fails if and only if the lower bound of the sum is strictly larger than b. When this is true, a possible conflict is the set containing every variable x_i such that either a_i is positive and $\min(\mathcal{D}(x_i))$ is strictly larger than its initial value, or a_i is negative and $\max(\mathcal{D}(x_i))$ is strictly lower than its initial value.

Element($\langle x_1, \ldots, x_k \rangle, n, v$) $\Leftrightarrow x_n = v$, where $\langle x_1, \ldots, x_k \rangle$ is a tuple of integer variables, n, v two integer variables.

We use the conflict set $\{n, v\} \cup \{x_i \mid i \in \mathcal{D}(n)\}$, that is, we put weight only on the "index" and "value" variables n and v, as well as every variable of the array pointed to by the index variable n.

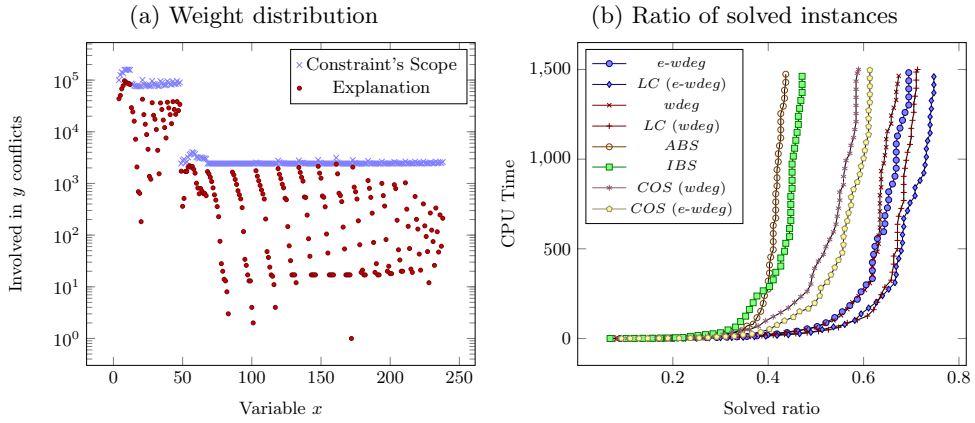
5 Experimental Evaluation

Figure 1a illustrates the difference that explanation can make. The data comes from instance `3-10-20.dzn` of the `ghoulomb.mzn` model. We plot, for each variable x, the number of failures of a constraints involving x (blue crosses) and the number of explanations of failures involving x (red dot). We observe a much wider distribution of the weight when using explanations. As a result, the optimal solution could be proven with *e-wdeg* in 15s whereas the best upper bound found with *wdeg* after 1200s was 1.83 times larger than the optimal.

Next, we experimentally evaluated the proposed variant of *wdeg*, denoted *e-wdeg*, against the state-of-the-art general-purpose branching heuristics *wdeg*, *ABS*, *IBS*, *COS* and *LC* with *wdeg* and *e-wdeg* as default heuristic for the two latter. We used lexicographic value ordering for every heuristic, except *IBS* and *ABS* since their default branching strategies were slightly better.[5] The decay factor γ was set to 0.95 for *ABS*, *wdeg* and *e-wdeg*, and the bias α was set to 8 for *ABS* and *IBS*. No probes were used to initialize weights. In all cases the initial values were set up to emulate the "minimum domain over degree"

[5] The extra space requirement was an issue for 5 optimization instances. However the impact on the overall results is quasi null.

Fig. 1: Weight distribution & Search efficiency on satisfaction instances



strategy. All the methods were implemented in Mistral-2.0 [5] and the same geometric restarts policy was used in all cases.

We used all the instances of the Minizinc challenge from 2012 to 2015 [15]. This data set contains 399 instances, 323 are optimization problems and 76 are satisfaction problems. In this set, 72 instances have at least one AllDifferent, 147 have at least one Element and all have at least one Linear Inequality. However, Element is often posted on arrays of constants and LinearInequality may be used to model clauses, which is not favorable to *e-wdeg* since the explanations are trivial. All the tests ran on Intel Xeon E5430 processors with Linux. The time cutoff was 1500s for each instance excluding the parsing time. Heuristics were randomized by choosing uniformly between the two best choices, except *COS* and *LC* for which only the default heuristics were randomized in the same way. Each configuration was given 5 randomized runs.

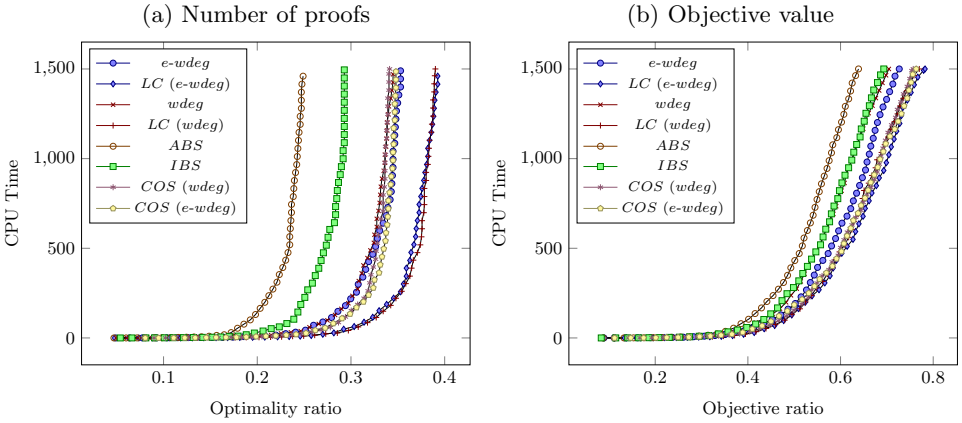
5.1 Satisfaction problems

We first report the results on satisfaction instances, where we plot, for every heuristic, the ratio of runs (among 76×5) in which the instance was solved (x-axis) over time (y-axis) in Figure 1b. The results are clear. Among previous heuristics, *wdeg* is very efficient, only outperformed by *LC* (using *wdeg* as default heuristic when the testing set is empty). Conflict Ordering Search comes next, followed by *IBS* and *ABS*. Notice that *IBS* and *ABS* are often initialised using probing. However, this method could be used for other heuristics (see [3]), and is unlikely to be sufficient to close this gap. Whether used as default for *COS* or *LC*, or as stand alone, *e-wdeg* always solves more instances than *wdeg*. The difference is not significant below a 500 seconds time limit, but patent above this mark. Overall, *e-wdeg* solves 4.5%, 3.7% and 2.9% more instances than *wdeg* when used as default for *LC*, *COS* or as stand alone, respectively.

5.2 Optimization problems

Second, we report the results for the 323 optimization instances (1615 runs). We first plot, for every heuristic, the ratio of instances proven optimal (x-axis) over time (y-axis) in Figure 2a. Here the gain of explanation-based weights is less clear. *e-wdeg* can prove 2.3% more instances, however *LC* (*e-wdeg*) finds only 0.6% more proofs than *LC* and *COS* (*e-wdeg*) 1.6% more than *COS*.

Fig. 2: Search efficiency, optimization instances



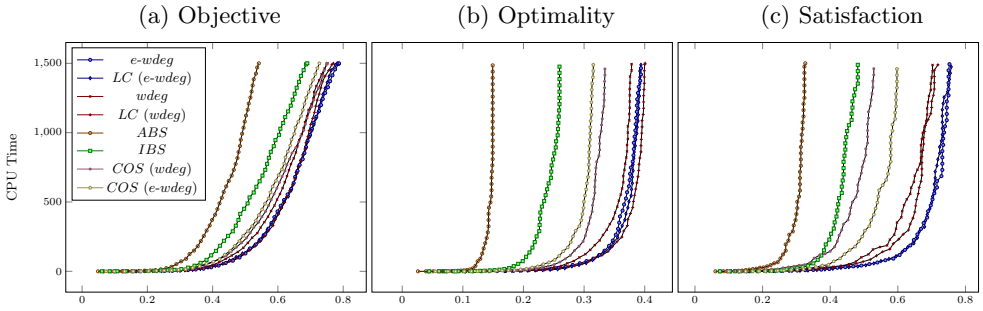
In Figure 2b we plot the normalized objective value of the best solution found by heuristic h (x-axis) after a given time (y-axis). Let $h(I)$ be the objective value of the best solution found using heuristic h on instance I and $lb(I)$ (resp. $ub(I)$) the lowest (resp. highest) objective value found by any heuristic on I. The formula below gives a normalized score in the interval $[0, 1]$:

$$score(h, I) = \begin{cases} \frac{h(I)-lb(I)+1}{ub(I)-lb(I)+1}, & \text{if } I \text{ is a maximization instance} \\ \frac{ub(I)-h(I)+1}{ub(I)-lb(I)+1}, & \text{otherwise} \end{cases}$$

Notice that we add 1 to the actual and maximum gap. Moreover, if an heuristic h does not find any feasible solution for instance I, we arbitrarily set $h(I)$ to $lb(I) - 1$ for a maximization problem, and $ub(I) + 1$ for a minimization problem. It follows that $score(h, I)$ is equal to 1 if h has found the best solution for this instance among all heuristics, decreases as $h(I)$ gets further from the optimal objective value, and is equal to 0 if and only if h did not find any solution for I.

We observe that using *e-wdeg*, the solver finds significantly better solutions faster than using *wdeg*. The same observation can be made for Last Conflict and Conflict Ordering Search using *e-wdeg* outperforming their counterparts relying on *wdeg*, although the gap is slightly less important in these cases. The

Fig. 3: Search efficiency, branching on auxiliary variables



overall improvement (respectively 3.2%, 1.6% and 0.8%) is modest, however, the number of data points makes it statistically significant.

Overall, the *wdeg* heuristic is clearly very competitive when considering a large sample of instances, and indeed it dominates *IBS* and *ABS* on the MiniZinc instances. Last Conflict and Conflict Ordering Search seem even more efficient, however, they rely on *wdeg* as default heuristic. These experiments show that computing specific conflict sets for constraints significantly boosts *wdeg*. Although the gain is less straightforward in the case of *LC* and *COS* (especially for the latter which relies less heavily on the default heuristic), this approach can be useful in those cases too, and in any case never hinders search efficiency.

Notice that we restricted the heuristic selection to the decision variables specified in the MiniZinc model. We also tried to let the heuristic branch on auxiliary variables, created internally to model expressions. The results, shown in Figure 3 are not exactly as cleanly cut in this case, and actually difficult to understand. The only two heuristics to actually benefit from this are *wdeg* and *e-wdeg*, by 6.7% and 8.3%, respectively for the objective value criteria. Two heuristics perform much worse, again for the objective value: *ABS* loses 15.2% and *COS* (*e-wdeg*) 4.7%. Other heuristics are all marginally worse in this setting. On other criteria, such as the number of optimality proofs, there is a similar, but not identical trend, with *LC* gaining the most from the extra freedom.

6 Conclusion

We showed that the *wdeg* heuristic can be made more robust to instances with large arity constraints through a relatively simple method. Whereas *wdeg* distributes weights equally among all variables of the constraint that triggered a failure, we propose to weight only the variables of that constraint participating in an explanation of the failure. Our empirical analysis shows that this technique improves the performance of *wdeg*. In particular, the Last Conflict heuristic [7] using the improved version of Weighted Degree, *e-wdeg*, is the most efficient overall on the benchmarks from the previous MiniZinc challenges.

References

1. Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. Boosting systematic search by weighting constraints. In *Proceedings of the 16th European Conference on Artificial Intelligence – ECAI*, pages 146–150, 2004.
2. Steven Gay, Renaud Hartert, Christophe Lecoutre, and Pierre Schaus. Conflict Ordering Search for Scheduling Problems. In Gilles Pesant, editor, *Principles and Practice of Constraint Programming – CP*, pages 140–148, 2015.
3. Diarmuid Grimes and Richard J. Wallace. Sampling strategies and variable selection in weighted degree heuristics. In *Principles and Practice of Constraint Programming – CP*, pages 831–838, 2007.
4. Robert M. Haralick and Gordon L. Elliott. Increasing Tree Search Efficiency for Constraint Satisfaction Problems. In *Proceedings of the 6th International Joint Conference on Artificial Intelligence – IJCAI*, pages 356–364, 1979.
5. Emmanuel Hebrard. Mistral, a Constraint Satisfaction Library. In *Proceedings of the CP-08 Third International CSP Solvers Competition*, pages 31–40, 2008.
6. Roberto Bayardo Jr and Robert C. Schrag. Using CSP Look-Back Techniques to Solve Real-World SAT Instances. In *Proceedings of the 14th National Conference on Artificial Intelligence – AAAI*, pages 203–208, 1997.
7. Christophe Lecoutre, Lakhdar Sas, Sbastien Tabary, and Vincent Vidal. Reasoning from last conflict(s) in constraint programming. *Artificial Intelligence*, 173(18):1592 – 1614, 2009.
8. Alejandro Lopez-Ortiz, Claude-Guy Quimper, John Tromp, and Peter van Beek. A fast and simple algorithm for bounds consistency of the alldifferent constraint. In *Proceedings of the 18th International Joint Conference on AI – IJCAI*, pages 245–250, 2003.
9. Laurent Michel and Pascal Van Hentenryck. Activity-Based Search for Black-Box Constraint Programming Solvers. In *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems – CPAIOR 2012*, pages 228–243, 2012.
10. Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Annual Design Automation Conference – DAC*, pages 530–535, 2001.
11. Philippe Refalo. Impact-based search strategies for constraint programming. In *Principles and Practice of Constraint Programming – CP*, pages 557–571, 2004.
12. Thomas Schiex and Gérard Verfaillie. Nogood Recording for Static and Dynamic Constraint Satisfaction Problems. In *ICTAI*, pages 48–55, 1993.
13. João P. Marques Silva and Karem A. Sakallah. GRASP - a New Search Algorithm for Satisfiability. In *Proceedings of the IEEE/ACM International Conference on Computer-aided Design – ICCAD*, pages 220–227, 1996.
14. João P. Marques Silva and Karem A. Sakallah. GRASP: a search algorithm for propositional satisfiability. *Computers, IEEE Transactions on*, 48(5):506–521, 1999.
15. Peter J. Stuckey, Thibaut Feydy, Andreas Schutt, Guido Tack, and Julien Fischer. The minizinc challenge 2008-2013. *AI Magazine*, 35(2):55–60, 2014.
16. Richard J. Wallace and Diarmuid Grimes. Experimental studies of variable selection strategies based on constraint weights. *Journal of Algorithms*, 63(1):114 – 129, 2008.