A Qualitative Analysis of Search Behavior: A Visual Approach

Ian Howell,[1,2] **Robert Woodward,**[1] **Berthe Y. Choueiry,**[1] **Hongfeng Yu**[2]
[1]Constraint Systems Laboratory, University of Nebraska-Lincoln, USA
[2]Visualization Laboratory, University of Nebraska-Lincoln, USA
{ihowell|rwoodwar|choueiry|yu}@cse.unl.edu

Abstract

In this paper, we propose visualizations that track the progress and behavior of backtrack search when solving an instance of a Constraint Satisfaction Problem. The goal of our visualizations is to provide insight in the difficulty of the particular instance at hand as well as in the effectiveness of various strategies for enforcing consistency during search. To this end, our visualizations track the number of backtracks and the number of calls to a consistency algorithm per depth of the search tree and superimpose the two measures while distinguishing effective and wasteful consistency calls. Using these numbers, we automatically derive qualitative regimes summarizing the evolution of the search process over time. We show that these instruments provide new insights into the performance of search on a particular instance and into the effectiveness of the various strategies for enforcing consistency during search. We present WORMHOLE, an extendable, solver-agnostic visualization tool that we built as a platform to implement these mechanisms. Currently WORMHOLE provides a 'post-mortem' analysis of search, but our ultimate goal is to provide an 'in-vivo' analysis and allow the user to intervene and guide the search process.

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 currently the only sound and complete algorithm 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 and new strategies for dynamically switching between consistency algorithms are being investigated [Borrett *et al.*, 1996; Freuder and Wallace, 1991; Epstein *et al.*, 2005; Eén and Biere, 2005; Stergiou, 2008; Lagerkvist and Schulte, 2009; Paparrizou and Stergiou, 2012; Balafrej *et al.*, 2013; Balafrej *et al.*, 2014; Woodward *et al.*, 2014; Wallace, 2015; Balafrej *et al.*, 2015; Woodward *et al.*, 2011; Paparrizou and Stergiou, 2017]. While consistency algorithms can dramatically reduce the size of the search space, their impact on the CPU cost of search can vary widely. It is likely poorly understood but certainly difficult to control.

In this paper, we propose three tools and their visualizations as a first step towards graphically summarizing and explaining the performance of search:

1. We track the number of *backtracks per depth* (BpD) at each level of search to understand where and how search struggles and where it smoothly proceeds.
2. To understand the impact of enforcing a given consistency property, we track the number of *calls to the consistency algorithm per depth* (CpD) in the search tree. Further, we split these calls into three categories: those that yield domain *wipeout* (i.e., detect inconsistency), those that effectively *filter* domains without detecting a dead-end, and those that yield *no filtering* (i.e., constitute a wasted effort).
3. To summarize the behavior of search over time, we structure the evolution of the above two measurements into qualitatively equivalent *regimes* by using three criteria that characterize growth rate and shape evolution.

We implement the above mechanisms in a visualization system called WORMHOLE that allows users to interactively examine the performance of search and graphically compare the performance of various algorithms on the same instance. WORMHOLE is a first step towards building a library of visualization tools aimed at providing an insight into the strengths and weaknesses of current algorithms for solving CSPs.

While our system does not generate verbal explanations, we claim that the graphical tools that it provides directly 'speak' to a user's intuitions. Our long-term goal is to allow users to actively intervene in the search process itself, trying alternatives and mixing strategies while observing their effects on the effectiveness of problem solving.

This paper is structured as follows. Section 2 recalls background information and reviews the relevant literature. Section 3 describes the proposed visualizations. Section 4 discusses how search evolution is organized into regimes of equivalent behaviors to facilitate user understanding. Section 5 is a case study showing how our visualizations allow

us to compare and understand the performance of three strategies for solving the same CSP instance. Section 6 briefly describes the architecture of WORMHOLE. Finally, Section 7 concludes the paper.

2 Background

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. A solution to a CSP is an assignment of values to variables such that all constraints are simultaneously satisfied.

Backtrack (BT) search is currently the only sound and complete algorithm for solving CSPs. In order to reduce thrashing, which is the main malady of search, it is common practice to enforce a given consistency property after every variable instantiation. This procedure 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 [Freuder and Elfe, 1996; Debruyne and Bessiere, 1997; Samaras and Stergiou, 2005; Bessière *et al.*, 2008], leading to a tradeoff between the search effort and the time for enforcing consistency. We claim that our visualizations are insightful tools for understanding such a tradeoff.

Prior research on search visualization has appeared in the Constraint Programming literature, which typically focuses on the 2-way branching search scheme in contrast to the k-way branching scheme adopted by the CSP community. 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, and event propagation at each node of the search tree [Simonis and Aggoun, 2000; Carro and Hermenegildo, 2000]. Many useful methodologies from the DiSCiPl project are implemented in CP-Viz [Simonis *et al.*, 2010] and other works [Shishmarev *et al.*, 2016]. The OZ Explorer displays the search tree allowing the user to access detailed information about the node at each tree node and to collapse and expand failing trees for closer examination [Schulte, 1996]. This work is currently incorporated into Gecode's Gist [Schulte *et al.*, 2015]. The above approaches focus on exploring the search tree (as well as a problem's components) while our work proposes particular projections (i.e., views, summaries) of the data *reflecting* (i.e., compiling) the cost and the effectiveness of both search and enforcing consistency. We believe that these visualizations are orthogonal and complementary.

Tracking search effort by depth was first proposed by Epstein *et al.* [2005] for the number of constraint checks and values removed per search and by Simonis *et al.* [2010] in CP-Viz for the number of nodes visited (also used for solving a packing problem [Simonis and O'Sullivan, 2011]). We claim that the number of constraint checks, values removed, and nodes visited are not accurate measures of the thrashing effort. Indeed, the number of constraint checks varies with the degree of the variables. The number of values removed and nodes visited vary with the size of the domain. In contrast, we claim that the number of backtracks per search depth (BpD) provides a more faithful representation of the thrashing effort, which is exactly the aspect of search that we aim to characterize.

Recently, techniques have appeared in Constraint Processing for dynamically choosing between a set of consistency properties based on the CPU time spent on exploring a given subtree [Balafrej *et al.*, 2015]. We claim that we better track the effectiveness of such decisions by following the number of backtracks per depth (BpD) and the number of consistency calls per depth (CpD) rather than the CPU time of searching a given subtree.

In the SAT community, inprocessing (in the form of the application of the resolution rule) interleaves search and inference steps [Järvisalo *et al.*, 2012; Wotzlaw *et al.*, 2013]. 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 the visualization provided by WORMHOLE can be used to identify where such an effort is best invested.

3 Analyzing Search Effectiveness

The BpD chart reflects various aspects of search effectiveness as we illustrate with an example. We consider the instance of a coloring problem called 4-insertions-3-3 of the benchmark graphColoring-k-insertions.[1] We find one solution of this instance using backtrack search while maintaining one of two consistency properties, namely GAC [Mackworth, 1977] and POAC [Bennaceur and Affane, 2001], and using the dom/wdeg ordering heuristic [Boussemart *et al.*, 2004]. The definitions of GAC and POAC are beyond the scope of this paper: it suffices to say that an algorithm that enforces GAC is generally quick but does little filtering while a POAC algorithm is typically (very) costly but can prune larger subtrees of the search space. In fact, enforcing POAC during search is so costly that, in practice, we use an adaptive version called APOAC [Balafrej *et al.*, 2014].

Table 1 reports the cost of the two search algorithms in terms of their CPU time, the number of nodes visited by search, the number of backtracks, and the maximum value reached by the respective BpD curves, which are shown in Figure 1. As we can see from Table 1, it is clear that our 'in-

Table 1: Cost of GAC and APOAC on 4-insertions-3-3. Note that GAC times out. In WORMHOLE, this data is displayed in a panel.

Algorithm	GAC	APOAC
CPU Time (sec)	>8,099.9	2,981.9
# Nodes Visited	348,276,252	17,078,644
# Backtracks	252,570,526	13,416,093
$\max_{\text{BpD}}$	15,863,603	693,829

[1] Source: `www.cril.univ-artois.fr/~lecoutre/benchmarks.html`.

vestment' in a stronger consistency algorithm is worthwhile because APOAC solves the instance in about 50 minutes while GAC does not terminate. Comparing the BpD charts of GAC (left) and APOAC (right) in Figure 1, we see that GAC thrashes around depth 50 with $\max_{\text{BpD}} = 15,863,603$ backtrack at depth 53. APOAC, which enforces a strictly stronger consistency throughout search, limits the severity of thrashing to only $\max_{\text{BpD}} = 693,829$ backtracks at depth 40. By detecting and pruning inconsistencies at the shallower search levels, APOAC solves the problem while GAC fails.

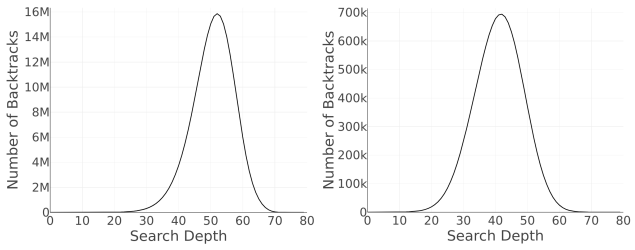

Figure 1: BpD for GAC (left) and APOAC (right) on instance 4-INSERTIONS-3-3.

Naturally, investing in a high-level consistency (HLC) is not always worthwhile. On easy instances, the cost of APOAC is can be an overkill. To examine the effectiveness of enforcing an HLC, we propose another visualization, which superimposes, to the BpD chart, the chart reporting the number of calls to POAC per depth (CpD). Figure 2 (left) unsurprisingly shows that the BpD and the CpD charts of APOAC largely overlap in shape (modulo their respective ranges shown on both sides of the chart), which is explained by the fact that APOAC is called at every variable instantiation during search. In other dynamic strategies where two or more levels of consistency are enforced, the CpD would allow us to differentiate between the impact of each consistency algorithm.

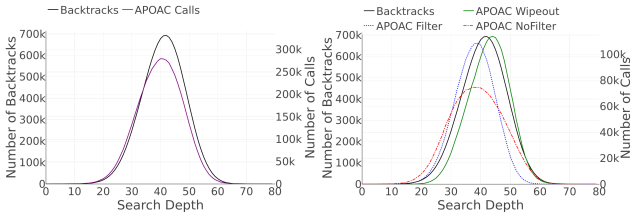

Figure 2: Superimposing CpD onto BpD for APOAC on 4-INSERTIONS-3-3: Showing CpD total (left) and detailed as wipeout, filtering, no filtering (right)

We propose a third visualization by splitting more finely the CpD into three categories depending on whether calls to POAC resulted in (1) in a domain wipeout, (2) filtering but no wipe out, and (3) no filtering. In Figure 2 (right), these three CpDs are shown in green (the most effective POAC calls, which cause backtracking), blue (which prune inconsistent subtrees, reduce the search space, but cannot detect inconsistencies), and red (which are wasteful calls to POAC resulting in no filtering). In the case of our particular example, we can see that the wasteful calls to POAC are fewer, and not as much time is wasted at deeper levels. This realization fully explains the ability of APOAC to prevent search from thrashing at deeper search levels and its effectiveness in solving this difficult instance.

4 Analyzing Search Evolution

The visualizations discussed in Section 3 provide interesting information at particular snapshots in time and at the end of search. However, in practice, search can last from a few milliseconds in duration to hours (before timeout). Thus, it is not practical, sometimes impossible, to examine the *evolution* of search from beginning to end. In order to help the user build an understanding of the evolution of search over time, gain insight in the difficulty of the problem at hand, perhaps even detect critical transitions in search behavior, we propose to automatically and dynamically organize the evolution of search over time in terms of a *history of successive episodes that are qualitatively meaningful*,[2] where each episode is a collection of equivalent behaviors.

We propose a two-step procedure to build such histories. First, we partition the time duration of the entire search into time intervals based on some criterion of equivalent behavior. We call these time intervals *regimes* [Kuipers, 1994]. Next, we provide a mechanism to dilate the time duration (i.e., by compressing or stretching it) of each regime, then concatenate them into a continuous animation of the history of the search.

Definition 1 (Regime, History) *Given a search procedure of total duration T, a given behavior function B and an equivalence relation $\sim$, a regime R_i is a contiguous interval of time $[s_i, e_i] \subseteq T$ during which the search features defining the behavior are qualitatively equivalent by some metric. The search history is a sequence of such regimes.*

$$
\begin{aligned}
H &= \langle R_1, \ldots, R_n \rangle \\
R_i &= \langle B_i, [s_i, e_i] \rangle \text{ where } [s_i, e_e] \subseteq T, \\
&\quad \forall t_i, t_j \in [s_i, e_e] \rightarrow B(t_i) \sim B_(t_j) \\
T &= \bigcup_{i=1}^{n} [s_i, e_i]
\end{aligned}
$$

While the set of desirable and useful behaviors and search features may be limitless, we integrate in WORMHOLE the following ones:

1. BASIC: Using the maximum value of BpD, $\max_{\text{BpD}}$, we partition the duration of search into a number of k regimes (k determined by the user), where the largest BpD value exceeds another k^{th} fraction of the value of $\max_{\text{BpD}}$.
2. GROWTH: From the beginning of search, we generate a new regime each time the largest BpD value increases by 10%.
3. SHAPE: From the start of search, we compute the Shannon Entropy of the derivatives over depth of the BpD to represent the relative shape of BpD curve [Rényi, 1961]. We start a new regime when this value changes by 20%.

[2] Our use of the term 'history' is in compliance with its initial meaning as proposed by Hayes [Hayes, 1990].

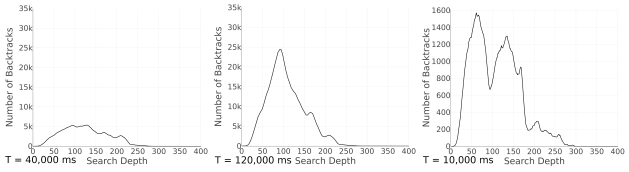

Figure 3: BASIC regimes of GAC on pseudo-aim-200-1-6-4 (cropped right-edges for space)

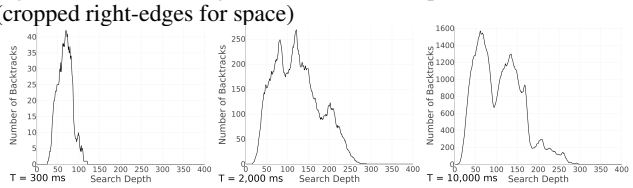

Figure 4: SHAPE regime of GAC on pseudo-aim-200-1-6-4 (cropped right-edges for space)

Of the above three regimes, SHAPE can recognize the most interesting changes in the BpD because it recognizes the change of the depth where thrashing occurs. Figures 3 and 4 show two regime progressions of GAC on the pseudo-aim-200-1-6-4 instance.[3] Figure 3 emphasizes the growth over time, while Figure 4 highlights the change in shape of the BpD.

We generate animations (of a user-specified duration) by applying time dilation on the histories generated with any of the three above behaviors. Currently, WORMHOLE implements two time-dilation methods, namely, EQUAL and PROPORTIONAL. EQUAL assigns to each regime an equal amount of time, while PROPORTIONAL assigns to each regime an animation time that is proportional to the regime's duration in search. In addition, WORMHOLE allows the user to directly modify the percentage of time that each regime takes independently of the above two dilation methods.

5 Case Study: PREPEAK$^+$

In this section, we use WORMHOLE to understand and compare the behavior of PREPEAK$^+$, a new reactive strategy for enforcing high level consistency during search [Woodward *et al.*, 2018]. To this end, we solve the CSP instance pseudo-aim-200-1-6-4 studied in Section 4 with backtrack search under three settings: (1) maintaining GAC, (2) with APOAC, and (3) PREPEAK$^+$. PREPEAK$^+$ is conservative in that it primarily enforces GAC. However, it triggers an HLC, such as POAC, when the number of backtracks per depth (BpD) reaches a given threshold value θ but only when search backtracks to levels shallower than the depth where the threshold is met. PREPEAK$^+$ keeps firing the HLC as long as the BpD at the considered depth is smaller than θ. Furthermore, every time it backtracks, PREPEAK$^+$ updates the values of θ by reducing it or increasing it according to three geometric laws depending on whether the HLC yields wipeout (i.e., it is effective), filters the search space, or yields no filtering (i.e., the HLC calls are wasteful). WORMHOLE reports the costs of three search algorithms as shown in Table 2.

Figure 5 shows the BpD for GAC at the end of search.

[3]Instance pseudo-aim-200-1-6-4 of the benchmark pseudo-aim from `www.cril.univ-artois.fr/~lecoutre/benchmarks.html`.

Table 2: Solving instance pseudo-aim-200-1-6-4 with GAC, APOAC, PREPEAK$^+$

	GAC	APOAC	PREPEAK$^+$
CPU time (sec)	185,045	66,816	17,836
#NV	3,978,074	47,457	284,289
$\max_{\text{BpD}}$	34,023	407	2,421
#HLC calls		7,739	228

This curve exhibits a peak around depth 100 with $\max_{\text{BpD}} = 34,023$ showing that GAC is too weak to filter out bad values: it spends much of its time thrashing around this depth level.

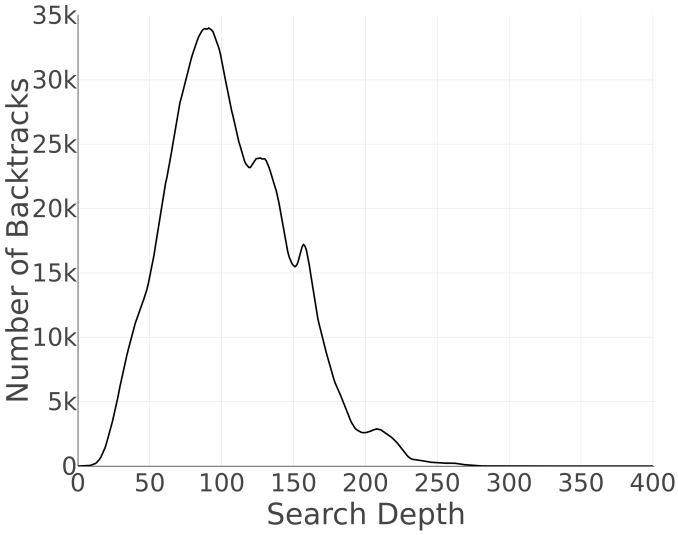

Figure 5: BpD and CpD of GAC on pseudo-aim-200-1-6-4

Figure 6 shows the BpD (black) and CpD (colored) curves for APOAC. Examining the BpD curve, we realize that

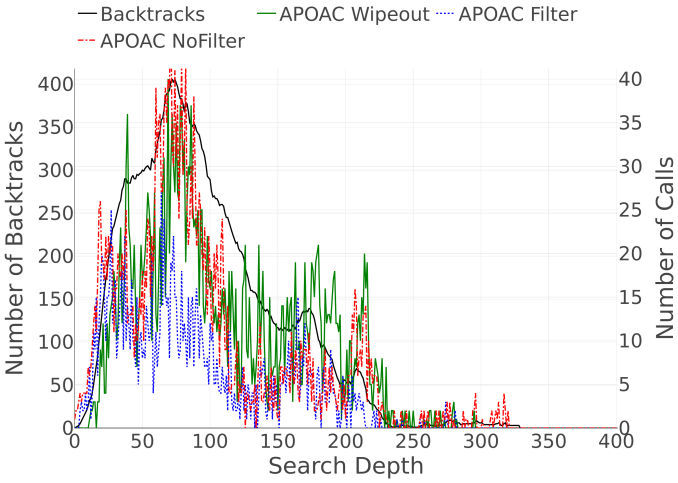

Figure 6: BpD (black) and CpD's (colored) of APOAC on pseudo-aim-200-1-6-4

APOAC so effectively prunes the 'bad subtrees' from the

search space that it dramatically reduces $\max_{\text{BpD}}$ down to 407 and the location of peak to around depth 75. We see that this instance benefits from enforcing an HLC such as POAC with a clear benefit on the CPU time (which is reduced by one order of magnitude from GAC). However, by observing the colored curves in Figure 6, we notice that the number of calls to POAC that are ineffective (red curve) are of the same order as those that yield wipeout (green curve). The detailed CpD curves hint to some savings that could be further obtained could one cancel the wasteful calls to POAC.

Figure 7 shows the BpD (black) and CpD (colored) curves for PREPEAK$^+$. PREPEAK$^+$ is conservative in that it calls

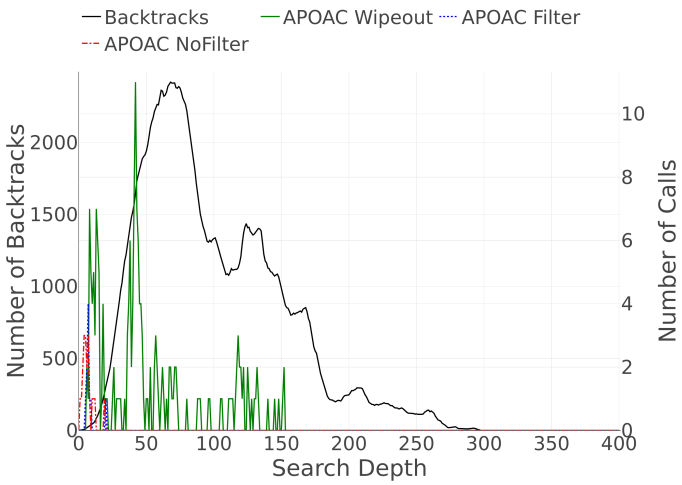

Figure 7: BpD (black) and CpD's (colored) of PREPEAK$^+$ on pseudo-aim-200-1-6-4

an HLC only when search thrashes, justifying the cost of a stronger but more costly consistency algorithm. Indeed, we observe that the peak value of BpD is smaller than for GAC but greater than for APOAC (2,421 versus 34,023 and 407, respectively). However, examining the detailed CpD curves shows that advantage of PREPEAK$^+$: Indeed, the wasteful calls to POAC (red) are almost eliminated and the total calls to POAC are reduced down to 228 for PREPEAK$^+$ from 7,739 for APOAC. This economy in the calls to POAC is immediately translated by the reduction of the CPU time. Thus, despite the fact that PREPEAK$^+$ explores a larger search tree than APOAC (see number of nodes visited) because it does not call the HLC at each variable instantiation, it effectively reacts to thrashing, calling the HLC only when it is needed, but spontaneously reverting to GAC otherwise.

This example illustrates the pertinence of the tools provided by WORMHOLE in visually explaining the behavior of search and the benefits of PREPEAK$^+$.

6 Architecture

Figure 8 shows the architecture of WORMHOLE. A constraint solver exports data to a generic, JSON-based log file that records the differences in tracked values over time (States 1-2). These logs are then loaded by the user into WORMHOLE, through (State 3), and parsed (State 4), into various data structures. At this point, the timeline data is pregenerated for

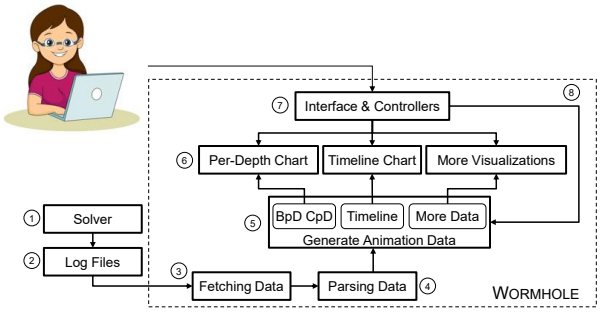

Figure 8: Architecture of WORMHOLE

immediate use while the BpD and CpD data are cached for further data generation (State 5). Timeline data is then sent immediately to the timeline chart and displayed to the user (State 6). Through the use of control panel interaction the user may generate new animations, select specific time information, or control their playback (State 7). The user input controller then calls (State 8) for the re-generation of data (State 5) which updates the animation data and sent to the user's view (State 6).

We designed WORMHOLE using web technologies of HTML, JavaScript, and CSS, based on the React-Redux model [Walke, ; Abramov and Clark,]. In this framework, React manages view rendering and user interaction, while Redux manages the application state. While standard web-application systems work adequately for logs with limited data (10's of MB), much search data is too large (100's of MB) to be handled by such systems. For the sake of performance, we enhance the system as follows.

JavaScript is a single-threaded language because it is based on an event-loop paradigm, which prevents the user from interacting with the application during periods of heavy computation. In WORMHOLE, we use Web Workers, a modern feature of the browser, to prevent blocking: it offloads most of the processing to a separate process. In this separate process, we perform initial data parsing, animation pre-fetching, and time-specific frame loading, which all are processor-intensive tasks.

To accelerate frame loading during animation, we pre-fetch all animation data in the data process and transfer it using JavaScript Transferables, which are data structures for rapidly transferring large data sets between web workers and the main thread [Mozilla and individual contributors,]. Further, when parsing the log, we perform regular sampling on the BpD and CpD data in order to reduce the work needed to calculate specific time frames because only data differences over time are stored in memory. When users request BpD and CpD data for a specific time, we perform nearest neighbor interpolation, only accepting samples of time not greater than the request before calculating with the rest of the difference data.

With the increasing size of the backtrack data, it is not viable to interactively visualize the entire backtrack data with limited memory size and rendering capacity. To address this issue, we apply a simple multi-resolution modeling technique that samples the backtrack data-set across multiple resolu-

tions and, for each resolution, partitions the data-set into blocks, each containing 1024 data points [Wynn *et al.*, 1997]. When users zoom on this chart to different resolutions, the blocks that have a resolution at least as great as the user's view and include points visible to the user are displayed on the chart. This optimization drastically increases performance on large searches from 0.2 frames per second (fps) to 20 fps.

7 Conclusions

WORMHOLE offers a number of new visualization and animation techniques that allow the user to explore and understand the behavior of backtrack search and compare the performance of different algorithms. We are currently applying our techniques to Choco, a popular constraint solver that uses binary search (2-way branching) while exploring new functionalities.

Acknowledgments

This research is supported by NSF Grant No. RI-1619344 and NSF CAREER Award No. III-1652846. Robert Woodward was supported by an NSF GRF Grant No. 1041000 and a Chateaubriand Fellowship. The experiments were completed utilizing the Holland Computing Center of the University of Nebraska, which receives support from the Nebraska Research Initiative.

References

[Abramov and Clark,] Dan Abramov and Andrew Clark. Redux. `https://redux.js.org/`. Accessed: 2017-04-21.

[Balafrej *et al.*, 2013] Amine Balafrej, Christian Bessiere, Remi Coletta, and El-Houssine Bouyakhf. Adaptive Parameterized Consistency. In *Proc. of CP 2013*, volume 8124 of *LNCS*, pages 143–158. Springer, 2013.

[Balafrej *et al.*, 2014] Amine Balafrej, Christian Bessiere, El-Houssine Bouyakhf, and Gilles Trombettoni. Adaptive Singleton-Based Consistencies. In *Proc. of AAAI 2014*, pages 2601–2607, 2014.

[Balafrej *et al.*, 2015] Amine Balafrej, Christian Bessière, and Anastasia Paparrizou. Multi-Armed Bandits for Adaptive Constraint Propagation. In *Proc. of IJCAI 2015*, pages 290–296, 2015.

[Bennaceur and Affane, 2001] Hachemi Bennaceur and Mohamed-Salah Affane. Partition-k-AC: An Efficient Filtering Technique Combining Domain Partition and Arc Consistency. In *Proceedings of* 7^{th} *International Conference on Principle and Practice of Constraint Programming (CP'01)*, volume 2239 of *LNCS*, pages 560–564. Springer, 2001.

[Bessière *et al.*, 2008] Christian Bessière, Kostas Stergiou, and Toby Walsh. Domain Filtering Consistencies for Non-Binary Constraints. *Artificial Intelligence*, 172:800–822, 2008.

[Borrett *et al.*, 1996] James E. Borrett, Edward P.K. Tsang, and Natasha R. Walsh. Adaptive Constraint Satisfaction: The Quickest First Principle. In *Proc. of ECAI 1996*, pages 160–164, 1996.

[Boussemart *et al.*, 2004] Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. Boosting Systematic Search by Weighting Constraints. In *Proc. of ECAI 2004*, pages 146–150, 2004.

[Carro and Hermenegildo, 2000] Manuel Carro and Manuel Hermenegildo. Tools for Constraint Visualisation: The VIFID/TRIFID Tool. In *Analysis and Visualization Tools for Constraint Programming: Constraint Debugging*, volume 1870 of *Lecture Notes in Computer Science*, pages 253–272. Springer, 2000.

[Debruyne and Bessiere, 1997] Romuald Debruyne and Christian Bessiere. Some Practicable Filtering Techniques for the Constraint Satisfaction Problem. In *Proc. of IJCAI 1997*, pages 412–417, 1997.

[Eén and Biere, 2005] Niklas Eén and Armin Biere. Effective Preprocessing in SAT Through Variable and Clause Elimination. In *Proc. of SAT 2005*, volume 3569 of *LNCS*, pages 61–75. Springer, 2005.

[Epstein *et al.*, 2005] Susan L. Epstein, Eugene C. Freuder, Richard M. Wallace, and Xingjian Li. Learning Propagation Policies. In *International Workshop on Constraint Propagation and Implementation*, pages 1–15, 2005.

[Freuder and Elfe, 1996] Eugene C. Freuder and Charles D. Elfe. Neighborhood Inverse Consistency Preprocessing. In *Proc. of AAAI 1996*, pages 202–208, 1996.

[Freuder and Wallace, 1991] Eugene C. Freuder and Richard J. Wallace. Selective Relaxation For Constraint Satisfaction Problems. In *Proc. of ICTAI 1991*, pages 332–339, 1991.

[Hayes, 1990] Patrick J. Hayes. *Readings in Qualitative Reasoning About Physical Systems*, chapter The Second Naive Physics Manifesto, pages 46–63. Morgan Kaufmann, 1990.

[Järvisalo *et al.*, 2012] Matti Järvisalo, Marijn J. H. Heule, and Armin Biere. Inprocessing Rules. In *Proc. of IJCAR 2012*, volume 7364 of *LNCS*, pages 355–370. Springer, 2012.

[Kuipers, 1994] Benjamin Kuipers. *Qualitative Reasoning - Modeling and Simulation with Incomplete Knowledge*. MIT Press, 1994.

[Lagerkvist and Schulte, 2009] Mikael Z. Lagerkvist and Christian Schulte. Propagator Groups. In *Proceedings of* 5^{th} *International Conference on Principle and Practice of Constraint Programming (CP'99)*, volume 5732 of *LNCS*, pages 524–538. Springer, 2009.

[Mackworth, 1977] Alan K. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8:99–118, 1977.

[Mozilla and individual contributors,] Mozilla and individual contributors. MDN Web Docs: Transferable. `https://developer.mozilla.org/en-US/docs/Web/API/Transferable`. Accessed: 2017-04-21.

[Paparrizou and Stergiou, 2012] Anastasia Paparrizou and Kostas Stergiou. Evaluating Simple Fully Automated Heuristics for Adaptive Constraint Propagation. In *Proc. of ICTAI 2012*, pages 880–885, 2012.

[Paparrizou and Stergiou, 2017] Anastasia Paparrizou and Kostas Stergiou. On Neighborhood Singleton Consistencies. In *Proc. of IJCAI 2017*, pages 736–742, 2017.

[Rényi, 1961] Alfréd Rényi. On Measures of Entropy and Information. Technical report, Hungarian Academy of Sciences, Budapest, Hungary, 1961.

[Samaras and Stergiou, 2005] Nikos Samaras and Kostas Stergiou. Binary Encodings of Non-binary Constraint Satisfaction Problems: Algorithms and Experimental Results. *JAIR*, 24:641–684, 2005.

[Schulte *et al.*, 2015] Christian Schulte, Guido Tack, and Mikael Z. Lagerkvist. *Modeling and programming with Gecode*, 2015.

[Schulte, 1996] Christian Schulte. Oz Explorer: A visual constraint programming tool. In *International Symposium on Programming Language Implementation and Logic Programming*, pages 477–478. Springer, 1996.

[Shishmarev *et al.*, 2016] Maxim Shishmarev, Christopher Mears, Guido Tack, and Maria Garcia de la Banda. Visual Search Tree Profiling. *Constraints*, 21(1):77–94, 2016.

[Simonis and Aggoun, 2000] Helmut Simonis and Abder Aggoun. Search-Tree Visualisation. In *Analysis and Visualization Tools for Constraint Programming: Constraint Debugging*, volume 1870 of *Lecture Notes in Computer Science*, pages 191–208. Springer, 2000.

[Simonis and O'Sullivan, 2011] Helmut Simonis and Barry O'Sullivan. Almost Square Packing. In *Proc. of CPAIOR 2011*, pages 196–209. Springer, 2011.

[Simonis *et al.*, 2010] Helmut Simonis, Paul Davern, Jacob Feldman, Deepak Mehta, Luis Quesada, and Mats Carlsson. A Generic Visualization Platform for CP. In *Proc. of CP 2010*, pages 460–474, 2010.

[Stergiou, 2008] Kostas Stergiou. Heuristics for Dynamically Adapting Propagation. In *Proc. of ECAI 2008*, pages 485–489, 2008.

[Walke,] Jordan Walke. React. `https://reactjs.org/`. Accessed: 2017-04-21.

[Wallace, 2015] Richard J. Wallace. SAC and Neighbourhood SAC. *AI Communications*, 28(2):345–364, 2015.

[Woodward *et al.*, 2011] Robert Woodward, Shant Karakashian, Berthe Y. Choueiry, and Christian Bessiere. Solving Difficult CSPs with Relational Neighborhood Inverse Consistency. In *Proc. of AAAI 2011*, pages 112–119, 2011.

[Woodward *et al.*, 2014] Robert J. Woodward, Anthony Schneider, Berthe Y. Choueiry, and Christian Bessiere. Adaptive Parameterized Consistency for Non-Binary CSPs by Counting Supports. In *Proc. of CP 2014*, volume 8656 of *LNCS*, pages 755–764. Springer, 2014.

[Woodward *et al.*, 2018] Robert J. Woodward, Berthe Y. Choueiry, and Christian Bessiere. A Reactive Strategy for High-Level Consistency During Search. In *Proc. of IJCAI 2018*, pages 1–8, 2018.

[Wotzlaw *et al.*, 2013] Andreas Wotzlaw, Alexander van der Grinten, and Ewald Speckenmeyer. Effectiveness of Pre- and Inprocessing for CDCL-based SAT Solving. Technical report, Institut für Informatik, Universität zu Köln, Germany, 2013.

[Wynn *et al.*, 1997] William C. Wynn, J. Fritz Barnes, Bernd Hamann, and Mark Miller. Multiresolution and Adaptive Rendering Techniques for Structured, Curvilinear Data. In *Scientific Visualization Conference (dagstuhl '97)*, pages 332–332, 1997.