Robust Benchmark Set Selection for Boolean Constraint Solvers

H. Hoos[1], B. Kaufmann[2], T. Schaub[2], and M. Schneider[2]

[1] Department of Computer Science, University of British Columbia,
Vancouver, BC (Canada), hoos@cs.ubc.ca
[2] Institute of Computer Science, University of Potsdam, Germany,
{kaufmann,torsten,manju}@cs.uni-potsdam.de

Abstract. We investigate the composition of representative benchmark sets for evaluating and improving the performance of robust Boolean constraint solvers in the context of satisfiability testing and answer set programming. Starting from an analysis of current practice, we isolate a set of desiderata for guiding the development of a parametrized benchmark selection algorithm. Our algorithm samples a benchmark set from a larger base set (or distribution) comprising a large variety of instances. This is done fully automatically, in a way that carefully calibrates instance hardness and avoids duplicates. We demonstrate the usefulness of this approach by means of empirical results showing that optimizing solvers on the benchmark sets produced by our method leads to better configurations than obtained based on the much larger, original sets.

1 Introduction

The availability of representative sets of benchmark instances is of crucial importance for the successful development of high-performance solvers for computationally challenging problems, such as propositional satisfiability (SAT) and answer set programming (ASP). Such benchmark sets play a key role for assessing solver performance and thus for measuring the computational impact of algorithms and/or their vital parameters. On the one hand, this allows a solver developer to gain insights on the strengths and weaknesses of features of interest. On the other hand, representative benchmark instances are indispensable to empirically underpin the claims of computational benefit of novel ideas.

A representative benchmark set is composed of benchmark instances stemming from a variety of different benchmark classes. Such benchmark sets have been assembled (manually) in the context of well-known solver competitions, such as the SAT and ASP competitions, and then widely used in the research literature. These sets of competition benchmarks are well-accepted, because they have been constituted by an independent committee using sensible criteria. Moreover, these sets evolve over time and thus usually reflect the capabilities (and limitations) of state-of-the-art solvers; they are also publicly available and well-documented.

However, instance sets from competitions are not always suitable for benchmarking scenarios where the same runtime cutoff is used for all instances. For example, in the last three ASP competitions, only $\approx 10\%$ of all instances were non-trivial (runtime over 9 second, i.e., 1% of the runtime cutoff) for the state-of-the-art ASP solver *clasp*, while all other instances were trivial or unsolvable for *clasp* within the time cutoff used in the competition. While benchmarking, results of benchmarks are (typically) aggregated over all instances. But if the percentage of interesting instances in the benchmark set is too small, the interesting instances have small influence on the aggregated result and the overall result is dominated by uninteresting, i.e., trivial or unsolvable, instances. Hence, a significant change of the runtime behaviour of a new algorithm is harder to identify on such degenerate benchmark sets. In addition, uninteresting instances unnecessarily waste computational resources and thus cause avoidable delays in the benchmarking process.

Moreover, in ASP, competition instances do not necessarily represent real world applications. In the absence of a common modelling language, benchmark instances are often formulated in the most basic common setting and thus bear no resemblance to how real world problems are addressed (e.g., they are usually free of any aggregates).[3] The situation is simpler in SAT, where a wide range of benchmark instances stems from real-world applications and are quite naturally encoded in a low-level format, without the modelling layer present in ASP. Notably, SAT competitions place considerable emphasis on a public and transparent instance selection procedure [1]. However, as we discuss in detail in Section 3, competition settings may differ from other benchmarking contexts.

In what follows, we elaborate upon the composition of representative benchmark sets for evaluating and improving the performance of Boolean constraint solvers in the context of ASP and SAT. Starting from an analysis of current practice of benchmark set selection in the context of SAT competitions (Section 2), we isolate a set of desiderata for representative benchmark sets (Section 3). For instance, sets with a large variety of instances are favourable when developing a default configuration of a solver that is desired to perform well across a wide range of instances. We rely on these desiderata for guiding the development of a parametrized benchmark selection algorithm (Section 4).

Overall, our approach makes use of (i) a large base set (or distribution) of benchmark instances; (ii) instance features; and (iii) a representative set of state-of-the-art solvers. Fundamentally, it constructs a benchmark set with desirable properties regarding difficulty and diversity by sampling from the given base set. It achieves diversity of the benchmark set by clustering instances based on their similarity w.r.t a given set of features, while ensuring that no cluster is overrepresented. The difficulty of the resulting set is calibrated based on the given set of solvers. Use of the benchmark sets thus obtained helps save computational resources during solver development, configuration and evaluation, while concentrating on interesting instances.

[3] In ASP competitions, this deficit is counterbalanced by a modelling track, in which each participant can use its preferred modelling language.

We empirically demonstrate in Section 5 that optimizing solvers on the obtained selection of benchmarks leads to better configurations than obtainable from the vast original set of benchmark instances. We close with a final discussion and some thoughts on future work in Section 6.

2 Current Practice

The generation or selection of benchmark sets is an important factor in the empirical analysis of algorithms. Depending on the goals of the empirical study, there are various criteria for benchmark selection. For example, in the field of Boolean constraint solving, regular competitions are used to asses new approaches and techniques as well as to identify and recognize state-of-the-art solvers. Over the years, competition organizers came up with sets of rules for selecting subsets of submitted instances to assess solver performance in a fair manner. To begin with, we investigate the rules used in the well-known and widely recognized SAT Competition[4], which try to achieve (at least) three overall goals. First, the selection should be broad, i.e., the selected benchmark set should contain a large variety of different kinds of instances to assess the robustness of solvers. Second, each selected instance should be significant w.r.t. the ranking obtained from the competition. Third, the selection should be fair, i.e., the selected set should not be dominated by a set of instances from the same source (either a domain or a benchmark submitter).

For the 2009 SAT Competition [2] and the 2012 SAT Challenge [1], instances were classified according to hardness, as assessed based on the runtime of a set of representative solvers. For instance, for the 2012 SAT Challenge, the organizers measured the runtimes of the best five SAT solvers from the Application and Crafted tracks of the last SAT Competition on all available instances and assigned each instance to one of the following classes: *easy* instances are solved by all solvers under 10% of the runtime cutoff, i.e., 90 CPU seconds; *medium* instances are solved by all solvers under 100% of the runtime cutoff; *too hard* instances are not solved by any solver within 300% of the runtime cutoff; and *hard* instances are solved by at least one solver within 300% of the runtime cutoff but not by all solvers within 100% of the runtime cutoff. Instances were then selected with the objective to have 50% *medium* and 50% *hard* instances in the final instance set and, at the same time, to allow at most 10% of the final instance set to originate from the same source.

While the *easy* instances are assumed to be solvable by all solvers, the *too hard* instances are presumably not solvable by any solver. Hence, neither class contributes to the solution count ranking used in the competition.[5] On the other hand, *medium* instances help to rank weaker solvers and to detect performance deterioration w.r.t. previous competitions. The *hard* instances are most useful for ranking the top-performing solvers and provide both a challenge and a chance to improve state-of-the-art SAT solving.

[4] `http://www.satcompetition.org`

[5] Solution count ranking assesses solvers based on the number of solved instances.

Although using a large variety of benchmark instances is clearly desirable for robust benchmarking, the rules used in the SAT Competition are not directly applicable to our identified use cases. First, the hardness criteria and distribution used are directly influenced by the use of the solution count ranking system. On the other hand, ranking systems that also consider measured runtimes, like the careful ranking[6] [3], might be better suited for differentiating solver performance. Second, limiting the number of instances from one source to achieve fairness is not needed in our setting. Furthermore, the origin of instances provides only an indirect way of achieving a heterogeneous instance set, as certain instances of different origin may in fact be more similar than other pairs of instances from the same source.

3 Desirable Properties of Benchmark Sets

Before diving into the details of our selection algorithm, let us first explicate the desiderata for a representative benchmark set (cf. [4]).

Large Variety of Instances. As mentioned, a large variety of instances is favourable to assess the robustness of solver performance and to reduce the risk of generalising from results that only apply to a limited class of problems. Such large variety can include different types of problems, i.e., real-world applications, crafted problems, and randomly generated problems; different levels of difficulty, i.e., easy, medium, and hard instances; different instance sizes; or instances with diverse structural properties. While the structure of an instance is hard to assess, a qualitative assessment could be based on visualizing the structure [5], and a quantitative assessment can be performed based on instance features [6, 7]. Such instance features have already proven useful in the context of algorithm selection [7, 8] and algorithm configuration [9, 10].

Adapted Instance Hardness. While easy problem instances are sometimes useful for investigating certain properties of specific solvers, intrinsically hard or difficult to solve problem instances are better suited to demonstrate state-of-the-art solving capabilities through benchmarking. However, in view of the nature of NP-hard problems, it is likely that many hard instances cannot be solved efficiently. Resource limitations, such as runtime cutoffs or memory limits, are commonly applied in benchmarking. Solver runs that terminated prematurely because of violations of resource limits are not helpful in differentiating solver performance. Hence, instances should be carefully selected so that such prematurely terminated runs for the given set of solvers are relatively rare. Therefore, the *distribution of instance hardness* within a given benchmark set should be adjusted based on the given resource limits and solvers under consideration. In particular, instances that are too hard (i.e., for which there is a high probability of a timeout) as well as instances that are too easy, should be avoided, where

[6] Careful ranking compares pairs of solvers based on statistically significant performance differences and ranks solvers based on the resulting ranking graph.

hardness is assessed using a representative set of state-of-the-art solvers, as is done, for example, in the instance selection process of SAT competitions [2].

Since computational resources are typically limited, the number of benchmark instances should also be carefully calibrated. While using too few instances can bias the results, using too many instances can cost computational resources without improving the information gained from benchmarking. Therefore, we propose to start with a broad base set of instances, e.g., generated by one or more (possibly parametrized) generators or a collection of previously used competition instance sets, and to select a subset of instances following our desiderata.

Free of Duplicates, Reproducible, and Publicly Available. Benchmark set should be free of duplicates, because using the same instance twice does not provide any additional information about solver performance. Nevertheless, non-trivially transformed instances can be useful for assessing the robustness of solvers [11]. To facilitate reproducibility and comparability, both the problem instances and the process of instance selection should be publicly available. Ideally, problem instances should originate from established benchmark sets and/or public benchmark libraries. To our surprise, these properties are not true for all competition sets. For example, we found duplicates in the SAT Challenge 2012, ASP Competitions 2007 and 2009 (for example, `15-puzzle.init1.gz` and `15puzzle_ins.lp.gz` in the latter).

4 Benchmark Set Selection

Based on our analysis of solver competitions and the resulting desiderata, we developed an instance selection algorithm. Its implementation is open source and freely available at `http://potassco.sourceforge.net`. In addition, we present a way to assess the relative robustness and quality of an instance set based on the idea of Q-scores [1].

4.1 Benchmark Set Selection Algorithm

Our selection process starts from a given base set of instances I. This set can be a benchmark collection or simply a mix of previously used instances from competitions.

Inspired by past SAT competitions, a representative set of solvers S – e.g., best solvers of the last competition, the state-of-the-art (SOTA) contributors identified in the last competition, or contributors to SOTA portfolios [12] – is used to assess the hardness $h(i) \in \mathbb{R}$ of an instance $i \in I$. Typically, the runtime $t(i, s)$ (measured in CPU seconds) is used to assess the hardness of an instance $i \in I$ for solver $s \in S$. The aggregation of the runtimes of all solvers $s \in S$ on a given instance i can be carried out in several ways, e.g., by considering the minimal ($\min_{s \in S} t(i, s)$) or the average runtime ($\frac{1}{|S|} \cdot \sum_{s \in S} t(i, s)$). The resulting hardness metric is closely related to the intended ranking scheme for solvers. For example, the minimal runtime is a lower bound of the portfolio runtime performance and represents a challenging hardness metric appropriate in the context of solution count ranking. In contrast, the average runtime would be better suited for a careful ranking [3], which uses pairwise comparisons between solvers for each instance, because the pairs of runtimes for two solvers are of limited value if neither of them solved the given instance within the given cutoff time. Since all solvers contribute to the average runtime per instance, this metric will assess instances as hard even if only some solvers time out on time, and selecting instances based on it (as explained in the following) can therefore be expected to result in fewer timeouts overall.

After selecting a hardness metric, we have to choose how the instance hardness should be distributed within the benchmark set. As stated earlier, and under the assumption that the set to be created will not be used primarily in the context of solution count ranking, the performance of solvers can be compared better, if the incidence of timeouts is minimized. This is important, for example, in the context of algorithm configuration (manual or automatic). The incidence of timeouts can be minimized by increasing the runtime cutoff, but this is infeasible or wasteful in many cases. Alternatively, we can ensure that not too many instances on which timeouts occur are selected for inclusion in our benchmark set. At the same time, as motivated previously, it is also undesirable to include too many easy instances, because they incur computational cost and, depending on the hardness metric used, can also distort final performance rankings determined on a given benchmark set.

One way to focus the selection process on the most useful instances w.r.t. hardness, namely those that are neither too easy nor too hard, is to use an appropriately chosen probability distribution to guide sampling from the given base set of instances. For example, the use of a normal (Gaussian) distribution of instance hardness in this context leads to benchmark sets consisting predominantly of instances of medium hardness, but also include some easy and hard instances. Alternatively, one could consider log-normal or exponential distributions, which induce a bias towards harder instances, as can be found in many existing benchmark sets. Compared to the instance selection approach used in SAT competitions [2, 1], this method does not require the classification of instances into somewhat arbitrary hardness classes.

The parameters of the distribution chosen for instance sampling, e.g., mean and variance in the case of a normal or log-normal distribution, can be determined based on the hardness metric and runtime limit; e.g., the mean could be chosen as half the cutoff time. By modifying the mean, the sampling distribution can effectively be shifted towards harder or easier benchmark instances.

As argued before, the origin of instances is typically less informative than their structure, as reflected, e.g., in informative sets of instance features. Such informative sets of instance features are available for many combinatorial problems, including SAT [7], ASP [13] and CSP [14], where they have been shown to correlate with the runtime of state-of-the-art solvers and have been used prominently in the context of algorithm selection (see, e.g., [7, 8]). To prevent the inclusion of too many similar instances in the benchmark sets, we cluster the instances based on their similarity in feature space. We then require that a clus-

```
Algorithm 1: Benchmark Selection Algorithm
   Input  : instance set $I$; desired number of instances $n$; representative set of
            solvers $S$; runtimes $t(i, s)$ with $(i, s) \in I \times S$; instance features $f(i)$ for
            each instance $i \in I$; hardness metric $h : I \rightarrow \mathbb{R}$ of instances; desired
            distribution $\mathcal{D}_h$ regarding $h$; clustering algorithm $ca$; cutoff time $t_c$;
            threshold $e$ for too easy instances;
   Output : selected instances $I^*$
1  remove instances from $I$ that are not solved by any $s \in S$ within $t_c$;
2  remove instances from $I$ that are solved by all $s \in S$ under $e\%$ of $t_c$ ;
3  cluster all instances $i \in I$ in the feature space $f(i)$ into clusters $S(i)$ using
   clustering algorithm $ca$;
4  while $|I^*| < n$ and $I \neq \emptyset$ do
5      sample $x \in \mathbb{R} \sim \mathcal{D}_h$;
6      select instance $i^* \in I$ with the nearest $h(i^*)$ to $x$;
7      remove $i^*$ from $I$;
8      if $S(i^*)$ is not over-represented then
9          add $i^*$ to $I^*$;
10     end
11 end
12 return $I^*$
```

ter must not be over-represented in the selected instance set; in what follows, roughly reminiscent of the mechanism used in SAT competitions, we say that a cluster is over-represented if it contributes more than 10% of the instances to the final benchmark set. While other mechanisms are easily conceivable, the experiments we report later demonstrate that this simple criterion works well.

Algorithm 1 implements these ideas with the precondition that the base instance set I is free of duplicates. (This can be easily ensured by means of simple preprocessing.) In Line 1, all instances are removed from the given base set that cannot be solved by all solver from the representative solver set S within the selection runtime cutoff t_c (*rejection of too hard instances*). If solution count ranking is to be used in the benchmarking scenario under consideration, the cutoff in the instance selection process should be larger than the cutoff for benchmarking, as was done in the 2012 SAT Challenge. In Line 2, all instances are removed that are solved by all solvers under $e\%$ of the cutoff time (*rejection of too easy instances*). For example, in the 2012 SAT Challenge [1], all instances were removed which were solved by all solvers under 10% of the cutoff. Line 3 performs clustering of the remaining instances based on their features. To perform this clustering, the well-known k-means algorithm could be used, and the number of clusters could be computed using G-means [15, 10] or by increasing the number of clusters until the clustering optimization does not improve further under a cross validation [16]. In our experiments, we used the latter, because the G-means algorithm relies on a normality assumption that is not necessarily satisfied for the instance feature data used here. Beginning with Line 4, instances are sampled within a loop until enough instances are selected or no more instances are left in the base set. To this end, $x \in \mathbb{R}$ is sampled from a distribution $\mathcal{D}_h$ induced by instance hardness metric h, such that for each sample r from hardness distribution $\mathcal{D}_h$, the instance i^* is selected whose hardness $h(i^*)$ is closest to r. After adding i^* to I^*, the benchmark set under construction, it is removed from the base instance set I. If the respective cluster $S(i^*)$ is not already over-represented in I^*, instance i^* is added to I^*.

4.2 Benchmark Set Quality

We would like to ensure that our benchmark selection algorithm produces instance sets that are in some way better than the respective base sets. At the same time, any benchmark set I^* it produces should be representative of the underlying base set I in the sense that if an algorithm performs better than a given baseline (e.g., some prominent solver) on I^* it should also be better on I. However, the converse may not hold, because specific kinds of instances may dominate I but not I^*, and excellent performance on those instances can lead to a situation where an algorithm that performs better on I does not necessarily perform better on I^*.

Bayless et al. [17] proposed a quantitative assessment of instance set utility. Their use case is the performance assessment of (new) algorithms on an instance set I_1 that has practical limitations, e.g., the instances are too large, too hard to solve, or not enough instances are available. Therefore, a second instance set I_2 without these limitations is assessed as to whether it can be regarded as a representative proxy for the instance set I_1 during solver development or configuration. The key idea is that any I_2 that is a representative proxy for I_1 can be used in lieu of I_1 to assess performance of a solver, with the assurance that good performance on I_2 (which is easier to demonstrate or achieve) implies, at least statistically, good performance on I_1.

To assess the utility of an instance set, they use algorithm configuration [18, 9, 10]. An algorithm configurator is used to find a configuration $c := s(c_I)$ of solver s on instance set I by optimizing, e.g., the runtime of s. If I_2 is a representative proxy for I_1, the algorithm configuration $s(c_{I_2})$ should perform on I_1 as well as a configuration optimized directly on I_1, i.e. $s(c_{I_1})$. The Q-score $Q(I_1, I_2, s, m)$ defined in Equation (1) is the performance ratio of $s(c_{I_1})$ and $s(c_{I_2})$ on I_1 with respect to a given performance metric m. A large Q-score means I_2 is a good proxy for I_1. The short form of $Q(I_1, I_2, s, m)$ is $Q_{I_1}(I_2)$.

To compare both sets, I_1 and I_2, we want to know whether I_2 is a better proxy for I_1 than vice versa. To this end, we extended the idea in [17] and propose the Q*-score of I_1 and I_2 by computing the ratio of $Q_{I_1}(I_2)$ and $Q_{I_2}(I_1)$ as per Equation (2). If I_1 is a better proxy for I_2 than vice versa, the Q*-score $Q^*(I_1, I_2)$ is larger than 1.

$$Q(I_1, I_2, s, m) = \frac{m(s(c_{I_1}), I_1)}{m(s(c_{I_2}), I_1)} \tag{1}$$

$$Q^*(I_1, I_2) = \frac{Q_{I_1}(I_2)}{Q_{I_2}(I_1)} \tag{2}$$

We use the Q*-score to assess the quality of the sets I^* obtained from our benchmark selection algorithm in comparison to the respective base sets I. Based on this score, we can assess the degree to which our benchmark selection algorithm succeeded in producing a set that is representative of the given base set in the way motivated earlier. Thereby, a Q^*-score ($Q^*(I_1, I_2)$) and a Q-score ($Q_{I_1}(I_2)$) of larger than 1.0 indicates that I_2 is better proxy for I_1 than vice versa and I_2 is a good proxy for I_1.

5 Evaluation

We evaluated our benchmark set selection approach by means of the Q^*-score criterion on widely studied instance sets from SAT and ASP competitions.

Instance Sets. We used three base instance sets to select our benchmark set: `SAT-Application` includes all instances of the *application* tracks from the 2009 and 2011 SAT Competition and 2012 SAT Challenge; `SAT-Crafted` includes instances of the *crafted* tracks (resp. hard combinatorial track) of the same competitions; and `ASP` includes all instances of the 2007 ASP Competition (SLparse track), the 2009 ASP Competition (with the encodings of the Potassco group [19]), the 2011 ASP Competition (decision NP-problems from the system track), and several instances from the ASP benchmark collection platform *asparagus*[7]. Duplicates were removed from all sets, resulting in 649 instances in `SAT-Application`, 850 instances in `SAT-Crafted`, and 2589 instances in `ASP`.

Solvers. In the context of the two sets of SAT instances, the best two solvers of the application track, i.e., *Glucose* [20] (2.1) and *SINN* [21], and of the hard combinatorial track, i.e., *clasp* [19] (2.0.6) and *Lingeling* [22] (agm), and the best solver of the random track, i.e., *CCASAT* [23], of the 2012 SAT Challenge were chosen as representative state-of-the-art SAT solvers. *clasp* [19] (2.0.6), *cmodels* [24] (3.81) and *smodels* [25] (2.34) were selected as competitive and representative ASP solvers capable of reading the *smodels*-input format [26].

Instance Features. We used efficiently computable, structural features to cluster instances. The fifty-four *base* features of the feature extractor of *SATzilla* [7] (2012) were utilized for analysing SAT instances. The seven structural features of *claspfolio* [13] were considered for ASP instances, namely, tightness (0 or 1), number of atoms, all rules, basic rules, constraint rules, choice rules, and weight rules of the grounded program. For feature computation, a runtime limit of 900 CPU seconds per instance was used. Any instance for which the complete set of features could not be computed within 900 seconds was removed from the set of candidate instances. This led to the removal of 52 instances from the `SAT-Application` set, 2 from the `SAT-Crafted` set, and 3 from the `ASP` set.

Execution Environment and Solver Settings. All our experiments were performed on a computer cluster with dual Intel Xeon E5520 quad-core processors (2.26 GHz, 8192 KB cache) and 48 GB RAM per node, running Scientific Linux

[7] `http://asparagus.cs.uni-potsdam.de/`

(2.6.18-308.4.1.el5). Each solver run was limited to a runtime cutoff of 900 CPU seconds. Furthermore, we set parameter e in our benchmark selection procedure to 10, i.e., instances solved by all solvers within 90 CPU seconds were discarded, and the number of instances to select (n) to 200 for SAT (because of the relatively small base sets) and 300 for ASP. After filtering out *too hard* instances (Line 1 of Algorithm 1), 404 instances remained in `SAT-Application`, 506 instances in `SAT-Crafted` and 2190 instances in `ASP`; after filtering out *too easy* instances (Line 2), we obtained sets of size 393, 425, and 1431, respectively.

Clustering. To cluster the instances based on their features (Line 3), we applied k-means 100 times with different randomised initial cluster centroids. To find the optimal number of clusters, we gradually increased the number of clusters (starting with 2) until the quality of the clustering, assessed via 10-fold cross validation and 10 randomised repetitions of k-means for each fold, did not improve any further [16]. This resulted in 13 clusters for each of the two SAT sets, and 25 clusters for the ASP set.

Selection. To measure the hardness of a given problem instance, we used the average runtime over all representative solvers. We considered a cluster to be over-represented (Line 8) if more than 20% of the final set size (n) were selected for SAT, and more than 5% in case of ASP; the difference in threshold was motivated by the fact that substantially more clusters were obtained for the `ASP` set than for `SAT-Application` and `SAT-Crafted`.

Algorithm Configuration. After generating the benchmark sets `SAT-Application`*, `SAT-Crafted`* and `ASP`* using our automated selection procedure, these sets were evaluated by assessing their Q^*-scores. To this end, we used the freely available, state-of-the-art algorithm configurator *ParamILS* [18] to configure the SAT and ASP solver *clasp* (2.0.6). *clasp* is a competitive solver in several areas of Boolean constraint solving[8] that is highly parameterized, exposing 46 performance-relevant parameters for SAT and 51 for ASP. This makes it particularly well suited as a target for automated algorithm configuration methods and hence for evaluating our instance sets. Following standard practice, for each set, we performed 10 independent runs of *ParamILS* of 2 CPU days each and selected from these the configuration with the best training performance as the final result of the configuration process for each instance set.

Sampling Distributions. One of the main input parameters of Algorithm 1 is the sampling distribution. With the help of our Q^*-score criterion, three distributions are assessed: a normal (Gaussian) distribution, a log-normal distribution, and an exponential distribution. The parameters of these distributions were set to the empirical statistics (e.g., empirical mean and variance) of the hardness distribution over the base sets. The log-normal and exponential distributions have fat right tails and typically reflect better the runtime behaviour of solvers for NP problems than the normal distribution. However, when using the average runtime as our hardness metric, the instances sampled using a normal distribution are not necessarily atypically easy. For instance, an instance i, on which

[8] *clasp* won several first places in previous SAT, PB and ASP competitions.

Sampling-	PAR10 on I			PAR10 on I^*			Q*-score
Distribution	c_{def}	c_I	c_{I^*}	c_{def}	c_I	c_{I^*}	
`SAT-Application`							
Normal	4629	4162	3997	3410	2667	1907	1.46
Log-Normal	4629	4162	4683	3875	2601	3487	0.66
Exponential	4629	4162	4192	2969	2380	2188	1.08
`SAT-Crafted`							
Normal	5226	5120	5056	2429	2155	1752	1.25
Log-Normal	5226	5120	5184	3359	3235	3184	1.04
Exponential	5226	5120	5072	1958	1819	1523	1.21
`ASP`							
Normal	2496	1239	1072	1657	705	557	1.46
Log-Normal	2496	1239	1128	3136	1173	678	1.90
Exponential	2496	1239	1324	1648	710	555	1.20

Table 1: Comparison of set qualities of the base sets I and benchmark sets I^* generated by Algorithm 1; evaluated with Q^*-Scores with $I_1 = I^*$, $I_2 = I$, *clasp* as algorithm A and PAR10-scores as performance metric m

half of the representative solvers have a timeout while the other half solve the instance in nearly no time, has an average runtime of half of the runtime cutoff. Therefore, the instance is medium hard and will be likely selected by using the normal distribution.

In Table 1, we compare the benchmark sets we obtained from the base sets `SAT-Application`, `SAT-Crafted` and `ASP` when using these three types of distributions, based on their Q^*-scores. On the left of the table, we show the PAR10 performance on the base set I of the default configuration of *clasp* (c_{def}; we use this as a baseline), the configuration c_I found on the base set I, and the configuration c_{I^*} found on the selected set I^*; this is followed by the performance on the benchmark sets I^* generated using our new algorithm. The last column reports the Q^*-score values for the pairs of sets I and I^*.

For all three instance sets, the Q^*-scores obtained via the normal distribution were larger than 1.0, indicating that c_{I^*} performed better than c_I and the set obtained from our benchmark selection algorithm, I^*, proved to be a good alternative to the entire base set I. Although on the `ASP` set, by using the log-normal distribution a larger Q^*-score (1.90) was obtained than for the normal distribution (1.46), on the `SAT-Application` set, using the log-normal distribution did not produce good benchmark sets. When using exponential distributions, Q^*-scores are larger than 1.0 in all three cases, but smaller than those obtained with normal distributions.

When using the normal distribution, configuration c_{I^*} performed better than c_I on both sets I and I^* (implying $Q_{I_1}(I_2) > 1.0$). Therefore, configuration on the selected set I^* leads to faster (and more robust) configurations than on the base set I. Furthermore, the benchmark sets produced by our algorithm

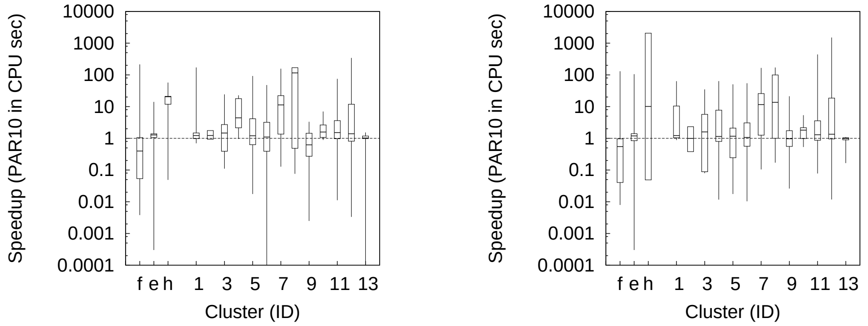

Fig. 1: Boxplots indicating the median, quartiles minimum and maximum speedup achieved on the instance clusters within the base set `SAT-Application`; (left) compares $c_{default}$ and c_I (high values are favourable for c_I); (right) compares $c_{default}$ and c_{I^*} (high values are favourable for c_{I^*}); special clusters: S_f uncompleted feature computation; S_e too easy, S_h too hard;

are smaller and easier than the respective base sets. Hence, less CPU time is necessary to assess the performance of an algorithm on those benchmark sets. For instance, the default configuration of *clasp* needed 215 CPU hours on the base `ASP` set and only 25 CPU hours on the benchmark set `ASP`*. For developing a new algorithm or configuring an algorithm (manually or automatically), fast and informative assessment, as facilitated by our new benchmark set generation algorithm, is very important.

Cluster Assessment. An additional advantage of Algorithm 1 is the fact that it produces a feature-based instance clustering, which can be further used to assess more precisely the performance of algorithms (or configurations). Normally, the performance of an algorithm is assessed over an entire instance set, but with the help of instance clusters, the performance can be assessed on different types of instances. This is useful, for example, in the context of developing a robust solver which should perform equally well across different types of instances. An example for such a solver is the CPLEX solver for mixed integer programming (MIP) problems, which is designed to perform well over a broad range of application contexts, each of which gives rise to different types of MIP instances.

The box plots in Figure 1 show the speedups (y-axis) of the configurations c_I (left) and c_{I^*} (right; while sampling with a normal distribution) against the default configuration c_{def} of *clasp* on each cluster $S_{1..13}$ (x-axis) within the `SAT-Application` base set. Furthermore, three special clusters contain the instances that were discarded in Algorithm 1 because, feature computation could not be completed (S_f), they were too easy (S_e), or too hard (S_h).

The comparison against a common baseline, here: the default configuration, helps to determine whether the new algorithm improved only on some types of instance or on all. For instance, configuration c_I (configured on the base set;

left plot) improved the performance by two orders of magnitude on cluster S_8 but is slightly slower on S_9. However, configuration c_{I^*} (configured on the set generated by Algorithm 1; right plot) achieved better median performance on all clusters except for S_f. In addition, the comparison between both plots reveals that c_{I^*} produces fewer outliers than c_I, especially on clusters S_6, S_9, S_{11} and S_{13}. Similar results (not shown here) were obtained for `SAT-Crafted` and `ASP`. Therefore, c_{I^*} can be considered to be a more robust improvement over c_{def} than c_I.

We believe that the reason for the robustness of configuration c_{I^*} lies in the fact that the (automatic) configuration process tends to be biased by instance types that are highly represented in a given training set. Since Algorithm 1 produces sets I^* that cover instance clusters more evenly than the respective base sets I, the configuration process is naturally guided more towards robust performance improvements across all clusters.

Particular attention should be paid to the special clusters S_f, S_e and S_h for the assessment of c_{I^*}, because the instances contained in these clusters are not at all represented in I^*. On none of our experiments with the three types of sampling distributions did we ever observe that the performance of c_{I^*} on the *too hard* instances S_h decreased; in fact, it sometimes increased. In contrast, the performance on the *too easy* instances S_e and instances with no features S_f was less consistent, and we observed speedups between 300 and 0.1 in comparison to c_I. Therefore, the threshold for filtering too easy instances e should be set conservatively (below 10%), to ensure that not too many too easy instances are discarded (we note that this is in contrast to common practice in SAT competitions).

Furthermore, our Algorithm 1 ensures that no cluster is over-represented, but does not ensure a sufficient representation of all clusters in the selected set. For instance, cluster S_4 has 141 instances in the base `ASP` set but only one instance in `ASP`* set (with normal distribution). Nevertheless, a low representation of a cluster in the selected set did not necessarily harm the configuration process, and in most observed cases, the configuration c_{I^*} performed as well as c_I on the under-represented clusters.

6 Conclusions and Future Work

In this work, we have introduced an algorithm for selecting instances from a base set or distribution to form an effective and efficient benchmark set. We consider a benchmark set to be effective, if a solver configured on it performs at least as well as when configured on the original set, and we consider it to be efficient, if the instances in it are on average easier to solve than those in the base set. By using such benchmark sets, the computational resources required for assessing the performance of a solver can be reduced substantially. Our benchmark selection procedure can use arbitrary sampling distributions; yet, in our experiments, we found that using a normal (Gaussian) distribution is particularly effective. Since our approach filters out instances considered too easy or too hard for the solver under consideration, it can lead to a situation where the performance of a given solver, when configured on the benchmark set, becomes worse on those discarded instances. However, the risk of worsening the performance on too hard instances can be reduced by setting the runtime cutoff of the selection process higher than in the actual benchmark. Then, the selected set contains very challenging instances under the runtime cutoff in the benchmark, which are yet known to be solvable. We have also demonstrated that clustering of instances based on instance features facilitates diagnostic assessments of the degree to which a solver performs well on specific types of instances or across an entire, heterogeneous benchmark set. Our work reported here is primarily motivated by the desire to develop solvers that perform robustly well across a wide range of problem instances, as has been (and continues to be) the focus in developing solvers for many hard combinatorial problems.

In future work, it may be interesting to ensure that semantically different types of instances, such as satisfiable and unsatisfiable instances in the case of SAT, are represented evenly or equivalently as in a given base set. Furthermore, one could consider more sophisticated ways to assess the over-representation of feature-based clusters and to automatically adjust the sampling process based on the number of clusters and their sizes. Finally, we believe that it would be interesting to study criteria for assessing the robustness of solver performance across clusters and to use such criteria for automatic algorithm configuration.

Acknowledgments. B. Kaufmann, T. Schaub and M. Schneider were partially supported by DFG under grants SCHA 550/8-3 and SCHA 550/9-1. H. Hoos was supported by an NSERC Discovery Grant and by the GRAND NCE.

References

1. Balint, A., Belov, A., Järvisalo, M., Sinz, C.: Application and hard combinatorial benchmarks in SAT challenge 2012. In Proc. of SAT Challenge 2012: Solver and Benchmark Descriptions. Volume B-2012-2 of Department of CS Series of Publications B, University of Helsinki (2012) 69–71
2. Berre, D., Roussel, O., Simon, L.: (2009) `http://www.satcompetition.org/2009/BenchmarksSelection.html`, last visited. 09-03-2012.
3. van Gelder, A.: Careful ranking of multiple solvers with timeouts and ties. In Proc. of SAT'11. Volume 6695 of LNCS., Springer-Verlag (2011) 317–328
4. Hoos, H., Stützle, T.: Stochastic Local Search: Foundations and Applications. Elsevier/Morgan Kaufmann (2004)
5. Sinz, C.: Visualizing SAT instances and runs of the DPLL algorithm. Journal of Automated Reasoning **39** (2007) 219–243
6. Nudelman, E., Leyton-Brown, K., Hoos, H., Devkar, A., Shoham, Y.: Understanding random SAT: Beyond the clauses-to-variables ratio. In Proc. of CP'04. Volume 3258 of LNCS., Springer-Verlag (2004) 438–452
7. Xu, L., Hutter, F., Hoos, H., Leyton-Brown, K.: SATzilla: Portfolio-based algorithm selection for SAT. Journal of Artificial Intelligence Research **32** (2008) 565–606

8. Kadioglu, S., Malitsky, Y., Sabharwal, A., Samulowitz, H., Sellmann, M.: Algorithm selection and scheduling. In Proc. of CP'11. Volume 6876 of LNCS., Springer-Verlag (2011) 454–469
9. Hutter, F., Hoos, H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In Proc. of LION'11. Volume 6683 of LNCS., Springer-Verlag (2011) 507–523
10. Kadioglu, S., Malitsky, Y., Sellmann, M., Tierney, K.: ISAC – instance-specific algorithm configuration. In Proc. of ECAI'10, IOS Press (2010) 751–756
11. Brglez, F., Li, X., Stallmann, F.: The role of a skeptic agent in testing and benchmarking of sat algorithms (2002)
12. Xu, L., Hutter, F., Hoos, H., Leyton-Brown, K.: Evaluating component solver contributions to portfolio-based algorithm selectors. In Proc. of SAT'12. Volume 7317 of LNCS., Springer-Verlag (2012) 228–241
13. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T., Schneider, M., Ziller, S.: A portfolio solver for answer set programming: Preliminary report. In Proc. of LPNMR'11. Volume 6645 of LNAI., Springer-Verlag (2011) 352–357
14. O'Mahony, E., Hebrard, E., Holland, A., Nugent, C., O'Sullivan, B.: Using case-based reasoning in an algorithm portfolio for constraint solving. In AICS'08. (2008)
15. Hamerly, G., Elkan, C.: Learning the k in k-means. In Proc. of NIPS'03, MIT Press (2003)
16. Hill, T., Lewicki, P.: Statistics: Methods and Applications. StatSoft (2005)
17. Bayless, S., Tompkins, D., Hoos, H.: Evaluating instance generators by configuration. Technical report, Department of Computer Science, University of British Columbia (2012). **To appear**
18. Hutter, F., Hoos, H., Leyton-Brown, K., Stützle, T.: ParamILS: An automatic algorithm configuration framework. Journal of Artificial Intelligence Research **36** (2009) 267–306
19. Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., Schneider, M.: Potassco: The Potsdam answer set solving collection. AI Communications **24**(2) (2011) 105–124
20. Audemard, G., Simon, L.: Glucose 2.1. in the SAT challenge 2012. In Proc. of SAT Challenge 2012: Solver and Benchmark Descriptions. Volume B-2012-2 of Department of CS Series of Publications B, University of Helsinki (2012) 23–23
21. Yasumoto, T.: Sinn. In Proc. of SAT Challenge 2012: Solver and Benchmark Descriptions. Volume B-2012-2 of Department of CS Series of Publications B, University of Helsinki (2012) 61–61
22. Biere, A.: Lingeling and friends entering the SAT challenge 2012. In Proc. of SAT Challenge 2012: Solver and Benchmark Descriptions. Volume B-2012-2 of Department of CS Series of Publications B, University of Helsinki (2012) 33–34
23. Cai, S., Luo, C., Su, K.: CCASAT: Solver description. In Proc. of SAT Challenge 2012: Solver and Benchmark Descriptions. Volume B-2012-2 of Department of CS Series of Publications B, University of Helsinki (2012) 13–14
24. Giunchiglia, E., Lierler, Y., Maratea, M.: Answer set programming based on propositional satisfiability. Journal of Automated Reasoning **36**(4) (2006) 345–377
25. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artificial Intelligence **138**(1-2) (2002) 181–234
26. Syrjänen, T.: Lparse 1.0 user's manual