Journal of Heuristics (2021) 27:951–990
https://doi.org/10.1007/s10732-021-09475-z

Conflict history based heuristic for constraint satisfaction problem solving

Djamal Habet[1] **· Cyril Terrioux**[1]

Received: 16 July 2020 / Revised: 11 February 2021 / Accepted: 4 May 2021 / Published online: 30 June 2021

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 BTD (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.

Keywords CSP solving · Variable ordering heuristic · Conflict history · Exponential recency weighted average

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. Cabon et al. 1999; Holland and

This paper is an extension of the work published in Habet and Terrioux (2019).

✉ Djamal Habet
djamal.habet@lis-lab.fr

Cyril Terrioux
cyril.terrioux@lis-lab.fr

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

O'Sullivan 2005; Rossi et al. 2006; Simonin et al. 2015). 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 (Rossi et al. 2006). 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 (Liberatore 2000).

Many heuristics have been proposed (e.g. Bessière et al. 2001; Bessière and Régin 1996; Boussemart et al. 2004; Brélaz 1979; Geelen 1992; Golomb and Baumert 1965; Hebrard and Siala 2017; Michel and Hentenryck 2012; Refalo 2004) aiming mainly to satisfy the *first-fail principle* (Haralick and Elliot 1980) which advises "to succeed, try first where you are likely to fail". Nowadays, the most efficient heuristics are adaptive and dynamic (Boussemart et al. 2004; Geelen 1992; Hebrard and Siala 2017; Michel and Hentenryck 2012; Refalo 2004), 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 (Michel and Hentenryck 2012; Refalo 2004). $dom/wdeg$ is one of the simplest, the most used and efficient variable ordering heuristic (Boussemart et al. 2004). 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 (Lecoutre et al. 2006) and COS (Gay et al. 2015), 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 (Sutton and Barto 1998). It was also recently used in defining powerful branching heuristics for solving the satisfiability problem (SAT) (Liang et al. 2016a, b). We have integrated CHS in solvers based on MAC (Maintaining Arc Consistency) (Sabin and Freuder 1994) and BTD (Backtracking with Tree-Decomposition) (Jégou and Terrioux 2003). The empirical evaluation on XCSP3 instances[1] 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).

[1] http://www.xcsp.org.

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 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 Sect. 5 and on COP ones in Sect. 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: $X = \{x_1, \ldots, x_n\}$ is a set of n variables, $D = \{D_1, \ldots, D_n\}$ is a set of finite domains, and $C = \{c_1, \ldots, c_e\}$ 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 $S(c_j) = \{x_{j_1}, \ldots, x_{j_k}\} \subseteq X$ and $R(c_j) \subseteq D_{j_1} \times \cdots \times D_{j_k}$. 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 (Rossi et al. 2006).

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 $O(e.d^n)$ where d denotes the size of the largest domain. In order to ensure an efficient solving, they commonly exploit jointly several techniques (see Rossi et al. 2006 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 Sect. 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 Sabin and Freuder 1994) or RFL (Real Full Look-ahead Nadel 1988) 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 $\Sigma = \langle \delta_1, \ldots, \delta_i \rangle$ (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 Lecoutre et al. 2007). 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 (Gomes et al. 2000). For efficiency reasons, they are usually exploited with some learning techniques like recording of nld-nogoods in Lecoutre et al. (2007). 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 $y = (y_1, y_2, \ldots, y_m)$, 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) (Sutton and Barto 1998) 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 (Sutton and Barto 1998). 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 (Sutton and Barto 1998):

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

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 (Sutton and Barto 1998). 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* (Haralick and Elliot 1980).

CHS applies the decreasing policy of α, which is successfully used for designing efficient branching heuristic for the satisfiability problem (Liang et al. 2016a, b). 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:

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

$Uvars(Y)$ 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 Eq. 1 to calculate chv as given below, where δ is a positive real number close to 0.

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

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 Eq. 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.

3.2 CHS and restarts

Restart techniques are known to be important for the efficiency of solving algorithms (see for example Lecoutre et al. 2007). 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.

Algorithm 1: CHS

Input: an event e

1 **switch** e **do**
2 **case** $initialization$
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** $conflict\ when\ filtering\ with\ c_j$
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** $select\ a\ new\ variable$
16 **return** a variable x s.t. $x \in \arg\min_{x_i \in Uvars(X)} \frac{\sum_{c_j \in C:\ x_i \in S(c_j) \wedge |Uvars(S(c_j))| > 1} (q(c_j) + \delta)}{|D_i|}$
17 **case** $restart$
18 $\alpha \leftarrow \alpha_0$
19 **for** $c_j \in C$ **do**
20 $q(c_j) \leftarrow q(c_j) \times 0.995^{Conflicts - Conflict(c_j)}$

3.3 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 (Lecoutre et al. 2006), 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 (Liang et al. 2016a, b).

4.1 Impact-based search (IBS)

This heuristic selects the variable which leads to the largest search space reduction (Refalo 2004). 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 kth 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 $dom/wdeg$ (Boussemart et al. 2004). 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:

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

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 Hebrard and Siala (2017), but are not widely used in practice. Very recently, a refined version of $wdeg$ (called $wdeg^{ca.cd}$) has been defined in Wattez et al. (2019). 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 (Michel and Hentenryck 2012). 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 (Schulte 2018). It follows the same steps of the first definition of CHB in the context of the satisfiability problem (Liang et al. 2016a, b). 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 (Lecoutre et al. 2006) 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) (Gay et al. 2015) 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) (Eén and Sörensson 2003; Marques-Silva and Sakallah 1999;

Moskewicz et al. 2001) 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 (Moskewicz et al. 2001) 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) (Liang et al. 2016a), 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 (Liang et al. 2016b) 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 (Balafrej et al. 2015). A linear regression method is used to learn the scoring function of value heuristics (Chu and Stuckey 2015). Rewards are calculated and used to select adaptively the backtracking strategy (Bachiri et al. 2015). Learning process based on Least Squares Policy Iteration technique is used to tune adaptively the parameters of stochastic local search algorithms (Battiti and Campigotto 2012).

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 (Xia and Yap 2018). 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, Wattez et al. have proposed another MAB approach (Wattez et al. 2020). Like in the work of Xia and Yap, each heuristic corresponds to an arm. In contrast, an 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 Wattez et al. (2020).

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 Sect. 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 Sect. 5.3, before studying the behavior of CHS when it is used jointly with LC or COS in Sect. 5.4. Finally, in Sect. 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 repository[2] and the XCSP3 competition 2018,[3] 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 1233 unsupported instances. We also remove 2813 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 (Lecoutre et al. 2007) before assessing the impact of our approach on a structural solving method, namely BTD-MAC+RST+Merge (Jégou et al. 2016). Roughly speaking, BTD-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 Robertson and Seymour 1986). While the search performed by MAC considers at each step all the remaining variables, one performed by

2 http://www.xcsp.org/series.

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

BTD-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 Jégou et al. 2016 for more details). For BTD-MAC+RST+Merge, the tree-decompositions are computed with the heuristic H_5-TD-WT (Jégou et al. 2016). 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 Jégou et al. (2016). Note that these settings except the variable ordering heuristic correspond to those used for the XCSP3 competitions 2017 and 2018 (Habet et al. 2018; Jégou et al. 2017, 2018).

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 Michel and Hentenryck (2012). Note that we do not exploit a probing step like one mentioned in Michel and Hentenryck (2012). So all the weights are initially set to 0. For CHB, we use the value parameters as given in Schulte (2018). 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 Liberatore 2000). 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 Hooker (1995).

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

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	**6530**	**4212**	**10742**	**1038.89**
0.2	6505	4206	1711	1049.55
0.3	6505	4203	10708	1052.04
0.4	6493	4204	10697	1056.14
0.5	6509	4202	10711	1058.13
0.6	6487	4205	10692	1062.14
0.7	6504	4207	10711	1055.46
0.8	6479	4197	10676	1072.28
0.9	6473	4203	10676	1071.43
VBS	6691	4242	10933	940.21

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 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 1072 and 1071 h. 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 6530 instances (resp. 4212 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

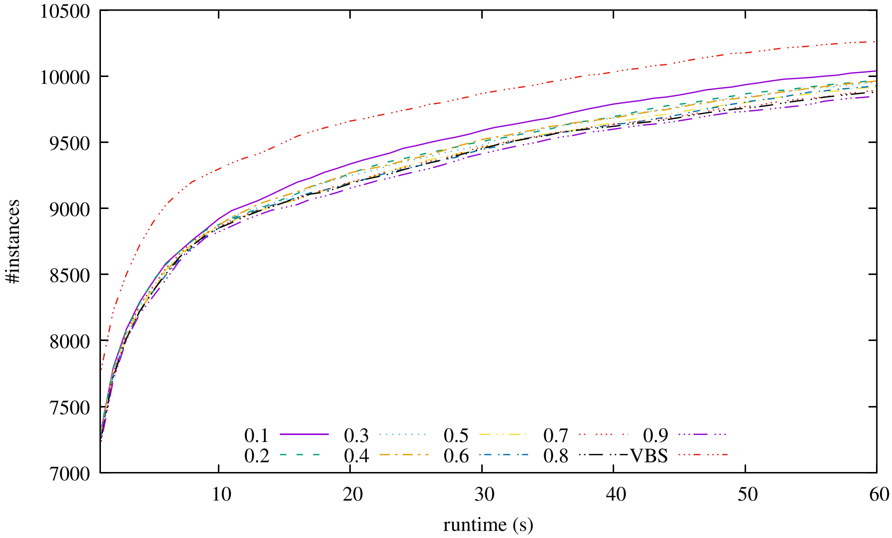

Fig. 1 Number of solved instances as a function of the elapsed time for α_0 varying between 0.1 and 0.9 and the VBS, for a runtime between 1 and 60 s

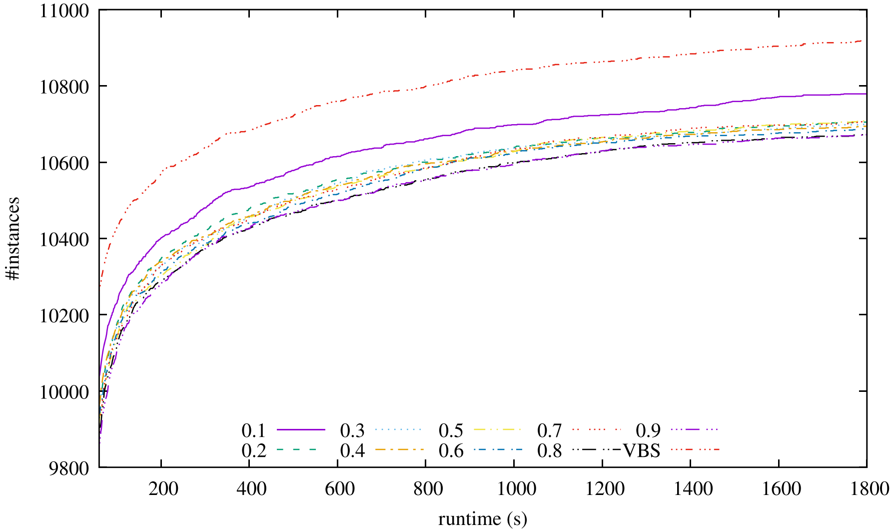

Fig. 2 Number of solved instances as a function of the elapsed time for α_0 varying between 0.1 and 0.9 and the VBS, a for runtime between 60 and 1800 s

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.

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 9069 instances are solved. At the same time, the

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	6507	4202	10709	1061.07
0.05	6512	4212	10724	1058.89
0.075	6500	4204	10704	1064.61
0.1	**6530**	**4212**	**10742**	**1038.89**
0.125	6519	4203	10722	1078.10
0.15	6503	4207	10710	1061.81

runtime is almost doubled with a cumulative runtime of 1921.35 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 (1089.37 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 $\alpha_0 = 0.1$ 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 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

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	6479	4204	10683	1079.25
10^{-5}	6519	4207	10726	1043.53
$\mathbf{10^{-4}}$	**6530**	**4212**	**10742**	**1038.89**
10^{-3}	6508	4199	10707	1044.41

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 $(+s+r)$	**6530**	**4212**	**10742**	**1038.89**
MAC+CHS+s-r	6520	4209	10729	1043.95
MAC+CHS-s-r	6484	4199	10683	1064.20
MAC+CHS-s+r	6482	4176	10658	1067.72

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.

Finally, these results are globally consistent with those presented in Habet and Terrioux (2019). Indeed, except that the best value of α_0 is now 0.1 instead of 0.4 in Habet and Terrioux (2019), we observe the same trends. The benchmark used in Habet and Terrioux (2019) was a subset of our initial benchmark. If we proceed similarly by removing arc-inconsistent instances, we obtain a benchmark with 7916 instances. From this benchmark, MAC solved respectively 6700 and 6706 instances with 0.1 and 0.4 for α_0 in Habet and Terrioux (2019), while in the current experiments, it succeeds in solving 6837 and 6829 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 versus 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, 6, 7 and 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 on rows C the number of instances solved by MAC with each considered heuristic (excluding

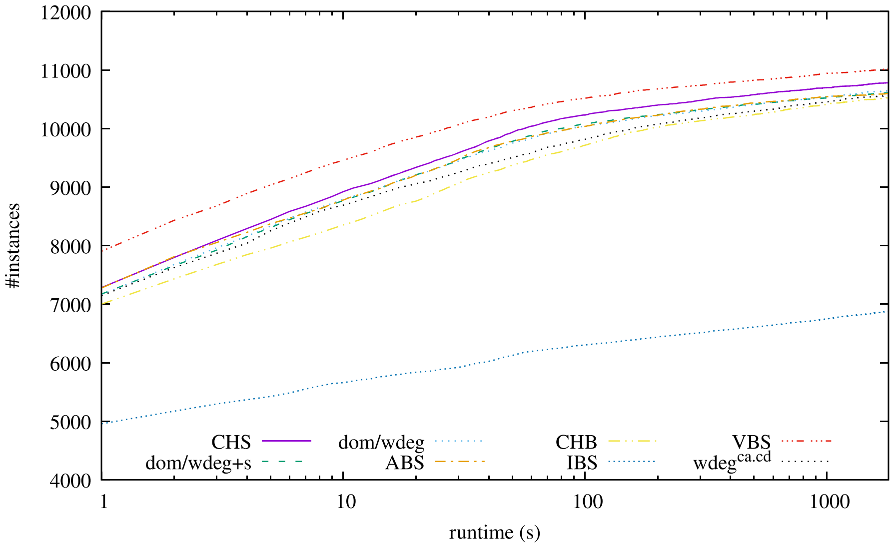

Fig. 3 Number of solved instances as a function of the elapsed time (with a logarithmic scale) for the considered heuristics (namely CHS, $dom/wdeg{+}s$, $dom/wdeg$, $wdeg^{ca.cd}$, ABS, CHB and IBS) and the VBS based on these seven heuristics

IBS[4]) 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 s. 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 Fig. 3 and Tables 5, 6, 7 and 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

[4] Given 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, $dom/wdeg$+s, $dom/wdeg$, $wdeg^{ca.cd}$, ABS or CHB for each considered family (Part 1 for Table 5)

	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	AllInterval	C	22		**0.20**		9.69		89.26		1080		0.20		0.20
		T	32	**32**	**3.95**	32	2810	24	15907	23	17284	32	4.08	32	4.07
	Basic	C	4		0.02		0.01		0.01		0.01		**0.01**		0.03
		T	4	4	0.02	4	0.01	4	0.01	4	0.01	**4**	**0.01**	4	0.03
	Bibd	C	73		574		5084		5625		8301		**170**		4686
		T	312	**152**	**141154**	98	246462	114	225675	110	222637	119	194960	107	226228
	Blackhole	C	84		1379		1798		416		10.03		1586		**7.17**
		T	112	84	51779	85	50398	85	49016	85	48610	85	50189	**85**	**48607**
	CarSequencing	C	3		10.29		62.82		485		291		481		**0.38**
		T	52	10	75804	9	77613	8	82166	**19**	**63172**	16	70836	5	84601
	ColouredQueens	C	5		0.89		0.90		0.71		0.39		0.74		**0.27**
		T	17	5	21601	5	21601	5	21601	5	21600	5	21601	**5**	**21600**
	CostasArray	C	8		79.16		773		468		62.99		428		**274**
		T	11	8	5479	8	6173	9	5503	9	4477	9	5311	**9**	**4196**
	CryptoPuzzle	C	10		0.01		0.01		0.01		**0.01**		0.01		0.01
		T	10	10	0.01	10	0.01	10	0.01	**10**	**0.01**	10	0.01	10	0.01

Table 5 Part 2 for Table 5

	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	DeBruijnSequence	C	12		534		**488**		529		510		530		528
		T	18	12	7765	12	11377	12	7775	12	11401	12	9570	**12**	**7752**
	DiamondFree	C	31		524		2274		1789		3127		13.31		**10.89**
		T	38	36	5009	33	12116	33	11537	36	9206	38	21.97	**38**	**19.23**
	Domino	C	37		256		256		**243**		278		261		261
		T	37	37	256	37	256	**37**	**243**	37	278	37	261	37	261
	Driver	C	7		23.10		33.15		16.75		30.15		50.89		**11.89**
		T	7	7	23.10	7	33.15	7	16.75	7	30.15	7	50.89	**7**	**11.89**
	Dubois	C	10		948		964		1386		1158		**237**		1271
		T	30	12	36080	10	36964	11	36992	11	36534	**16**	**27440**	11	36863
	GracefulGraph	C	12		**0.04**		2.07		33.81		5.86		0.69		1.14
		T	104	**19**	**153785**	17	157580	16	160081	14	162902	16	158843	14	162121
	Hanoi	C	6		3.36		3.72		4.39		**2.33**		4.19		3.56
		T	7	6	3.36	6	3.72	6	4.39	**6**	**2.33**	6	4.19	6	3.56
	Haystacks	C	2		4.20		1.57		3.16		**0.04**		2.47		0.51
		T	51	2	88204	2	88202	2	88203	**2**	**88200**	2	88203	2	88201

Table 6 Part 3 for Table 5

	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	Kakuro	C	1084		96.75		**68.80**		205		805		1554		6110
		T	1102	1100	3896	**1101**	**2623**	1101	4275	1096	14150	1088	27207	1086	36617
	Knights	C	12		**293**		461		438		525		1019		444
		T	19	**13**	**11836**	13	12334	13	12295	13	12477	12	13625	13	12074
	KnightTour	C	5		186		198		185		**18.42**		2.26		387
		T	25	7	32527	6	32753	6	32724	12	26651	9	28114	**14**	**22352**
	Langford	C	46		1614		1387		1312		1378		**512**		2657
		T	125	49	141969	49	141486	49	141478	49	138236	**67**	**108978**	58	124732
	LatinSquare	C	266		660		875		**413**		1250		604		1628
		T	366	**281**	**160383**	276	166675	275	166858	274	172571	271	174580	276	166455
	MagicSequence	C	83		536		473		473		1170		**443**		982
		T	85	83	536	83	473	83	473	83	1170	**83**	**443**	83	982
	MagicSquare	C	18		382		836		**350**		3271		434		2640
		T	86	44	70825	**54**	**59015**	42	76997	19	100476	58	49814	43	88153
	MarketSplit	C	10		266		261		261		254		**103**		280
		T	10	10	266	10	261	10	261	10	254	**10**	**103**	10	280

Table 6 Part 4 for Table 5

	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	MaxCSP	C	2186		126		75.89		**55.43**		238		55.89		156
		T	2186	2186	126	2186	75.89	**2186**	**55.43**	2186	238	2186	55.89	2186	156
	Mixed	C	11		**213**		472		384		264		442		296
		T	18	**17**	**4258**	14	6601	17	5760	15	5458	14	7674	16	4160
	MultiKnapsack	C	28		11.87		13.97		38.68		36.92		**2.59**		227
		T	31	31	211	31	332	31	605	31	592	**31**	**12.42**	28	5627
	Nonogram	C	351		162		77.09		247		597		**45.90**		280
		T	356	354	3781	**355**	**2956**	354	4631	355	2998	354	5005	354	3903
	NumberPartitioning	C	12		61.07		1651		209		114		8.59		**4.37**
		T	50	20	58781	13	68253	14	65070	16	63436	27	43591	**28**	**43068**
	Ortholatin	C	4		3.69		**1.61**		3.52		357		16.43		616
		T	27	6	37816	4	41406	4	41407	**7**	**36368**	4	41418	4	42021
	PigeonsPlus	C	29		85.21		89.18		**81.29**		143		2745		89.45
		T	38	37	4179	37	4065	**37**	**4044**	36	5914	29	18945	37	4261
	Primes	C	134		838		240		139		1014		**64.35**		527
		T	160	141	35478	143	32080	**145**	**29488**	143	37623	143	31633	137	43784

Table 7 Part 5 for Table 5

	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	PseudoBoolean	C	109		1524		1793		1927		**1195**		1892		2967
		T	337	130	361038	129	385566	114	390684	121	399516	**137**	**347537**	135	354105
	QRandom	C	607		**6180**		7697		7331		13907		8259		21394
		T	614	614	10287	613	11569	**614**	**9706**	613	18716	612	14170	609	33109
	QuasiGroups	C	24		482		528		442		499		**156**		622
		T	148	**26**	**203502**	26	222532	25	204709	24	223707	24	205359	24	205826
	QueenAttacking	C	4		3.78		22.46		**0.71**		9.50		172.81		7.17
		T	10	**5**	**9021**	5	9171	5	9082	4	10810	4	10973	4	10807
	Queens	C	18		497		85.04		65.89		2081		**34.66**		208
		T	24	20	4805	21	2353	21	2362	18	9283	**22**	**2355**	20	4633
	QueensKnights	C	10		**55.19**		86.39		83.74		88.47		175.85		101
		T	18	**16**	**5939**	15	6766	15	6949	15	6830	10	14576	16	6397
	RadarSurveillance	C	81		2.54		2.30		2.35		**2.19**		3.69		1006
		T	90	90	3.58	90	3.90	90	3.64	**90**	**2.89**	90	5.53	81	17206
	Random	C	1591		65626		71650		**59499**		126920		75109		194100
		T	1955	1699	605428	1678	635177	**1703**	**585527**	1664	705018	1674	633027	1623	819665

Table 7 Part 6 for Table 5

Family			# instances	CHS #solv.	CHS Time	$dom/wdeg{+}s$ #solv.	$dom/wdeg{+}s$ Time	$dom/wdeg$ #solv.	$dom/wdeg$ Time	$wdeg^{ca.cd}$ #solv.	$wdeg^{ca.cd}$ Time	ABS #solv.	ABS Time	CHB #solv.	CHB Time
Academic	RoomMate	C	17		289		398		363		309		276		**257**
		T	30	17	289	17	398	17	363	17	309	17	276	**17**	**257**
	Sat	C	356		6620		3915		5071		**2867**		3467		7149
		T	366	361	18203	361	13996	361	15163	361	12897	**361**	**12822**	357	17004
	Scheduling	C	80		1076		1230		1147		543		**29.27**		67.29
		T	107	90	33332	85	40923	86	41489	92	29932	88	34230	**92**	**28486**
	SchurrLemma	C	7		374		409		**286**		522		308		564
		T	10	8	3974	8	4009	**9**	**2700**	8	4123	8	3908	7	5964
	Steiner3	C	4		4.45		0.07		**0.06**		0.08		6.62		0.96
		T	10	4	7204.45	5	7213.70	5	7212.11	**5**	**5516.92**	4	7206.62	4	7200.96
	Subisomorphism	C	1616		21942		**19362**		20467		29582		28134		43863
		T	1878	1704	264950	1699	275951	**1707**	**264534**	1681	307266	1692	281407	1676	321878
	Sudoku	C	92		**0.22**		0.19		0.20		0.37		0.25		0.23
		T	92	**92**	**0.22**	92	0.19	92	0.20	92	0.37	92	0.25	92	0.23

Table 8 Part 7 for Table 5

	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	SuperSolutions	C	80		324		**76.95**		1436		581		2024		677
		T	330	110	404455	92	428717	**119**	**391364**	117	396668	89	439039	107	411775
	TravellingSalesman	C	45		456		269		395		1347		**247**		728
		T	45	45	456	45	269	45	395	45	1347	**45**	**247**	45	728
Real-world	Renault	C	14		**0.03**		0.05		0.05		0.03		0.04		0.03
		T	14	**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	**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		1087.50
		T	60	58	4301	57	7553	59	3608	**59**	**4294**	56	7507	57	8095
	SocialGolfers	C	322		2632		2791		2780		1265		**1036**		3650
		T	460	335	231754	337	232076	338	228481	**346**	**210490**	336	228857	343	217181
	SportsScheduling	C	3		0.12		0.04		0.12		0.05		**0.02**		0.03
		T	19	5	25305	4	27003	4	27005	**6**	**25160**	5	26659	3	28800
	Wwtpp	C	234		**459**		721		12811		17854		23687		2251
		T	371	**316**	**111155**	312	114889	300	166108	281	186078	266	214019	288	158115
Total	Academic	C	9346		32.03 h		35.13 h		**31.78** h		57.43 h		36.70 h		82.80 h
		T	11590	**9846**	**857.42** h	9731	924.33 h	9778	895.38 h	9702	954.69 h	9769	884.90 h	966	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	10107		**33.94** h		37.36 h		37.65 h		63.58 h		44.29 h		87.32 h
		T	12829	**10742**	**1038.89 h**	10615	1112.85 h	10650	1096.54 h	10568	1155.69 h	10608	1097.14 h	10529	1176.29 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

the domain sizes Refalo 2004). 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, 6, 7 and 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.

Finally, our observations are consistent with ones in Habet and Terrioux (2019). 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

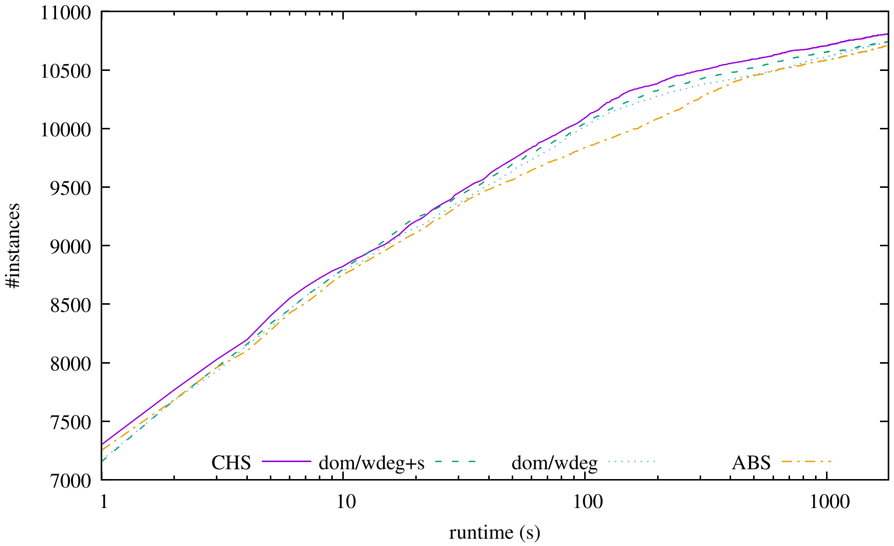

Fig. 4 Number of solved instances as a function of the elapsed time (with a logarithmic scale) for LC with the heuristics CHS, $dom/wdeg{+}s$, $dom/wdeg$ or ABS

heuristics according to the results of the previous subsection, namely $dom/wdeg{+}s$, $dom/wdeg$ and 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 1017.03 hours against 10,742 instances solved in 1038.89 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 Fig. 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

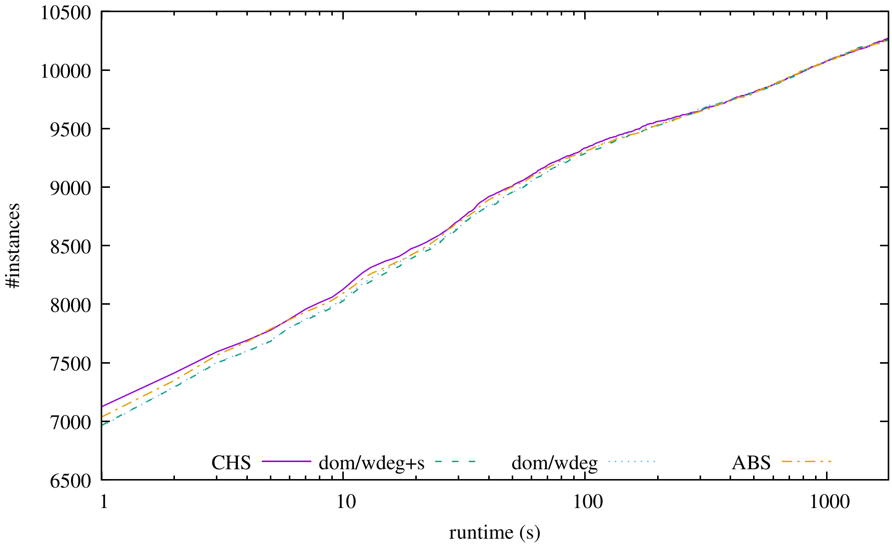

Fig. 5 Number of solved instances as a function of the elapsed time (with a logarithmic scale) for COS with the heuristics CHS, $dom/wdeg{+}s$, $dom/wdeg$ or ABS

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 #solved	LC Time (h)	COS #solved	COS Time (h)
CHS	**10812**	**1017.03**	**10281**	**1363.86**
$dom/wdeg{+}s$	10752	1057.91	10265	1368.66
$dom/wdeg$	10741	1067.28	10259	1367.17
ABS	10718	1090.23	10262	1368.99

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.

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 (Jégou et al. 2016) and the heuristics CHS, $dom/wdeg{+}s$, $dom/wdeg$ and ABS combined or not with LC. As shown in Fig. 6 and Table 11, the trends observed for MAC are still valid for BTD-MAC+RST+Merge.

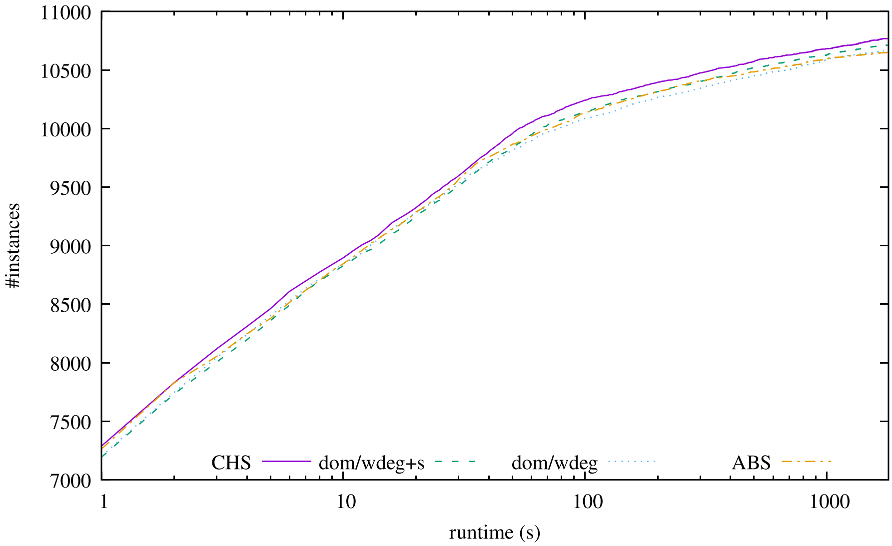

Fig. 6 Number of instances solved by BTD-MAC+RST+Merge as a function of the elapsed time (with a logarithmic scale) with the heuristics CHS, $dom/wdeg{+}s$, $dom/wdeg$ or ABS

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 #solved	Time (h)	With LC #solved	Time (h)
CHS	**10770**	**1035.59**	**10839**	**1011.22**
$dom/wdeg{+}s$	10712	1065.01	10805	1032.30
$dom/wdeg$	10672	1089.00	10767	1061.63
ABS	10650	1082.71	10705	1093.49

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 1035 h) against 10,712 instances (in 1065 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 Fig. 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 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 Habet and Terrioux (2019). They are also consistent with ones of the XCSP3 competition 2018. For instance, BTD-MAC+RST+Merge participated in the mini-solvers track of the competition by using respectively $dom/wdeg$ (for the solver miniBTD Jégou et al. 2018) and CHS (for

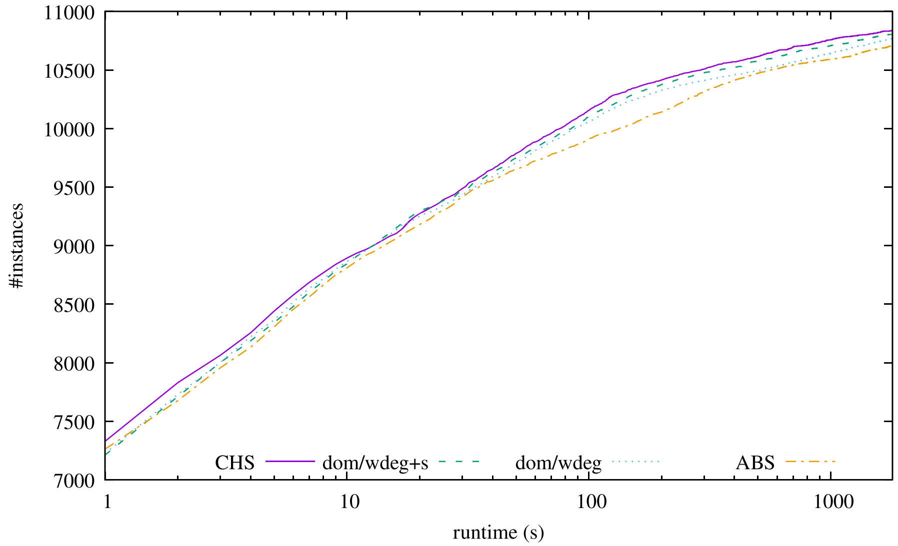

Fig. 7 Number of instances solved by BTD-MAC+RST+Merge as a function of the elapsed time (with a logarithmic scale) with LC combined with the heuristics CHS, $dom/wdeg{+}s$, $dom/wdeg$ or ABS

the solver miniBTD_12 Habet et al. 2018) as variable ordering heuristic. miniBTD_12 finished in the second place by solving 79 instances while miniBTD 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 (Rossi et al. 2006).

We first describe the experimental protocol we use. Then, in Sect. 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 Sect. 6.3.

6.1 Experimental protocol

We consider the COP instances from the 2019 XCSP3 competition.[5] Like for CSP instances, we discard 36 instances containing some global constraints which are not

[5] http://www.cril.univ-artois.fr/XCSP19.

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

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 min), 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 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

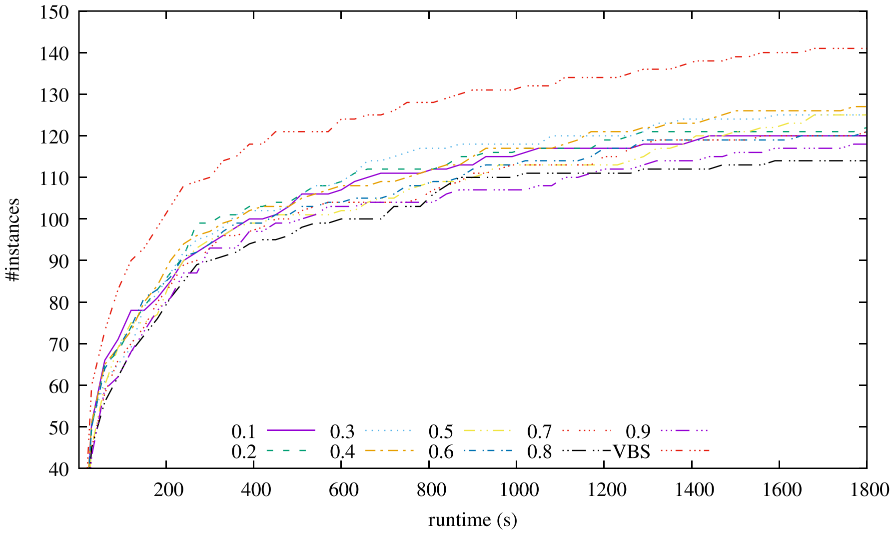

Fig. 8 Number of solved COP instances as a function of the elapsed time for α_0 varying between 0.1 and 0.9 and the VBS

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 Fig. 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 h. 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 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-

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

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.

6.3 CHS versus 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.

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 (Part 1 for Table 15)

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
	T	15	2	24505	0	27003	0	27002	**3**	**21687**	0	27001
ChessboardColoration	C	3		17.17		**6.17**		6.81		24.26		10.82
	T	6	3	5417	**3**	**5406**	3	540681	3	5424	3	5410
Cutstock	C	1		**0.78**		37.29		2.45		4.70		2.85
	T	15	**12**	**6013**	3	21886	4	21443	7	15838	3	21612
Fastfood	C	5		776		983		1319		856		**568**
	T	15	8	17091	5	18983	5	19319	7	15845	**12**	**12093**
GolombRuler	C	3		62.12		121		97.94		170		**52.13**
	T	11	**6**	**9503**	3	14533	5	13852	6	11735	6	9543
GraphColoring	C	5		1043		101		104		712		**53.41**
	T	15	6	17457	**6**	**16546**	6	16586		17404	5	18053
Knapsack	C	6		2624		2640		2571		2555		**72.41**
	T	15	6	18824	6	18840	6	18771	6	18755	**15**	**338**

Table 15 Part 2 for Table 15

Family	# instances		CHS		$dom/wdeg{+}s$		$dom/wdeg$		$wdeg^{ca.cd}$		ABS	
			#solv.	Time	#solv.	Time	#solv.	Time	#solv.	Time	#solv.	Time
LowAutocorrelation	C	4		644		417		489		448		**98.20**
	T	12	4	15044	4	14817	4	14889	4	14848	**4**	**14498**
NurseRostering	C	1		387		221		1439		**38.41**		132
	T	2	1	2187	1	2021	1	3238	**1**	**1838**	1	1932
Opd	C	0		0		0		0		0		0
	T	15	1	25200	2	23870	**3**	**21910**	2	23483	1	25200
OpenStacks	C	4		234		954		706		**71**		1479
	T	15	15	5666	8	16107	8	17029	**15**	**1106**	4	21279
Pb	C	8		508		470		472		456		**256**
	T	15	9	12644	9	12284	10	11378	10	11313	**10**	**10505**
PeacableArmies	C	2		171		102		**98.55**		160		158
	T	2	2	171	2	102	**2**	**98.56**	2	160	2	158
PizzaVoucher	C	0		0		0		0		0		0
	T	1	0	1800	0	1800	0	1800	**1**	**1747**	0	1800
PrizeCollecting	C	3		7.30		4.76		**4.35**		736		12.38
	T	15	15	472	15	319	**15**	**320**	3	22336	15	1848

Table 16 Part 3 for Table 15

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	**7**	**4197**	7	4458	7	4470	7	4555	5	8147
QueenAttacking	C	0		0		0		0		0		0
	T	1	**1**	**1455**	0	1800	0	1800	0	1800	0	1800
Ramsey	C	3		1.06		0.92		3.38		**0.50**		27.91
	T	4	3	1801	3	1801	3	1803	**3**	**1801**	3	1827
Rlfap	C	1		3.76		2.66		3.12		3.14		0.28
	T	4	**2**	**3605**	2	3607	2	3610	1	5403	2	3625
StillLife	C	7		448		**217**		683		247		1023
	T	15	12	11485	12	9275	**13**	**10281**	12	8778	7	15423
Taillard	C	0		0		0		0		0		0
	T	15	9	12815	9	10815	**9**	**10813**	0	27001	9	13347
Tal	C	1		258		434		468		**233**		1627
	T	2	2	659	1	2234	2	1854	**2**	**587**	1	3427
TravellingSalesman	C	7		1369		2201		2003		3226		**1053**
	T	15	7	15769	7	16601	7	16403	7	17626	**7**	**15453**

Table 16 Part 4 for Table 15

Family	# instances		CHS		$dom/wdeg$+s		$dom/wdeg$		$wdeg^{ca.cd}$		ABS	
			#solv.	Time	#solv.	Time	#solv.	Time	#solv.	Time	#solv.	Time
Vrp	C	1		747		418		**374**		619		480
	T	3	1	4348	1	4018	**1**	**3974**	1	4219	1	4080
Warehouse	C	2		997		**901**		1082		1027		2349
	T	2	2	997	**2**	**901**	2	1082	2	1027	2	2349
All	C	72		2.96 h		2.96 h		3.44 h		3.45 h		**2.89 h**
	T	264	**127**	**66.31 h**	102	75.45 h	109	75.20 h	105	72.70 h	114	69.97 h

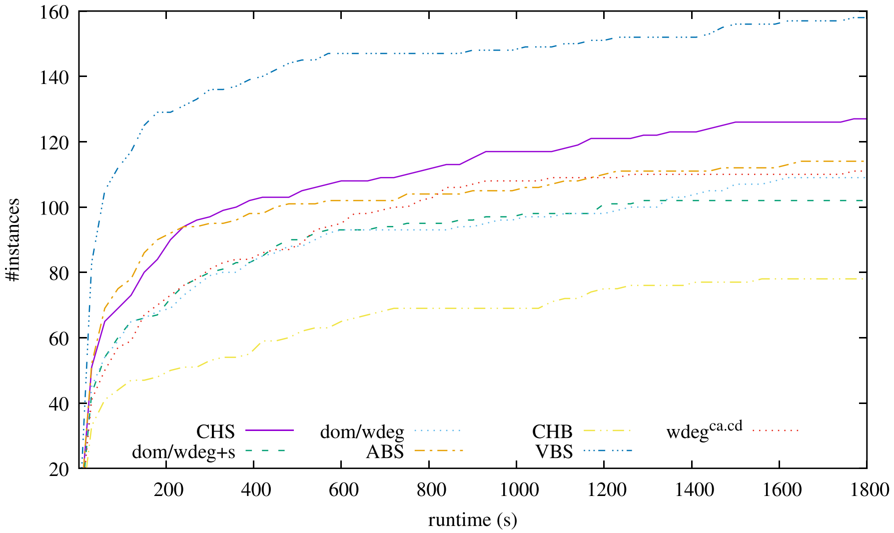

Fig. 9 Number of solved instances as a function of the elapsed time for the considered heuristics (namely CHS, $dom/wdeg$+s, $dom/wdeg$, $wdeg^{ca.cd}$, and ABS) and the VBS based on these five heuristics

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

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)

Balafrej, A., Bessiere, C., Paparrizou, A.: Multi-armed bandits for adaptive constraint propagation. In: Proceedings of IJCAI, pp. 290–296 (2015)

Battiti, R., Campigotto, P.: An Investigation of Reinforcement Learning for Reactive Search Optimization, pp. 131–160. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
Bessière, C., Chmeiss, A., Saïs, L.: Neighborhood-based variable ordering heuristics for the constraint satisfaction problem. In: Proceedings of CP, pp. 565–569 (2001)
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)
Boussemart, F., Hemery, F., Lecoutre, C., Saïs, L.: Boosting systematic search by weighting constraints. In: Proceedings of ECAI, pp. 146–150 (2004)
Brélaz, D.: New methods to color vertices of a graph. Commun. ACM **22**(4), 251–256 (1979)
Cabon, C., de Givry, S., Lobjois, L., Schiex, T., Warners, J.P.: Radio link frequency assignment. Constraints **4**, 79–89 (1999)
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)
Eén, N., Sörensson, N.: An extensible SAT-solver. In: Proceedings of SAT, pp. 502–518 (2003)
Gay, S., Hartert, R., Lecoutre, C., Schaus, P.: Conflict Ordering search for scheduling problems. In: Pesant, G. (ed.) Proceedings of CP, pp. 140–148 (2015)
Geelen, P.A.: Dual viewpoint heuristics for binary constraint satisfaction problems. In: Proceedings of ECAI, pp. 31–35 (1992)
Golomb, S.W., Baumert, L.D.: Backtrack programming. J. ACM **12**, 516–524 (1965)
Gomes, C.P., Selman, B., Crato, N., Kautz, H.A.: Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. J. Autom. Reason. **24**(1/2), 67–100 (2000)
Habet, D., Jégou, P., Kanso, H., Terrioux, C.: BTD_12 and miniBTD_12. In: Proceedings of the XCSP3 Competition, pp. 68–69 (2018)
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)
Haralick, R.M., Elliot, G.L.: Increasing tree search efficiency for constraint satisfaction problems. AIJ **14**, 263–313 (1980)
Hebrard, E., Siala, M.: Explanation-based weighted degree. In: Proceedings of CPAIOR, pp. 167–175 (2017)
Holland, A., O'Sullivan, B.: Weighted super solutions for constraint programs. In: Proceedings of AAAI, pp. 378–383 (2005)
Hooker, J.N.: Testing heuristics: we have it all wrong. J. Heuristics **1**(1), 33–42 (1995)
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)
Jégou, P., Kanso, H., Terrioux, C.: BTD and miniBTD. In: XCSP3 Competition (2017)
Jégou, P., Kanso, H., Terrioux, C.: BTD and miniBTD. In: Proceedings of the XCSP3 Competition, pp. 66–67 (2018)
Jégou, P., Terrioux, C.: Hybrid backtracking bounded by tree-decomposition of constraint networks. AIJ **146**, 43–75 (2003)
Lecoutre, C., Sais, L., Tabary, S., Vidal, V.: Last conflict based reasoning. In: Proceedings of ECAI, pp. 133–137 (2006)
Lecoutre, C., Sais, L., Tabary, S., Vidal, V.: Nogood recording from restarts. In: Proceedings of IJCAI, pp. 131–136 (2007)
Lecoutre, C., Sais, L., Tabary, S., Vidal, V.: Recording and minimizing nogoods from restarts. JSAT **1**(3–4), 147–167 (2007)
Liang, J.H., Ganesh, V., Poupart, P., Czarnecki, K.: Exponential recency weighted average branching heuristic for SAT solvers. In: Proceedings of AAAI, pp. 3434–3440 (2016a)
Liang, J.H., Ganesh, V., Poupart, P., Czarnecki, K.: Learning rate based branching heuristic for SAT solvers. In: Proceedings of SAT, pp. 123–140 (2016b)
Liberatore, P.: On the complexity of choosing the branching literal in DPLL. Artif. Intell. **116**(1–2) (2000)
Marques-Silva, J., Sakallah, K.A.: GRASP: a search algorithm for propositional satisfiability. IEEE Trans. Comput. **48**(5), 506–521 (1999)
Michel, L., Hentenryck, P.V.: Activity-based search for black-box constraint programming solvers. In: Proceedings of CPAIOR, pp. 228–243 (2012)
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)

Nadel, B.: Tree search and arc consistency in constraint-satisfaction algorithms, pp. 287–342. In: Search in Artificial Intelligence. Springer-Verlag (1988)
Refalo, P.: Impact-based search strategies for constraint programming. In: Proceedings of CP, pp. 557–571 (2004)
Robertson, N., Seymour, P.D.: Graph minors II: algorithmic aspects of treewidth. Algorithms **7**, 309–322 (1986)
Rossi, F., van Beek, P., Walsh, T.: Handbook of Constraint Programming, Foundations of Artificial Intelligence, vol. 2. Elsevier (2006)
Sabin, D., Freuder, E.C.: Contradicting conventional wisdom in constraint satisfaction. In: Proceedings of ECAI, pp. 125–129 (1994)
Schulte, C.: Programming branchers. In: Schulte, C., Tack, G., Lagerkvist, M.Z. (eds.) Modeling and Programming with Gecode (2018). Corresponds to Gecode 6.0.1
Simonin, G., Artigues, C., Hebrard, E., Lopez., P.: Scheduling scientific experiments for comet exploration. Constraints **20(1)**, 77–99 (2015)
Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction, 1st edn. MIT Press, Cambridge (1998)
Wattez, H., Koriche, F., Lecoutre, C., Paparrizou, A., Tabary, S.: Learning variable ordering heuristics with multi-armed bandits and restarts. In: Proceedings of ECAI (2020)
Wattez, H., Lecoutre, C., Paparrizou, A., Tabary, S.: Refining constraint weighting. In: Proceedings of ICTAI, pp. 71–77 (2019)
Xia, W., Yap, R.H.C.: Learning robust search strategies using a bandit-based approach. In: Proceedings of AAAI, pp. 6657–6665 (2018)