Conflict History based Search for Constraint Satisfaction Problem
Djamal Habet, Cyril Terrioux

To cite this version:
Djamal Habet, Cyril Terrioux. Conflict History based Search for Constraint Satisfaction Problem. Proceedings of the 34th ACM/SIGAPP Symposium On Applied Computing (SAC), Apr 2019, Limassol, Cyprus. 10.1145/3297280.3297389 . hal-02090618

HAL Id: hal-02090618
https://hal-amu.archives-ouvertes.fr/hal-02090618
Submitted on 20 May 2020

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 Search
for Constraint Satisfaction Problem

Djamal Habet, Cyril Terrioux
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
{djamal.habet,cyril.terrioux}@lis-lab.fr

Abstract
Branching heuristic is an important module in algorithms dedicated to solve Constraint Satisfaction Problems (CSP). It impacts the efficiency of exploring the search space and exploiting the problem structure. In this paper, we propose Conflict-History Search (CHS), a dynamic and adaptive branching heuristic for CSP solving. It is based on the search failures and considers the temporality of these failures throughout the resolution process.

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 obtains competitive results and can improve those obtained by other heuristics of the state of the art, such as dom/wdeg and ABS.

Keywords. CSP, Branching Heuristic, Conflict Based Search, Search History, Exponential Recency Weighted Average

DOI. https://doi.org/10.1145/3297280.3297389

1 Introduction
The Constraint Satisfaction Problem (CSP) is a powerful framework to model and efficiently solve problems that occur in various fields, both academic and industrial [21]. A CSP instance is defined on a set of variables which must be assigned in their respective finite domains by satisfying a set of constraints which express restrictions between different assignments. A solution is an assignment of each variable which satisfies all constraints.

CSP solving, based on search tree algorithms, has made significant progress in recent years thanks to research on several aspects which receive considerable efforts such as global constraints, filtering techniques, learning and restarts. An important component in CSP solvers is the variable branching rule. 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.

Many heuristics have been proposed (e.g. [1, 2, 3, 4, 6, 7, 18, 20, 9]) and aim to satisfy the famous first-fail principle [8] which advises “to succeed, try first where you are likely to fail”. Nowadays, the most efficient heuristics are adaptive and dynamic [3, 6, 18, 20, 9]. Indeed, the order of branchings 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 [18, 20]. Defined since 2004, the dom/wdeg heuristic remains one of the simplest, the most popular and efficient one. It is based on the hardness of constraints to reflect 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 [3].

In this paper, we propose Conflict-History Search (CHS), a new dynamic and adaptive branching 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 [24]. It was also recently used in defining powerful branching heuristics for solving the satisfiability problem (SAT) [15, 16]. We have integrated CHS in solvers based on MAC (Maintaining Arc Consistency) [22] and BTD (Backtracking with Tree Decomposition) [12]. The empirical evaluation on XCSP3 instances (XCSP3, for XML-CSP version 3, is an XML-based format to represent instances of combinatorial constrained problems) shows that CHS is competitive and brings improvements to the heuristics of the state of the art.

*This work has been funded by the french Agence Nationale de la Recherche, reference ANR-16-C40-0028.
The paper is organized as follows. Section 2 includes some necessary definitions and notations. Section 3 describes related work on branching heuristics for CSP and SAT. Section 4 presents and details our contribution which is evaluated experimentally in Section 5. Finally, we conclude and give future work.

2 Preliminaries

We give some definitions including CSP and Exponential Recency Weighted Average (ERWA).

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.

Each constraint \(c_i\) is defined by \(S(c_i)\) and \(R(c_i)\), where \(S(c_i) = \{x_{i_1}, \ldots, x_{i_k}\} \subseteq X\) defines the scope of \(c_i\) and \(R(c_i) \subseteq D_{i_1} \times \cdots \times D_{i_k}\) is its compatibility relation. The constraint satisfaction problem asks for an assignment of a value from \(D_i\) to each variable \(x_i\) of \(X\) that satisfies each constraint in \(C\). Checking whether a CSP instance has a solution (i.e. a consistent assignment of \(X\)) is NP-complete.

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 \(\frac{1}{m}\sum_{i=1}^{m}y_i\) where each \(y_i\) has the same weight \(\frac{1}{m}\). However, recent data may be more pertinent than the older ones to characterize the current situation. The Exponential Recency Weighted Average (ERWA) [24] takes into account such considerations by giving to the recent data higher weights than the older ones. In fact, the exponential moving average \(\bar{y}_m\) is computed by:

\[
\bar{y}_{m+1} = (1 - \alpha) \bar{y}_m + \alpha y_{m+1}.
\]

ERWA was used to solve the bandit problem to estimate the expected reward of different actions in non-stationary environments [24]. In bandit problems, there is a set of actions and the agent must select the action to play in order to maximize its long term expected reward.

3 Related Work

We present the most efficient branching heuristics for CSP and SAT. The recalled heuristics share the same behavior. Indeed, the variables and/or constraints are weighted dynamically throughout the search by considering the collected information since the beginning of the search. Also, some heuristics smooth (or decay) these weights as it will be explained further.

3.1 Impact-Based Search (IBS)

This heuristic selects the variable which leads to the largest search space reduction [20]. This 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.

3.2 Conflict-Driven Heuristic

A popular branching heuristic for CSP solving is \(dom/wdeg\) [3]. It guides the search towards the variables appearing in the constraints which seem to be 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:
the decision has been made. Let indeed, at each node of the search tree, constraint propagation may filter the domains of some variables after filtering information and maintains measures of how often the variable domains are reduced during the search. This heuristic is motivated by the prominent role of filtering techniques in CSP solving [18]. It exploits this

### 3.3 Activity-Based Heuristic (ABS)

This heuristic is motivated by the prominent role of filtering techniques in CSP solving [18]. It exploits this filtering information and maintains measures of how often the variable domains are reduced during the search. Indeed, at each node of the search tree, constraint propagation may filter the domains of some variables after the decision has been made. 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 \( x_i \in X_f \) and
- \( A(x_i) = \gamma \times A(x_i) \) if \( x_i \notin X_f \).

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

### 3.4 Branching Heuristics for SAT

In the context of the satisfiability problem (SAT), modern solvers based on Conflict-Driven Clause Learning (CDCL) [5, 17, 19] 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 [19] 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) [15], 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 [16] extends CHB by exploiting locality and introducing the learning rate of the variables.

Dedicated to constraint programming, Gecode solver implements CHB since version 5.1.0 released in April 2017 [23]. Indeed, in this version of Gecode, the variables are weighted in the same manner as in the SAT context following ERWA [15, 16].

As we will describe it in the next sections, the branching heuristic that we propose (CHS) uses ERWA to weight the constraints and not the variables. The constraint weights are then used in a next phase to calculate the variable scores which are used to select the branching variable.

### 4 Conflict-History Search for CSP

Inspired by the CHB heuristic for SAT, we define a new branching heuristic for CSP solving which we call Conflict-History Search (CHS). The central idea is to consider the history of constraint failures and favor the variables that often appear in recent failures.

So, the conflicts are dated and the constraints are weighted on the basis of the exponential recency weighted average. These weights are coupled to the variable domains to calculate the Conflict-History scores of the variables.

#### 4.1 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 by propagation then \( q(c_j) \) is updated by the formula below derived from ERWA:

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

The parameter \( 0 < \alpha < 1 \) is the step-size and \( r(c_j) \) is the reward value. It defines the importance given to the old value of \( q \) at the expense of the reward \( r \). The value of \( \alpha \) decreases over time as it is applied in ERWA [24].
Indeed, starting from its initial value \( \alpha_0 \), \( \alpha \) decreases by \( 10^{-6} \) at each constraint failure to a minimum of 0.06. Decreasing the \( \alpha \) value 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.

The reward value \( r(c_j) \) is based on how recently \( c_j \) occurred in conflicts. The goal is 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. Also initialized to 0 for each constraint \( c_j \in C \), Conflict\((c_j)\) stores the last \( \#Conflicts \) value where \( c_j \) led to a failure. Once \( r(c_i) \) and \( q(c_i) \) are updated, \( \#Conflicts \) is incremented by 1.

At this stage, we are able to define the Conflict-History score of the variables \( x_i \in X \), which will be used in selecting the branching variable as follows:

\[
\text{chv}(x_i) = \frac{\sum_{c_j \in C \mid x_i \in S(c_j) \land |Uvars(c_j)| > 1} q(c_j)}{|D_i|}
\]

CHS keeps the variable to branch on with the highest \( \text{chv} \) value. In this manner, CHS focuses branching on the variables with small size of domain while 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 equal to 0. To avoid random selection of the branching variable, we reformulate the calculation of \( \text{chv} \) as given below, where \( \delta \) is a positive real number close to 0.

\[
\text{chv}(x_i) = \frac{\sum_{c_j \in C \mid x_i \in S(c_j) \land |Uvars(c_j)| > 1} (q(c_j) + \delta)}{|D_i|}
\]

Thus, at the beginning of the search, 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.

### 4.2 CHS and Restarts

Nowadays, restart techniques are important for the efficiency of solving algorithms (see for example [13]). Restarts may allow to reduce the impact of irrelevant choices done during search according to heuristics such as variable selection.

As it will be detailed in the next section, 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 later). Keeping this information unchanged reinforces learning from the search history.

Concerning the step-size \( \alpha \), 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 step-size value \( \alpha \) to \( \alpha_0 \) at each restart. This may guide the search through different parts of the search space.

### 4.3 CHS and Smoothing

At each conflict and as in the \textit{dom/wdeg} heuristic, CHS updates the \( \text{chv} \) 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 high for the current part of the search. To avoid this situation, we propose to smooth the scores \( q(c_i) \) 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. Decaying is also used in other heuristics such as ABS [18] for CSP and VSIDS [19] for SAT. However, it is applied to the score of the variables and not that of the constraints (or clauses).
5 Experimental Evaluation

5.1 Experimental Protocol

We consider 10,785 instances from the XCSP3 repository\(^1\), including notably structured instances and discarding fully random instances. This latter restriction is quite natural since adaptive heuristics aim to exploit the underlying structure of the instances to solve.

Regarding the solving step, we exploit MAC with restarts \([14]\) before assessing the impact of our approach on a structural solving method, namely BTD-MAC+RST+Merge \([10]\). MAC uses 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 have written our own code to implement all the compared branching heuristics in this section (\(\text{dom/wdeg}\), \(\text{ABS}\), \(\text{CHS}\), \(\text{CHB}\), \(\text{ABS}\) and \(\text{dom/wdeg}\)), as well as the solvers that exploit them (MAC and BTD). All the algorithms are written in C++.

The experiments are performed on Dell PowerEdge M610 blade servers with Intel Xeon E5620 processors under Ubuntu 18.04. Each solving process is allocated a slot of 30 minutes and at most 12 GB of memory per instance. In the following tables, \#solv denotes the number of solved instances by a given solver and time is the cumulative runtime.

5.2 Impact of CHS Settings

In this part, we assess the sensitivity of CHS with respect to the chosen values for \(\alpha\) or \(\delta\). First, we fix \(\delta\) to \(10^{-4}\) (to start the search by considering the variable degrees then quickly exploit ERWA-based computation) and vary the value of \(\alpha_0\) between 0.1 and 0.9 with a step of 0.1. Figure 1 presents the number of instances solved by MAC depending on the value of \(\alpha\) and the corresponding cumulative runtime. We also provide the results of the Virtual Best Solver (VBS) which counts the number of the instances solved (and the cumulative running time) at least one time when varying the value of \(\alpha\).

We observe that the value \(\alpha_0 = 0.4\) allows MAC to solve more instances (9,525 solved instances with a cumulative solving time of 493 hours) than the other considered values. The worst case is \(\alpha_0 = 0.7\) with 9,515 solved instances in 496 hours. This shows the robustness of CHS w.r.t. the \(\alpha\) parameter.

Regarding the Virtual Best Solver (VBS), we note that it can solve 64 additional instances than MAC+CHS when \(\alpha_0 = 0.4\). Among these instances, some of them seem to be hard. Indeed, often, only one of the checked values of \(\alpha\) allows MAC to solve them and the required runtime generally exceeds several minutes. Therefore, a finer adjustment of the value of \(\alpha_0\) or its adaptation to the treated instance would allow MAC+CHS to perform even better.

Now, we set \(\alpha_0\) to 0.4 and evaluate different values of \(\delta\). From Table 1, the observations are similar to those presented previously, showing the robustness of CHS regarding \(\delta\).

Also, it is interesting to highlight that MAC+CHS with \(\delta = 0\) solves 9,517 instances while it solves 9,525 instances if \(\delta = 10^{-4}\). This illustrates the relevance of introducing \(\delta\) in CHS while it allows to solve 8 more instances.

---

\(^1\)http://www.xcsp.org/series
Table 1: Impact of $\delta$ value on MAC+CHS regarding the number of solves instances and the cumulative runtime in hours

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>#solv.</th>
<th>time (h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>9,517</td>
<td>498.17</td>
</tr>
<tr>
<td>$10^{-3}$</td>
<td>9,520</td>
<td>494.07</td>
</tr>
<tr>
<td>$10^{-4}$</td>
<td>9,525</td>
<td>493.41</td>
</tr>
<tr>
<td>$10^{-3}$</td>
<td>9,524</td>
<td>493.91</td>
</tr>
</tbody>
</table>

instances with this last setting.

Finally, Table 2 gives the results of MAC+CHS ($\alpha_0 = 0.4$, $\delta = 10^{-4}$) with smoothing (+$s$) or not (-$s$) the constraint scores and/or with resetting (+$r$) or not (-$r$) the value of $\alpha$ to 0.4 at each new restart.

Table 2: Number of instances solved by MAC with CHS with/without smoothing and reset of $\alpha$ and cumulative runtime in hours

<table>
<thead>
<tr>
<th>Solver</th>
<th>#solv.</th>
<th>time (h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAC+CHS (+$s$+$r$)</td>
<td>9,525</td>
<td>493.41</td>
</tr>
<tr>
<td>MAC+CHS+$s$-$r$</td>
<td>9,509</td>
<td>498.80</td>
</tr>
<tr>
<td>MAC+CHS-$s$+$r$</td>
<td>9,482</td>
<td>514.73</td>
</tr>
<tr>
<td>MAC+CHS-$s$-$r$</td>
<td>9,478</td>
<td>518.05</td>
</tr>
</tbody>
</table>

The observed behaviors clearly support the importance of smoothing and restarts for CHS. For example, MAC+CHS+$s$-$r$ solves 16 less instances than MAC+CHS, while MAC+CHS-$s$+$r$ solves 47 instances less.

5.3 CHS vs. Other Search Heuristics

Now, we compare CHS (the settings are: $\alpha_0 = 0.4$, $\delta = 10^{-4}$) to other search strategies: dom/wdeg, ABS and CHB as implemented in Gecode. For ABS, we fix the decay parameter $\gamma$ to 0.999 as in [18]. For CHB, we use the value parameters as given in [23]. We add a variant dom/wdeg+$s$ which is dom/wdeg but the weights of constraints are smoothed exactly as in CHS.

Table 3: Number of instances solved by MAC with dom/wdeg, dom/wdeg+$s$, ABS, CHB and CHS and cumulative runtime in hours

<table>
<thead>
<tr>
<th>Solver</th>
<th>#solv.</th>
<th>time (h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAC+CHS</td>
<td>9,525</td>
<td>493.41</td>
</tr>
<tr>
<td>MAC+dom/wdeg</td>
<td>9,501</td>
<td>507.17</td>
</tr>
<tr>
<td>MAC+dom/wdeg+$s$</td>
<td>9,500</td>
<td>505.13</td>
</tr>
<tr>
<td>MAC+ABS</td>
<td>9,476</td>
<td>515.17</td>
</tr>
<tr>
<td>MAC+CHB</td>
<td>9,458</td>
<td>525.38</td>
</tr>
</tbody>
</table>

From Table 3, it is clear that MAC with CHS performs better than with the other heuristics. Indeed, it solves 24 instances more than MAC+dom/wdeg, 49 instances more than MAC+ABS and 67 instances more than MAC+CHB. Interestingly, whatever the value of $\alpha_0$, MAC with CHS remains better than all its competitors. Indeed, the worst case is when $\alpha_0 = 0.7$ where MAC+CHS solves 9,515 instances. Moreover, the results obtained by MAC+CHB show 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). Furthermore, the smoothing phase introduced in dom/wdeg allows MAC+dom/wdeg+$s$ to reduce slightly the computation time when compared to MAC+dom/wdeg, while solving one less instance.

dom/wdeg and ABS are two powerful and popular branching heuristics. In particular, dom/wdeg is widely used in the literature and integrated in many solvers. A careful reading of the results of Table 3 shows that dom/wdeg solves 25 instances more than ABS. This remark is to highlight the improvement brought by our heuristic CHS. Indeed, it allows MAC to solve 24 additional instances compared to dom/wdeg.

Finally, Table 4 provides the results of MAC variants on some instance families chosen from a representative panel of the used benchmark in order to show the different trends we observed. First, we note that no heuristic is
Table 4: Number of instances solved by MAC with dom/wdeg, ABS, CHB and CHS and cumulative runtime in seconds for some instance families.

<table>
<thead>
<tr>
<th>Family</th>
<th>Name</th>
<th>#inst</th>
<th>#solv.</th>
<th>time (s)</th>
<th>#solv.</th>
<th>time (s)</th>
<th>#solv.</th>
<th>time (s)</th>
<th>#solv.</th>
<th>time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Academic</td>
<td>AllInterval-m1-s1</td>
<td>32</td>
<td>25</td>
<td>15,406</td>
<td>32</td>
<td>9</td>
<td>32</td>
<td>9</td>
<td>32</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>Blackhole-xcsp2-s04</td>
<td>10</td>
<td>10</td>
<td>5.51</td>
<td>10</td>
<td>4.82</td>
<td>10</td>
<td>5.15</td>
<td>10</td>
<td>5.46</td>
</tr>
<tr>
<td></td>
<td>GracefulGraph-m1-s1</td>
<td>104</td>
<td>17</td>
<td>160,007</td>
<td>16</td>
<td>160,090</td>
<td>16</td>
<td>160,355</td>
<td>18</td>
<td>155,667</td>
</tr>
<tr>
<td></td>
<td>Kakuro-sundiff-hard</td>
<td>187</td>
<td>187</td>
<td>285</td>
<td>185</td>
<td>4,216</td>
<td>180</td>
<td>15,044</td>
<td>187</td>
<td>811</td>
</tr>
<tr>
<td></td>
<td>Nonogram-table-s1</td>
<td>176</td>
<td>167</td>
<td>1,991</td>
<td>168</td>
<td>34.56</td>
<td>168</td>
<td>35.19</td>
<td>168</td>
<td>331.67</td>
</tr>
<tr>
<td></td>
<td>PigeonsPlus-m1-s1</td>
<td>38</td>
<td>37</td>
<td>4,860</td>
<td>29</td>
<td>20,120</td>
<td>37</td>
<td>5,192</td>
<td>37</td>
<td>4,878</td>
</tr>
<tr>
<td></td>
<td>Sat-xcsp2-bmc</td>
<td>24</td>
<td>24</td>
<td>1,816</td>
<td>24</td>
<td>518</td>
<td>20</td>
<td>51</td>
<td>24</td>
<td>4,708</td>
</tr>
<tr>
<td></td>
<td>Subisomorphism-m1-LV</td>
<td>1,176</td>
<td>1,100</td>
<td>151,661</td>
<td>1,108</td>
<td>136,868</td>
<td>1,101</td>
<td>147,664</td>
<td>1,109</td>
<td>134,787</td>
</tr>
<tr>
<td></td>
<td>SuperSolutions-Taillard-os05</td>
<td>30</td>
<td>23</td>
<td>14,102</td>
<td>19</td>
<td>20,036</td>
<td>26</td>
<td>11,125</td>
<td>21</td>
<td>16,386</td>
</tr>
<tr>
<td></td>
<td>TravellingSalesman-xcsp2-s20a4</td>
<td>15</td>
<td>15</td>
<td>64.29</td>
<td>15</td>
<td>60.77</td>
<td>15</td>
<td>311.45</td>
<td>15</td>
<td>116.59</td>
</tr>
<tr>
<td>Real-world</td>
<td>OpenStacks</td>
<td>76</td>
<td>40</td>
<td>4,664</td>
<td>40</td>
<td>6,342</td>
<td>41</td>
<td>5,757</td>
<td>41</td>
<td>5,007</td>
</tr>
<tr>
<td></td>
<td>RenaultMod-m1-s1</td>
<td>50</td>
<td>50</td>
<td>1.70</td>
<td>50</td>
<td>0.61</td>
<td>50</td>
<td>0.99</td>
<td>50</td>
<td>0.52</td>
</tr>
<tr>
<td></td>
<td>SocialGolfers-xcsp2-s1</td>
<td>12</td>
<td>4</td>
<td>14,576</td>
<td>4</td>
<td>16,300</td>
<td>5</td>
<td>14,030</td>
<td>6</td>
<td>11,404</td>
</tr>
</tbody>
</table>

always better than the others. However, if we sort the heuristics with respect to the number of solved instances per family, CHS is ranked at the first place for 88% of the 141 considered families, by performing better or similarly than the two other heuristics. This percentage exceeds respectively 93% and 99% if we consider the first two places or the first three places. Hence, CHS is clearly competitive.

Also, one might think that dom/wdeg performs worse than ABS and CHB. This impression is explained by the fact that, when MAC+dom/wdeg is better on a given family, it solves only few additional instances. In contrast, when it is outperformed, this is done by several additional solved instances. Finally, if we compare the results on the instances labeled real-world in the XCSP3 repository, we observe that MAC with CHS solves more instances and performs faster, between 10% and 30%, than any other combination.

5.4 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 at exploiting 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 [10]. The parameters of BTD-MAC+RST+Merge are set like in [11] except the variable heuristic which can be one of the two best heuristics considered previously, namely dom/wdeg or CHS.

Like for MAC, the solving is more efficient with CHS than with dom/wdeg. Indeed, BTD-MAC+RST+Merge with CHS solves 9,525 instances (in 485 h) against 9,495 instances (in 501 h) for dom/wdeg: 30 additional instances are solved. This observation shows that exploiting both CHS and tree-decomposition may be of interest and that these two strategies can be complementary.

6 Conclusion

We have proposed CHS, a new branching heuristic for CSP based on the search history and designed following techniques coming from reinforcement learning. The experimental results confirm the relevance of CHS which is competitive with the powerful heuristics dom/wdeg and ABS, when implemented in solvers based on MAC or tree-decomposition exploitation.

The experimental study suggests that the 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 the appropriate value of α. Furthermore, similarly to the ABS heuristic, we will also consider to include information provided by filtering operations in CHS. Finally, we will extend CHS to deal with constraint optimization problems.

References


