



Visualizations to Summarize Search Behavior

Ian S. Howell^{1,2(✉)}, Berthe Y. Choueiry¹, and Hongfeng Yu²

¹ Constraint Systems Laboratory, University of Nebraska-Lincoln,
Lincoln, NE 68508, USA

{[ihowell](mailto:ihowell@cse.unl.edu), [choueiry](mailto:choueiry@cse.unl.edu)}@cse.unl.edu

² Visualization Laboratory, University of Nebraska-Lincoln,
Lincoln, NE 68508, USA

yu@cse.unl.edu

Abstract. In this paper, we argue that metrics that assess the performance of backtrack search for solving a Constraint Satisfaction Problem should not be visualized and examined only at the end of search, but their evolution should be tracked throughout the search process in order to provide a more complete picture of the behavior of search. We describe a process that organizes search history by automatically recognizing qualitatively significant changes in the metrics that assess search performance. To this end, we introduce a criterion for quantifying change between two time instants and a summarization technique for organizing the history of search at controllable levels of abstraction. We validate our approach in the context of two algorithms for enforcing consistency: one that is activated by a surge of backtracking and the second that modifies the structure of the constraint graph. We also introduce a new visualization for exposing the behavior of variable ordering heuristics and validate its usefulness both as a standalone tool and when displayed alongside search history.

Keywords: Search · Visualization · Constraint Satisfaction

1 Introduction

In this paper, we propose a new perspective and visualization tools to understand and analyze the behavior of the backtrack-search procedure for solving Constraint Satisfaction Problems (CSPs). Backtrack search is commonly used for solving CSPs. However, its performance is unpredictable and can differ widely on similar instances. Further, maintaining a given consistency property during search has become a common practice [13, 22, 30, 42, 43] and new strategies for

This research is supported by NSF Grant No. RI-1619344 and NSF CAREER Award No. III-1652846. The experiments were completed utilizing the Holland Computing Center of the University of Nebraska, which receives support from the Nebraska Research Initiative.

dynamically switching between consistency algorithms are being investigated [1–3, 9, 15, 16, 18, 25, 29, 41, 46, 48]. While consistency algorithms can dramatically reduce the size of the search space, their impact on the CPU cost of search can vary widely. This cost is currently poorly understood, hard to predict, and difficult to control.

To gain an understanding of the behavior of search and its performance, most previous work has focused on the visualization of various metrics at the *end* of search. In contrast, we claim that it is sometimes advantageous to *inspect the history* [20] or evolution of these metrics along the search process in order to detect bottlenecks that can be addressed by specific tools (e.g., modify the topological structure of the constraint network [43] or enforce a particular high-level consistency). As we demonstrate in this paper, it is sometimes not practical, or even feasible, to manually carry out such an analysis. For this reason, we advocate to automatically organize the history of search (i.e., its evolution over time) as a sequence of *regimes* [23] of qualitatively distinct behaviors detected by sampling the performance metrics during search. We claim that our visualizations are useful to the researcher designing new adaptive strategies and consistency algorithms as well as to the savvy engineer deploying constraint-programming solutions.

Building on the visualizations proposed by Woodward et al. [46] for describing search performance, we introduce the following contributions:

1. A criterion for computing the distance between two samples, quantifying the change in search behavior.
2. A clustering technique for automatically summarizing the history of search and its progress over time in a hierarchical structure that a human user can inspect at various levels of details.
3. A new visualization that reveals and summarizes the behavior of a dynamic variable ordering heuristic.

This paper is structured as follows. We first review background information and the relevant literature. Then, we describe our contributions. For each contribution, we highlight its usefulness with illustrative examples. Finally, we discuss the usefulness of our approach and conclude with directions for future research.

2 Background and Related Work

Constraint Satisfaction Problems (CSPs) are used to model many combinatorial decision problems of practical importance in Computer Science, Engineering, and Management. A CSP is defined by a tuple (X, D, C) , where X is a set of *variables*, D is the set of the variables' *domains*, and C a set of *constraints* that restrict the combinations of values that the variables can take at the same time. We denote the number of variables $|X|=n$. A solution to a CSP is an assignment of values to variables such that all constraints are simultaneously satisfied. Determining whether or not a given CSP has a solution is known to be NP-complete. The *constraint network* of a CSP instance is a graph where the vertices represent the

variables in the CSP and the edges represent the constraints and connect the variables in their scope.

To date, backtrack search (BT) is the only sound and complete algorithm for solving CSPs [7]. Backtracking suffers from *thrashing* when it repeatedly explores similar subtrees without making progress. Strategies for reducing thrashing include intelligent backtracking, ordering heuristics, learning no-goods, and enforcing a given *consistency property* after every variable instantiation. Enforcing consistency reduces the size of the search space by deleting, from the variables' domains, values that cannot appear in a consistent solution given the current search path (i.e., conditioning). In recent years, the research community has investigated *higher-level consistencies* (HLC) as inference techniques to prune larger portions of the search space at the cost of increased processing effort [6, 13, 17, 33], leading to a trade-off between the search effort and the time for enforcing consistency. We claim that our visualizations are insightful tools to understand why and where to apply HLCs and to assess their effectiveness.

In this paper, we use by default dom/wdeg [10], a *dynamic* variable ordering heuristic (and dom/deg [5] where indicated). Below, we summarize the consistency properties that we enforce. We always maintain Generalized Arc Consistency (GAC) [26] during search. In addition to GAC, we enforce, as HLCs, Partition-One Arc-Consistency (POAC) [4], which filters the domains of the variables by performing singleton testing, and Partial Path Consistency (PPC) [8], which filters the constraints after triangulating the constraint graph. Both POAC and (strong) PPC are strictly stronger than GAC. In order to control their cost, we enforce the HLCs in a 'selective/adaptive' manner using the PREPEAK⁺ strategy of Woodward et al. [46] for POAC (denoted PREPEAK⁺-POAC) and the Directional Partial Path Consistency of Woodward [45] (denoted DPPC⁺). PREPEAK⁺ triggers POAC by watching the numbers of backtracks and follows geometric laws to increase or decrease the calls to POAC. DPPC⁺ chooses a subset of the triangles of the triangulated constraint network on which it calls PPC in a manner to reduce the addition of new constraints while increasing propagation across the network. Finally, we use a *k*-way branching strategy for search. However, preliminary tests on Choco [12], on which we tested GAC and implemented POAC, show similar shapes of the visualizations.

Prior research on search visualization has appeared in the Constraint Programming literature. Most previous approaches, such as the ones discussed below, can be viewed as tools for *inspecting* and *debugging* search. The DiSCiPl project provides extensive visual functionalities to develop, test, and debug constraint logic programs such as displaying variables' states, effect of constraints and global constraints, event propagation at each node of the search tree, and identifying isomorphic subtrees [11, 38]. Many useful methodologies from the DiSCiPl project are implemented in CP-Viz [39] and other works [37]. The OZ Explorer displays the search tree allowing the user to access detailed information about each tree node and to collapse and expand failing trees for closer examination [34]. This work is currently incorporated into Gecode's Gist [35]. The above approaches focus on exploring in detail the search tree to examine

and inspect the impact of individual search decisions on a particular variable, constraint, variable domain, or even a subtree of the search. In contrast, our work proposes to *summarize* the evolution of search along a given dimension (i.e., projection), providing a qualitative, abstract view of a search tree that is too large to be explored. We believe that the above approaches are orthogonal and complementary to ours.

Ghoniem et al. [19] propose a variety of heatmap visualizations to illustrate the operation of constraint propagation across an instance. Their heatmaps illustrate the impact of the activity of a variable or of a constraint on other variables or constraints in terms of filtering the variables’ domains or causing further constraint propagation. These visualizations are fine grained and inform the user of the effect of a particular decision. They have pedagogical value and can also be useful for debugging. They differ from our approach in that the user has the burden of identifying the animation frames of interests. Relatively short behaviors can go unnoticed and long behaviors can be wasteful of the user’s time. In contrast, our regime-identification process automatically organizes search history into a sequence of ‘interesting’ episodes. We consider their contribution too to be orthogonal and complementary to our approach.

Tracing the search effort by *depth* of the search tree was first proposed for the number of constraint checks and values removed per search level (Epstein et al. [16]) and for the number of nodes visited (Simonis et al. [39] in CP-Viz, also used for solving a packing problem [40]). More recently, Woodward et al. [46] propose to focus on the number of *backtracks* per depth (BpD) to assess the severity of thrashing and on the number of *calls* to an HLC per depth (CpD) to explain the cost of enforcing an HLC, see Fig. 1. Further, they split the CpD into three line charts, showing in green the HLC calls that prune inconsistent subtrees and yield domain wipeouts (most effective), in blue those that filter the domains of future variables (effective), and in red those that do not filter the

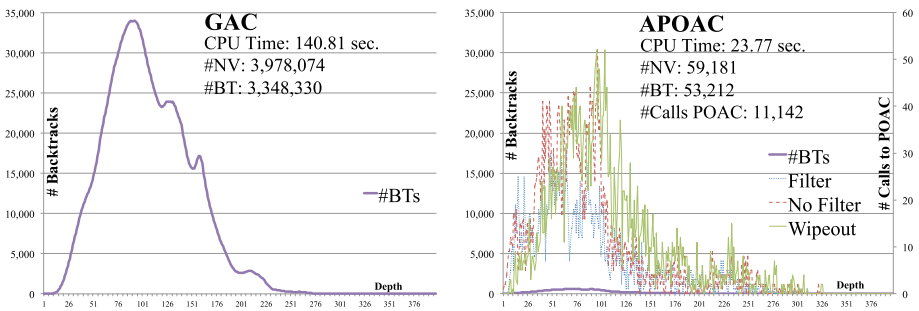


Fig. 1. From Woodward et al. [46]: Number of backtracks (BpD) and calls (CpD) to POAC per depth at the end of search for benchmark instance pseudo-aim-200-1-6-4 using dom/wdeg. Left: GAC. Right: APOAC, an adaptive POAC by Balafrej et al. [1]. The CPU time, #NV (number of nodes visited), and #BT (number of backtracks) are typically used to indicate the cost of search. (Color figure online)

domain of any future variable (wasteful). Finally, they superimpose the CpD to the BpD chart in order to reveal the effectiveness of the consistency enforced (and that of the adaptive strategy for enforcing an HLC, when applicable).

3 Tracking Search History

To the best of our knowledge, previous approaches to visualizing the various search ‘metrics per depth’ of the search tree (e.g., number of nodes visited, constraint checks, or backtracks) present the values reached at the *end* of search thus providing a ‘blanket’ summary where thrashing occurred and search invested the most resources. They fail to identify the problems that occur *during* search. We argue that these visualizations may mask important information, such as transient behaviors of search, that an algorithm developer or a savvy user need to examine and address in order to solve the problem. In this paper, we propose to track the progress of search by collecting samples of the BpD ‘profile’ during search, see Fig. 2.¹ In the examples discussed in this paper, we use a sampling rate of 100 ms. To this end, we propose a criterion for deciding whether two consecutive samples are ‘equivalent’ and pertain to the same qualitative *regime* [23] of search. The sequence of regimes yields a *history* [20] of the search.

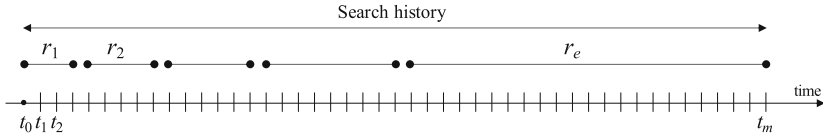


Fig. 2. BpD and CpD are sampled, during search, at times t_i ($0 < i \leq \max$), and the corresponding line charts are portioned into successive regimes of equivalent behavior that collectively describe the history of search progress

Below, we first introduce a criterion for assessing distance between two BpD samples. Then, we describe how to build a summarization tree using agglomerative hierarchical clustering and how the tree can be used to allow the human user to examine the search history at the desired level of detail. Finally, we provide two illustrative examples.

3.1 Distance Between Two BpD Samples

We adopt the following notations. $D = [1, d_{\max}]$ is the domain of the depth of the search tree for solving the CSP. In a k -way backtrack search, $d_{\max} = n$; in a binary-branching scheme, $d_{\max} \leq na$, where a is the maximum domain size. We sample search, recording the cumulative number of backtracks per depth (BpD)

¹ Note that we can interrupt search at any time to conduct the analysis and need not wait until the end of search.

at each sampling instant. $\text{BT}(d, t)$ designates the number of backtracks executed by search at depth $d \in D$ from the beginning of search until time t . The first regime starts at $t = 0$ and we set $\text{BT}_{d \in D}(d, 0) \equiv 0$. The last regime ends at t_m and corresponds to the last sample collected. We say that two time instants t and t' do not belong to the same qualitative regime if their corresponding BpD curves ‘differ enough’ according to some divergence or distance criterion. Below, we introduce one such criterion.

To measure the distance between BpDs at time stamps t and t' , we adapt, to our context, the Kullback-Leibler (KL) divergence (also called *relative entropy*) [24]. Given two discrete probability distributions P and Q defined on the same probability space, χ , the KL-divergence from Q to P , which is always non-negative, is defined as: $D_{\text{KL}}(P \parallel Q) = \sum_{x \in \chi} P(x) \log \left(\frac{P(x)}{Q(x)} \right)$.

In order to adapt the KL-divergence to our context, we introduce the probability that $\text{BT}(d, t)$ take a given value as $p(d, t) = \frac{\text{BT}(d, t)}{\sum_{d \in D} \text{BT}(d, t)}$. In order to avoid having this probability be zero when $\text{BT}(d, t) = 0$, we use Laplace Smoothing [27], approximating this probability with $\hat{p}(d, t) = \frac{\text{BT}(d, t) + \alpha}{\sum_{d \in D} \text{BT}(d, t) + \alpha d_{\text{max}}}$ and using $\alpha = 0.1$ in our experiments. Finally, we define the divergence $\text{div}(t, t') = \text{div}(t', t)$ between two BpD curves at time t and t' as

$$\text{div}(t, t') = \max \left(\sum_{d \in D} \hat{p}(d, t) \log \left(\frac{\hat{p}(d, t)}{\hat{p}(d, t')} \right), \sum_{d \in D} \hat{p}(d, t') \log \left(\frac{\hat{p}(d, t')}{\hat{p}(d, t)} \right) \right) \quad (1)$$

Further, we define the distance between two regimes as the divergence between the two BpDs located at the *middle* sample of the corresponding regimes. Another alternative, a little more costly, is to use the largest divergence between any two samples in the regimes. Other options exist, preliminary evaluations did not exhibit a significant sensitivity to this choice.

3.2 Clustering Tree and Summarization Tree

We propose to summarize the history of the search following the idea of *agglomerative hierarchical clustering* described by Nielsen [28, Chapter 8], which yields a binary tree.

Starting from the sequence of BpD samples (collected during search and ordered along the time line), we form the leaf nodes of the *clustering tree* storing one sample per node. We place these nodes in a vector Q . Then,

1. We determine the divergence between every two consecutive nodes in Q (those that store temporally *adjacent* samples).
2. We generate a parent for the two nodes with the smallest divergence value.
3. We replace, in Q , the two nodes with their parent.
4. We associate, to the parent, the concatenation of the sample sequences of the two children, ordered in increasing time.
5. We store, for the parent, the divergence value between its children.

6. We choose as representative of the new node the middle sample, breaking ties by using the earlier sample.

We iteratively apply the process until $|Q|= 1$. We call this tree the *clustering tree* and reuse it every time a new summarization is needed.

Finally, we identify the deepest nodes in the tree with a stored divergence value larger than or equal to a user-defined value δ and replace each such node by a new *regime node*. The regime node inherits from the node that it replaces, its representative middle sample, its divergence value, and the sequence of samples stored at the leaves of the corresponding subtree. We call the resulting structure a δ -*summarization tree*. The dendrogram shown in Fig. 3 illustrates this process.

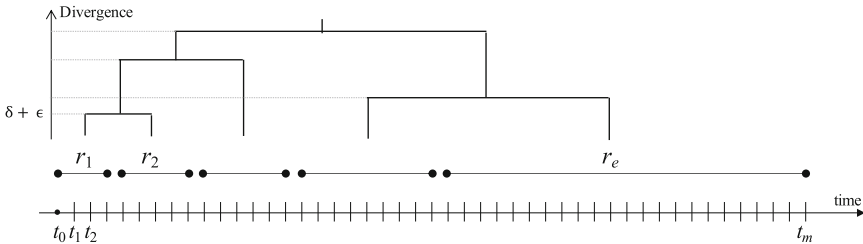


Fig. 3. Dendrogram of the summarization tree, showing the sequence of ‘cluster’ mergers and the divergence value at which each merger occurs (with $\epsilon \geq 0$)

The clustering tree stores samples at its leaves and does not use or depend on δ . The summarization tree uses δ to ‘trim’ the *lower* levels of the clustering tree, replacing them with regimes for explanation, which can be done interactively and is computationally cheap. Thus, if the summarization is too detailed or too abstract, the user may adjust δ to quickly regenerate a new history from the clustering tree to reach a level of detail that is cognitively satisfying. Further, iteratively adding samples to the clustering tree as search proceeds can be done, online, in logarithmic time. Consequently, building and maintaining the clustering tree can efficiently be done online, during search. However, this direction remains to be investigated.

In addition to the data structures used for the hierarchical clustering [36], we introduce two data structures: first, Q , implemented as a vector of size m , where m is the number of samples collected. Then, another vector to store the data of the non-leaf nodes of the clustering tree. This size of this vector is $m - 1$ because the clustering tree is a full binary tree. The clustering tree can be built in $\mathcal{O}(m^2)$ time and $\mathcal{O}(m)$ space [14].

3.3 Illustrative Example: Explaining Adaptive Consistency

In this section, we compare the behavior of search for solving the SAT instance `aim-200-2-0-sat-1` [50]. GAC takes less than 10 min and generates 14 million

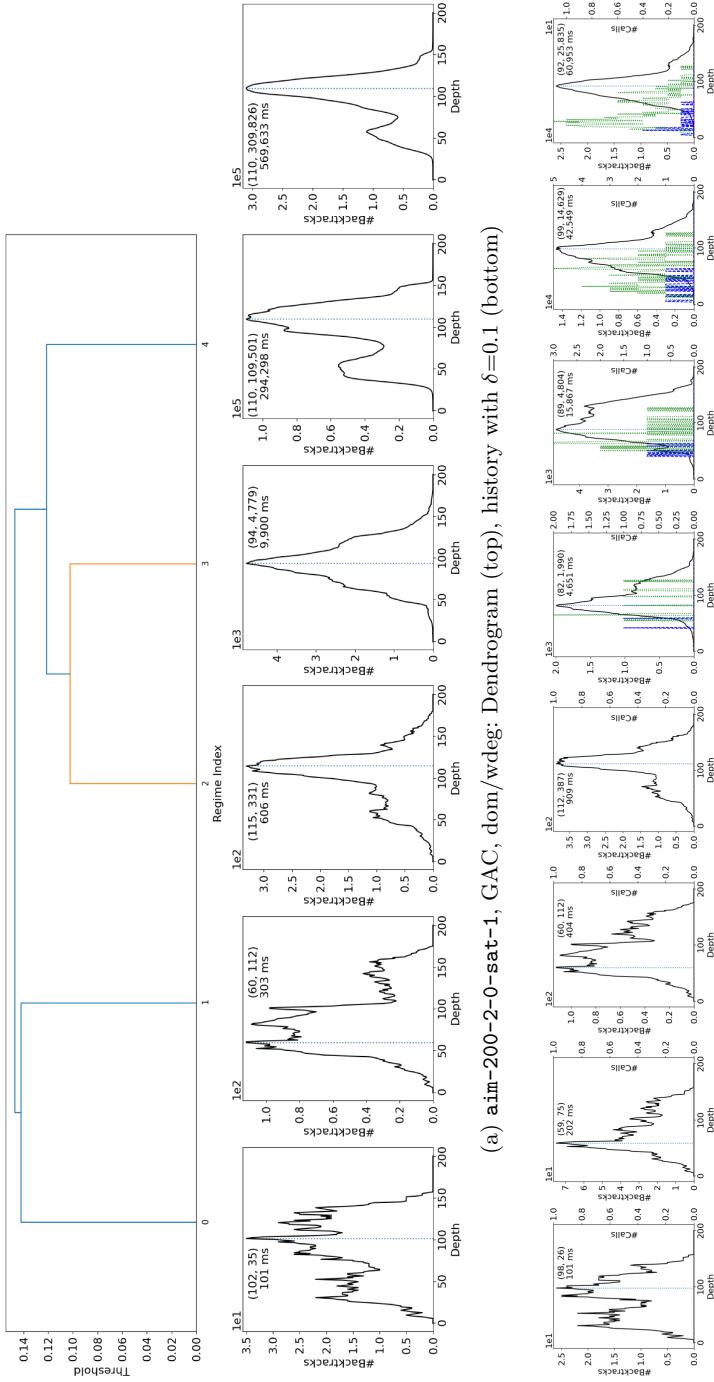


Fig. 4. aim-200-2-0-sat-1: Histories with GAC (top) and with PREPEAK⁺-POAC (bottom), using dom/wdeg. Showing the middle BpD sample in each regime and, for reference, the final sample collected. (Color figure online)

backtracks. PREPEAK⁺-POAC, on the other hand, solves the instance in little over a minute, netting only 1.2 million backtracks. To see why, we examine the corresponding search histories shown in Fig. 4a and 4b. We see that initially, both searches proceed exactly in the same manner. The behavior of PREPEAK⁺-POAC starts to diverge from that of GAC at the fifth regime of PREPEAK⁺-POAC (Fig. 4b) where PREPEAK⁺ reacts to the sharp increase of the number of backtracks around depth 100 and triggers POAC. POAC successfully prunes inconsistent subtrees (green calls) or decreases the size of search tree (blue calls). By examining the last three regimes of PREPEAK⁺ (Fig. 4b), we see how the HLC successfully reduces the severity of the backtracking, reducing the peak in height, width, and even in position. By investing in opportunistic and effective (even if costly) calls to POAC, we significantly reduce the search time.

3.4 Illustrative Example: Analyzing Structure

In this section, we show how the identification and visualization of the regimes of search provide insight into the structure of a problem instance and guide the choice of the appropriate type of consistency for solving it.

We consider the graph-coloring instance mug100-25-3 [50] and try to solve it with GAC using dom/wdeg for dynamic variable ordering. Search fails to terminate within two hours. The inspection of the BpD of GAC (left of Fig. 5) at the end of the unsuccessful search reveals the presence of a ‘dramatic’ peak of the number of backtracks at depth 83 (i.e., relatively deep in the search tree). This behavior hints that search may have made a bad decision at a shallower level of search from which it could not recover.

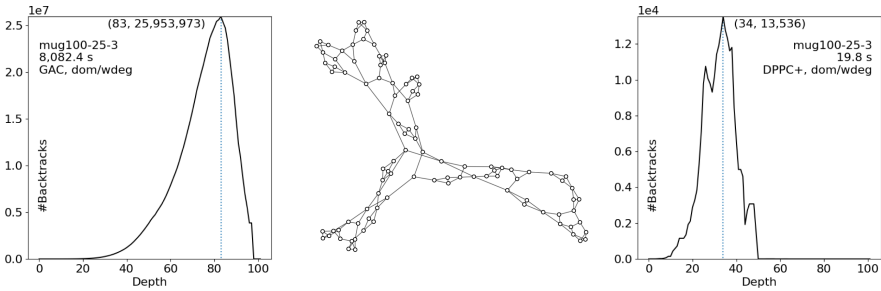
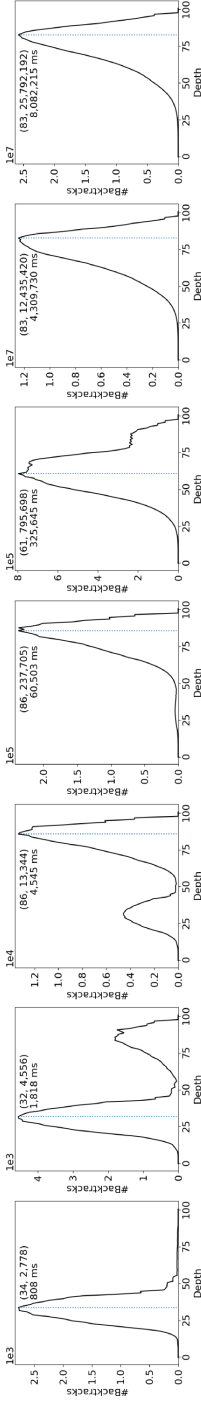
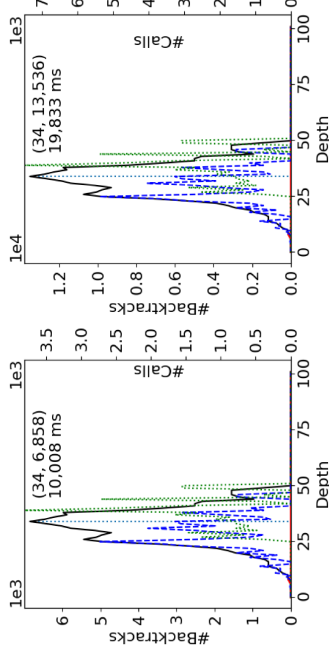


Fig. 5. mug100-25-3: GAC (left); constraint network (center); DPPC⁺ (right).

The inspection of the history of search, shown in Fig. 6a, reveals six regimes. Search generates a first peak at depth 34 (Regimes 1 and 2) before generating a second peak at depth 86 (Regimes 3 and 4). Then, this second peak grows larger, dwarfs the first, shifts to depth 61, and settles at depth 83. Further, Regime 1 reveals a peak of magnitude 2834 backtracks at depth 34, which corresponds to instantiating about one third of the variables in the problem. Regime 2



(a) History ($\delta=0.1$) of GAC, dom/wdeg shows six regimes



(b) History ($\delta=0.1$) of DPPC⁺, dom/wdeg shows a unique regime

Fig. 6. Histories of mug100-25-3 with GAC and DPPC⁺. Showing the middle BpD sample in each regime and, for reference, the final sample collected.

shows that search has overcome the initial bottleneck in the instance occurring at depth 34 with a magnitude of 4556 backtracks but is struggling again with a second bottleneck at depth 86 with a magnitude of 4624 backtracks. Regime 3 shows that the severity of the first bottleneck is dwarfed by a dramatic increase of the second bottleneck, which reaches 237705 backtracks at depth 86. As time progresses, the peak moves to shallower levels, to depth 61. Although the peak keeps moving to deeper levels, search struggles and is unable to conclude. Another noteworthy information is the time scale at which the regimes are identified. The first three regimes take place within the first five seconds of search whereas the last three regimes occur at larger time scales. It seems presumptuous to expect a human user to identify and isolate behavior occurring at such small time scales.

The observation of the two bottlenecks prompts us to examine the constraint network of the problem instance looking for some structural explanation for the non-termination of search. Indeed, we expect the graph to exhibit some particular configuration such as two overlapping cliques. The examination of the constraint graph, shown at the center of Fig. 5, refutes our two-cliques hypothesis but reveals the existence of a few large cycles as well as many cycles of size three or more. Previous work argued that the presence of cycles can be detrimental to the effectiveness of constraint propagation and showed how triangulation of the constraint network allows us to remedy the situation [43, 44, 47]. With this insight, we consider enforcing, during search, Partial Path Consistency (PPC) instead of GAC because the algorithm for PPC operates on existing triangles and on triangulated cycles of the constraint network [8]. Because PPC is too expensive to enforce during search, Woodward proposed a computationally competitive algorithm to enforce a relaxed version of Directional Partial Path Consistency (DPPC⁺) [45]. By enforcing a strong consistency along cycles, search is able to detect the insolvability of `mug100-25-3` and terminates in less than 20s. The corresponding history of search, shown in Fig. 6b, exhibits a unique regime.

Figure 5 (right) shows the BpD of search while enforcing DPPC⁺ on the `mug100-25-3` instance. This chart shows a peak of 13536 backtracks at depth 34. Table 1 compares the cost of search with GAC and with DPPC⁺. The sign ‘>’ indicates that search does not terminate within the allotted two hours.

Table 1. Performance of search (dom/wdeg) on `mug100-25-3`: GAC versus DPPC⁺

	GAC	DPPC ⁺
CPU time (sec)	>8082.4	19.8
#Nodes visited	851130992	288976
#Backtracks	697346084	236356
$\max_{d \in D}(\text{BT}(d, t_{max}))$	25953973	13536
$\arg \max_{d \in D}(\text{BT}(d, t_{max}))$	83	34

Above, we showed how the visualizations and regimes of search behavior allow us to form a hypothesis about the issues encountered by search and attempt an

alternative technique that is usually avoided because of its cost, culminating in successfully solving a difficult instance.

4 Visualizing Variable Ordering

Heuristics for variable ordering remain an active area of research in Constraint Programming. Currently, dom/wdeg [10] is considered the most effective general-purpose heuristic. When running experiments on benchmark problems, we encountered a puzzling case where search did not terminate when using dom/wdeg but safely terminated, determining inconsistency, when using dom/deg. We propose *variable-instantiation per depth* (VIpD) as a visualization to summarize the behavior of a variable ordering heuristic.

4.1 Variable-Instantiation per Depth (VIpD)

We denote by $I(v, d, t)$ the number of times variable v is instantiated at a given depth d from the beginning of search until time t . For a variable $v \in X$, we define the weighted depth:

$$d_w(v, t) = \frac{\sum_{d \in D} I(v, d, t) \cdot d}{\sum_{d \in D} I(v, d, t)} \quad (2)$$

We introduce the VIpD chart at time t as a two-dimensional heatmap of $I(\cdot, \cdot, t)$ using, as x -axis, the depth of search and, as y -axis, the variables of the CSP listed in their increasing $d_w(\cdot, t)$ values (ties broken using a lexicographical ordering of the names of the variables).

4.2 Illustrative Example

We try to solve the instance mug100-1-3 [50] of a graph coloring problem using GAC, but searches with dom/wdeg and with dom/deg fail to terminate after two hours of wall-clock time. In a next attempt, we maintain POAC during search; search with dom/wdeg did not terminate but, with dom/deg, it terminated after 47 min and determined that the instance is inconsistent. Table 2 compares the performance of POAC with dom/wdeg and dom/deg. The sign ‘>’ indicates that search did not terminate within two hours.

Table 2. Performance of search on mug100-1-3 using POAC

POAC	dom/wdeg	dom/deg
CPU time (sec)	>8085.4	2836.5
#Nodes visited	37811011	25178511
#Backtracks	32031950	14955444
$\max_{d \in D}(\text{BT}(d, t_{max}))$	1042601	1393204
$\arg \max_{d \in D}(\text{BT}(d, t_{max}))$	54	57

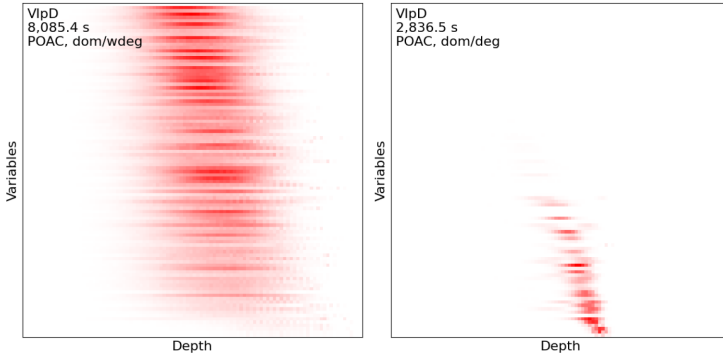


Fig. 7. Variable Instantiation per Depth (VIPD) on `mug100-1-3`. Left: POAC with `dom/wdeg` after two hours. Right: POAC with `dom/deg` after 47 min.

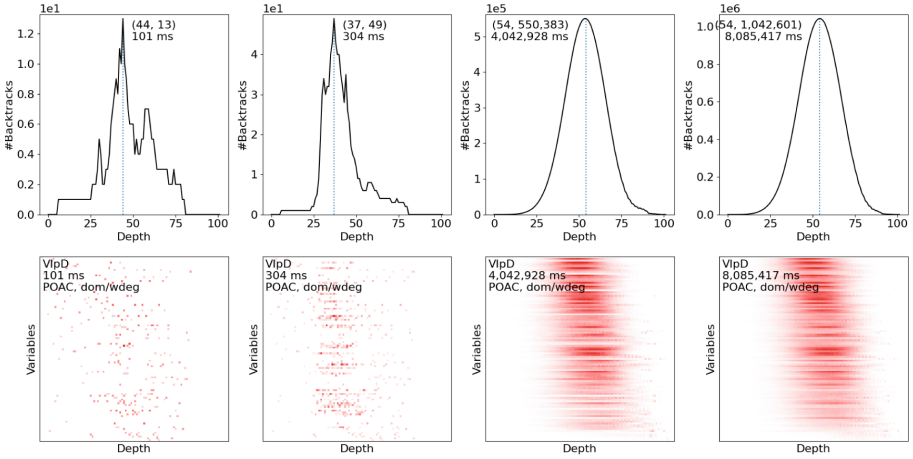
The comparison of the VIPD of POAC with `dom/wdeg` and that with `dom/deg` reveals an erratic behavior of `dom/wdeg` (Fig. 7). Indeed, `dom/wdeg`, which learns from domain wipeouts, is unable to determine the most conflicting variables on which it should focus. Instead, it continually instantiates a large number of variables over a wide range of depth levels in search. In contrast, `dom/deg` exhibits a more stable, less chaotic behavior. It focuses on the variables that yield inconsistency and is able to successfully terminate. The VIPD charts are also available as an Excel file where the user may analyze individual cells to examine their $I(\cdot, \cdot, t)$ values.

Comparing the histories alongside the VIPD charts (see Fig. 8), we see clearly that `dom/wdeg` is erratic and chaotic since the beginning of search whereas `dom/deg` is able to quickly focus on a relatively localized conflict to determine inconsistency of the problem instance. The peak of the BpD is sharper for `dom/deg` than for `dom/wdeg` and the VIPD shows a smaller set of variables is affected by the backtracking in `dom/deg` than in `dom/wdeg`. Future work could analyze this information to identify the source of conflict.

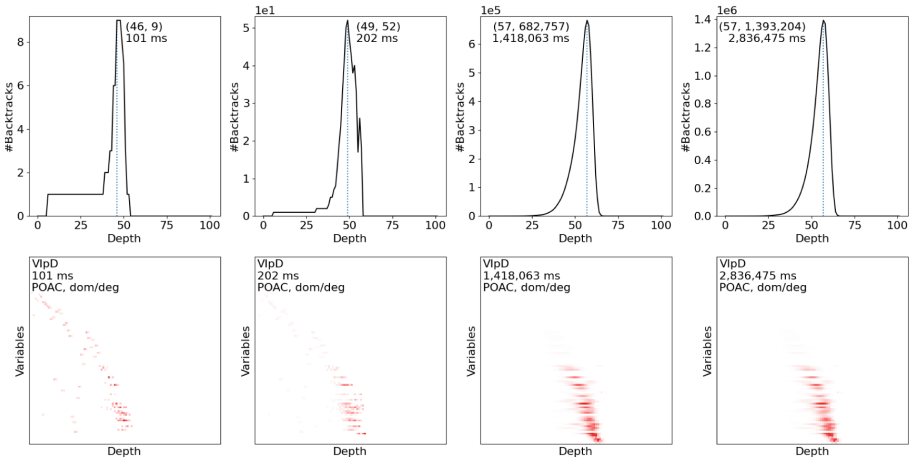
5 Discussion

We illustrated the usefulness of our approach in three case studies (summarized in Table 3). The first example (Sect. 3.3) compares two search histories to explain the operation and effectiveness of an adaptive technique (perhaps to a student or to a layperson). The same process can be used by a researcher or an application developer to explore the impact of, and adjust, the parameters of a new method. In the second example (Sect. 3.4), search fails because of a transient behavior at a tiny time scale detected by our automatic regime identification. Without this tool, the user may never notice the issue in order to explore effective solutions.

Beyond the scenarios and the metrics discussed above, we believe that building and comparing search histories are useful to explore, understand, and explain



(a) POAC with dom/wdeg after two hours.



(b) POAC with dom/deg after 47 minutes.

Fig. 8. History and VipD of POAC on mug100-1-3: dom/wdeg (top), dom/deg (bottom)

the impact, on the behavior and performance of search, of many advanced techniques such as no-good/clause learning, restart strategies, consistency algorithms, variable ordering heuristics, as well as the design of new such algorithms.

We strongly believe that our approach is beneficial in the context of existing constraint solvers such as Choco [12] and Gecode [35]. As stated above, preliminary tests on GAC and POAC [32] on Choco have shown similar BpD and CpD line-charts to those we see in k -way branching. Further, propagators in such solvers are often implemented on the individual constraints themselves, which

Table 3. Illustrative examples: the instances used and the corresponding search history

	aim-200-2-0-sat-1		mug100-25-3		mug100-1-3	
Consistency	GAC	PREPEAK ⁺ -POAC	GAC	DPPC ⁺	POAC	
Displayed in...	Fig. 4a	Fig. 4b	Figs. 5, 6a	Figs. 5, 6b	Figs. 7, 8a	Figs. 7, 8b
CPU (ms)	569633.2	60953.1	8082215.3	19833.0	8085417.6	2836475.7
Threshold (δ)	0.1	0.05	0.1	0.1	0.1	0.1
# Samples	5639	604	80014	197	79,345	28025
# Regimes	5	7	6	1	3	3
History	H1	H2	H3	H4	H5	H6

The histories with the durations of their regimes, listed chronologically:

- H1: [202 ms, 101 ms, 606 ms, 17982 ms, 550742 ms]
- H2: [101 ms, 202 ms, 101 ms, 1010 ms, 6370 ms, 16175 ms, 36994 ms]
- H3: [1515 ms, 606 ms, 4848 ms, 107067 ms, 423119 ms, 7545060 ms]
- H4: [19833 ms]
- H5: [101 ms, 304 ms, 8085012 ms]
- H6: [101 ms, 101 ms, 2836273 ms]

makes the accounting of the calls to various types of consistencies particularly well adapted.

The SAT community is another one that could benefit from our work. In SAT solvers, *inprocessing* (in the form of the application of the resolution rule) interleaves search and inference steps [21, 49]. Resolution is allocated a fixed percentage of the CPU time (e.g., 10%) and sometimes its effectiveness is monitored for early termination. We believe that inference should be targeted at the ‘areas’ where search is struggling rather than following a predetermined and fixed effort allocation. We claim that understanding where search struggles and how that struggle changes can be used to identify where inference is best invested.

While our work does not generate verbal explanations, we claim that the graphical tools directly ‘speak’ to a user’s intuitions.

6 Conclusions

In this paper, we presented a summarization technique and a visualization to allow the user to understand search behavior and performance on a given problem instance.

Currently, our approach provides a ‘post-mortem’ analysis of search, but our ultimate goal is to provide an ‘in-vivo’ analysis and allow the user to intervene in, and guide, the search process, trying alternatives and mixing strategies while observing their effects on the effectiveness of problem solving. A significantly more ambitious objective is to use the tools discussed in this paper as richer representations than mere numerical values to automatically guide search [31].

Future work includes the development of animation techniques based on the proposed visualizations.

References

1. Balafrej, A., Bessiere, C., Bouyakhf, E., Trombetta, G.: Adaptive singleton-based consistencies. In: Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI 2014), pp. 2601–2607 (2014)
2. Balafrej, A., Bessiere, C., Coletta, R., Bouyakhf, E.H.: Adaptive parameterized consistency. In: Schulte, C. (ed.) CP 2013. LNCS, vol. 8124, pp. 143–158. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40627-0_14
3. Balafrej, A., Bessière, C., Paparrizou, A.: Multi-armed bandits for adaptive constraint propagation. In: Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015), pp. 290–296 (2015)
4. Bennaceur, H., Affane, M.-S.: Partition-k-AC: an efficient filtering technique combining domain partition and arc consistency. In: Walsh, T. (ed.) CP 2001. LNCS, vol. 2239, pp. 560–564. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45578-7_39
5. Bessière, C., Régin, J.-C.: MAC and combined heuristics: two reasons to forsake FC (and CBJ?) on hard problems. In: Freuder, E.C. (ed.) CP 1996. LNCS, vol. 1118, pp. 61–75. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-61551-2_66
6. Bessière, C., Stergiou, K., Walsh, T.: Domain filtering consistencies for non-binary constraints. *Artif. Intell.* **172**, 800–822 (2008)
7. Bitner, J.R., Reingold, E.M.: Backtrack programming techniques. *Commun. ACM* **18**(11), 651–656 (1975). <http://doi.acm.org/10.1145/361219.361224>
8. Bliet, C., Sam-Haroud, D.: Path consistency for triangulated constraint graphs. In: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI 1999), pp. 456–461 (1999)
9. Borrett, J.E., Tsang, E.P., Walsh, N.R.: Adaptive constraint satisfaction: the quickest first principle. In: Proceedings of the Twelfth European Conference on Artificial Intelligence (ECAI 1996), pp. 160–164 (1996)
10. Boussemart, F., Hemery, F., Lecoutre, C., Sais, L.: Boosting systematic search by weighting constraints. In: Proceedings of the Seventeenth European Conference on Artificial Intelligence (ECAI 2006), pp. 146–150 (2004)
11. Carro, M., Hermenegildo, M.: Tools for constraint visualisation: the VIFID/TRIFID tool. In: Deransart, P., Hermenegildo, M.V., Maluszynski, J. (eds.) *Analysis and Visualization Tools for Constraint Programming*. LNCS, vol. 1870, pp. 253–272. Springer, Heidelberg (2000). https://doi.org/10.1007/10722311_11
12. Choco: An Open-Source Java Library for Constraint Programming. <https://choco-solver.org>
13. Debruyne, R., Bessière, C.: Some practicable filtering techniques for the constraint satisfaction problem. In: Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI 1997), pp. 412–417 (1997)
14. Defays, D.: An efficient algorithm for a complete link method. *Comput. J.* **20**(4), 364–366 (1977)
15. Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 61–75. Springer, Heidelberg (2005). https://doi.org/10.1007/11499107_5
16. Epstein, S.L., Freuder, E.C., Wallace, R.M., Li, X.: Learning propagation policies. In: Working Notes of the Second International Workshop on Constraint Propagation and Implementation, Held in Conjunction with CP-05, pp. 1–15 (2005)
17. Freuder, E.C., Elfe, C.D.: Neighborhood inverse consistency preprocessing. In: Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI 1996), pp. 202–208 (1996)

18. Freuder, E.C., Wallace, R.J.: Selective relaxation for constraint satisfaction problems. In: Proceedings of the IEEE Third International Conference on Tools with Artificial Intelligence (ICTAI 1991), pp. 332–339 (1991)
19. Ghoniem, M., Cambazard, H., Fekete, J.D., Jussien, N.: Peeking in solver strategies using explanations visualization of dynamic graphs for constraint programming. In: Proceedings of the 2005 ACM Symposium on Software Visualization (SoftVis 2005), pp. 27–36 (2005)
20. Hayes, P.J.: The second Naive physics manifesto. In: Readings in Qualitative Reasoning About Physical Systems, pp. 46–63. Morgan Kaufmann (1990)
21. Järvisalo, M., Heule, M.J.H., Biere, A.: Inprocessing rules. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. LNCS (LNAI), vol. 7364, pp. 355–370. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31365-3_28
22. Karakashian, S., Woodward, R., Reeson, C., Choueiry, B.Y., Bessiere, C.: A first practical algorithm for high levels of relational consistency. In: Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2010), pp. 101–107 (2010)
23. Kuipers, B.: Qualitative Reasoning - Modeling and Simulation with Incomplete Knowledge. MIT Press, Cambridge (1994)
24. Kullback, S., Leibler, R.A.: On information and sufficiency. *Ann. Math. Stat.* **22**(1), 79–86 (1951)
25. Lagerkvist, M.Z., Schulte, C.: Propagator groups. In: Gent, I.P. (ed.) CP 2009. LNCS, vol. 5732, pp. 524–538. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04244-7_42
26. Mackworth, A.K.: On reading sketch maps. In: Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI 1977), pp. 598–606 (1977)
27. Manning, C.D., Raghavan, P., Schütze, H.: Introduction to Information Retrieval. Cambridge University Press, Cambridge (2008)
28. Nielsen, F.: Introduction to HPC with MPI for Data Science. UTCS. Springer, Cham (2016). <https://doi.org/10.1007/978-3-319-21903-5>
29. Paparrizou, A., Stergiou, K.: Evaluating simple fully automated heuristics for adaptive constraint propagation. In: Proceedings of the IEEE Twenty-Fourth International Conference on Tools with Artificial Intelligence (ICTAI 2012), pp. 880–885 (2012)
30. Paparrizou, A., Stergiou, K.: On neighborhood singleton consistencies. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI 2017), pp. 736–742 (2017)
31. Petit, T.: Personal communication (2020)
32. Phan, K.: Extending the Choco constraint-solver with high-level consistency with evaluation on the nonogram puzzle. Undergraduate thesis. Department of Computer Science and Engineering, University of Nebraska-Lincoln (2020)
33. Samaras, N., Stergiou, K.: Binary encodings of non-binary constraint satisfaction problems: algorithms and experimental results. *JAIR* **24**, 641–684 (2005)
34. Schulte, C.: Oz explorer: a visual constraint programming tool. In: Kuchen, H., Doaitse Swierstra, S. (eds.) PLILP 1996. LNCS, vol. 1140, pp. 477–478. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-61756-6_108
35. Schulte, C., Tack, G., Lagerkvist, M.Z.: Modeling and Programming with Gecode (2015). <http://www.gecode.org/doc-latest/MPG.pdf>
36. SciPy.org: `scipy.cluster.hierarchy.linkage`. <https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.linkage.html>. Accessed 26 May 2020
37. Shishmarev, M., Mears, C., Tack, G., de la Banda, M.G.: Visual search tree profiling. *Constraints* **21**(1), 77–94 (2016)

38. Simonis, H., Aggoun, A.: Search-tree visualisation. In: Deransart, P., Hermenegildo, M.V., Mahuszynski, J. (eds.) *Analysis and Visualization Tools for Constraint Programming*. LNCS, vol. 1870, pp. 191–208. Springer, Heidelberg (2000). https://doi.org/10.1007/10722311_8
39. Simonis, H., Davern, P., Feldman, J., Mehta, D., Quesada, L., Carlsson, M.: A generic visualization platform for CP. In: Cohen, D. (ed.) *CP 2010*. LNCS, vol. 6308, pp. 460–474. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15396-9_37
40. Simonis, H., O’Sullivan, B.: Almost square packing. In: Achterberg, T., Beck, J.C. (eds.) *CPAIOR 2011*. LNCS, vol. 6697, pp. 196–209. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21311-3_19
41. Stergiou, K.: Heuristics for dynamically adapting propagation. In: *Proceedings of the Eighteenth European Conference on Artificial Intelligence (ECAI 2008)*, pp. 485–489 (2008)
42. Wallace, R.J.: SAC and neighbourhood SAC. *AI Commun.* **28**(2), 345–364 (2015)
43. Woodward, R., Karakashian, S., Choueiry, B.Y., Bessiere, C.: Solving difficult CSPs with relational neighborhood inverse consistency. In: *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2011)*, pp. 112–119 (2011)
44. Woodward, R.J.: Relational neighborhood inverse consistency for constraint satisfaction: a structure-based approach for adjusting consistency and managing propagation. Master’s thesis, Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, NE, December 2011
45. Woodward, R.J.: Higher-level consistencies: where, when, and how much. Ph.D. thesis, University of Nebraska-Lincoln, December 2018
46. Woodward, R.J., Choueiry, B.Y., Bessiere, C.: A reactive strategy for high-level consistency during search. In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI 2018)*, pp. 1390–1397 (2018)
47. Woodward, R.J., Karakashian, S., Choueiry, B.Y., Bessiere, C.: Revisiting neighborhood inverse consistency on binary CSPs. In: Milano, M. (ed.) *CP 2012*. LNCS, pp. 688–703. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33558-7_50
48. Woodward, R.J., Schneider, A., Choueiry, B.Y., Bessiere, C.: Adaptive parameterized consistency for non-binary CSPs by counting supports. In: O’Sullivan, B. (ed.) *CP 2014*. LNCS, vol. 8656, pp. 755–764. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10428-7_54
49. Wotzlaw, A., van der Grinten, A., Speckenmeyer, E.: Effectiveness of pre- and inprocessing for CDCL-based SAT solving. Technical report, Institut für Informatik, Universität zu Köln, Germany (2013). <http://arxiv.org/abs/1310.4756>
50. XCSP3: Benchmark Series in XCSP3 Format. <http://xcsp.org/series>. Accessed 24 May 2020