Quantifying the Similarity of Algorithm Configurations

Lin Xu[1], Ashiqur R. KhudaBukhsh[2], Holger H. Hoos[1], and Kevin Leyton-Brown[1]

[1] University of British Columbia
{xulin730, hhoos, kevinlb}@cs.ubc.ca
[2] Carnegie Mellon University
akhudabu@cs.cmu.edu

Abstract A natural way of attacking a new, computationally challenging problem is to find a novel way of combining design elements introduced in existing algorithms. For example, this approach was made systematic in SATenstein (14), a highly parameterized stochastic local search (SLS) framework for SAT that unifies techniques across a wide range of well-known SLS solvers. The focus of such work so far has been on building frameworks and identifying high-performing configurations. Here, we focus on analyzing such frameworks, a problem that currently requires considerable manual effort and domain expertise. We propose a quantitative alternative: a new metric that measures the similarity between a new configuration and previously known algorithm designs. We first introduce concept DAGs, a data structure that preserves the hierarchical structure of configurations induced by conditional parameter dependencies. We then quantify the degree of similarity between two configurations as the transformation cost between the respective concept DAGs. In the context of analyzing SATenstein configurations, we demonstrate that visualizations based on transformation costs can provide useful insights into the similarities and differences between existing SLS-based SAT solvers and novel solver configurations.[3]

Keywords: SAT; stochastic local search; algorithm configuration similarity

1 Introduction

When faced with a new, computationally hard problem to solve, researchers do not typically want to reinvent the wheel. Instead, it makes sense to draw on design ideas from existing high-performance solvers. Such an approach can be made systematic by designing a single, highly parameterized solver that incorporates these different ideas, and then identifying a parameter configuration that achieves good performance via an automatic algorithm configuration method (14; 8). Indeed, many powerful configuration procedures have recently become available

[3] Lin Xu and Ashiqur R. KhudaBukhsh contributed equally to this work.

to meet this challenge (13; 10; 9; 19). The types of solvers configured in this way can range from simple heuristic switching (30) to a complex combination of multiple algorithms (28). While the result is often an algorithm with excellent performance characteristics, it can be difficult to understand such an algorithm, e.g., in terms of how similar (or dissimilar) it is to the existing solvers from which design ideas were drawn—a problem that has received little attention to date by the research community. This work seeks to address this gap. We propose a new metric for quantitatively assessing the similarity between configurations for highly parametric solvers, which computes the distance between two algorithm configurations in two steps. In the first step, the hierarchical structure of algorithm parameters is represented by a novel data structure called a concept DAG. In the second step, we estimate the similarity of two configurations as the transformation cost from one configuration to another, using concept DAGs.

In order to demonstrate the effectiveness of our approach, we investigate the configurations of `SATenstein`, a well-known, highly parameterized SLS solver. `SATenstein` has a rich and complex design space with 43 parameters, drawing design ideas from several existing solvers, and is one of the most complex SLS solvers in the literature. We show that visualizations based on transformation costs can provide useful insights into similarities and differences between solver configurations. In addition, we argue that this metric can help to suggest potential links between algorithm structure and algorithm performance.

To our knowledge, there is little previous work directly relevant to the problem of quantifying the similarity of algorithm configurations. Visualization techniques have been used previously to characterize the structure of instances of the well-known propositional satisfiability problem (SAT) (25); instead, we focus on algorithm design elements. Most similar to our work, Nikolić et al. (20) used the notion of edit distance to automatically quantify algorithm similarity. Our main innovation is to address hierarchies of conditional parameters by saying that edits to lower-level parameters are less significant than edits to higher-level parameters. Conditional parameters are increasingly important as algorithm development shifts to rely on algorithm configuration tools and hence parameter spaces become richer and more complex; see e.g., recent work on assessing parameter importance (12) and finding critical parameters (3).

The remainder of this paper is organized as follows. We present a high-level description of `SATenstein` in Section 2. Next, we describe concept DAGs (Section 3) and then present our experimental setup (Section 4). We describe our results on quantifying similarities between algorithm configurations in Section 5 and then conclude (Section 6).

2 SATenstein

In this section, we provide a short description of the overall design of `SATenstein`. A detailed description of `SATenstein` is given in (15). As shown in the high-level algorithm outline, any instantiation of `SATenstein` proceeds as follows:

1. Optionally execute $B1$, which performs search diversification.

2. Execute either $B2$, $B3$ or $B4$, thus performing WalkSAT-based local search, dynamic local search or G^{2}WSAT-based local search, respectively.
3. Optionally execute $B5$ to update data structures such as promising list, clause penalties, dynamically adaptable parameters or tabu attributes.

`SATenstein` consists of five building blocks and eight components, some of which are shared across different building blocks. It has 43 parameters in total. The choice of building block is encoded by several high-level categorical parameters, while the design strategies within each component are determined by a larger number of low-level parameters.

3 Concept DAGs

We now introduce *concept DAGs*, a novel data structure for representing algorithm configurations that preserves the hierarchical structure of parameter dependencies. Our notion of a concept DAG is based on that of a concept tree (31). We work with a DAG-based data structure because parameters may have more than one parent, where the child is only active if the parents take certain values (e.g., SATenstein's noise parameter *phi* is only activated when both *useAdaptiveMechanism* and *singleClauseAsNeighbor* are turned on). We then define four operators whose repeated application can be used to map between arbitrary concept DAGs, and assign each operator a cost. To compare two parameter configurations, we first represent them using concept DAGs and then define their similarity as the minimal total cost of transforming one DAG into the other.

A *concept DAG* is a six-tuple $G = (V, E, L^V, R, D, M)$, where V is a set of nodes, E is a set of directed edges between the nodes in V such that (V, E) is an acyclic graph, L^V is a set of lexicons (terms) for concepts used as node labels, R is a distinguished node called the root, D is the domain of discourse (i. e., , the set of all possible node labels), and M is an injective mapping from V to L^V that assigns a unique label to every node. A parameter configuration can be expressed as a concept DAG in which each node in V represents a parameter, and each directed edge in E represents the conditional dependence relationship between two parameters. L^V is the set of parameter values used in a particular configuration (i. e., a set containing exactly one value from the domain of each parameter), D is the union of the domains of all parameters, and M specifies which value assigned to each parameter $v \in V$ in the given configuration. We add an artificial root node R, which connects to all parameter nodes that do not have any parent, and refer to these parameters as *top-level parameters*.

We can transform one concept DAG into another by a series of delete, insert, relabel and move operations, each of which has an associated cost. For measuring the degree of similarity between two algorithm configurations, we first express them as concept DAGs, DAG_1 and DAG_2. We define the distance between these DAGs as the minimal total cost required for transforming DAG_1 into DAG_2. Obviously, the distance between two identical configurations is 0.

The parameters with the biggest impact on an algorithm's execution path are likely to appear high in the DAG (i.e., to be conditional upon few or no

```
    Input: CNF formula φ; real number cutoff;
           Booleans performDiversification, singleClauseAsNeighbor,
                    usePromisingList;
    Output: Satisfying variable assignment
    Start with random assignment A;
    Initialize parameters;
    while runtime < cutoff do
        if A satisfies φ then
            return A;
        end
        varFlipped ← FALSE;
        if performDiversification then
B1          with probability diversificationProbability() do
B1              c ← selectClause();
B1              y ← diversificationStrategy(c) ;
B1              varFlipped ← TRUE;
        end
        if not varFlipped then
            if not usePromisingList then
                if singleClauseAsNeighbor then
B2                  c ← selectClause();
B2                  y ← selectHeuristic(c) ;
                else
B3                  sety ← selectSet();
B3                  y ← tieBreaking(sety);
                end
            else
B4              if promisingList is not empty then
B4                  y ← selectFromPromisingList() ;
                else
B4                  c ← selectClause();
B4                  y ← selectHeuristic(c) ;
                end
            end
            flip y ;
B5          update();
        end
    end
```

Procedure SATenstein(...)

other parameters) and/or to turn on a complex mechanism (i.e., to have many parameters conditional upon them). Therefore, we say that the importance of a parameter v is a function of its depth (the length of the longest path from the root R of the given concept DAG to v) and the total number of other parameters conditional on it. To capture this definition of importance, we define the cost of each of the four DAG-transforming operations as follows.

Deletion cost $C(\text{delete}(v)) = \frac{1}{|V|} \cdot (height(DAG) - depth(v) + 1 + |DE(v)|)$, where $height(DAG)$ is the height of the DAG, $depth(v)$ is the depth of node v and $DE(v)$ is the set of descendants of node v. This captures the idea that it is more costly to delete top-level parameters and parameters that (de-)activate complex mechanisms.

Insertion cost $C(\text{insert}(u, v)) = \frac{1}{|V|} \cdot (height(DAG) - depth(u) + 1 + |DE(v)|)$, where $DE(v)$ is the set of descendants of v after the insertion.

Moving cost $C(\text{move}(u, v)) = \frac{|V|-2}{2 \cdot |V|} \cdot [C(\text{delete}(v)) + C(\text{insert}(u, v))]$, where $|V| > 2$.

Relabelling cost $C(\text{relabel}(v, l^v, l^{v^*}) = [C(\text{delete}(v)) + C(\text{insert}(u, v))] \cdot s(l^v, l^{v^*})$, where $s(l^v, l^{v^*})$ is a measure of the distance between two labels l^v and l^{v^*}. For parameters with continuous domains, $s(l^v, l^{v^*}) = |l^v - l^{v^*}|$. For parameters whose domains are some finite, ordinal and discrete set $\{l^{v_1}, l^{v_2}, \ldots, l^{v_k}\}$, $s(l^v, l^{v^*}) = \text{abs}(v - v^*)/(k-1)$, where $\text{abs}(v - v^*)$ measures the number of intermediate values between v and v^*. For categorical parameters, $s(l^v, l^{v^*}) = 0$ if $l^v = l^{v^*}$ and 1 otherwise.

4 Experimental Setup

Our quantitative analysis of `SATenstein` configurations is based on performance comparisons with eleven high-performance SLS solvers on six well-known SAT distributions, listed in Table 1 (we call each of these solvers a challenger) and Table 2, respectively.

We performed algorithm configuration using ParamILS (10), a well-known automatic algorithm configurator. On each benchmark distribution, we configured `SATenstein` on the training set, and evaluated its performance of the configuration on the test set. For each test set instance, we ran each solver 25 times with a per-run cutoff of 600 CPU seconds. Following (10), we evaluate performance in terms of penalized average run time (PAR), which is defined as average run time with each timed out run counted as having completed in 10 times the cutoff time (in this case, 6000 CPU seconds). For a particular solver, we consider an instance solved if a majority of runs found a satisfying assignment. In practice, PAR can be sensitive to the choice of cutoff; however, in past work (14), we showed that PAR did not affect the qualitative evaluation of `SATenstein`'s performance in all six distributions we considered.

We conducted all of our experiments on a cluster of 55 machines each equipped with dual 3.2GHz Intel Xeon CPUs with 2MB cache and 2GB RAM, running OpenSuSE Linux 11.1 and managed by Sun Grid Engine (version 6.0).

5 Quantitative Comparison of Algorithm Configurations

In previous work, we performed an extensive performance evaluation on six well-known benchmark distributions, finding that `SATenstein` outperformed all challengers in every distribution (15). Moreover, we found that `SATenstein`

Algorithm	Abbrev	Reason for Inclusion	Parameters
Ranov (21)	Ranov	gold 2005 SAT Competition (random)	wp
G^2WSAT (16)	G2	silver 2005 SAT Competition (random)	novNoise, dp
VW (23)	VW	bronze 2005 SAT Competition (random)	c, s, wpWalk
gNovelty$^+$ (22)	GNOV	gold 2007 SAT Competition (random)	novNoise, wpWalk, ps
adaptG^2WSAT$_0$ (17)	AG20	silver 2007 SAT Competition (random)	NA
adaptG^2WSAT$_+$ (18)	AG2+	bronze 2007 SAT Competition (random)	NA
adaptNovelty$^+$ (7)	ANOV	gold 2004 SAT Competition (random)	wp
adaptG^2WSAT$_p$ (18)	AG2p	performance comparable to G^2WSAT (16), Ranov, and adaptG^2WSAT$_+$; see (17)	NA
SAPS (11)	SAPS	prominent DLS algorithm	alpha, ps, rho, sapsthresh, wp
RSAPS (11)	RSAPS	prominent DLS algorithm	alpha, ps, rho, sapsthresh, wp
PAWS (27)	PAWS	prominent DLS algorithm	maxinc, pflat

Table 1. Our eleven challenger algorithms.

Distribution	Description	generator Parameters	Train/Test size
QCP	SAT-encoded quasi-group completion problems (5)	order $O \in [10, 30]$; holes $H = h * O^{1.55}$, $h \in [1.2, 2.2]$	1000/1000
SW-GCP	SAT-encoded small-world graph-colouring problems (4)	ring lattice size $S \in [100, 400]$; nearest neighbors connected: 10; rewiring probability: 2^{-7}; chromatic numbers: 6	1000/1000
R3SAT	uniform-random 3-SAT instances (24)	variable: 600; clauses-to-variables ratio: 4.26	250/250
HGEN	random instances generated by HGEN2 (6)	variable $n \in [200, 400]$	1000/1000
FAC	SAT-encoded factoring problems (29)	prime number $\in [3000, 4000]$	1000/1000
CBMC(SE)	SAT-encoded bounded model checking (1), preprocessed by SatELite (2)	array size $s \in [1, 2000]$; loop unwinding $n \in 4, 5, 6$	302/302

Table 2. Our six benchmark distributions.

outperformed tuned challengers as well, albeit to a reduced extent. In order to refer to them in what follows, we summarize these results in Tables 4 and 5.

Table 3 gives a high-level description of SATenstein solvers in terms of building blocks used and overall SLS category. Recall that SATenstein draws components from three major SLS solver categories: WalkSAT, dynamic local search and G^2WSAT-based algorithms.

5.1 Comparison of SATenstein Configurations

We now compare our automatically identified SATenstein solver designs to all of the challengers. As shown in Table 1, 3 of our 11 challengers (AG2p, AG2+, and AG20) are parameter-less. Furthermore, RANOV only differs from ANOV by the addition of a preprocessing step, and so can be understood as a variant of the same algorithm. This leaves us with 7 parameterized challengers to consider. For each, we sampled 50 configurations (consisting of the default configuration, one configuration optimized for each of our 6 benchmark distributions, and 43 random configurations). We then computed the pairwise transformation cost between the resulting 359 configurations (7×50 challengers' configurations + 6 SATenstein solvers + AG2p + AG2+ + AG20). The result can be understood

Solver	Uses building blocks	Broad category
SATenstein[QCP]	1, 2 and 5	WalkSAT
SATenstein[SW-GCP]	2 and 5	WalkSAT
SATenstein[R3SAT]	1, 3 and 5	Dynamic local search
SATenstein[HGEN]	1, 2 and 5	WalkSAT
SATenstein[FAC]	3 and 5	Dynamic local search
SATenstein[CBMC(SE)]	1, 3 and 5	Dynamic local search

Table 3. High-level summary of SATenstein solvers.

SATenstein[D] (14)	**0.08** **100%**	**0.03** **100%**	**1.11** **100%**	**0.02** **100%**	**10.89** **100%**	**4.75** **100%**
Solvers	**QCP**	**SW-GCP**	**R3SAT**	**HGEN**	**FAC**	**CBMC(SE)**
AG20 (17)	1054.99 81.2%	0.64 **100%**	2.14 **100%**	137.02 98.1%	3594.40 35.9%	2169.77 61.1%
AG2p (18)	1119.96 80.1%	0.43 **100%**	2.35 **100%**	105.30 98.4%	1954.83 80.6%	2294.24 61.1%
AG2+ (18)	1091.37 80.3%	0.67 **100%**	3.04 **100%**	148.28 98.0%	1450.89 91.0%	2181.92 61.1%
ANOV (7)	25.42 99.6%	4.86 **100%**	11.17 **100%**	109.94 98.6%	2897.52 51.4%	2021.22 61.1%
G2 (16)	2942.13 50.9%	4092.29 31.0%	3.69 **100%**	104.55 98.7%	5947.80 0%	2139.12 65.4%
GNOV (22)	414.69 93.3%	1.20 **100%**	11.14 **100%**	52.58 99.4%	5935.39 0%	2236.85 61.5%
PAWS (27)	1127.84 81.0%	4495.50 24.3%	1.77 **100%**	62.18 99.4%	22.05 **100%**	1693.82 70.8%
RANOV (21)	73.38 99.1%	0.15 **100%**	18.29 **100%**	151.11 98.2%	887.33 96.8%	1227.07 79.7%
RSAPS (11)	1255.94 79.2%	5635.54 5.4%	18.42 **100%**	33.28 99.7%	17.86 **100%**	827.81 85.0%
SAPS (11)	1248.34 79.4%	3864.74 34.2%	22.93 **100%**	40.17 99.5%	16.41 **100%**	646.89 89.7%
VW (23)	1022.69 81.9%	161.74 99.4%	12.45 **100%**	176.18 97.8%	3382.02 35.3%	385.12 93.4%

Table 4. Performance of SATenstein and the 11 challengers. Every algorithm was run 25 times on each instance with a cutoff of 600 CPU seconds per run. Each cell $\langle i, j \rangle$ summarizes the test-set performance of algorithm i on distribution j as a/b, where a (top) is the the penalized average runtime; b (bottom) is the percentage of instances solved (i.e., those with median runtime $<$ cutoff). The best-scoring algorithm(s) in each column are indicated in bold, and the best-scoring challenger(s) are underlined.

as a graph with 359 nodes and 128 522 edges, with nodes corresponding to concept DAGs, and edges labeled by the minimum transformation cost between them. To visualize this graph, we used a dimensionality reduction method to map it onto a plane, with the aim of positioning points so that the Euclidean distance between every pair of points approximates their transformation cost as accurately as possible. In particular, we used the Isomap algorithm (26), which

Solvers	QCP	SW-GCP	R3SAT	HGEN	FAC	CBMC(SE)
ANOV[D] (7)	26.13 99.6%	0.06 **100%**	2.68 **100%**	119.75 98.2%	1731.16 90.1%	994.94 83.4%
G2[D] (16)	514.29 91.4%	**0.05** **100%**	3.64 **100%**	98.70 99.1%	617.83 97.8%	1084.60 81.4%
GNOV[D] (22)	417.33 92.9%	0.22 **100%**	8.87 **100%**	68.24 99.4%	5478.75 0.3%	2195.76 61.8%
PAWS[D] (27)	68.06 99.2%	0.70 **100%**	1.91 **100%**	64.48 99.4%	22.01 **100%**	1925.56 67.7%
RANOV[D] (21)	75.06 98.9%	0.15 **100%**	13.85 **100%**	141.61 98.1%	336.27 100%	1223.83 80.4%
RSAPS[D] (11)	868.37 85.2%	0.19 **100%**	1.32 **100%**	42.99 99.5%	12.17 **100%**	67.59 99.0%
SAPS[D] (11)	27.69 99.8%	0.31 **100%**	1.54 **100%**	**31.77** 99.6%	**10.68** **100%**	62.63 99.0%
VW[D] (23)	**0.33** **100%**	417.71 94.8%	**1.26** **100%**	57.44 99.6%	32.38 100%	**16.45** **100%**

Table 5. Performance summary of the automatically configured versions of 8 challengers (three challengers have no parameters). Every algorithm was run 25 times on each problem instance with a cutoff of 600 CPU seconds per run. Each cell $\langle i, j \rangle$ summarizes the test-set performance of algorithm i on distribution j as a/b, where a (top) is the the penalized average runtime; b (bottom) is the percentage of instances solved (i.e., having median runtime $<$ cutoff). The best-scoring algorithm(s) in each column are indicated in bold.

builds on a multidimensional scaling technique but has the ability to preserve the intrinsic geometry of the data, as captured in the geodesic manifold distances between all pairs of data points. It is capable of discovering the nonlinear degrees of freedom that underlie complex natural observations.

The final layout of similarities among 359 configurations (16 algorithms) is shown in Figure 1. Observe that in most cases the 50 different configurations for a given challenger solver were so similar that they mapped to virtually the same point in the graph.

As noted earlier, the distance between any two configurations shown in Figure 1 only approximates their true distance. In addition, the result of the visualization also depends on the number of configurations considered: adding an additional configuration may affect the position of many or all other configurations. Thus, before drawing further conclusions about the results illustrated in Figure 1, we validated the fidelity of the visualization to the original distance data. As can be seen from Figure 2, although `Isomap` tended to underestimate the true distances between configurations, there was a strong correlation between the computed and mapped distances (Pearson correlation coefficient: 0.93). Also, the mapping preserved the relative ordering of the true distances between configurations quite well (Spearman correlation coefficient 0.91)—in other words, distances that appear similar in the 2D plot tend to correspond to similar true distances

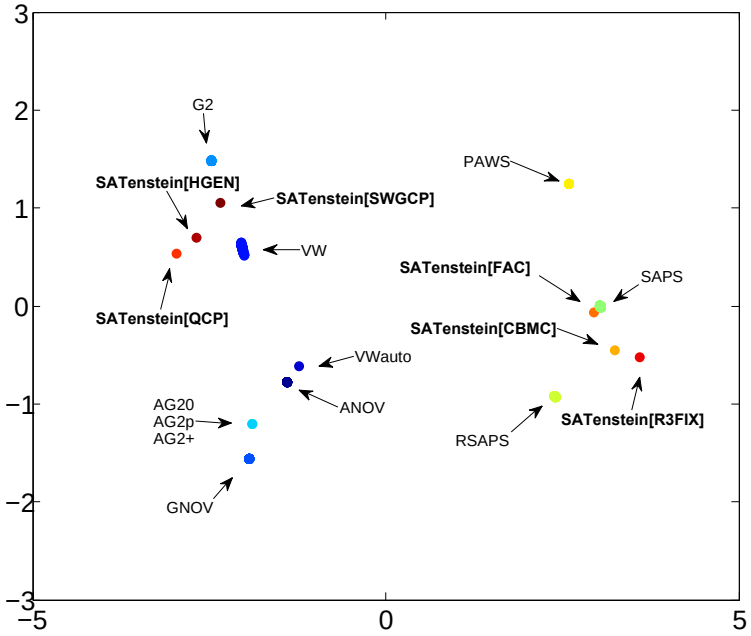


Figure 1. Visualization of the transformation costs in the design of 16 high-performance solvers (359 configurations) obtained via `Isomap`.

(and vice versa). Digging deeper, we confirmed that the challenger closest in Figure 1 to each given `SATenstein` solver was indeed the one having the lowest true transformation cost. This was not true for the most distant challengers; however, we find this acceptable, since we are mainly interested in examining which configurations are similar to each other.

Having confirmed that our dimensionality reduction method is performing reliably, let us examine Figure 1 in more detail. Overall, and unsurprisingly, we first note that the transformation cost between two configurations in the design space is very weakly related to their performance difference (quantitatively, the Spearman correlation coefficient between performance difference (PAR ratio) and configuration difference (transformation cost) was 0.25). Examining algorithms by type, we note that all dynamic local search algorithms are grouped together, on the right side of Figure 1; likewise, the algorithms using adaptive mechanisms are grouped together at the bottom of Figure 1. `SATenstein()` solvers were typically more similar to each other than to challengers, and fell into two broad clusters. The first cluster also includes the `SAPS` variants (`SAPS`, `RSAPS`), while the second also includes `G2` and `VW`. None of the `SATenstein` solvers uses an adaptive mechanism to automatically adjust other parameters. In fact, as shown in Table 5, the same is true of the best performance-optimized challengers as neither `SAPS`, `G2`, or `VW` use adaptive mechanism. This suggests that in many cases, contrary to common belief (see, e.g., (7; 18)) it may be preferable to expose

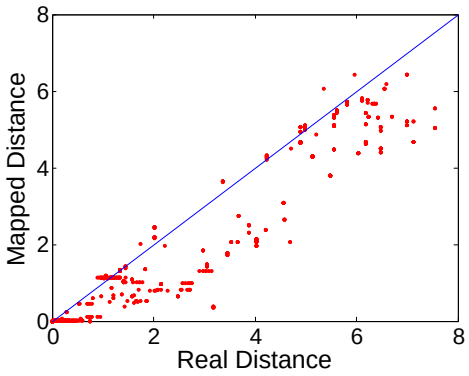


Figure 2. True *vs* mapped distances in Figure 1. The data points correspond to the complete set of SATenstein[D] for all domains and all challengers with their default and domain-specific, optimized configurations.

parameters so they can be instantiated by sophisticated configurators rather than automatically adjusting them at running time using a simple adaptive mechanism.

We now consider benchmarks individually. For the FAC benchmark, SATenstein[FAC] had similar performance to SAPS[FAC]; as seen in Figure 1, both solvers are structurally very similar as well. Overall, for the 'industrial' distributions, CBMC(SE) and FAC, dynamic local search algorithms often yielded the best performance amongst all challengers. Our automatically-constructed SATenstein solvers for these two distributions are also dynamic local search algorithms. Due to the larger search neighbourhood and the use of clause penalties, dynamic local search algorithms are more suitable for solving industrial SAT instances, which often have some special global structure.

For R3SAT, a well-studied distribution, many challengers showed good performance (the top three challengers were VW, RSAPS, and SAPS). The performance of SATenstein[R3SAT] is only slightly better than that of VW[R3SAT]. Figure 1 shows that SATenstein[R3SAT] is a dynamic local search algorithm similar to RSAPS and SAPS.

For HGEN, even the best performance-optimized challengers, RSAPS[HGEN] and SAPS[HGEN], performed poorly. SATenstein[HGEN] achieves more than 1 000-fold speedups against all challengers. Its configuration is far away from any dynamic local search algorithm (the best challengers), and closest to VW, a WalkSAT algorithm, and G2.

For QCP, VW[QCP] does not reach the performance of SATenstein[QCP], but significantly outperforms all other challengers. Our transformation cost analysis shows that VW is the closest neighbour to SATenstein[QCP]. For SWGCP, many challengers achieve similar performance to SATenstein[SWGCP].

Figure 1 shows that SATenstein[SWGCP] is close to G2[SWGCP], which is the best performing challenger on SWGCP.

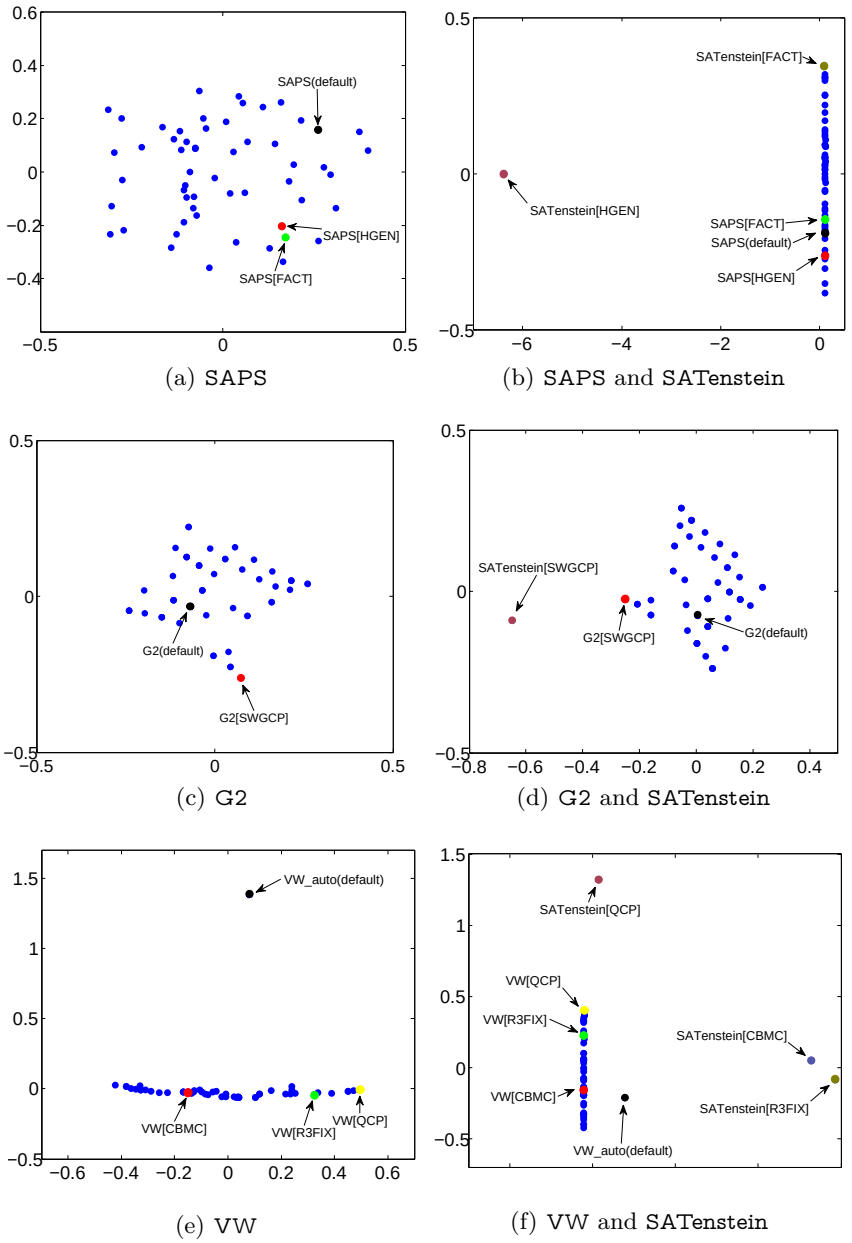
5.2 Comparison to Configured Challengers

Since there were large performance gaps between default and configured challengers, we were also interested in the transformation cost between the configurations of individual challenger solvers. Table 5 shows that after configuring each challenger for each distribution, we found that SAPS was best on HGEN and FAC; G2 was best on SWGCP, and VW was best on CBMC(SE), QCP, and R3FIX. Figure 3 (left) visualizes the parameter spaces for each of these three solvers (43 random configurations + default configuration + 6 optimized configurations). Figure 3 (right) shows the same thing, but also adds the best SATenstein() configurations for each benchmark on which the challenger exhibited top performance.

Examining these figures in the left column of Figure 3, we first note that the SAPS configurations optimized for FAC and HGEN are very similar but differ substantially from SAPS's default configuration. On SWGCP, the optimized configuration of G2 not only performs much better than the default but, as seen in Figure 3(c), is also quite different. All three top-performing VW configurations are rather different from VW's default, and none of them uses the adaptive mechanism for choosing parameter *wpWalk*, *s*, and *c*. Since the parameter useAdaptiveMechanism is a top-level parameter and many other parameters are conditionally dependent on it, the transformation costs between VW default and optimized configurations of VW are very large, due to the high relabelling cost for these nodes in our concept DAGs.

The right column of Figure 3 illustrates the similarity between optimized SATenstein() solvers and the best performing challenger for each benchmark. As previously noted, SATenstein[FAC] and SAPS[FAC] are not only very similar in performance, but also structurally similar. Likewise, SATenstein[SWGCP] is similar to G2SWGCP. On R3SAT, many challengers had similar performance. SATenstein[R3SAT] (PAR=1.11) was quite different from the best challenger VW[R3SAT] (PAR=1.26), but resembled SAPS[R3SAT] (PAR=1.53). For the three remaining benchmarks, SATenstein() solvers exhibited much better performance than the best optimized challengers, and their configurations likewise differed substantially from the challengers' configurations.

As an aside, it might initially be surprising that qualitative features of the visualizations in Figures 3 appear to be absent from Figure 1. In particular, the sets of randomly sampled challenger configurations that are quite well-separated in Figure 3 are nearly collapsed into single points in Figure 1, although the scales are not vastly different. The reason for this lies in the fact that the 2D-mapping of the highly non-planar pairwise distance data performed by Isomap focuses on minimal overall distortion. For example, when visualizing the differences within a set of randomly sampled SAPS configurations (Figure 3 (a)), Isomap spreads these out into a cloud of points to represent their differences. However, the presence of a single SATenstein configuration that has large transformation



(a) SAPS (b) SAPS and SATenstein

(c) G2 (d) G2 and SATenstein

(e) VW (f) VW and SATenstein

Figure 3. The transformation costs of configuration of individual challengers and selected SATenstein solvers. (a): SAPS (best on HGEN and FACT); (b): SAPS and SATenstein[HGEN, FACT]; (c): G2 (best on SWGCP); (d): G2 and SATenstein[SWGCP]; (e): VW (best on CBMC(SE), QCP, and R3FIX); (f): VW and SATenstein[CBMC, QCP, R3FIX].

costs from all of these SAPS configurations forces Isomap to use one dimension to capture those differences, leaving essentially only one dimension to represent the much smaller differences between the SAPS configurations (Figure 3 (b)). Adding further very different configurations (as present in Figure 1) leads to mappings in which the smaller differences between configurations of the same challenger become insignificant.

6 Conclusion

We have proposed a new metric for quantitatively assessing the similarity between configurations of highly parametric solvers. Our metric is based on a data structure, concept DAGs, that preserves the internal hierarchical structure of parameters. We estimate the similarity of two configurations as the transformation cost from one configuration to another. In the context of SATenstein, a highly parameterized SLS-based SAT solver, we have demonstrated that visualizations based on transformation cost can provide useful insights into similarities and differences between solver configurations. In addition, we believe that this metric could be useful for suggesting potential links between algorithm structure and algorithm performance further exploration of which could be an interesting future research direction.

Bibliography

[1] Clarke, E., Kroening, D., Lerda, F.: A tool for checking ANSI-C programs. In: Proceedings of the Tenth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'2004). pp. 168–176 (2004)

[2] Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT'05). pp. 61–75 (2005)

[3] Fawcett, C., Hoos, H.H.: Analysing differences between algorithm configurations through ablation. Journal of Heuristics pp. 1–28 (2013)

[4] Gent, I.P., Hoos, H.H., Prosser, P., Walsh, T.: Morphing: Combining structure and randomness. In: Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI'99). pp. 654–660 (1999)

[5] Gomes, C.P., Selman, B.: Problem structure in the presence of perturbations. In: Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI'97). pp. 221–226 (1997)

[6] Hirsch, E.A.: Random generator hgen2 of satisfiable formulas in 3-CNF (2002), `http://logic.pdmi.ras.ru/~hirsch/benchmarks/hgen2-1.01.tar.gz`. Last accessed on Dec. 18, 2015.

[7] Hoos, H.H.: An adaptive noise mechanism for WalkSAT. In: Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI'02). pp. 655–660 (2002)

[8] Hoos, H.H.: Programming by optimization. Communications of the ACM 55(2), 70–80 (2012)

[9] Hutter, F., Hoos, H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Learning and Intelligent Optimization (LION). pp. 507–523 (2011)

[10] Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ParamILS: An automatic algorithm configuration framework. Joural of Artificial Intelligence Research (JAIR) 36(1), 267–306 (2009)

[11] Hutter, F., Tompkins, D.A.D., Hoos, H.H.: Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In: Eighth International Conference on Principles and Practice of Constraint Programming (CP'02). pp. 233–248 (2002)

[12] Hutter, F., Hoos, H., Leyton-Brown, K.: An efficient approach for assessing hyperparameter importance. In: Proceedings of the 31st International Conference on Machine Learning (ICML-14). pp. 754–762 (2014)

[13] Kadioglu, S., Malitsky, Y., Sellmann, M., Tierney, K.: ISAC - instance specific algorithm configuration. In: European Conference on Artificial Intelligence (ECAI). vol. 2010, pp. 751–756 (2010)

[14] KhudaBukhsh, A.R., Xu, L., Hoos, H.H., Leyton-Brown, K.: SATenstein: Automatically building local search SAT solvers from components. In: Proceedings of the Twenty-first International Joint Conference on Artificial Intelligence (IJCAI'09). pp. 517–524 (2009)

[15] KhudaBukhsh, A.R., Xu, L., Hoos, H.H., Leyton-Brown, K.: Satenstein: Automatically building local search sat solvers from components. Artificial Intelligence 232(0), 20–42 (Mar 2016)

[16] Li, C.M., Huang, W.: Diversification and determinism in local search for satisfiability. In: Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT'05). pp. 158–172 (2005)
[17] Li, C.M., Wei, W., Zhang, H.: Combining adaptive noise and promising decreasing variables in local search for SAT (2007), solver description, SAT competition 2007
[18] Li, C.M., Wei, W.X., Zhang, H.: Combining adaptive noise and look-ahead in local search for SAT. In: Proceedings of the Tenth International Conference on Theory and Applications of Satisfiability Testing (SAT'07). pp. 121–133 (2007)
[19] Malitsky, Y., Sabharwal, A., Samulowitz, H., Sellmann, M.: Algorithm portfolios based on cost-sensitive hierarchical clustering. In: Proceedings of the Twenty-Third international joint conference on Artificial Intelligence. pp. 608–614. AAAI Press (2013)
[20] Nikolić, M., Marić, F., Janičić, P.: Instance-based selection of policies for sat solvers. In: Theory and Applications of Satisfiability Testing-SAT 2009, pp. 326–340. Springer (2009)
[21] Pham, D.N., Anbulagan: Resolution enhanced SLS solver: R+AdaptNovelty+ (2007), solver description, SAT competition 2007
[22] Pham, D.N., Thornton, J., Gretton, C., Sattar, A.: Combining adaptive and dynamic local search for satisfiability. Journal on Satisfiability, Boolean Modeling and Computation (JSAT) 4, 149–172 (2008)
[23] Prestwich, S.: Random walk with continuously smoothed variable weights. In: Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT'05). pp. 203–215 (2005)
[24] Simon, L.: SAT competition random 3CNF generator (2002), www.satcompetition.org/2003/TOOLBOX/genAlea.c. Last accessed on Dec. 18, 2015.
[25] Sinz, C.: Visualizing the internal structure of sat instances (preliminary report). In: SAT. Citeseer (2004)
[26] Tenenbaum, J.B., de Silva, V., Langford, J.C.: A global geometric framework for nonlinear dimensionality reduction. Science 290(5500), 2319–2323 (2000)
[27] Thornton, J., Pham, D.N., Bain, S., Ferreira, V.: Additive versus multiplicative clause weighting for SAT. In: Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI'04). pp. 191–196 (2004)
[28] Tompkins, D.A., Balint, A., Hoos, H.H.: Captain jack: new variable selection heuristics in local search for sat. In: Theory and Applications of Satisfiability Testing-SAT 2011, pp. 302–316. Springer (2011)
[29] Uchida, T., Watanabe, O.: Hard SAT instance generation based on the factorization problem (1999), originally available from http://www.is.titech.ac.jp/~watanabe/gensat/a2/GenAll.tar.gz
[30] Wei, W., Li, C.M., Zhang, H.: A switching criterion for intensification, and diversification in local search for sat. Journal on Satisfiability, Boolean Modeling and Computation 4, 219–237 (2008)
[31] Xue, Y., Wang, C., Ghenniwa, H., Shen, W.: A tree similarity measuring method and its application to ontology. Journal of Universal Computer Science 15(9), 1766–1781 (2001)