



HAL
open science

Conflict history based heuristic for constraint satisfaction problem solving

Djamal Habet, Cyril Terrioux

► **To cite this version:**

Djamal Habet, Cyril Terrioux. Conflict history based heuristic for constraint satisfaction problem solving. *Journal of Heuristics*, Springer Verlag, 2021, 27 (6), pp.951-990. 10.1007/s10732-021-09475-z . hal-03276535

HAL Id: hal-03276535

<https://hal-amu.archives-ouvertes.fr/hal-03276535>

Submitted on 2 Jul 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Conflict History Based Heuristic for Constraint Satisfaction Problem Solving*

Djamal Habet Cyril Terrioux

Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France

{djamal.habet,cyril.terrioux}@lis-lab.fr

Abstract

The variable ordering heuristic is an important module in algorithms dedicated to solve Constraint Satisfaction Problems (CSP), while it impacts the efficiency of exploring the search space and the size of the search tree. It also exploits, often implicitly, the structure of the instances. In this paper, we propose Conflict-History Search (CHS), a dynamic and adaptive variable ordering heuristic for CSP solving. It is based on the search failures and considers the temporality of these failures throughout the solving steps. The exponential recency weighted average is used to estimate the evolution of the hardness of constraints throughout the search. The experimental evaluation on XCSP3 instances shows that integrating CHS to solvers based on MAC (Maintaining Arc Consistency) and BTM (Backtracking with Tree Decomposition) achieves competitive results and improvements compared to the state-of-the-art heuristics. Beyond the decision problem, we show empirically that the solving of the constraint optimization problem (COP) can also take advantage of this heuristic.

1 Introduction

The Constraint Satisfaction Problem (CSP) is an important formalism in Artificial Intelligence (AI) which allows to model and efficiently solve problems that occur in various fields, both academic and industrial (e.g. [8, 19, 37, 40]). A CSP instance is defined on a set of variables, which must be assigned in their respective finite domains. Variable assignments must satisfy a set of constraints, which express restrictions on assignments. A solution is an assignment of each variable, which satisfies all constraints.

CSP solving is often based on backtracking algorithms. In recent years, it has made significant progress thanks to research on several aspects. In particular, considerable effort is devoted to global constraints, filtering techniques, learning and restarts [37]. An important component in CSP solvers is the variable ordering heuristic. Indeed, the corresponding heuristics define, statically or dynamically, the order in which the variables will be assigned and, thus, the way that the search space will be explored and the size of the search tree. The problem of finding the best variable to assign (i.e. one which minimizes the search tree size) is NP-Hard [30].

Many heuristics have been proposed (e.g. [4, 5, 6, 7, 12, 13, 18, 32, 35]) aiming mainly to satisfy the *first-fail principle* [17] which advises "to succeed, try first where you are likely to fail". Nowadays, the most efficient heuristics are adaptive and dynamic [6, 12, 18, 32, 35], where the variable ordering is defined according to the collected information since the beginning of the search. For instance, some heuristics consider the effect of filtering when decisions and propagations are applied [32, 35]. *dom/wdeg* is one of the simplest, the most used and efficient variable ordering heuristic [6]. It is based on the hardness of constraints and, more specifically, reflects how often a constraint fails. It uses a weighting process to focus on the variables appearing in constraints with high weights which are assumed to be hard to satisfy. In addition, some heuristics, such as LC [25] and COS [11], attempt to consider the search history while they require the use of auxiliary heuristics.

In this paper, we propose *Conflict-History Search (CHS)*, a new dynamic and adaptive variable ordering heuristic for CSP solving. It is based on the history of search failures, which happen as soon as a domain of a variable is emptied after constraint propagations. The goal is to reward the scores of constraints that have recently been involved in conflicts and therefore to favor the variables appearing in these constraints. The scores of constraints are estimated on the basis of the exponential recency weighted average technique, which comes from reinforcement learning [41]. It was also recently used in defining powerful branching heuristics for solving the satisfiability problem (SAT) [28, 29]. We have integrated CHS in solvers based on MAC (Maintaining Arc Consistency) [38] and BTM (Backtracking with Tree-Decomposition) [24]. The empirical evaluation on XCSP3 instances¹ shows that CHS is competitive and brings improvements to the state-of-the-art heuristics. In addition, this evaluation provides an extensive study of the performance of state-of-the-art search heuristics on more than 12,000 instances. Finally, we also study, from a practical viewpoint, the benefits of the proposed heuristic for solving constraint optimization problems (COP).

The paper is structured as follows. Section 2 includes some necessary definitions and notations. Section 3 presents and details our contribution, the CHS variable ordering heuristic. Section 4 describes related work on variable ordering heuristics for CSP and

*The final authenticated version is available online at <https://doi.org/10.1007/s10732-021-09475-z>. This work is an extension of the work published in [16].

¹<http://www.xcsp.org>

on branching heuristics for the satisfiability problem. CHS is evaluated experimentally and compared to the main powerful heuristics of the state-of-the-art on CSP instances in Section 5 and on COP ones in Section 6. Finally, we conclude and give some perspectives on extending the application of CHS.

2 Preliminaries

This section is dedicated to the definition of CSP and Exponential Recency Weighted Average, which we use to propose our heuristic.

2.1 Constraint Satisfaction Problem

An instance of a Constraint Satisfaction Problem (CSP) is given by a triple (X, D, C) , such that: $X = \{x_1, \dots, x_n\}$ is a set of n variables, $D = \{D_1, \dots, D_n\}$ is a set of finite domains, and $C = \{c_1, \dots, c_e\}$ is a set of e constraints. The domain of each variable x_i is D_i . Each constraint c_j is defined by its scope $S(c_j)$ and its compatibility relation $R(c_j)$, where $S(c_j) = \{x_{j_1}, \dots, x_{j_k}\} \subseteq X$ and $R(c_j) \subseteq D_{j_1} \times \dots \times D_{j_k}$. The constraint satisfaction problem asks for an assignment of the variables $x_i \in X$ within their respective domains D_i ($1 \leq i \leq n$) that satisfies each constraint in C . Such consistent assignment is a solution. Checking whether a CSP instance has a solution is NP-complete [37].

In the past decades, many solvers have been proposed for solving CSPs. Generally, from a practical viewpoint, they succeed in solving efficiently a large kind of instances despite of the NP-completeness of the CSP decision problem. In most cases, they rely on optimized backtracking algorithms whose time complexity is at least in $O(e \cdot d^n)$ where d denotes the size of the largest domain. In order to ensure an efficient solving, they commonly exploit jointly several techniques (see [37] for more details) among which we can cite:

- variable ordering heuristics which aim to guide the search by choosing the next variable to assign (we discuss about some state-of-the-art heuristics in Section 4),
- constraint learning and non-chronological backtracking which aim to avoid some redundancies during the exploration of the search space,
- filtering techniques enforcing some consistency level which aim to simplify the instance by removing some values from domains or tuples from constraint relations which cannot participate to a solution.

For instance, most state-of-the-art solvers maintain some consistency level at each step of the search, like MAC (Maintaining Arc-Consistency [38]) or RFL (Real Full Look-ahead [34]) do for arc-consistency. This latter turns out to be a relevant tradeoff between the number of removed values and the runtime.

We now recall MAC with more details. During the solving, MAC develops a binary search tree whose nodes correspond to decisions. More precisely, it can make two kinds of decisions: *positive decisions* $x_i = v_i$ which assign the value v_i to the variable x_i and *negative decisions* $x_i \neq v_i$ which ensure that x_i cannot be assigned with v_i . Let us consider $\Sigma = \langle \delta_1, \dots, \delta_i \rangle$ (where each δ_j may be a positive or negative decision) as the current decision sequence. At each node of the search tree, MAC takes either a positive decision or negative one. When reaching a new level, it starts by a positive decision which requires to choose a variable among the unassigned variables and a value. Both choices are achieved thanks to heuristics. Then, once the decision made, MAC applies an arc-consistency filtering. This filtering deletes some values of unassigned variables which are not consistent with the last taken decision and Σ . By so doing, a domain may become empty. In such a case, we say that a *dead-end* or a *conflict* occurs. This means that the current set of decisions cannot lead to a solution. If no dead-end occurs, the search goes on to the next level by choosing a new positive decision. Otherwise, the current decision is called into question. If it is a positive decision $x_i = v_i$, MAC makes the corresponding negative decision $x_i \neq v_i$, that is the value v_i is deleted from the domain D_i . Otherwise, it is a negative decision and MAC backtracks to the last positive decision $x_\ell = v_\ell$ in Σ and makes the decision $x_\ell \neq v_\ell$. If no such decision exists, it means that the instance has no solution. In contrast, if MAC succeeds in assigning all the variables, the corresponding assignment is, by construction, a solution of the considered instance.

More recently, restart techniques have been introduced in the CSP framework (e.g. in [27]). They generally allow to reduce the impact of bad choices performed thanks to heuristics (like the variable ordering heuristic) or of the occurrence of heavy-tailed phenomena [14]. For efficiency reasons, they are usually exploited with some learning techniques like recording of nld-nogoods in [27]. These nogoods can be seen as a set of decisions which cannot be extended to a solution. They are used to avoid visiting again a part of the search space which has already been visited by MAC. These nogoods are recorded each time a restart occurs.

2.2 Exponential Recency Weighted Average

Given a time series of m numbers $y = (y_1, y_2, \dots, y_m)$, the simple average of y is $\sum_{i=1}^m \frac{1}{m} y_i$ where each y_i has the same weight $\frac{1}{m}$. There are situations where recent data are more relevant than old data to describe the current situation. The Exponential Recency

Weighted Average (ERWA) [41] takes into account such considerations by giving higher weights to the recent data than the older ones. More precisely, the *exponential moving average* \bar{y}_m is computed as follows:

$$\bar{y}_m = \sum_{i=1}^m \alpha \times (1 - \alpha)^{m-i} \times y_i$$

where $0 < \alpha < 1$ is a step-size parameter which controls the relative weights between recent and past data. The moving average can also be calculated incrementally by the formula:

$$\bar{y}_m = (1 - \alpha) \times \bar{y}_{m-1} + \alpha \times y_m.$$

The base case is $\bar{y}_0 = 0$. ERWA is used to solve the bandit problem to estimate the expected reward of different actions in nonstationary environments [41]. In bandit problems, the agent must select an action to play, from a given set of actions, while maximizing its long term expected reward.

3 Conflict-History Search for CSP

This section is dedicated to our contribution by defining and describing a new variable ordering heuristic for CSP solving, which we call *Conflict-History Search* (CHS). The main idea is to consider the history of constraint failures and favor the variables that often appear in recent failures. In this order, the conflicts are dated and the constraints are weighted on the basis of the exponential recency weighted average. These weights are coupled with the variable domains to calculate the Conflict-History scores of the variables.

3.1 CHS Description

Formally, CHS maintains for each constraint c_j a score $q(c_j)$ which is initialized to 0 at the beginning of the search. If c_j leads to a failure during the search because the domain of a variable in $S(c_j)$ is emptied then $q(c_j)$ is updated by the formula below derived from ERWA [41]:

$$q(c_j) = (1 - \alpha) \times q(c_j) + \alpha \times r(c_j)$$

The parameter $0 < \alpha < 1$ is the step-size and $r(c_j)$ is the reward value. The parameter α fixes the importance given to the old value of q at the expense of the reward r . The value of α decreases over time as it is applied in reinforcement learning to converge towards relevant values of q [41]. In other words, decreasing the value of α amounts to giving more importance to the last value of q and considering that the values of q are more and more relevant as the search progresses. Furthermore, we are interested by the constraint failure to follow the *first-fail principle* [17].

CHS applies the decreasing policy of α , which is successfully used for designing efficient branching heuristic for the satisfiability problem [28, 29]. More precisely, starting from an initial value α_0 , α decreases by 10^{-6} at each constraint failure to a minimum of 0.06. This minimum value of α controls the number of steps before considering that a convergence is reached.

The reward value $r(c_j)$ is based on how recently c_j occurred in conflicts. More precisely, it relies on the proximity between the previous conflict in which c_j is involved and the current one. By so doing, we aim to give a higher reward to constraints that fail regularly over short periods of time during the search space exploration. The reward value is calculated according to the formula:

$$r(c_j) = \frac{1}{Conflicts - Conflict(c_j) + 1}$$

Initialized to 0, *Conflicts* is the number of conflicts which have occurred since the beginning of the search. *Conflict*(c_j) is also initialized to 0 for each constraint $c_j \in C$. When a conflict occurs on c_j , $r(c_j)$ and $q(c_j)$ are computed. Then *Conflicts* is incremented by 1 and *Conflict*(c_j) is updated to the new value of *Conflicts*.

At this stage, we define the Conflict-History score of a variable $x_i \in X$ as follows:

$$chv(x_i) = \frac{\sum_{c_j \in C: x_i \in S(c_j) \wedge |Uvars(S(c_j))| > 1} q(c_j)}{|D_i|} \quad (1)$$

$Uvars(Y)$ is the set of unassigned variables in Y . D_i is the current domain of x_i and its size may be reduced by the propagation process in the current step of the search. CHS chooses the variable to assign with the highest *chv* value. In this manner, CHS focuses branching on the variables with a small domain size belonging to constraints which appear recently and repetitively in conflicts.

One can observe that at the beginning of the search, all the variables have the same score, which is equal to 0. To avoid random selection, we update Equation 1 to calculate *chv* as given below, where δ is a positive real number close to 0.

$$chv(x_i) = \frac{\sum_{c_j \in C: x_i \in S(c_j) \wedge |Uvars(S(c_j))| > 1} (q(c_j) + \delta)}{|D_i|} \quad (2)$$

Thus, when the search starts, the branching will be oriented according to the degree of the variables without having a negative influence on the ERWA-based calculation later in the search. CHS selects the branching variable with the highest *chv* value calculated according to Equation 2.

The heuristic CHS is described in Algorithm 1 with an event-driven approach. Lines 2-7 correspond to the initialization step. If a conflict occurs when enforcing the filtering with the constraint c_j , the associated event is triggered and the score is update (Lines 8-14). The selection of a new variable is achieved thanks to Lines 15-16.

Algorithm 1: CHS

Input: an event e

```

1 switch  $e$  do
2   case initialization do
3      $\alpha \leftarrow \alpha_0$ 
4      $Conflicts \leftarrow 0$ 
5     for  $c_j \in C$  do
6        $Conflict(c_j) \leftarrow 0$ 
7        $q(c_j) \leftarrow 0$ 
8   case conflict when filtering with  $c_j$  do
9      $r(c_j) \leftarrow \frac{1}{Conflicts - Conflict(c_j) + 1}$ 
10     $q(c_j) \leftarrow (1 - \alpha) \times q(c_j) + \alpha \times r(c_j)$ 
11     $Conflicts \leftarrow Conflicts + 1$ 
12     $Conflict(c_j) \leftarrow Conflicts$ 
13    if  $\alpha > 0.06$  then
14       $\alpha \leftarrow \alpha - 10^{-6}$ 
15  case select a new variable do
16    return a variable  $x$  s.t.  $x \in \arg \min_{x_i \in Uvars(X)} \frac{\sum_{c_j \in C: \#i \in S(c_j) \wedge |Uvars(S(c_j))| > 1} (q(c_j) + \delta)}{|D_i|}$ 
17  case restart do
18     $\alpha \leftarrow \alpha_0$ 
19    for  $c_j \in C$  do
20       $q(c_j) \leftarrow q(c_j) \times 0.995^{Conflicts - Conflict(c_j)}$ 

```

3.2 CHS and Restarts

Restart techniques are known to be important for the efficiency of solving algorithms (see for example [26]). Restarts may allow to reduce the impact of irrelevant choices done during the search according to heuristics, such as variable selection.

As it will be detailed later, CHS is integrated into CSP solving algorithms, which include restarts. In the corresponding implementations, the $Conflict(c_j)$ value of each constraint c_j is not reinitialized when a restart occurs. It is the same for $q(c_j)$. However, a *smoothing* may be applied and will be explained below. Keeping this information unchanged reinforces learning from the search history.

Concerning the step-size α , which defines the importance given to the old value of $q(c_j)$ at the expense of the reward $r(c_j)$, CHS reinitializes the value of α to α_0 at each restart (Line 18 of Algorithm 1). This may guide the search through different parts of the search space.

3.3 CHS and Smoothing

At each conflict, CHS updates the *chv* score of one constraint at a time: the constraint c_j which is used to wipe out the domain of a variable in $S(c_j)$. As long as they do not appear in new conflicts, some constraints can have their weights unchanged for several search steps. These constraints may have high scores while their importance does not seem significant for the current part of the search. To avoid this situation, we propose to smooth the scores $q(c_j)$ of all the constraints $c_j \in C$ at each restart by the following formula:

$$q(c_j) = q(c_j) \times 0.995^{Conflicts - Conflict(c_j)}$$

Hence, the scores of constraints are decayed according to the date of their last appearances in conflicts (Lines 19-20 of Algorithm 1).

4 Related Work

Before providing a detailed experimental evaluation of CHS and its components, we present the most efficient and common variable ordering heuristics for CSP. As CHS, the recalled heuristics share the same behavior. In effect, the variables and/or constraints are weighted dynamically throughout the search by considering the collected information since its beginning. Some of these heuristics, such as Last Conflict [25], require the use of an auxiliary heuristic as it will be explained later. We also recall briefly branching heuristics for the satisfiability problem. It should be recalled that ERWA was first used in the context of the satisfiability problem [28, 29].

4.1 Impact-Based Search (IBS)

This heuristic selects the variable which leads to the largest search space reduction [35]. The impact on the search space size is approximated as the reduction of the product of the variable domain sizes. Formally, the impact of assigning the variable x_i to the value $v_i \in D_i$ is defined by:

$$I(x_i = v_i) = 1 - \frac{P_{after}}{P_{before}}$$

P_{after} and P_{before} are respectively the products of the domain cardinalities after and before branching on $x_i = v_i$ and applying constraint propagations. By doing so, selecting the next branching variable requires the computation of the impact of each variable assignment, by simulating filtering at each node of the search tree. This can be very time consuming. Hence, IBS considers the impact of an assignment at a given node as the average of its observed impacts. More precisely, if K is the index set of impacts observed of $x_i = v_i$, IBS estimates an averaged impact of this assignment as follows, where I_k is k^{th} impact value:

$$\bar{I}(x_i = v_i) = \frac{\sum_{k \in K} I_k(x_i = v_i)}{|K|}$$

Finally, the impact of a variable according to its current domain, which may be filtered, is defined as follows:

$$\mathcal{I}(x_i) = \sum_{v \in D_i} 1 - \bar{I}(x_i = v)$$

IBS selects the variable with the highest impact value $\mathcal{I}(x_i)$.

4.2 Conflict-Driven Heuristic

A popular variable ordering heuristic for CSP solving is *dom/wdeg* [6]. It guides the search towards the variables appearing in the constraints which seem hard to satisfy. For each constraint c_j , the *dom/wdeg* heuristic maintains a weight $w(c_j)$, initially set to 1, counting the number of times that c_j has led to a failure (i.e. the domain of a variable x_i in $S(c_j)$ is emptied during propagation from c_j). The weighted degree of a variable x_i is defined as:

$$wdeg(x_i) = \sum_{c_j \in \mathcal{C}: x_i \in S(c_j) \wedge |Uvars(S(c_j))| > 1} w(c_j)$$

The *dom/wdeg* heuristic selects the variable x_i to assign with the smallest ratio $|D_i|/wdeg(x_i)$, such that D_i is the current domain of x_i (the size of D_i may be reduced in the current search step). Note that the constraint weights are not smoothed in *dom/wdeg*. Also, variants of *dom/wdeg* were introduced, such as in [18], but are not widely used in practice. Very recently, a refined version of *wdeg* (called *wdeg^{ca.cd}*) has been defined in [43]. When a conflict occurs for a constraint c_j , instead of increasing its weight by 1 as in *dom/wdeg*, *wdeg^{ca.cd}* increases its weight by a value depending on the number of unassigned variables in the scope of c_j and their current domain size.

4.3 Activity-Based Heuristic (ABS)

ABS is motivated by the prominent role of filtering techniques in CSP solving [32]. It exploits this filtering information and maintains measures of how often the variable domains are reduced during the search. In practice, at each node of the search tree, constraint propagation may filter the domains of some variables after the decision process. Let X_f be the set of such variables. Accordingly, the activities $A(x_i)$, initially set to 0, of the variables $x_i \in X$ are updated as follows:

- $A(x_i) = A(x_i) + 1$ if $x_i \in X_f$
- $A(x_i) = \gamma \times A(x_i)$ if $x_i \notin X_f$

γ is a decay parameter, such that $0 \leq \gamma \leq 1$. The ABS heuristic selects the variable x_i with the highest ratio $A(x_i)/|D_i|$.

4.4 CHB in Gecode

Dedicated to constraint programming, Gecode solver implements Conflict-History based Branching (CHB) heuristic since version 5.1.0 released in April 2017 [39]. It follows the same steps of the first definition of CHB in the context of the satisfiability problem [28, 29]. In Gecode, the following parameters are used to update the Q -score of each variable x_i of the CSP instance, denoted $qs(x_i)$. f is the number of failures encountered since the beginning of the search and $lf(x_i)$ is the last failure number of x_i , corresponding to the last time that D_i is emptied.

Initialized to 0.05 for each variable x_i , CHB update the Q -score $qs(x_i)$ of x_i during the constraint propagation as follows:

- If D_i is not reduced then $qs(x_i)$ remains unchanged
- If D_i is pruned and the search leads to a failure, $lf(x_i)$ is set to f and $qs(x_i)$ is updated by:

$$qs(x_i) = (1 - \alpha) \times qs(x_i) + \alpha \times r$$

The step-size α , initialized to 0.4, is updated to $\alpha - 10^{-6}$ if $\alpha > 0.06$. The value of the reward r is given by:

$$r = \frac{1}{f - lf(x_i) + 1}$$

- If D_i is pruned and the search does not lead to a failure, $qs(x_i)$ is also updated by:

$$qs(x_i) = (1 - \alpha) \times qs(x_i) + \alpha \times r$$

In this case, the reward value is defined by:

$$r = \frac{0.9}{f - lf(x_i) + 1}$$

CHB in Gecode selects the variable with the highest Q -score.

4.5 Last Conflict (LC)

Last Conflict (LC) reasoning [25] aims to better identify and exploit nogoods in a binary tree search, where each node has a first branch corresponding to a positive decision ($x_i = v_i$) and eventually a second branch with a negative decision ($x_i \neq v_i$).

If a positive decision $x_i = v_i$ leads to a conflict then LC records the variable x_i as a conflicting variable. The value v_i is removed from the domain D_i of x_i . After developing the negative branch $x_i \neq v_i$, LC continues the search by assigning a new value v'_i to x_i instead of choosing a new decision variable. This treatment is repeated until a successful assignment of x_i is achieved. In this case, the variable x_i is unrecorded as a conflicting one and the next decision variable is decided by an auxiliary variable ordering heuristic. Hence, this last one is used when no conflicting variable is recorded by LC.

4.6 Conflict Order Search (COS)

Conflict Order Search (COS) [11] is intended to focus the search on the variables which lead to recent conflicts. When a branching on a variable x_i fails, x_i is stamped by the total number of failures since the beginning of the search (the initial stamp value of each variable is 0). COS prefers the variable with the highest stamp value. An auxiliary heuristic is used if all the unassigned variables have the stamp value 0.

4.7 Branching Heuristics for the Satisfiability Problem

In the context of the satisfiability problem, modern solvers based on Conflict-Driven Clause Learning (CDCL) [10, 31, 33] employ variable branching heuristics correlated to the ability of the variable to participate in producing learnt clauses when conflicts arise (a conflict is a clause falsification). The Variable State Independent Decaying Sum (VSIDS) heuristic [33] maintains an activity value for each Boolean variable. The activities are modified by two operations: the bump (increase the activity of variables appearing in the process of generating a new learnt clause when a conflict is analyzed) and the multiplicative decay of the activities (often applied at each conflict). VSIDS selects the variable with the highest activity to branch on.

Recently, a conflict history based branching heuristic (CHB) [28], based on the exponential recency weighted average, was introduced. It rewards the activities to favor the variables that were recently assigned by decision or propagation. The rewards are higher if a conflict is discovered. The Learning Rate Branching (LRB) heuristic [29] extends CHB by exploiting locality and introducing the learning rate of the variables.

4.8 Discussion

Reinforcement learning techniques have already been studied in constraint programming. The multi-armed bandit framework is used to select adaptively the consistency level of propagation at each node of the search tree [2]. A linear regression method is used to learn the scoring function of value heuristics [9]. Rewards are calculated and used to select adaptively the backtracking strategy [1]. Learning process based on Least Squares Policy Iteration technique is used to tune adaptively the parameters of stochastic local search algorithms [3].

More recently, upper confidence bound and Thompson Sampling techniques are employed to select automatically a variable ordering heuristic for CSP, among a set of candidate ones, at each node of the search tree [44]. The considered candidate set contains notably IBS, ABS and *dom/wdeg*. Knowing that no heuristic always outperforms another, Xia and Yap exploit reinforcement learning (under the form of a multi-armed bandit) to choose the search heuristic to employ at each node of the search rather than choosing a particular heuristic before the solving. More recently, Watez et al. have proposed another MAB approach [42]. Like in the work of Xia and Yap, each heuristic corresponds to an arm. In contrast, a new arm is chosen at each restart instead of each node. On the other hand, in CHS, reinforcement learning allows to select the branching variable based on ERWA. Note also that CHS can be used as an additional arm in the work of Xia and Yap while it is already exploited as an arm in [42].

To return to the heuristics detailed in this section, LC, COS and CHB are also conceptually interested in the search history as CHS. They act directly on the variable scores while CHS considers this history by weighting the constraints that are responsible for failures before scoring the variables. As an illustration, CHB in Gecode updates the Q -score values of variables according to ERWA while CHS uses ERWA to update the weight of constraints to calculate the score of the variables. The update of the α parameter is also different between CHS and CHB, especially during restarts.

Weight and score decaying is also used in other heuristics such as ABS. However, it is applied to the score of the variables and not that of the constraints such as in CHS. It is also important to note that there is no decaying in CHB. Furthermore, CHS and *dom/wdeg* calculate differently the score of the constraints leading to failures. In the first case, the score of the constraint is always incremented by a constant value 1. In the second case, the new score is a tradeoff between the current one and the reward that varies at each failure. Moreover, the scores of constraints are not decayed in *dom/wdeg* contrary to CHS. Finally, unlike LC and COS, CHS does not require the use of an auxiliary heuristic.

5 Experimental Evaluation on CSP Instances

This section is devoted to the evaluation of the behavior of our heuristic when solving CSP instances (decision problem). We first describe the experimental protocol we use. In subsection 5.2, we assess the sensitivity of our heuristic CHS to its parameters and the benefits of smoothing and resetting. Afterwards, we compare CHS with state-of-the-art variable ordering heuristics in subsection 5.3, before studying the behavior of CHS when it is used jointly with LC or COS in subsection 5.4. Finally, in subsection 5.5, we evaluate the practical interest of CHS in the particular case where the search is guided by a tree-decomposition.

5.1 Experimental Protocol

We consider all the CSP instances from the XCSP3 repository² and the XCSP3 competition 2018³, resulting in 16,947 instances. XCSP3, for XML-CSP version 3, is an XML-based format to represent instances of combinatorial constrained problems. Our solvers are compliant with the rules of the competition except that the global constraints *cumulative*, *circuit* and some variants of the *allDifferent* constraint (namely *except* and *list*) or the *noOverlap* constraint are not supported yet. Consequently, from the 16,947 obtained instances, we first discard 1,233 unsupported instances. We also remove 2,813 instances which are detected as inconsistent by the initial arc-consistency preprocessing and having no interest for the present comparison. Finally, we have noted that some instances appear more than once. In such a case, we keep only one copy. In the end, our benchmark contains 12,829 instances, including notably structured instances and instances with global constraints.

Regarding the solving step, we exploit MAC with restarts [27] before assessing the impact of our approach on a structural solving method, namely BTM-MAC+RST+Merge [21]. Roughly speaking, BTM-MAC+RST+Merge differs from MAC by the exploitation of the structure via the notion of tree-decomposition (i.e. a collection of subsets of variables, called *clusters*, which are arranged in the form of a tree [36]). While the search performed by MAC considers at each step all the remaining variables, one performed by BTM-MAC+RST+Merge only takes into account the unassigned variables of the current cluster. The clusters of the computed tree-decomposition are processed according to a depth-first traversal of the tree-decomposition starting from a cluster called the *root cluster* (see [21] for more details). For BTM-MAC+RST+Merge, the tree-decompositions are computed with the heuristic H_5 -TD-WT [21]. The first root cluster is the cluster having the maximum ratio number of constraints to its size minus one. At each restart, the selected root cluster is one which maximizes the sum of the weights of the constraints whose scope intersects the cluster. The merging heuristic is the one provided in [21]. Note that these settings except the variable ordering heuristic correspond to those used for the XCSP3 competitions 2017 and 2018 [15, 22, 23].

²<http://www.xcsp.org/series>

³<http://www.cril.univ-artois.fr/XCSP18/>

MAC and BTD-MAC+RST+Merge use a geometric restart strategy based on the number of backtracks with an initial cutoff set to 100 and an increasing factor set to 1.1. In order to make the comparison fair, the lexicographic ordering is used for the choice of the next value to assign. We consider the following heuristics $dom/wdeg$, $wdeg^{ca.cd}$, ABS, IBS and CHB as implemented in Gecode. For ABS, we fix the decay parameter γ to 0.999 as in [32]. Note that we do not exploit a probing step like one mentioned in [32]. So all the weights are initially set to 0. For CHB, we use the value parameters as given in [39]. We also introduce a new variant $dom/wdeg+s$ which we define as $dom/wdeg$ where the weights of constraints are smoothed at each restart, exactly as in CHS. For all the heuristics, ties (if any) are broken by using the lexicographic ordering.

We have written our own C++ code to implement all the compared variable ordering heuristics in this section, as well as the solvers that exploit them (MAC and BTD). By so doing, we avoid any bias related to the way the heuristics and solvers are implemented. In particular, the variable ordering heuristics are all implemented with equal refinement and care. Moreover, when comparing the variable ordering heuristics for a given solver, the only thing which differs is the variable ordering heuristic. Indeed, we use exactly the same propagators, the same value heuristic, etc. This ensures that we make a fair comparison. Finally, given a solver and a CSP instance, we consider that a variable ordering heuristic h_1 is better than another one h_2 if h_1 allows the solver to solve the instance faster than h_2 . Indeed, the aim of variable ordering heuristic is to make a good tradeoff between the size of the explored search tree and the runtime spent for choosing a relevant variable (remember that finding the best one is an NP-Hard task [30]). Since all the other parts of the solver are identical, the solving runtime turns to be a relevant measure of the quality of this tradeoff. Thus, when the comparison relies on a collection of instances, h_1 is said better than h_2 if it leads the solver to solve more instances than h_2 . If both lead to solve the same number of instance, ties are broken by considering the smaller cumulative runtime. At the end, note that our protocol is consistent with the recommendations outlined in [20].

The experiments are performed on Dell PowerEdge R440 servers with Intel Xeon Silver 4112 processors (clocked at 2.6 GHz) under Ubuntu 18.04. Each solving process is allocated a slot of 30 minutes and at most 16 GB of memory per instance. In the following tables, #solved (abbreviated sometimes #solv.) denotes the number of solved instances for a given solver and time is the cumulative runtime, i.e. the sum of the runtime over all the considered instances.

5.2 Impact of CHS Settings

In this part, we assess the sensitivity of CHS with respect to the chosen values for α_0 or δ . First, we observe the impact of α_0 value. Hence, we fix δ to 10^{-4} to start the search by considering the variable degrees then quickly exploit ERWA-based computation. We then vary the value of α_0 .

Table 1: Number of instances solved by MAC+CHS depending on the value of α_0 (between 0.1 and 0.9) for consistent instances (SAT), inconsistent ones (UNSAT), and all the instances (ALL) and the cumulative runtime (in hours) of MAC+CHS for all the instances.

α_0	# solved instances			time (h)
	SAT	UNSAT	ALL	
0.1	6,530	4,212	10,742	1,038.89
0.2	6,505	4,206	10,711	1,049.55
0.3	6,505	4,203	10,708	1,052.04
0.4	6,493	4,204	10,697	1,056.14
0.5	6,509	4,202	10,711	1,058.13
0.6	6,487	4,205	10,692	1,062.14
0.7	6,504	4,207	10,711	1,055.46
0.8	6,479	4,197	10,676	1,072.28
0.9	6,473	4,203	10,676	1,071.43
VBS	6,691	4,242	10,933	940.21

Table 1 presents the number of instances solved by MAC depending on the initial value of α_0 and the corresponding cumulative runtime. Here, we first vary α_0 between 0.1 and 0.9 with a step of 0.1. We also provide the results of the Virtual Best Solver (VBS). The VBS is a theoretical/virtual solver that returns the best answer obtained by MAC with a given α_0 among those considered here. Roughly, it allows to count the number of the instances solved at least one time when varying the value of α_0 , while considering the smaller corresponding runtime. Table 1 shows that the results obtained for the different values of α_0 are relatively close to each others. However, we can observe that the value $\alpha_0 = 0.1$ allows MAC to solve more instances (10,742 solved instances with a cumulative solving time of 1,038.89 hours) than the other considered values. More precisely, MAC with CHS and $\alpha_0 = 0.1$ solves at least 31 additional instances. The worst cases are $\alpha_0 = 0.8$ and $\alpha_0 = 0.9$ with 10,676 instances solved respectively in 1,072 and 1,071 hours. If we discard the value 0.1 for α_0 , we observe that the results for the remaining considered values are quite close. This shows that CHS is relatively robust w.r.t. the α_0 parameter. Moreover, we can also remark that these observations are still valid if we focus on SAT instances (respectively on UNSAT instances). For example, the choice $\alpha_0 = 0.1$ leads to solving the largest number of SAT instances (resp. UNSAT instances), exactly 6,530 instances (resp. 4,212 instances). Figures 1 and 2 also show that $\alpha_0 = 0.1$ is the best choice among the experimented values. Indeed, we can note that the curve corresponding to $\alpha_0 = 0.1$ is almost always

above the others in both figures. These two figures also highlight the robustness of CHS w.r.t. the value of α_0 since all the curves are quite close.

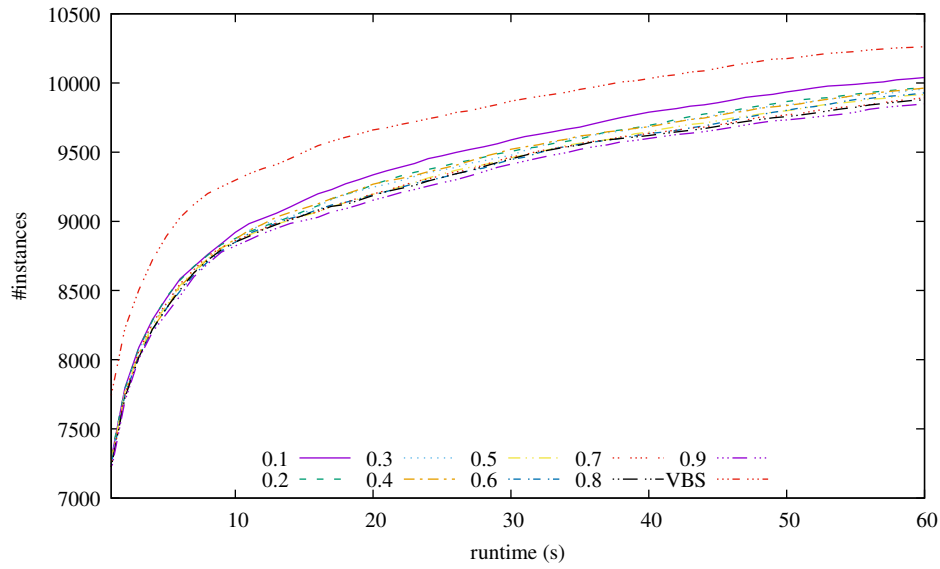


Figure 1: Number of solved instances as a function of the elapsed time for α_0 varying between 0.1 and 0.9 and the VBS, for a runtime between 1 s and 60 s.

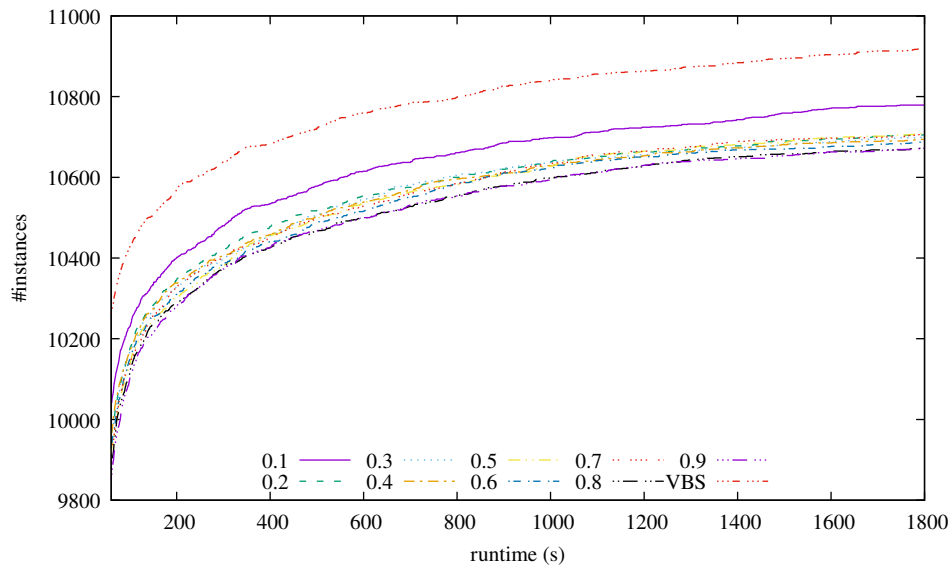


Figure 2: Number of solved instances as a function of the elapsed time for α_0 varying between 0.1 and 0.9 and the VBS, a for runtime between 60 s and 1,800 s.

Since the value $\alpha_0 = 0.1$ leads to the best result, a natural question is what happens if we consider the value $\alpha_0 = 0$ (which is normally a forbidden value since $0 < \alpha < 1$). So we run MAC+CHS with $\alpha_0 = 0$. In this case, the number of solved instances decreases significantly since only 9,069 instances are solved. At the same time, the runtime is almost doubled with a cumulative runtime of 1,921.35 hours. Consequently, the benefit of CHS is highly related to the tradeoff between the rewards of the past conflicts and the reward of the last one and so choosing a positive value for α_0 is crucial. The impact of this tradeoff is reinforced by the fact that MAC+CHS with $\alpha_0 = 1$ (a forbidden value too) performs worse than most of the combinations of MAC with α_0 between 0.1 and 0.9. Indeed, it only solves 10,667 instances while spending more time (1,089.37 h).

Likewise, we can wonder what happens if we choose a value slightly different from 0.1. Hence, we now vary α_0 between 0.025 and 0.15 with a step of 0.025 (see Table 2). Again, MAC+CHS with $\alpha_0 = 0.1$ turns to be the best case by solving more instances and obtaining the smallest cumulative runtime. Furthermore, the robustness of CHS w.r.t. the α_0 parameter is strengthened since we can note that the other values of α_0 obtain close results.

Regarding the Virtual Best Solver (VBS) in Table 1, we note that it can solve 191 additional instances than MAC+CHS when

Table 2: Number of instances solved by MAC+CHS depending on the value of α_0 (between 0.025 and 0.15) for consistent instances (SAT), inconsistent ones (UNSAT), and all the instances (ALL). and the cumulative runtime (in hours) of MAC+CHS for all the instances.

α_0	# solved instances			time (h)
	SAT	UNSAT	ALL	
0.025	6,507	4,202	10,709	1,061.07
0.05	6,512	4,212	10,724	1,058.89
0.075	6,500	4,204	10,704	1,064.61
0.1	6,530	4,212	10,742	1,038.89
0.125	6,519	4,203	10,722	1,078.10
0.15	6,503	4,207	10,710	1,061.81

$\alpha_0 = 0.1$ with the best runtime of 940.21 h. We can also remark that most of these additional instances are consistent (161 SAT instances vs. 30 UNSAT). If we consider the results instance per instance, we observe that 10,478 instances are solved whatever the chosen value for α_0 , which shows again the robustness of CHS w.r.t. the value of α_0 . Furthermore, among the 455 remaining ones, there exists 106 instances which are only solved by MAC with a particular value for α_0 (of course this value depends on the considered instance) and for 32% of the instances, MAC needs more than 1,200 seconds in order to solve each of them. Accordingly, some instances seem to be harder to solve. Finally, we observe that these 455 instances belong to several families. Indeed, more than half of the considered families are involved here, which shows that this phenomenon is more related to the instances themselves than to a particular feature of their family.

Now, we set α_0 to 0.1 and evaluate different values of δ (see Table 3). The observations are similar to those presented previously, showing the robustness of CHS regarding δ . Also, it is interesting to highlight that MAC+CHS with $\delta = 0$ solves 10,683 instances while it solves 10,742 instances if $\delta = 10^{-4}$. This illustrates the relevance of introducing δ in CHS since it allows to solve 59 more instances with this last setting.

Table 3: Impact of the value of δ on MAC+CHS regarding the number of solved instances and the cumulative runtime in hours.

δ	SAT	UNSAT	ALL	time (h)
0	6,479	4,204	10,683	1,079.25
10^{-5}	6,519	4,207	10,726	1,043.53
10^{-4}	6,530	4,212	10,742	1,038.89
10^{-3}	6,508	4,199	10,707	1,044.41

Table 4 gives the results of MAC+CHS ($\alpha_0 = 0.1$, $\delta = 10^{-4}$) with smoothing (+s) the constraint scores or without (-s) and/or with resetting (+r) the value of α to 0.1 at each new restart or without (-r). The observed behaviors clearly support the importance of smoothing and restarts for CHS. For example, MAC+CHS_{+s-r} solves 13 less instances than MAC+CHS, while MAC+CHS_{-s+r} solves 84 instances less.

Table 4: Number of instances solved by MAC with CHS with/without smoothing and reset of α and cumulative runtime in hours.

Solver	SAT	UNSAT	ALL	time (h)
MAC+CHS (+s+r)	6,530	4,212	10,742	1,038.89
MAC+CHS _{+s-r}	6,520	4,209	10,729	1,043.95
MAC+CHS _{-s-r}	6,484	4,199	10,683	1,064.20
MAC+CHS _{-s+r}	6,482	4,176	10,658	1,067.72

Finally, these results are globally consistent with those presented in [16]. Indeed, except that the best value of α_0 is now 0.1 instead of 0.4 in [16], we observe the same trends. The benchmark used in [16] was a subset of our initial benchmark. If we proceed similarly by removing arc-inconsistent instances, we obtain a benchmark with 7,916 instances. From this benchmark, MAC solved respectively 6,700 and 6,706 instances with 0.1 and 0.4 for α_0 in [16], while in the current experiments, it succeeds in solving 6,837 and 6,829 instances. In both cases, the gap between the two values of α_0 is very small. Note that the increase in the number of solved instances is mainly related to some improvements in our implementation and the difference of hardware configurations. Both impact all the heuristics in the same manner.

5.3 CHS vs. Other Search Heuristics

Now, we compare CHS to other search strategies from the state-of-the-art, namely *dom/wdeg*, *wdeg^{ca.cd}*, ABS, IBS and CHB. In the remaining part of the paper, by default, we consider CHS with $\alpha_0 = 0.1$ and $\delta = 10^{-4}$. We also consider the variant *dom/wdeg+s* that we introduced for *dom/wdeg*.

Figure 3 presents the number of solved instances as a function of the elapsed time for each considered heuristic. Since no heuristic outperforms another for all instances or families of instances, Tables 5-8 give some details for each family of instances. They allow to have a better insight of the kind of instances for which CHS is relevant. More accurately, for each family, they provide

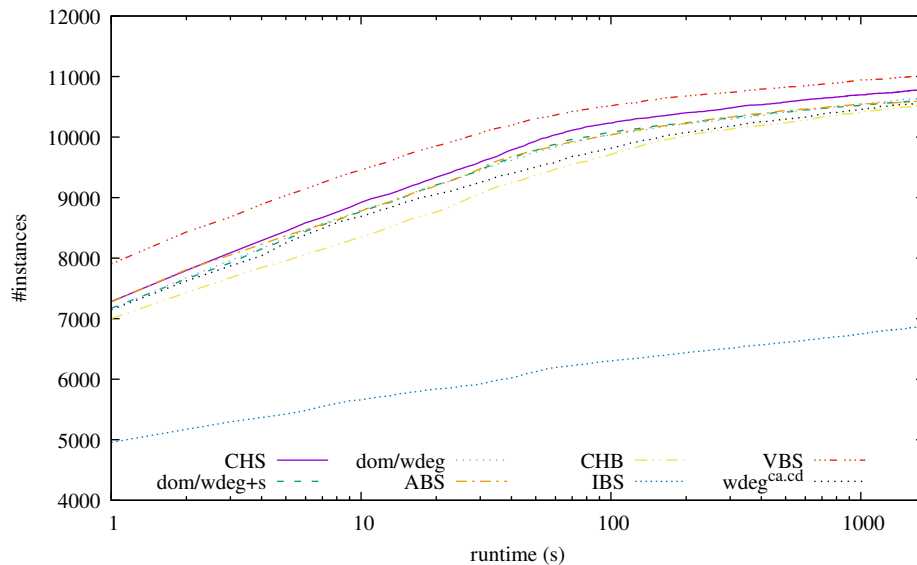


Figure 3: Number of solved instances as a function of the elapsed time (with a logarithmic scale) for the considered heuristics (namely CHS, $dom/wdeg+s$, $dom/wdeg$, $wdeg^{ca.cd}$, ABS, CHB and IBS) and the VBS based on these seven heuristics.

on rows C the number of instances solved by MAC with each considered heuristic (excluding IBS⁴) and the cumulative runtime for solving them for each heuristic, and on rows T the total number of instances of the family, the number of solved instances and the corresponding cumulative runtime for each heuristic. For each row, we write in bold the result of the best heuristic. As mentioned in our experimental protocol and like the solver competitions, we first consider the number of solved instances and we break ties by considering the cumulative runtime (given in seconds, except for the total runtimes which are expressed in hours). We only provide two digits after the decimal dot when the runtime does not exceed 100 seconds. Beyond, such details do not bring a significant information. We divide the instance families into three categories: academic, real-world and XCSP3 2018 competition. For that, we use the labeling from the XCSP3 repository.

From Figure 3 and Tables 5-8, it is clear that MAC with CHS performs better than any other heuristics whether in terms of the number of solved instances or runtime. Indeed, for example, $dom/wdeg$ is the heuristic closest to CHS but leads to solve 92 instances less. At the same time, CHS solves 127 instances more than $MAC+dom/wdeg+s$ and 174 more than $MAC+wdeg^{ca.cd}$. Likewise, it solves 134 additional instances w.r.t. $MAC+ABS$.

Now, if we consider the heuristic CHB which is based on conflict history like CHS, the gap with CHS is even greater (213 instances). This last result shows that the calculation of weights by ERWA on the constraints (as done in CHS) is more relevant than its calculation on the variables (as done in CHB). Note that the poor score of IBS is mainly related to the estimation of the size of the search tree (i.e. the product of the domain sizes [35]). In fact, we observe that, for many instances, the value of the estimation exceeds the capacity of representation of `long double` in C++. Finally, these trends are still valid if we focus on SAT instances or UNSAT ones.

Interestingly, whatever the value of α_0 , MAC with CHS remains better than all its competitors. Indeed, the worst case is observed when the value of α_0 is equal to 0.8 or 0.9 with 10,676 solved instances. This observation also holds for the version of CHS in which we disable the smoothing or the resetting of α . This clearly highlights the practical interest of our approach.

If we look at the results more closely, i.e. for each family (see Tables 5-8), we observe that no heuristic dominates the others. Indeed, if CHS is the heuristic that leads most often to the best results (for 13 families), the other heuristics are close (notably 10 families for $wdeg^{ca.cd}$, ABS and CHB). This makes the choice of a particular heuristic difficult, as it is highly dependent on the instance or the family of instances to be processed. This probably explains the gap between VBS and MAC with any heuristic (e.g. 10,982 solved instances for the VBS against 10,812 for MAC with CHS). Curiously, $dom/wdeg+s$ only ranks first for 3 families while being globally ranked at the third place. As CHS, it rarely performs significantly worse than the other heuristics.

To illustrate this phenomenon, let us consider the difference between the number of instances solved by the VBS and the corresponding number for MAC, for each family, with each heuristic. This number can be seen as a measure of the robustness of the heuristic. Table 9 provides the mean and the standard deviation of this difference for each heuristic. It shows that CHS is the most robust heuristic by obtaining the smallest mean and standard deviation.

⁴Given the poor results of MAC with IBS, including IBS leads to a less relevant comparison on instances solved by MAC with each heuristic since it significantly decreases the number of such instances.

Table 5: Detailed results (number of instances and runtime) of MAC with CHS, $dom/wdeg++$, $dom/wdeg$, $wdeg^{ca,cd}$, ABS or CHB for each considered family.

Family	# instances	CHS		$dom/wdeg++$		$dom/wdeg$		$wdeg^{ca,cd}$		ABS		CHB	
		#solv.	time	#solv.	time	#solv.	time	#solv.	time	#solv.	time	#solv.	time
AllInterval	C	22	0.20	32	9.69	24	89.26	23	1,080	32	0.20	32	0.20
	T	32	3.95	4	2,810	4	15,907	4	17,284	4	4.08	4	4.07
Basic	C	4	0.02	4	0.01	4	0.01	4	0.01	4	0.01	4	0.03
	T	4	0.02	4	0.01	4	0.01	4	0.01	4	0.01	4	0.03
Bibd	C	73	574	98	5,084	114	5,625	110	8,301	119	170	107	4,686
	T	312	141,154	246,462	246,462	225,675	225,675	222,637	194,960	194,960	226,228	226,228	
Blackhole	C	84	1,379	85	1,798	85	416	85	10,03	85	1,586	85	7.17
	T	112	51,779	50,398	50,398	49,016	49,016	48,610	48,610	48,610	50,189	48,610	48,607
CarSequencing	C	3	10.29	9	62.82	8	485	9	291	16	481	5	0.38
	T	52	75,804	77,613	77,613	82,166	82,166	63,172	70,836	70,836	84,601	84,601	
ColouredQueens	C	5	0.89	5	0.90	5	0.71	5	0.39	5	0.74	5	0.27
	T	17	21,601	21,601	21,601	21,601	21,601	21,600	21,600	21,601	21,601	21,600	21,600
CostasArray	C	8	79.16	8	773	9	468	9	62.99	9	428	9	274
	T	11	5,479	6,173	6,173	5,503	5,503	4,477	4,477	5,311	5,311	4,196	4,196
CryptoPuzzle	C	10	0.01	10	0.01	10	0.01	10	0.01	10	0.01	10	0.01
	T	10	0.01	10	0.01	10	0.01	10	0.01	10	0.01	10	0.01
DeBruijnSequence	C	12	534	12	488	12	529	12	510	12	530	12	528
	T	18	7,765	11,377	11,377	7,775	7,775	11,401	11,401	9,570	9,570	12	7,752
DiamondFree	C	31	524	33	2,274	33	1,789	36	3,127	38	13.31	38	10.89
	T	38	5,009	12,116	12,116	11,537	11,537	9,206	9,206	21.97	21.97	38	19.23
Domino	C	37	256	37	256	37	243	37	278	37	261	37	261
	T	37	256	256	256	243	243	278	278	261	261	37	261
Driver	C	7	23.10	7	33.15	7	16.75	7	30.15	7	50.89	7	11.89
	T	7	23.10	33.15	33.15	16.75	16.75	30.15	30.15	50.89	50.89	7	11.89
Dubois	C	10	948	10	964	11	1,386	11	1,158	11	237	11	1,271
	T	30	36,080	36,964	36,964	36,992	36,992	36,534	36,534	27,440	36,863	36,863	
GracefulGraph	C	12	0.04	17	2.07	16	33.81	14	5.86	16	0.69	14	1.14
	T	104	153,785	157,580	157,580	160,081	160,081	162,902	162,902	158,843	158,843	14	162,121
Hanoi	C	6	3.36	6	3.72	6	4.39	6	2.33	6	4.19	6	3.56
	T	7	3.36	3.72	3.72	4.39	4.39	2.33	4.19	4.19	4.19	6	3.56
Haystacks	C	2	4.20	2	1.57	2	3.16	2	0.04	2	2.47	2	0.51
	T	51	88,204	88,202	88,202	88,203	88,203	88,200	88,203	88,203	88,203	2	88,201

Table 6: Detailed results (number of instances and runtime) of MAC with CHS, dom/wdeg+s, dom/wdeg, wdeg^{ca.cd}, ABS or CHB for each considered family (Table 5 continued).

Family	# instances	CHS		dom/wdeg+s		dom/wdeg		wdeg ^{ca.cd}		ABS		CHB	
		#solv.	time	#solv.	time	#solv.	time	#solv.	time	#solv.	time	#solv.	time
Kakuro	C	1,084	96.75										
	T	1,102	3,896	1,101	68.80	1,101	205	1,096	805	1,088	1,554	1,086	6,110
Knights	C	12	293		461		438		525		1,019		444
	T	19	11,836	13	12,334	13	12,295	13	12,477	12	13,625	13	12,074
KnightTour	C	5	186		198		185		18.42		2.26		387
	T	25	32,527	6	32,753	6	32,724	12	26,651	9	28,114	14	22,352
Langford	C	46	1,614		1,387		1,312		1,378		512		2,657
	T	125	141,969	49	141,486	49	141,478	49	138,236	67	108,978	58	124,732
LatinSquare	C	266	660		875		413		1,250		604		1,628
	T	366	160,383	276	166,675	275	166,858	274	172,571	271	174,580	276	166,455
MagicSequence	C	83	536		473		473		1,170		443		982
	T	85	536	83	473	83	473	83	1,170	83	443	83	982
MagicSquare	C	18	382		836		350		3,271		434		2,640
	T	86	70,825	54	59,015	42	76,997	19	100,476	58	49,814	43	88,153
MarketSplit	C	10	266		261		261		254		103		280
	T	10	266	10	261	10	261	10	254	10	103	10	280
MaxCSP	C	2,186	126		75.89		55.43		238		55.89		156
	T	2,186	2,186	2,186	75.89	2,186	55.43	2,186	238	2,186	55.89	2,186	156
Mixed	C	11	213		472		384		264		442		296
	T	18	4,258	14	6,601	17	5,760	15	5,458	14	7,674	16	4,160
MultiKnapsack	C	28	11.87		13.97		38.68		36.92		2.59		227
	T	31	211	31	332	31	605	31	592	31	12.42	28	5,627
Nonogram	C	351	162		77.09		247		597		45.90		280
	T	356	3,781	355	2,956	354	4,631	355	2,998	354	5,005	354	3,903
NumberPartitioning	C	12	61.07		1,651		209		114		8.59		4.37
	T	50	58,781	13	68,253	14	65,070	16	63,436	27	43,591	28	43,068
Ortholatin	C	4	3.69		1.61		3.52		357		16.43		616
	T	27	37,816	4	41,406	4	41,407	7	36,368	4	41,418	4	42,021
PigeonsPlus	C	29	85.21		89.18		81.29		143		2,745		89.45
	T	38	4,179	37	4,065	37	4,044	36	5,914	29	18,945	37	4,261
Primes	C	134	838		240		139		1,014		64.35		527
	T	160	35,478	143	32,080	145	29,488	143	37,623	143	31,633	137	43,784

Table 7: Detailed results (number of instances and runtime) of MAC with CHS, $dom/wdeg+s$, $dom/wdeg$, $wdeg^{ca,cd}$, ABS or CHB for each considered family (Table 6 continued).

Family	# instances	CHS		$dom/wdeg+s$		$dom/wdeg$		$wdeg^{ca,cd}$		ABS		CHB	
		#solv.	time	#solv.	time	#solv.	time	#solv.	time	#solv.	time	#solv.	time
PseudoBoolean	C	109	1,524		1,793		1,927		1,195		1,892		2,967
	T	337	361,038	129	385,566	114	390,684	121	399,516	137	347,537	135	354,105
QRandom	C	607	6,180		7,697		7,331		13,907		8,259		21,394
	T	614	10,287	613	11,569	614	9,706	613	18,716	612	14,170	609	33,109
QuasiGroups	C	24	482		528		442		499		156		622
	T	148	203,502	26	222,532	25	204,709	24	223,707	24	205,359	24	205,826
QueenAttacking	C	4	3.78		22.46		0.71		9.50		172.81		7.17
	T	10	9,021	5	9,171	5	9,082	4	10,810	4	10,973	4	10,807
Queens	C	18	497		85.04		65.89		2,081		34.66		208
	T	24	4,805	21	2,353	21	2,362	18	9,283	22	2,355	20	4,633
QueensKnights	C	10	55.19		86.39		83.74		88.47		175.85		101
	T	18	5,939	15	6,766	15	6,949	15	6,830	10	14,576	16	6,397
RadarSurveillance	C	81	2.54		2.30		2.35		2.19		3.69		1,006
	T	90	3.58	90	3.90	90	3.64	90	2.89	90	5.53	81	17,206
Random	C	1,591	65,626		71,650		59,499		126,920		75,109		194,100
	T	1,955	605,428	1,678	635,177	1,703	585,527	1,664	705,018	1,674	633,027	1,623	819,665
RoomMate	C	17	289		398		363		309		276		257
	T	30	289	17	398	17	363	17	309	17	276	17	257
Sat	C	356	6,620		3,915		5,071		2,867		3,467		7,149
	T	366	18,203	361	13,996	361	15,163	361	12,897	361	12,822	357	17,004
Scheduling	C	80	1,076		1,230		1,147		543		29.27		67.29
	T	107	33,332	85	40,923	86	41,489	92	29,932	88	34,230	92	28,486
SchurrLemma	C	7	374		409		286		522		308		564
	T	10	3,974	8	4,009	9	2,700	8	4,123	8	3,908	7	5,964
Steiner3	C	4	4.45		0.07		0.06		0.08		6.62		0.96
	T	10	7,204.45	5	7,213.70	5	7,212.11	5	5,516.92	4	7,206.62	4	7,200.96
Subisomorphism	C	1,616	21,942		19,362		20,467		29,582		28,134		43,863
	T	1,878	264,950	1,699	275,951	1,707	264,534	1,681	307,266	1,692	281,407	1,676	321,878
Sudoku	C	92	0.22		0.19		0.20		0.37		0.25		0.23
	T	92	0.22	92	0.19	92	0.20	92	0.37	92	0.25	92	0.23

Table 8: Detailed results (number of instances and runtime) of MAC with CHS, $dom/wdeg+s$, $dom/wdeg$, $wdeg^{ca,cd}$, ABS or CHB for each considered family (Table 7 continued).

Family	# instances	CHS		$dom/wdeg+s$		$dom/wdeg$		$wdeg^{ca,cd}$		ABS		CHB	
		#solv.	time	#solv.	time	#solv.	time	#solv.	time	#solv.	time	#solv.	time
Academic	C	80	324				1,436		581		2,024		677
	T	330	404,455	92	76,95 428,717	119	391,364 395	117	396,668	89	439,039	107	411,775
TravellingSalesman	C	45	456		269		395		1,347		247		728
	T	45	456	45	269	45	395	45	1,347	45	247	45	728
Renault	C	14	0.03		0.05		0.05		0.03		0.04		0.03
	T	14	0.03	14	0.05	14	0.05	14	0.03	14	0.04	14	0.03
RenaultMod	C	50	0.29		0.48		0.84		0.76		0.34		0.50
	T	50	0.29	50	0.48	50	0.84	50	0.76	50	0.34	50	0.50
Rlfap	C	54	45.57		50.82		40.77		109.34		299.73		1,087.50
	T	60	4,301	57	7,553	59	3,608	59	4,294	56	7,507	57	8,095
SocialGolfers	C	322	2,632		2,791		2,780		1,265		1,036		3,650
	T	460	231,754	337	232,076	338	228,481	346	210,490	336	228,857	343	217,181
SportsScheduling	C	3	0.12		0.04		0.12		0.05		0.02		0.03
	T	19	25,305	4	27,003	4	27,005	6	25,160	5	26,659	3	28,800
Wwtpp	C	234	459		721		12,811		17,854		23,687		2,251
	T	371	111,155	312	114,889	300	166,108	281	186,078	266	214,019	288	158,115
Academic	C	9,346	32.03 h		35.13 h		31.78 h		57.43 h		36.70 h		82.80 h
	T	11,590	9,846 857.42 h	9,731	924.33 h	9,778	895.38 h	9,702	954.69 h	9,769	884.90 h	9,666	978.93 h
Real-World	C	677	0.87 h		0.99 h		4.34 h		5.34 h		6.95 h		1.94 h
	T	974	778 103.48 h	774	105.98 h	765	118.11 h	756	118.34 h	727	132.51 h	755	114.50 h
Competitions	C	84	1.03 h		1.24 h		1.53 h		0.80 h		0.64 h		2.57 h
	T	265	118 78.00 h	110	82.55 h	107	83.04 h	110	82.66 h	112	79.73 h	108	82.86 h
All	C	10,107	33.94 h		37.36 h		37.65 h		63.58 h		44.29 h		87.32 h
	T	12,829	10,742 1,038.89 h	10,615	1,112.85 h	10,650	1,096.54 h	10,568	1,155.69 h	10,608	1,097.14 h	10,529	1,176.29 h

Table 9: Mean and standard deviation of the difference between the number of instances solved by the VBS and the corresponding number for MAC with each heuristic.

	CHS	$dom/wdeg+s$	$dom/wdeg$	$wdeg^{ca.cd}$	ABS	CHB
Mean	5.11	7.38	6.75	8.21	7.50	8.91
Standard deviation	9.25	13.93	11.17	15.98	14.91	19.11

Finally, our observations are consistent with ones in [16]. In particular, MAC clearly performs better with CHS than with any other heuristic, notably the two powerful and popular variable ordering heuristics $dom/wdeg$ and ABS. The gap between CHS and the other heuristics has widened with the increase in the number of instances taken into account.

5.4 Combination with LC and COS

LC and COS are two branching strategies based on conflicts which require an auxiliary variable ordering heuristic in order to choose a variable when no conflict can be exploited. In this subsection, we study the behavior of CHS and some heuristics of the state-of-the-art when they are used jointly with LC or COS. We only keep the three best heuristics according to the results of the previous subsection, namely $dom/wdeg+s$, $dom/wdeg$ and ABS.

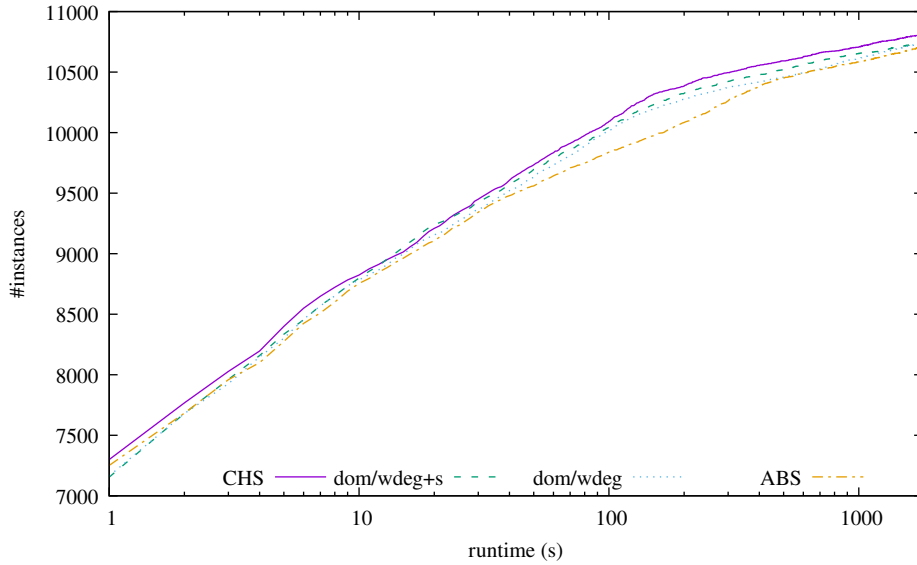


Figure 4: Number of solved instances as a function of the elapsed time (with a logarithmic scale) for LC with the heuristics CHS, $dom/wdeg+s$, $dom/wdeg$ or ABS.

First, we consider the case of LC. Figure 4 presents the number of solved instances as a function of the elapsed time for LC combined with each considered heuristic. As a first observation, we can note that using LC does not change the ranking obtained in the previous subsection. Namely, LC combined with CHS leads to the best results followed by $dom/wdeg+s$, $dom/wdeg$ and ABS. Indeed, as we can see in Table 10, MAC with LC and CHS solves more instances and solves them more quickly than MAC with LC and any other heuristic. Moreover, for any considered heuristic h , we can also remark that MAC with LC and h performs better and faster than MAC with h . For instance, MAC with LC and CHS solves 10,812 in 1,017.03 hours against 10,742 instances solved in 1,038.89 hours for MAC with CHS. We can also observe that the gain in the number of solved instances thanks to MAC with LC and h w.r.t. MAC with h varies according to h (70 instances for CHS and 110 instances for ABS). This probably reflects the fact that the less efficient the heuristic is, the easier it is to solve additional instances. To this end, LC with CHS turns to be the most interesting variable ordering heuristic among all the heuristics we consider in our experiments.

Now, we assess the behavior of MAC when using COS with any auxiliary heuristic among CHS, $dom/wdeg+s$, $dom/wdeg$ and ABS. As shown in Figure 5 and Table 10, combining COS with any heuristic leads to decrease significantly the ability of MAC to solve instances. Indeed, we can observe that MAC using COS and any heuristic solves at least 346 instances less than MAC using solely the auxiliary heuristic. Thus, if the ranking remains the same, the gap between MAC with COS and CHS and MAC with COS and any other auxiliary heuristic is narrower (from 92 instances when the heuristics are exploited alone to 16 instances with COS). A possible explanation of this behavior is that MAC only exploits the auxiliary heuristic when there is no more variable appearing in conflicts. This occurs at the beginning of the search when no conflict has been encountered yet or when all the variables appearing in past conflicts are assigned. Clearly, the first case concerns few nodes in the search tree. For the second case, it may be the same too as soon as many variables are involved in the encountered conflicts. In addition, a potential drawback of COS is that the conflicts

exploited by COS may be old and so have less sense at some steps of the search.

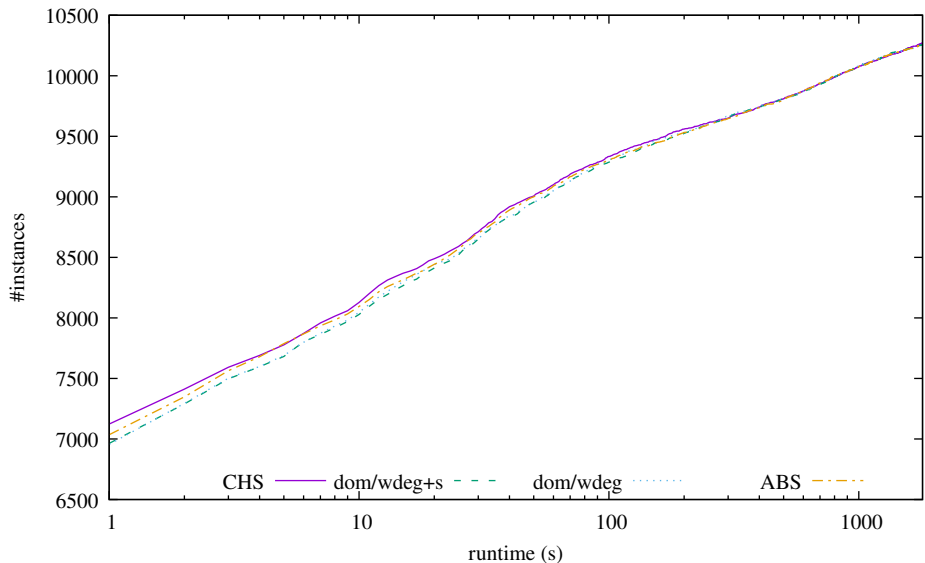


Figure 5: Number of solved instances as a function of the elapsed time (with a logarithmic scale) for COS with the heuristics CHS, $dom/wdeg+s$, $dom/wdeg$ or ABS.

Table 10: Number of instances solved by MAC with LC/COS with any auxiliary heuristic among CHS, $dom/wdeg+s$, $dom/wdeg$ or ABS, and cumulative runtime in hours.

Auxiliary heuristic	LC		COS	
	#solved	time (h)	#solved	time (h)
CHS	10,812	1,017.03	10,281	1,363.86
$dom/wdeg+s$	10,752	1,057.91	10,265	1,368.66
$dom/wdeg$	10,741	1,067.28	10,259	1,367.17
ABS	10,718	1,090.23	10,262	1,368.99

5.5 CHS and Tree-Decomposition

We now assess the behavior of CHS when the search is guided by a tree-decomposition. Studying this question is quite natural since CHS aims to exploit the structure of the instance, but in a way different from what the tree-decomposition does. With this aim in view, we consider BTD-MAC+RST+Merge [21] and the heuristics CHS, $dom/wdeg+s$, $dom/wdeg$ and ABS combined or not with LC. As shown in Figure 6 and Table 11, the trends observed for MAC are still valid for BTD-MAC+RST+Merge.

Table 11: Number of instances solved by BTD-MAC+RST+Merge with the heuristics CHS, $dom/wdeg+s$, $dom/wdeg$ and ABS combined or not with LC, and cumulative runtime in hours.

(Auxiliary) heuristic	without LC		with LC	
	#solved	time (h)	#solved	time (h)
CHS	10,770	1,035.59	10,839	1,011.22
$dom/wdeg+s$	10,712	1,065.01	10,805	1,032.30
$dom/wdeg$	10,672	1,089.00	10,767	1,061.63
ABS	10,650	1,082.71	10,705	1,093.49

Indeed, the solving is more efficient with CHS than with any other used heuristic by at least 58 additional instances. For example, BTD-MAC+RST+Merge with CHS solves 10,770 instances (in 1,035 h) against 10,712 instances (in 1,065 h) for $dom/wdeg+s$. Moreover, we can note that using BTD-MAC+RST+Merge instead of MAC does not change the ranking of the heuristics in terms of the number of solved instances or the cumulative runtime.

Likewise, we can make the same observations if we exploit LC (see Figure 7 and Table 11). Above all, BTD-MAC+RST+Merge with LC and CHS turns out to be more efficient than MAC with LC and any auxiliary heuristic. For example, it solves 27 additional

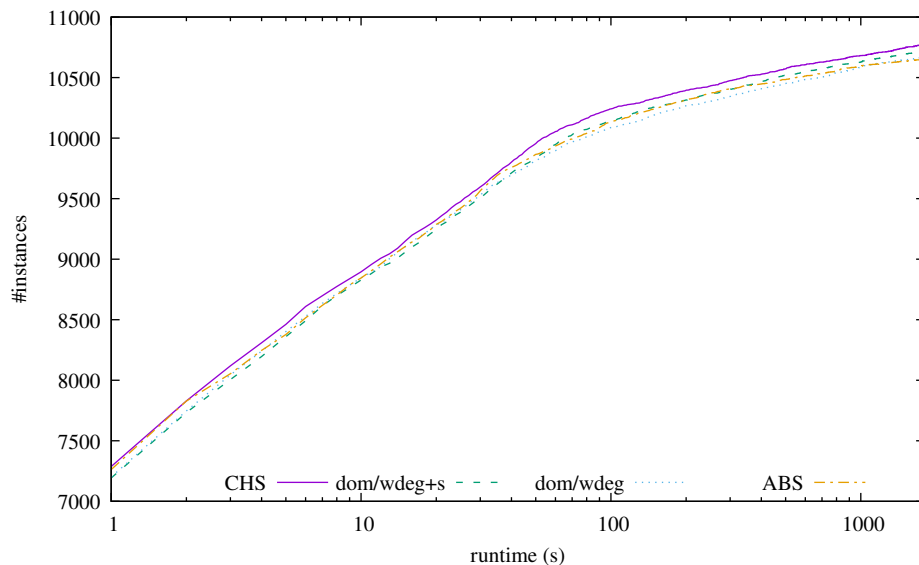


Figure 6: Number of instances solved by BT-D-MAC+RST+Merge as a function of the elapsed time (with a logarithmic scale) with the heuristics CHS, $dom/wdeg+s$, $dom/wdeg$ or ABS.

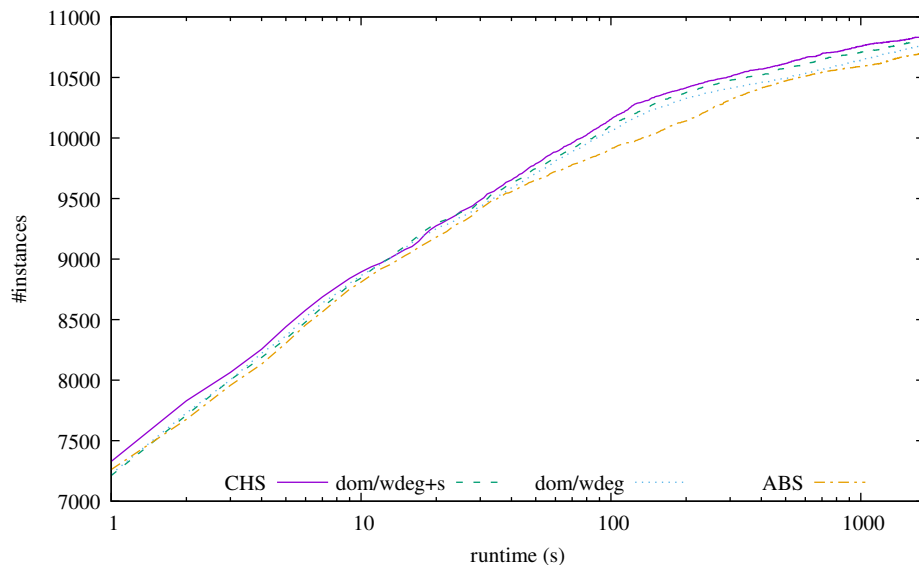


Figure 7: Number of instances solved by BT-D-MAC+RST+Merge as a function of the elapsed time (with a logarithmic scale) with LC combined with the heuristics CHS, $dom/wdeg+s$, $dom/wdeg$ or ABS.

instances compared to MAC with LC and CHS. All these observations show that exploiting both CHS and tree-decomposition may be of interest and that these two strategies can be complementary.

Finally, these results are consistent with the ones in [16]. They are also consistent with ones of the XCSP3 competition 2018. For instance, BT-D-MAC+RST+Merge participated in the mini-solvers track of the competition by using respectively $dom/wdeg$ (for the solver miniBT-D [23]) and CHS (for the solver miniBT-D_12 [15]) as variable ordering heuristic. miniBT-D_12 finished in the second place by solving 79 instances while miniBT-D was ranked third with 74 solved instances.

6 Experimental Evaluation on COP Instances

This section is devoted to the evaluation of the behavior of our heuristic when solving COP instances (optimization problem). Note that the constraint optimization problem (COP) differs from the constraint satisfaction problem by only the addition of an objective function to optimize. So solving a COP instance consists in assigning all the variables while satisfying all the constraints and optimizing the objective function. It is an NP-hard task [37].

We first describe the experimental protocol we use. Then, in subsection 6.2, we assess the sensitivity of our heuristic CHS to its

parameters and the benefits of smoothing and resetting. Finally, we compare CHS with state-of-the-art variable ordering heuristics in subsection 6.3.

6.1 Experimental Protocol

We consider the COP instances from the 2019 XCSP3 competition⁵. Like for CSP instances, we discard 36 instances containing some global constraints which are not handled by our library yet. In the end, our benchmark contains 264 instances, including notably structured ones and instances with global constraints.

The experiments are performed in the same conditions as for CSP instances. In particular, we use the same value heuristic, the same settings for variable ordering heuristics, restarts, Regarding the solving step, we exploit a branch and bound algorithm based on MAC with restarts and denoted MAC-BnB. We distinguish three statuses when solving a COP instance. If the solver finds an optimal solution and proves the optimality within the allocated time slot (30 minutes), the instance has the status OPT meaning that it is has been optimally solved. However, if the solver has found a solution but cannot establish its optimality, the instance has the status SAT meaning that a solution has been found in the CSP sense but with no guarantee with respect to the objective function. In such a case, the solver has only produced an upper bound (resp. a lower bound) if the instance aims to minimize (resp. maximize) the objective function. Finally, if the solver proves that the instance has no solution, the instance has the status UNSAT. In the following, an instance is said *solved* if it has the status OPT or UNSAT.

6.2 Impact of CHS Settings

In this part, we assess the sensitivity of CHS with respect to the chosen values for α_0 or δ when solving COP instances. First, we study the impact of α_0 value. With this aim in view, we set δ to 10^{-4} and then vary the value of α_0 between 0.1 and 0.9 with a step of 0.1.

Table 12: Number of instances having the status OPT, UNSAT or SAT depending on the value of α_0 (between 0.1 and 0.9) and the cumulative runtime (in hours) for all the instances.

α_0	# instances			time (h)
	OPT	UNSAT	SAT	
0.1	119	1	86	67.48
0.2	121	1	83	66.75
0.3	124	1	80	66.10
0.4	126	1	78	62.38
0.5	124	1	73	66.31
0.6	120	1	84	68.18
0.7	120	1	83	68.73
0.8	113	1	91	70.65
0.9	117	1	85	70.08
VBS	140	1	66	59.04

Table 12 provides the number of instances having the status OPT, UNSAT or SAT depending on the initial value of α_0 and the corresponding cumulative runtime. We also provide the results of the Virtual Best Solver (VBS) built on the basis of this nine combinations of MAC-BnB and CHS. Table 12 shows that the results obtained for the different values of α_0 are relatively close to each others. Indeed, if we consider the number of solved instances, the best combination ($\alpha_0 = 0.4$) solves in average 6 additional instances and the gap with the worst one is 13 instances. Regarding the runtime, MAC-BnB and CHS with $\alpha_0 = 0.4$ correspond again to the best combination with a cumulative runtime of 62.38 h. The other combinations are generally 5% slower, except for the values 0.8 and 0.9 of α_0 for which the rate is about 10%. Globally, these results are consistent with ones obtained when solving CSP instances and show again the robustness of CHS with respect to the value of α_0 . This robustness is also highlighted by the fact that all the curves in Figure 8 are quite close. Moreover, from this figure, we can note that $\alpha_0 = 0.4$ is the best choice among the experimented values. Indeed, the corresponding curve is almost always above the others.

Regarding the Virtual Best Solver (VBS) in Table 12, we note that it can solve 14 additional instances than MAC-BnB and CHS with $\alpha_0 = 0.4$ while saving 3.34 hours. If we consider the results instance per instance, we observe that 103 instances among the ones solved by the VBS are solved whatever the chosen value for α_0 . Furthermore, 20 instances among the 38 remaining ones are solved by more than half of the combinations. Finally, the 18 remaining instances seem harder to solve with an average runtime for the VBS about 819 seconds.

Now, we set α_0 to 0.4 and consider different values of δ (see Table 13). The observations are similar to those presented previously, showing the robustness of CHS regarding δ . It turns out that using a non-zero values for δ allows MAC-BnB to perform better. This

⁵<http://www.cril.univ-artois.fr/XCSP19>

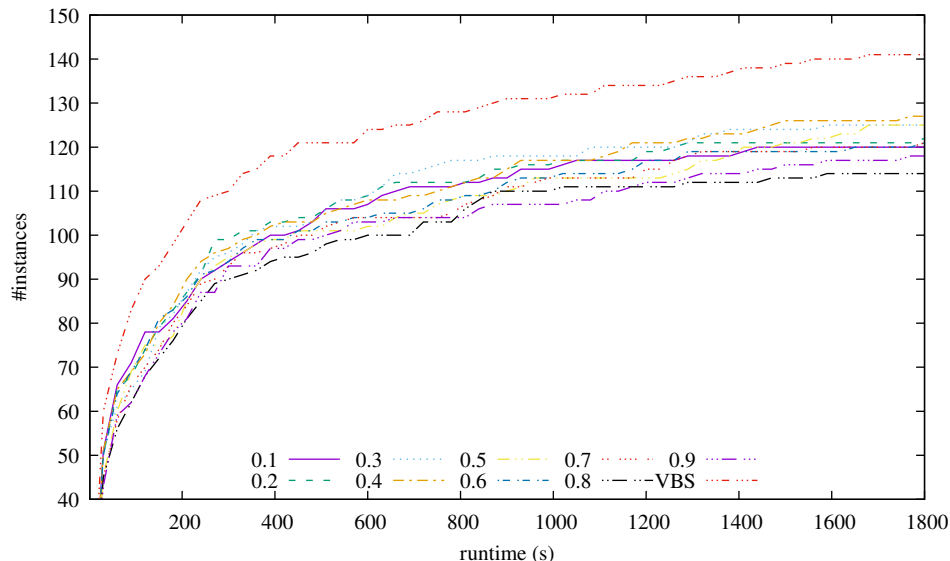


Figure 8: Number of solved COP instances as a function of the elapsed time for α_0 varying between 0.1 and 0.9 and the VBS.

shows the relevance of introducing δ in CHS. Finally, like for the CSP solving, the value 10^{-4} leads to obtain the best results in terms of the number of solved instances as well as the runtime.

Table 14 gives the results of MAC-BnB+CHS ($\alpha_0 = 0.4$, $\delta = 10^{-4}$) with smoothing (+s) the constraint scores or without (-s) and/or with resetting (+r) the value of α to 0.4 at each new restart or without (-r). The observed behaviors clearly support the importance of smoothing and restarts for CHS. For example, MAC-BnB with CHS+s-r solves 5 less instances than MAC-BnB with CHS, while MAC-BnB with CHS-s-r solves 11 instances less. In addition, it can be noted that removing the smoothing or the resetting lead to an increase in runtime.

Table 13: Impact of the value of δ regarding the number of instances having the status OPT, UNSAT or SAT and the cumulative runtime in hours.

δ	# instances			time (h)
	OPT	UNSAT	SAT	
0	120	1	84	67.89
10^{-5}	123	1	82	67.21
10^{-4}	126	1	78	62.38
10^{-3}	121	1	84	68.47

Table 14: Number of instances which are solved optimally (OPT), proved as inconsistent (UNSAT) or for which a solution is found (SAT) with CHS with/without smoothing and reset of α and the cumulative runtime (in hours) for all the instances.

Variant	# instances			time (h)
	OPT	UNSAT	SAT	
CHS(+s+r)	126	1	78	62.38
CHS+s-r	121	1	84	66.82
CHS-s-r	115	1	81	69.73
CHS-s+r	116	1	82	70.16

6.3 CHS vs. Other Search Heuristics

In this part, we compare CHS (with $\alpha_0 = 0.4$ and $\delta = 10^{-4}$) to other search strategies from the state-of-the-art, namely *dom/wdeg*, *wdeg^{ca.cd}*, ABS and CHB. We also consider the variant *dom/wdeg+s* that we introduced for *dom/wdeg*.

Figure 9 presents the number of solved instances as a function of the elapsed time for each considered heuristic. Clearly, CHS turns to be the more efficient heuristics. Indeed, MAC-BnB with CHS solves at least 13 additional instances than with any other considered heuristic while performing faster. More interestingly, CHS outperforms CHB with 49 additional solved instances. Nevertheless, no heuristic outperforms another for all instances or families of instances. So, Tables 15 and 16 give some details for each family of instances considered in the competition. They allow to have a better insight of the kind of instances for which CHS

is relevant. Note that we do not consider CHB in order to have a relevant comparison for instances which are solved with all the heuristics. Indeed, considering CHB dramatically reduces the number of instances solved by all the heuristics. Like for the decision problem, CHS is not always the better heuristic, but, it turns to be the more robust one. Finally, we can also remark that whatever the values chosen for α_0 or δ among the considered one, CHS performs better than the state-of-the-art heuristics. This observation still holds if CHS does not exploit smoothing and/or reset of α .

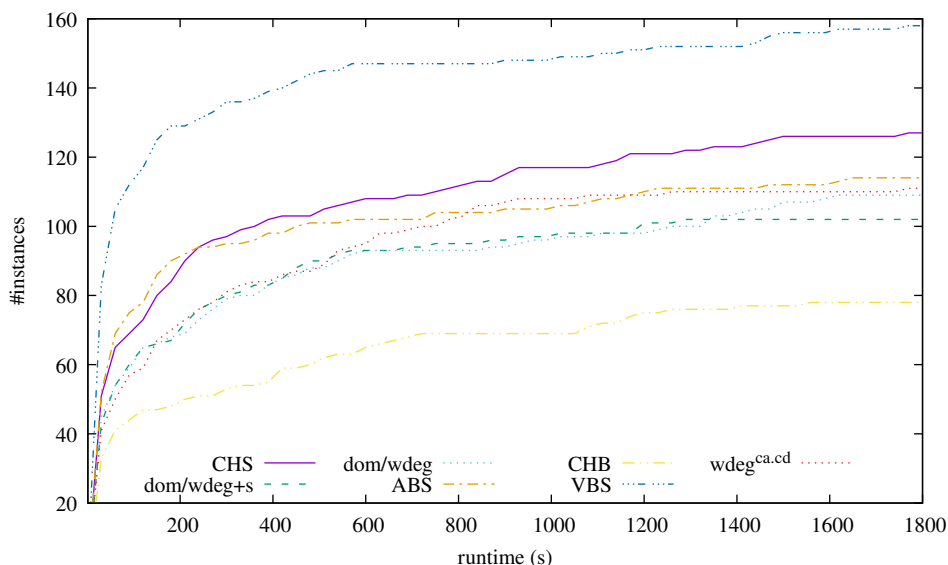


Figure 9: Number of solved instances as a function of the elapsed time for the considered heuristics (namely CHS, *dom/wdeg+s*, *dom/wdeg*, *wdeg^{ca.cd}*, and ABS) and the VBS based on these five heuristics.

7 Conclusion

We have proposed CHS, a new variable ordering heuristic for CSP based on the search history and designed following techniques inspired from reinforcement learning. The experimental results confirm the relevance of CHS, which is competitive with the most powerful heuristics, when implemented in solvers based on MAC or tree-decomposition exploitation. Our experiments also shows that CHS turns to be relevant for solving COP instances.

The experimental study suggests that the initial value of α parameter value could be refined. We will explore the possibility of defining its value depending on the instance to be solved. For example, we will look for probing techniques to fix its appropriate value. Furthermore, similarly to the ABS heuristic, we will also consider including information provided by filtering operations in CHS. Finally, we will measure the impact of CHS on solving other problems under constraints, such as counting and optimization when modeled as weighted CSP.

Acknowledgements

This work has been funded by the French Agence Nationale de la Recherche, reference ANR-16-CE40-0028.

References

- [1] Bachiri, I., Gaudreault, J., Quimper, C., Chaib-draa, B.: RLBS: An Adaptive Backtracking Strategy Based on Reinforcement Learning for Combinatorial Optimization. In: Proceedings of ICTAI, pp. 936–942 (2015)
- [2] Balafrej, A., Bessiere, C., Paparrizou, A.: Multi-Armed Bandits for Adaptive Constraint Propagation. In: Proceedings of IJCAI, pp. 290–296 (2015)
- [3] Battiti, R., Campigotto, P.: An Investigation of Reinforcement Learning for Reactive Search Optimization, pp. 131–160. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
- [4] Bessière, C., Chmeiss, A., Saïs, L.: Neighborhood-based variable ordering heuristics for the constraint satisfaction problem. In: Proceedings of CP, pp. 565–569 (2001)

Table 15: Detailed results (number of solved instances and runtime) with CHS, $dom/wdeg++s$, $dom/wdeg$, $wdeg^{ca,cd}$ or ABS for each considered family.

Family	# instances	CHS		$dom/wdeg++s$		$dom/wdeg$		$wdeg^{ca,cd}$		ABS	
		#solv.	time	#solv.	time	#solv.	time	#solv.	time	#solv.	time
BinPacking	C	0	0	0	0	0	0	0	0	0	0
	T	15	24,505	0	27,003	0	27,002	3	21,687	0	27,001
ChessboardColoration	C	3	17.17	6.17	6.81	3	5,40681	3	24.26	3	10.82
	T	6	5,417	5,406	5,40681	3	5,40681	3	5,424	3	5,410
Cutstock	C	1	0.78	37.29	2.45	3	21,886	4	4.70	7	2.85
	T	15	6,013	21,886	21,443	3	21,443	4	15,838	3	21,612
Fastfood	C	5	776	983	1,319	5	18,983	5	856	7	568
	T	15	17,091	18,983	19,319	5	19,319	7	15,845	12	12,093
GolombRuler	C	3	62.12	121	97.94	3	14,533	5	170	6	52.13
	T	11	9,503	14,533	13,852	5	13,852	6	11,735	6	9,543
GraphColoring	C	5	1,043	101	104	6	16,546	6	712	5	53.41
	T	15	17,457	16,546	16,586	6	16,586	6	17,404	5	18,053
Knapsack	C	6	2,624	2,640	2,571	6	18,840	6	2,555	6	72.41
	T	15	18,824	18,840	18,771	6	18,771	6	18,755	15	338
LowAutocorrelation	C	4	644	417	489	4	14,817	4	448	4	98.20
	T	12	15,044	14,817	14,889	4	14,889	4	14,848	4	14,498
NurseRostering	C	1	387	221	1,439	1	2,021	1	38.41	1	132
	T	2	2,187	2,021	3,238	1	3,238	1	1,838	1	1,932
Opd	C	0	0	0	0	2	23,870	3	0	2	0
	T	15	25,200	23,870	21,910	3	21,910	2	23,483	1	25,200
OpenStacks	C	4	234	954	706	8	16,107	8	71	4	1,479
	T	15	5,666	16,107	17,029	8	17,029	15	1,106	4	21,279
Pb	C	8	508	470	472	9	12,284	10	456	10	256
	T	15	12,644	12,284	11,378	10	11,378	10	11,313	10	10,505
PeacableArmies	C	2	171	102	98.55	2	102	2	160	2	158
	T	2	171	102	98.56	2	102	2	160	2	158
PizzaVoucher	C	0	0	0	0	0	0	0	0	0	0
	T	1	1,800	1,800	1,800	0	1,800	1	1,747	0	1,800
PrizeCollecting	C	3	7.30	4.76	4.35	15	319	15	736	3	12.38
	T	15	472	319	320	15	319	3	22,336	15	1,848

Table 16: Detailed results (number of solved instances and runtime) with CHS, $dom/wdeg+s$, $dom/wdeg$, $wdeg^{ca,cd}$ or ABS for each considered family (Table 15 continued).

Family	# instances	CHS		$dom/wdeg+s$		$dom/wdeg$		$wdeg^{ca,cd}$		ABS	
		#solv.	time	#solv.	time	#solv.	time	#solv.	time	#solv.	time
QuadraticAssignment	C	5	339		424		453		825		948
	T	9	4,197	7	4,458	7	4,470	7	4,555	5	8,147
QueenAttacking	C	0	0		0		0		0		0
	T	1	1,455	0	1,800	0	1,800	0	1,800	0	1,800
Ramsey	C	3	1.06		0.92		3.38		0.50		27.91
	T	4	1,801	3	1,801	3	1,803	3	1,801	3	1,827
Rifap	C	1	3.76		2.66		3.12		3.14		0.28
	T	4	3,605	2	3,607	2	3,610	1	5,403	2	3,625
StillLife	C	7	448		217		683		247		1,023
	T	15	11,485	12	9,275	13	10,281	12	8,778	7	15,423
Taillard	C	0	0		0		0		0		0
	T	15	12,815	9	10,815	9	10,813	0	27,001	9	13,347
Tal	C	1	258		434		468		233		1,627
	T	2	659	1	2,234	2	1,854	2	587	1	3,427
TravellingSalesman	C	7	1,369		2,201		2,003		3,226		1,053
	T	15	15,769	7	16,601	7	16,403	7	17,626	7	15,453
Vrp	C	1	747		418		374		619		480
	T	3	4,348	1	4,018	1	3,974	1	4,219	1	4,080
Warehouse	C	2	997		901		1,082		1,027		2,349
	T	2	997	2	901	2	1,082	2	1,027	2	2,349
All	C	72	2.96 h		2.96 h		3.44 h		3.45 h		2.89 h
	T	264	127	102	75.45 h	109	75.20 h	105	72.70 h	114	69.97 h

- [5] Bessière, C., Régin, J.C.: MAC and Combined Heuristics: Two Reasons to Forsake FC (and CBJ?) on Hard Problems. In: Proceedings of CP, pp. 61–75 (1996)
- [6] Boussemart, F., Hemery, F., Lecoutre, C., Saïs, L.: Boosting systematic search by weighting constraints. In: Proceedings of ECAI, pp. 146–150 (2004)
- [7] Brélaz, D.: New Methods to Color Vertices of a Graph. *Communications of the ACM* **22**(4), 251–256 (1979)
- [8] Cabon, C., de Givry, S., Lobjois, L., Schiex, T., Warners, J.P.: Radio Link Frequency Assignment. *Constraints* **4**, 79–89 (1999)
- [9] Chu, G., Stuckey, P.J.: Learning value heuristics for constraint programming. In: *Integration of AI and OR Techniques in Constraint Programming*, pp. 108–123. Springer International Publishing, Cham (2015)
- [10] Eén, N., Sörensson, N.: An Extensible SAT-solver. In: Proceedings of SAT, pp. 502–518 (2003)
- [11] Gay, S., Hartert, R., Lecoutre, C., Schaus, P.: Conflict Ordering Search for Scheduling Problems. In: G. Pesant (ed.) *Proceedings of CP*, pp. 140–148 (2015)
- [12] Geelen, P.A.: Dual Viewpoint Heuristics for Binary Constraint Satisfaction Problems. In: Proceedings of ECAI, pp. 31–35 (1992)
- [13] Golomb, S.W., Baumert, L.D.: Backtrack programming. *Journal of the ACM* **12**, 516–524 (1965)
- [14] Gomes, C.P., Selman, B., Crato, N., Kautz, H.A.: Heavy-Tailed Phenomena in Satisfiability and Constraint Satisfaction Problems. *Journal of Automated Reasoning* **24**(1/2), 67–100 (2000)
- [15] Habet, D., Jégou, P., Kanso, H., Terrioux, C.: BTD₁₂ and miniBTDD₁₂. In: Proceedings of the XCSP3 Competition, pp. 68–69 (2018)
- [16] Habet, D., Terrioux, C.: Conflict history based search for constraint satisfaction problem. In: *Proceeding of SAC, Knowledge Representation and Reasoning Technical Track*, pp. 1117–1122 (2019)
- [17] Haralick, R.M., Elliot, G.L.: Increasing tree search efficiency for constraint satisfaction problems. *AIJ* **14**, 263–313 (1980)
- [18] Hebrard, E., Siala, M.: Explanation-Based Weighted Degree. In: Proceedings of CPAIOR, pp. 167–175 (2017)
- [19] Holland, A., O’Sullivan, B.: Weighted Super Solutions for Constraint Programs. In: Proceedings of AAI, pp. 378–383 (2005)
- [20] Hooker, J.N.: Testing Heuristics: We Have It All Wrong. *Journal of Heuristics* **1**(1), 33–42 (1995)
- [21] Jégou, P., Kanso, H., Terrioux, C.: Towards a Dynamic Decomposition of CSPs with Separators of Bounded Size. In: Proceedings of CP, pp. 298–315 (2016)
- [22] Jégou, P., Kanso, H., Terrioux, C.: BTDD and miniBTDD. In: XCSP3 Competition (2017)
- [23] Jégou, P., Kanso, H., Terrioux, C.: BTDD and miniBTDD. In: Proceedings of the XCSP3 Competition, pp. 66–67 (2018)
- [24] Jégou, P., Terrioux, C.: Hybrid backtracking bounded by tree-decomposition of constraint networks. *AIJ* **146**, 43–75 (2003)
- [25] Lecoutre, C., Sais, L., Tabary, S., Vidal, V.: Last Conflict Based Reasoning. In: Proceedings of ECAI, pp. 133–137 (2006)
- [26] Lecoutre, C., Sais, L., Tabary, S., Vidal, V.: Nogood recording from restarts. In: Proceedings of IJCAI, pp. 131–136 (2007)
- [27] Lecoutre, C., Sais, L., Tabary, S., Vidal, V.: Recording and Minimizing Nogoods from Restarts. *JSAT* **1**(3-4), 147–167 (2007)
- [28] Liang, J.H., Ganesh, V., Poupart, P., Czarnecki, K.: Exponential Recency Weighted Average Branching Heuristic for SAT Solvers. In: Proceedings of AAI, pp. 3434–3440 (2016)
- [29] Liang, J.H., Ganesh, V., Poupart, P., Czarnecki, K.: Learning Rate Based Branching Heuristic for SAT Solvers. In: Proceedings of SAT, pp. 123–140 (2016)
- [30] Liberatore, P.: On the complexity of choosing the branching literal in DPLL. *Artificial Intelligence* **116**(1-2) (2000)
- [31] Marques-Silva, J., Sakallah, K.A.: GRASP: A Search Algorithm for Propositional Satisfiability. *IEEE Transactions on Computers* **48**(5), 506–521 (1999)
- [32] Michel, L., Hentenryck, P.V.: Activity-based search for black-box constraint programming solvers. In: Proceedings of CPAIOR, pp. 228–243 (2012)

- [33] Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an Efficient SAT Solver. In: Proceedings of DAC, pp. 530–535 (2001)
- [34] Nadel, B.: Tree Search and Arc Consistency in Constraint-Satisfaction Algorithms, pp. 287–342. In *Search in Artificial Intelligence*. Springer-Verlag (1988)
- [35] Refalo, P.: Impact-based search strategies for constraint programming. In: Proceedings of CP, pp. 557–571 (2004)
- [36] Robertson, N., Seymour, P.D.: Graph minors II: Algorithmic aspects of treewidth. *Algorithms* **7**, 309–322 (1986)
- [37] Rossi, F., van Beek, P., Walsh, T.: Handbook of Constraint Programming, *Foundations of Artificial Intelligence*, vol. 2. Elsevier (2006)
- [38] Sabin, D., Freuder, E.C.: Contradicting Conventional Wisdom in Constraint Satisfaction. In: Proceedings of ECAI, pp. 125–129 (1994)
- [39] Schulte, C.: Programming branchers. In: C. Schulte, G. Tack, M.Z. Lagerkvist (eds.) *Modeling and Programming with Gecode* (2018). Corresponds to Gecode 6.0.1
- [40] Simonin, G., Artigues, C., Hebrard, E., Lopez, P.: Scheduling Scientific Experiments for Comet Exploration. *Constraints* **20(1)**, 77–99 (2015)
- [41] Sutton, R.S., Barto, A.G.: *Reinforcement Learning: An Introduction*, 1st edn. MIT Press, Cambridge, MA, USA (1998)
- [42] Watez, H., Koriche, F., Lecoutre, C., Paparrizou, A., Tabary, S.: Learning Variable Ordering Heuristics with Multi-Armed Bandits and Restarts. In: Proceedings of ECAI (2020)
- [43] Watez, H., Lecoutre, C., Paparrizou, A., Tabary, S.: Refining Constraint Weighting. In: Proceedings of ICTAI, pp. 71–77 (2019)
- [44] Xia, W., Yap, R.H.C.: Learning robust search strategies using a bandit-based approach. In: Proceedings of AAAI, pp. 6657–6665 (2018)