Quantifying Homogeneity of Instance Sets for Algorithm Configuration

Marius Schneider[1] and Holger H. Hoos[2]

[1] University of Potsdam, manju@cs.uni-potsdam.de
[2] Unversity of British Columbia, hoos@cs.ubc.ca

Abstract. Automated configuration procedures play an increasingly prominent role in realising the performance potential inherent in highly parametric solvers for a wide range of computationally challenging problems. However, these configuration procedures have difficulties when dealing with inhomogenous instance sets, where the relative difficulty of problem instances varies between configurations of the given parametric algorithm. In the literature, instance set homogeneity has been assessed using a qualitative, visual criterion based on heat maps. Here, we introduce two quantitative measures of homogeneity and empirically demonstrate these to be consistent with the earlier qualitative criterion. We also show that according to our measures, homogeneity increases when partitioning instance sets by means of clustering based on observed runtimes, and that the performance of a prominent automatic algorithm configurator increases on the resulting, more homogenous subsets.

Keywords: Quantifying Homogeneity, Empirical Analysis, Parameter Optimization, Algorithm Configuration

1 Introduction

The automated configuration of highly parametric solvers has recently lead to substantial improvements in the state of the art in solving a broad range of challenging computational problems, including propositional satisfiability (SAT) [1,2,3], mixed integer programming (MIP) [4] and AI planning [5]. Broader adoption of this approach is likely to lead to a fundamental change in the way effective algorithms for NP-hard problems are designed [6], and the design and application of automated algorithm configuration techniques is a very active area of research (see, e.g., [7,8,9]).

One fundamental challenge in automated algorithm configuration arises from the fact that the relative difficulty of problem instances from a given set or distribution may vary between different configurations of the algorithm to be configured. This poses the risk that an iterative configuration process is misguided by the problem instances considered at early stages. For this reason, the performance of ParamILS [10,9], one of the strongest and most widely configuration procedures currently available, significantly depends on the ordering of the problem instances used for training [1,9], and the same can be expected to hold for other algorithm configuration techniques. Therefore, the question to which degree the relative difficulty of problem instances varies between configurations of a parametric algorithm is of considerably interest. Indeed, precisely this question has been addressed in recent work by Hutter et al. [11], who refer to instance sets for which the same instances are easy and hard for different configuration as *homogeneous* and ones for which this is markedly not the case as *inhomogeneous*. They state that inhomogeneous instance sets are "problematic to address with both manual and automated methods for offline algorithm configuration" [11] and list three approaches for addressing this issue: clustering of homogeneous instance sets [12,13], portfolio-based algorithm selection [14,15] and per-instance algorithm configuration [16,17]. They furthermore use a heat map visualization to qualitatively assess homogeneity.

In the work presented in the following, we introduce two *quantitative* measures of instance set homogeneity and explore to which extent these may provide a basis for assessing the efficacy of state-of-the-art algorithm configuration approaches. Like the heat maps of Hutter et al., our homogeneity measures are solely based on the performance of a given set of configurations and do not make use of problem-specific instance features. We show that on well-known algorithm configuration scenarios from the literature, our new measures are consistent with previous qualitative assessments. We further demonstrate that clustering instance sets can produce more homogenous subsets, and we provide preliminary evidence that automated algorithm configuration applied to those subsets tends to produce better results, which indicates that our homogeneity measures behave as intended.

The remainder of this paper is structured as follows: After a brief survey of related work (Section 2), we present our homogeneity measures (Section 3). This is followed by a brief description of clustering methods, which we subsequently use to partition instance set into more homogenous subsets (Section 4). We then present an experimental evaluation of our new measures, in terms of their consistency with the earlier qualitative assessment by Hutter et al., and in terms of the extent to which they are affected by clustering based on run-time measurements and on problem-dependent features (Section 5). Finally, we provide some general insights as well as a brief outlook on future work (Section 6).

2 Related Work

We are not aware of any existing work focused on homogeneity of instance sets given a solver as a central theme. However, there is a close conceptual relationship with prior work on portfolio-based algorithm selection and feature-based clustering of instances.

2.1 Portfolio-based algorithm selection

The key idea behind portfolio-based algorithm selection is, given a set S of solvers for a given problem, to map any given problem instance to a solver from S that can be expected to solve it most effectively (see [18]). Perhaps the best-known realization of this idea is the *SATzilla*approach [15], which has produced SAT solvers that won numerous medals in the 2007 and 2009 SAT competitions. *SATzilla* uses so-called empirical hardness models [19,20] for predicting performance based on cheaply computable instance features, and selects the solver to be applied to a given instance based on these performance predictions. Similar techniques have been successfully applied to several other problems (see, e.g., [4,21,22,23]).

In general, the problem of learning a mapping from solvers to instances can be seen as a multiclass classification problem and attacked using various machine learning techniques (see, e.g., [24,25]). Regardless of how the mapping is learned and carried out, a portfolio-based algorithm selector partitions any given set of instances Ω into subsets Ω_j, such that each Ω_j consists of the instances for which one particular solver from the given portfolio, say, $s_j \in S$, is selected. Ideally, each instance gets mapped to the solver that performs best on it; in this case, s_j dominates all other solvers in subset Ω_j. Although, unlike portfolio-based algorithm selection methods such as *SATzilla*, our approach does not use any problem-specific instance features, one of the homogeneity measures we introduce in Section 3.1 is based on the intuition that a given instance set is perfectly homogeneous if, and only if, a single solver (or in our case: solver configuration) dominates all others on all instances from the set.

2.2 Feature-based instance clustering

A rather different approach to algorithm selection underlies the more recent *ISAC* procedure [12]. Here, instances are clustered based on problem-dependent instance features. Next, an automated configuration procedure is used to find a good configuration for each of the instance subsets thus obtained. Unlike the clustering approaches we consider in our work, *ISAC* does not use runtimes of certain configurations of the given target algorithm for the clustering. The algorithm selector produced by *ISAC* maps each instance to the configuration associated with the nearest cluster, as determined based on instance features.

CluPaTra [13] also partitions instance sets based on instance features and subsequently optimizes parameters of the given target solver for each cluster. Lindawati et al. [13] have shown that clustering instance sets in this way leads to better performance than using random clustering of the same instances. This supports the idea that a parametric solver has a higher configuration potential on clustered instance sets if the clustering improves the homogeneity of these sets with respect to the parametric solver.

Our approach to clustering instance sets, explained in detail in Section 4, differs from *ISAC*and *CluPaTra*by not using any problem-specific instance features. While such features can be quite cheap to compute, expert knowledge is required to define and implement them. In addition, their use is based on the assumption that "[...] instances with alike features behave similarly under the same algorithm" [12]. We are not aware of any published work that provides stringent support for this hypothesis, and existing work (such as [12,13]) does not directly investigate it.[3] The homogeneity measures we introduce here offer a way of assessing to which extent clustering based on problem-

[3] The fact that problem instances that are indistinguishable with respect to simple syntactic features, such as critically constrained Random-3-SAT instances with a fixed number of variables, have been observed to vary substantially in difficulty for state-of-the-art solvers for the respective problems seems to contradict this hypothesis; however, one could conjecture that such instances might differ in more sophisticated, yet still cheaply computable features.

dependent features produces sets of instances for which different configurations of the same target algorithm show consistent performance rankings.

3 Homogeneity Measures

In this section, we deal with theoretical considerations to develop homogeneity measures for analyzing instance sets as motivated in Section 1. Intuitively, we characterize homogeneity as follows: An instance set $\varOmega$ is homogeneous for a given set $\varPhi$ of configurations of a parametric solver if the relative performance of the configurations in $\varPhi$ does not vary across the instances in $\varOmega$. In many cases, there will be deviations from perfect homogeneity, and because the degree to which these variations occur is of interest, we want to consider real-valued measures of instance set homogeneity.

Unfortunately, for most interesting parametric solvers, the configuration spaces are far too big to permit the evaluation of all configurations. For example, the discretized configuration space of the highly parametric SAT and ASP solver *Clasp* [26] is of size $\approx 10^{18}$. Therefore, following Hutter et al. [11], we consider sets of randomly sampled configurations as a proxy for the entire space. Somewhat surprisingly, even for relatively small samples, this rather simplistic approach turns out to be quite effective for optimizing the homogeneity of instance sets, as will become evident from the empirical results presented in Section 5.

To formally define homogeneity measures, we use $\varPhi$ to denote the space of all configurations ($\phi \in \varPhi$ for individual configurations), $\varPhi_r \subset \varPhi$ for a subset of n configurations sampled uniformly at random from $\varPhi$, and $\varOmega$ for an instance set ($\omega \in \varOmega$ for individual instances).

3.1 Ratio Measure - Similarity to the Oracle

Our first measure is motivated by our practical approach to determine whether a portfolio solver approach is useful for an instance set. The runtimes of all configurations (or solvers) in the portfolio are measured for each instance. Based on the sum of their runtimes over the given instance set, we compare the performance of the best configuration and that of the oracle (sometimes also called *virtual best solver*)[4] constructed from all the sampled configurations; if their performance is equal, there is one dominant configuration for the entire instance set, and portfolio-based selection offers no advantage over statically choosing this dominant configuration. We call such an instance set homogenous w.r.t. the given set of configurations. This approach corresponds to the interpretation of portfolio solvers given in Section 2.1.

Following this intuition, we define the *ratio measure*, Q_{Ratio}, to measure homogeneity based on the ratio of the runtimes between the best configuration and the oracle, as shown in Equations 1 to 3, where $t'_{Oracle(\varPhi_r)}(\varOmega)$ represents the performance of the oracle and $t'_{\phi^*}(\varOmega)$ that of the best configuration in $\varPhi_r$.

[4] The performance of the oracle solver on a given instance is the minimum runtime over all given configurations/solvers.

$$Q_{Ratio}(\varPhi_r, \varOmega) = 1 - \frac{t'_{Oracle(\varPhi_r)}(\varOmega)}{t'_{\phi^*}(\varOmega)} \text{ with } Q_{Ratio} \in \; [0, 1[\tag{1}$$

s.t.

$$t'_{\phi}(\varOmega) = \sum_{i}^{|\varOmega|} t(\omega_i, \phi) \text{ and } t'_{Oracle(\varPhi_r)}(\varOmega) = \sum_{i}^{|\varOmega|} \min_{\phi \in \varPhi_r} t(\omega_i, \phi) \tag{2}$$

$$\phi^* \in \arg\min_{\phi \in \varPhi_r} t'_{\phi}(\varOmega) \tag{3}$$

Q_{Ratio} is defined such that a value of 0 corresponds to minimal inhomogeneity, and higher values characterize increasingly inhomogeneous instance sets.

3.2 Variance Measure - Performance Similarity

The intuition behind our second measure is closely related to the question whether different evaluators rate a set of products similarly. More precisely, we want to determine whether m products (configurations) are rated similarly by n evaluators (instances) based on a given evaluation measure (runtime). This setting is similar to that addressed by the Friedman hypothesis test; however, the Friedman test is not directly applicable in our context, in part because we are typically dealing with noisy and censored runtimes.

Our *variance measure* is based on the general idea of assessing instance set homogeneity by means of the variances in runtimes over instances for each given configuration. An instance set is perfectly homogenous, if (after compensating for differences in instance difficulty independent of the configurations considered, i.e., for situations in which certain instances are solved faster than others by all configurations) for every given configuration, all instances are equally difficult.

To account for differences in instance difficulty that are independent of the configurations considered, we perform a standardized z-score normalization of the performance of configurations on instances such that for any given instance, the distribution of performance (here: log-transformed runtime) over configurations has mean zero and variance one. As we will see in Section 5.2, these distributions are often close to log-normal, which justifies standardized z-score transformation on log-transformed runtime measurements.

Formally, if $Var(t^*_{\phi}(\varOmega))$ is the variance of the log-transformed, standardized z-score normalized runtimes of configuration $\phi \in \varPhi_r$ over the instances in the given set $\varOmega$, we define the *variance measure* Q_{Var} as follows:

$$Q_{Var}(\varPhi_r, \varOmega) = \frac{1}{|\varPhi_r|} \sum_{\phi \in \varPhi_r} Var(t^*_{\phi}(\varOmega)) \tag{4}$$

As in the case of Q_{Ratio}, $Q_{Var} \geq 0$, where $Q_{Var} = 0$ characterizes perfectly homogenous instance sets, while higher values correspond to increasingly inhomogeneous sets.

4 Clustering of Homogeneous Subsets

Based on the homogeneity measures introduced in Section 3, instances within the given set Ω can be clustered with the goal of optimizing homogeneity of the resulting subsets of Ω.

We note that in order to calculate the values of Q_{Ratio} and Q_{Var}, runtimes for each configuration on each instance have to be measured; each of these runtimes can be interpreted as an observation on the behavior of the given parametric algorithm on the instance. Under this interpretation of the data, classical clustering approaches can be applied, in particular, K-Means [27], Gaussian Mixtures [28] and Hierarchical Agglomerative Clustering [29]. While many more clustering approaches can be found in the literature, these methods are amongst the most prominent and widely used classical clustering approaches based on observations.

Agglomerative Hierarchical Clustering[29] iteratively merges the clusters with the lowest distance, where distance between clusters can be defined in various ways. As an alternative to clustering based on distances between the observation vectors, we also explored a variant that always merges the two clusters resulting in the best homogeneity measure (of the merged cluster). Clusters will be merged until a termination criterion is satisfied (e.g., a given number of desired clusters is reached). Unfortunately, the property $\Omega_1 \subset \Omega_2 \Rightarrow Q(\Omega_1) < Q(\Omega_2)$ cannot be guaranteed for arbitrary instance sets Ω_1 and Ω_2, where Q is either of our homogeneity measures. This means that merging two clusters of instances does not necessarily result in a strict improvement in homogeneity. Therefore, we analyzed how our homogeneity measures vary as the number of clusters increases (see Section 5.4).

5 Experiments

In this section, we evaluate empirically how our approach can be used to analyze the homogeneity of instance sets on different kinds of solvers. First, we explain our experimental setting. Next, we characterize the distributions of runtimes on a given instance over solver configurations, which matter in terms of the standardized z-score normalization underlying our variance-based homogeneity measure, Q_{Var}. Then, we investigate to which degree our homogeneity measures agree with the earlier, qualitative analysis of homogeneity by Hutter et al. [11]. Finally, we investigate the question whether algorithm configurators, here ParamILS, perform better on more homogeneous instance sets, as obtained by clustering instances from large and diverse sets.

5.1 Data and Solvers

We used the runtime measurements produced by Hutter et al. [11][5]; their data includes runtimes of the mixed integer programming (MIP) solver *CPLEX* on the instance sets *Regions100* (`CPLEX-Regions100`: 2000 instances, 50 sec cutoff) and *Orlib* (`CPLEX-Orlib`: 140 instances, 3000 sec cutoff); the local search SAT solver

[5] See `http://www.cs.ubc.ca/labs/beta/Projects/AAC/empirical_analysis/index.html`

Spear on the instance sets *IBM* (SPEAR-IBM: 100 instances, 3000 sec cutoff) and *SWV* (SPEAR-SWV: 100 instances, 3000 sec. cutoff); and the SAT solver *Satenstein* on the instance sets *QCP* (SATenstein-QCP: 2000 instances, 50 sec cutoff) and *SWGCP* (SATenstein-SWGCP: 2000 instances, 50 sec cutoff). In each case, runtime measurements were provided for 1000 solver configurations chosen uniformly at random. Instances that could not be solved by any configuration were excluded from further analysis of our homogeneity measures, as were configurations that could not solve any of our instances. (However, for the clustering performed in later experiments, these instances and configurations were *not* eliminated.)

We augments this extensive data set with addition runtime data for the successful ASP [30] and SAT solver *Clasp* [26] (in version 2.0.2). *Clasp* won the system competitions in the ASP competitions 2009 and 2011 and several gold medals in the 2009 and 2011 SAT competitions. As an open source project[6], *Clasp* is freely available. It also is a highly parametric solver with over fifty parameters, 38 of which we considered in this work (these all influence the solving process for SAT instances).

We applied *Clasp* to two subsets of the crafted and industrial/application benchmarks used in the SAT competitions between 2003 and 2009, dubbed CLASP-Crafted and CLASP-Industrial. Furthermore, we used a set of SAT-encoded bounded model checking problems [31] dubbed CLASP-IBM. We removed all instances for which the running time of every configurations of *Clasp* from a manually chosen set required less than 3 seconds or more than 600 seconds; this was done in order to avoid problems with inaccurate runtime measurements and excessive occurrence of timeouts as well as to ensure that all experiments could be completed within reasonable time. After this filtering step, we were left with 505 instances in the CLASP-Crafted set, 552 instances in CLASP-Industrial, and 148 instances in CLASP-IBM.

We measured runtimes for 32 configurations of *Clasp* chosen uniformly at random (from a total of $\approx 10^{18}$) for each instance, using a cutoff of 600 seconds per run.[7] These runtime measurements required a total of about 350 CPU hours. In the same way as done with the data of Hutter et al., instances that could not be solved by any of our 32 *Clasp* configuration were excluded from further analysis of our homogeneity measures, as were configurations that could not solve any of our instances.

All runs of *Clasp* were carried out on a Dell PowerEdge R610 with an Intel Xeon E5520 (2.26GHz), 48GB RAM running 64-bit Scientific Linux, while the runtime data of Hutter et al. [11] was measured on a 3.2GHz Intel Xeon dual core CPUs with 2GB RAM running Open SuseLinux 10.1.

5.2 Normalization and Distributions

Clearly, the distribution of runtimes over target algorithm configurations on a given problem instance depends on the semantics on the given target algorithm's parameters. As motivated in Section 3.2, our variance-based homogeneity measure requires normalization. The approach we have chosen for this normalization is based on our finding that

[6] http://potassco.sourceforge.net/

[7] This runtime data is available at
http://www.cs.uni-potsdam.de/wv/clusteredHomogeneity.

Instance Sets	normal	log-normal	exponential	Weibull
CPLEX-Orlib	0.988(0.002)	0.494(0.160)	0.906(0.016)	0.859(0.036)
SPEAR-IBM	0.911(0.048)	0.533(0.155)	0.911(0.025)	0.867(0.055)
SPEAR-SWV	0.473(0.419)	0.243(0.496)	0.689(0.134)	0.351(0.421)
SATenstein-QCP	0.992(0.003)	0.063(0.474)	0.840(0.033)	0.055(0.413)

Table 1: Quality of fit for distributions of runtime on given problem instances over randomly sampled sets of algorithm configurations, assessed using the Kolmogorov-Smirnov goodness-of-fit test: average rejection rate of test over instances (low values are good) and average p-values (for details see text).

the distributions of log-transformed running times tends to be normal, described in the following.

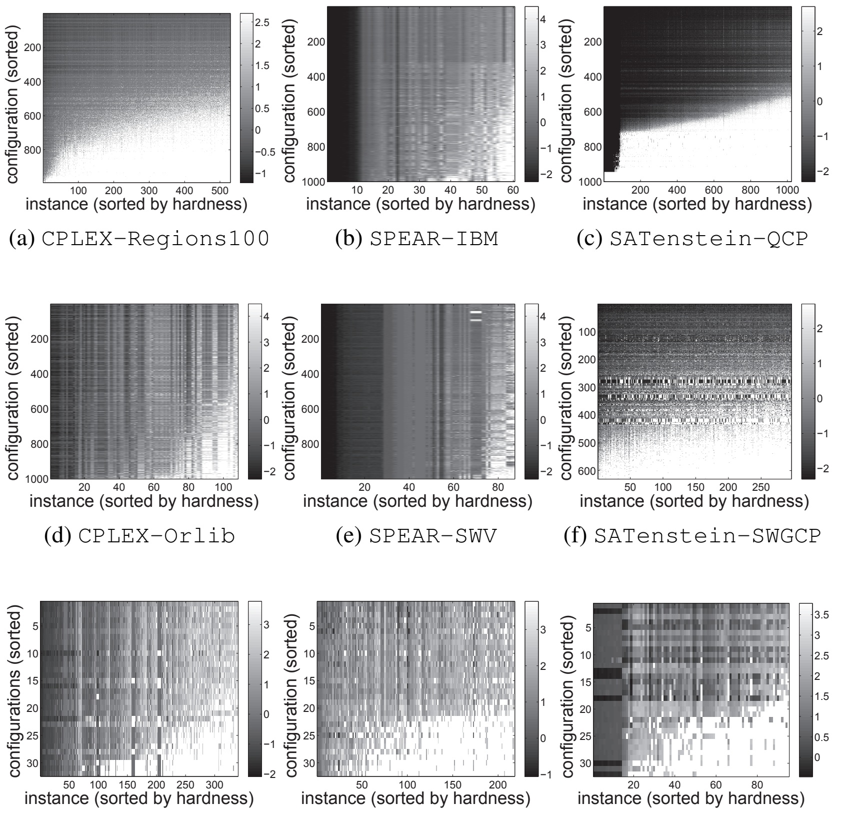
We used the Kolmogorov-Smirnov goodness-of-fit test (KS test) with a significance level of 0.05 to evaluate for each instance, whether the empirical distribution of runtimes over configurations was consistent with a log-normal, normal, exponential or Weibull distribution. We excluded the *Clasp* data from this analysis, since each distribution was only based on 32 data points, resulting in very low power of the KS test. Since the occurrence of a significant numbers of timeouts for a given instance renders the characterization of the underlying distributional family via a KS test impossible, we also eliminated all instances from our test on which more than half the given configuration timed out; since this would have left very few instances in the sets CPLEX-Regions100 and SATenstein-SWGCP, we did not consider these sets in our distributional analysis.

Table 1 shows the averaged test results over all remaining instances, where a result of 1 was recorded, if the respective KS test rejected the null hypothesis of distribution of the given type, and 0 otherwise; we also reported average p-values for each set. As can be seen from these results, in most cases, the distributions tend to be log-normal, whereas the three other types of distributions have much weaker support.

5.3 Evaluation of Homogeneity

Runtimes of instance sets on a set of configurations can be visualized with heat maps, as illustrated in Figure 1; following Hutter et al. [11], we have sorted configurations and instances according to their average PAR-10 scores and represented log-transformed runtimes using different shades of gray (where darker grays correspond to shorter runtimes). As noted in their work, cases where the relative difficulty of the instances varies between different configurations give rise to checkerboard patterns in these plots, and using this qualitative criterion, the CPLEX-Orlib and CLASP-IBM configuration scenarios appear to be rather inhomogeneous, in contrast to the homogeneous instance sets CPLEX-Regions100 and SATenstein-QCP.

As can be seen from the column labeled *unclustered* in Table 2, our variance-based measure is consistent with these earlier qualitative observations. (The remaining columns are discussed later.) In particular, the values for the homogenous sets CPLEX-Regions100 and SATenstein-QCP are low compared to the remaining instance sets, which show all clear signs of qualitative inhomogeneity. On the other



(a) CPLEX-Regions100 (b) SPEAR-IBM (c) SATenstein-QCP

(d) CPLEX-Orlib (e) SPEAR-SWV (f) SATenstein-SWGCP

(g) CLASP-Crafted (h) CLASP-Industrial (i) CLASP-IBM

Fig. 1: Heat maps of log transformed runtime data from the configuration scenarios studied by Hutter et al. [11] and three new scenarios using the *Clasp* ASP solver. (The diagrams were generated with the Matlab code provided by Hutter et al.)

hand, coarser checkerboard patterns do not always correspond to instance sets with higher variance measures for three reasons: (1) the data for *Clasp* is based on far fewer configuration, leading necessarily to coarser patterns in the plots; (2) deviations from uniformity appear more prominent towards the middle of our gray scale than towards the dark and light ends of the spectrum; and (3) large local deviations can have a large influence on the variance measure, but are not necessarily visually as prominent as smaller global variations.

Our *ratio measure* also shows the lowest values for the qualitatively homogenous sets CPLEX-Regions100 and SATenstein-QCP, but ranks the remaining sets differently from the variance measure. In fact, considering the definition of *ratio measure*, it becomes clear that in extreme cases, it may substantially disagree with the qualitative visual measure of Hutter et al.: For example, if one configuration dominates all others,

Instance Sets	*unclustered*		*configuration-based*		*feature-based*	
	Q_{Ratio}	Q_{Var}	Q_{Ratio}	Q_{Var}	Q_{Ratio}	Q_{Var}
`CPLEX-Regions100`	0.41	0.23	0.11	0.23	–	–
`CPLEX-Orlib`	0.50	0.71	0.35	0.40	–	–
`SPEAR-IBM`	0.68	0.75	0.40	0.64	0.58	0.64
`SPEAR-SWV`	0.74	0.89	0.20	0.72	0.43	0.77
`SATenstein-QCP`	0.30	0.20	0.25	0.11	0.37	0.17
`SATenstein-SWGCP`	0.62	0.35	0.62	0.35	0.68	0.41
`CLASP-Crafted`	0.86	0.39	0.82	0.41	0.79	0.35
`CLASP-Industrial`	0.81	0.58	0.71	0.50	0.71	0.54
`CLASP-IBM`	0.57	0.41	0.37	0.30	0.40	0.36

Table 2: Homogeneity measures Q_{Ratio} and Q_{Var} on the entire instance set (*unclustered*), after configuration-based Gaussian Mixture clustering in four sets (*configuration-based*), and after feature-based K-Means clustering also in four sets (*feature-based*). All measures are averages based on 4-fold cross validation (for details see text).

but the remaining configurations are highly inconsistent with each other in terms of their relative performance over the given instance set, the *ratio measure* would be very low, yet, the corresponding heat map would display a prominent checker-board pattern. This illustrates that reasonable and interesting measures of homogeneity, such as the *ratio measure* provide information that is not easily apparent from the earlier qualitative criterion. It also indicates that a single quantitative measure of homogeneity, such as our variance measure, may not capture all aspects of instance set homogeneity of interest in a given context.

5.4 Comparison of different Clustering Algorithms

We now turn our attention to the question whether partitioning a given instance set into subsets by means of clustering techniques leads to more homogenous subsets according to our ratio and variance measures, as one would intuitively expect. To investigate this question, we used the clustering approaches from Section 4, based on the observed runtimes in conjunction with Gaussian Mixtures and Agglomerative Hierarchical Clustering as well as for the direct optimization of the homogeneity measures using Agglomerative Hierarchical Clustering.

Inspired by *ISAC* [12], we also clustered our instance sets based on cheaply computable instance features [15], using ten runs of the K-Means algorithm for each set. (Preliminary experiments suggested that Gaussian Mixtures clustering on instances features does not yield results better than those produced by K-Means.) In addition, the instances were clustered uniformly at random to obtain a baseline against which the other clustering results could be compared. The SAT instance features were generated with the instance feature generator of *SATzilla* 2011 [15], which provides features based on graph representations of the instance, LP relaxation, DPLL probing, local search probing, clause learning, and survey propagation. Since we did not have feature computation code for MIP instances, we did not perform feature-based clustering on CPLEX-Orlib and CPLEX-Regions100.

To assess the impact of configuration-based clustering on instance set homogeneity, we used a 4-fold cross validation approach, where 3/4 of the configurations were used as a basis for clustering the instance set, and the remaining 1/4 was used for measuring instance homogeneity. (More then 4 folds could not be used, since that would have left too few configurations for measuring homogeneity.) The results in Table 2 and Figure 2 are averaged over the 4 folds, where within each fold, we combined the homogeneity measures for each cluster in the form of an average weighted by cluster size.

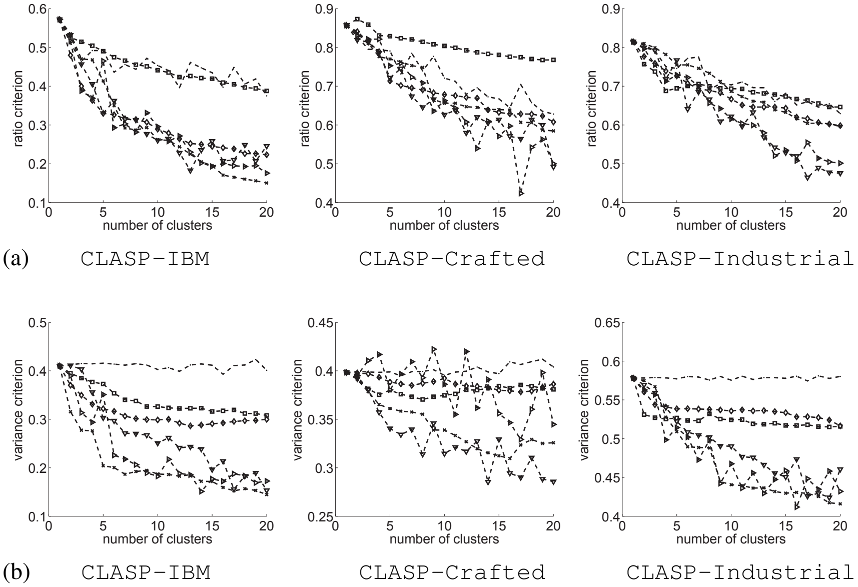
Figure 2 shows how our homogeneity measures vary with the number of clusters for instance sets CLASP-Crafted, CLASP-Industrial, and CLASP-IBM (the results for the other instance sets are qualitatively similar and have been omitted due to limited space). In most cases, Gaussian Mixture($\triangleright$) and the feature-based clustering ($\bigtriangledown$) lead to considerable improvements in the *ratio measure* (Figure 2a) compared to random clustering. The same holds w.r.t. the *variance measure* (Figure 2b), which also tends to be improved by agglomerative clustering ($\times$). The reasons for the oscillations seen for Gaussian Mixture clustering on CLASP-Crafted are presently unclear. Overall, with the exception of CLASP-Crafted, configuration-based Gaussian Mixture clustering tends to produce the biggest improvements in instance set homogeneity.

Interestingly, agglomerative clustering in which we directly optimized the *variance measure* or *ratio measure* tended to give good results on our training sets, but those results did not generalize well to our testing scenarios (in which a disjoint set of configurations was used for measuring homogeneity).

In Table 2, we present numerical results for Gaussian Mixture clustering of our instance sets into four subsets. (We chose four subsets, because the efficiency, measured as the number of clusters in proportion to the optimization of our homogeneity measures, peaked around this number of clusters.) As can be seen from these results, configuration- and feature-based clustering resulted in improvements in homogeneity for almost all instance sets, and configuration-based clustering, although computationally considerably more expensive, tends to produce more homogenous subsets than feature-based clustering. (Preliminary observations from further experiments currently underway suggest that even better results can be obtained from configuration-based clustering using K-Means with multiple restarts.) The fact that these results were obtained using 4-fold cross-validation on our configuration sets indicates that improved homogeneity w.r.t. the configurations considered in the clustering process generalizes to previously unseen configurations.

5.5 Evaluation of Configuration Improvement

The goal of our final experiment was to investigate the hypothesis that automatic algorithm configuration yields better results on more homogenous instance sets. Therefore, we compared the results from applying the same standard configuration protocol to some of our original instance sets and to their more homogenous subsets obtained by clustering. This should not be misunderstood as an attempt to design a practically useful configuration strategy based on homogeneity-improving clustering, which, in order to



(a) CLASP-IBM CLASP-Crafted CLASP-Industrial

(b) CLASP-IBM CLASP-Crafted CLASP-Industrial

Fig. 2: Q_{Ratio} (a) and Q_{Var} (b) for a varying number of clusters; $-$ random, $\times$ Agglomerative Clustering, $\triangleright$ Gaussian Mixture, $\square$ Agglomerative Clustering with Q_{Var} optimization, $\Diamond$ Agglomerative Clustering with Q_{Ratio} optimization, $\bigtriangledown$ feature-based K-Means clustering

be practical, would have to use cheaply computable features rather than the ones based on runtimes of a set of configurations used here (see, e.g., [12,17]).

For this experiment, we configured *Clasp* on the instance sets of CLASP-Crafted, CLASP-Industrial, and CLASP-IBM with the FocusedILS variant of the *ParamILS* framework (version 2.3.5) [9]. For each of CLASP-Crafted and CLASP-IBM, we conducted four independent runs of FocusedILS with a total time budget of 48 CPU hours per run, and for CLASP-Industrial, we performed 10 runs of FocusedILS of 72 CPU hours each, since this scenario was considerably more challenging. For each set, we then compared the default configuration of *Clasp* for SAT solving (*Default*) against a configuration optimized on the entire instance set (*Entire*), configurations optimized for each of the four subsets obtained by clustering (4 *Clusters*), the oracle performance (also called virtual best solver) over those four configurations (*Oracle*), and the performance obtained when running on each instance a version of *Clasp* optimized specifically for that instance by means of a single, 3 CPU hour run of FocusedILS (*Single Configuration*). With the exception of the last of these scenarios, the performance measurements reported in Table 3 were based on a set of test instances disjoint from the instances used for configuration, and those sets were obtained by random stratified splitting of each original clustered set into equal training and test sets.

The clustering method used in the context of these experiments was Gaussian Mixture clustering, based on the assumption that clusters should be normally distributed [12,32]

Instance Set	#	*Default*	*Entire Set*	4 *Clusters*	*Oracle*	*Single Configuration*
`CLASP-Crafted`	254	883(34)	422(15)	361(12)	51(9)	35(0)
`CLASP-Industrial`	276	1607(70)	1310(56)	1164(50)	721(30)	83(0)
`CLASP-IBM`	75	2125(26)	1220(15)	1216(15)	1210(13)	135(0)

Table 3: Runtimes in terms of PAR10 in CPU seconds (and number of timeouts) obtained by various configurations of *Clasp* along with (idealized) oracle and per-instance configuration performance (for details, see text).

and the of the clustering methods we considered, Gaussian Mixtures performed best on average in five out of six cases in Figure 2b. The target number of clusters was chosen to be four for the reasons explained in Section 5.4. Each instance subset thus obtained was split into a training and test set as previously explained, and *Clasp* was then configured for each of these training sets. After evaluating these configurations on the corresponding test sets, we aggregated the performance using weighted averaging, where the weights were given by the cluster sizes.

The results shown in Table 3 confirm that automated configuration of *Clasp* on the more homogenous instance sets obtained by clustering is more effective than configuration on the original, less homogenous instance sets. Not too surprisingly, algorithm selection between the resulting configurations can in principle yield additional improvements (as seen from the oracle results), and configuration on individual instances has the potential to achieve further, dramatic performance gains. We note that single-instance sets are, intuitively and by definition, completely homogenous and therefore represent an idealistic best-case scenario for automated algorithm configuration. Furthermore, the oracle performance provides a good estimate of the performance of a parallel portfolio of the respective set of configurations, whose performance is evaluated solely based on wallclock time.

6 Conclusions and Future Work

In this work, we introduced two quantitative measures of instance set homogeneity in the context of automated algorithm configuration. Our measures provide an alternative to an earlier qualitative visual criterion based on heat maps [11]; one of them, the *variance measure*, gives results that are highly consistent with the visual criterion, and both of them capture aspects of instance set homogeneity not easily seen from heat maps. Furthermore, we provided evidence that our measures are consistent with the previously informal intuition that more homogenous instance sets are more amenable to automated algorithm configuration (see, e.g., [13]).

The proposed homogeneity measures can be used directly to assess whether automated configuration of a given parametric algorithm using a particular instance set might be difficult due to instance set inhomogeneity. In addition, the *ratio measure* helps to assess the specific potential of portfolio-based approaches in a given configuration scenario, including instance-based algorithm configuration [15,12], portfolio multithreading [33] and sequential portfolio solving [34,35].

Unfortunately, like the previous qualitative approach, our quantitative homogeneity measures are computationally expensive. In future work, we plan to investigate how this computational burden can be reduced, for example, by using promising configurations encountered during algorithm configuration instead of randomly sampled ones.

Acknowledgements

We thank F. Hutter for providing the data and Matlab scripts used in [11], L. Xu for providing the newest version of the feature generator of *SATzilla*, and A. König and M. Möller for valuable comments on an early version of this work.
This work was partially funded by the DFG under grant SCHA 550/8-2 and by NSERC through a discovery grant to H. Hoos.

References

1. Hutter, F., Babić, D., Hoos, H., Hu, A.: Boosting verification by automatic tuning of decision procedures. In: Procs. FMCAD'07, IEEE Computer Society Press (2007) 27–34
2. KhudaBukhsh, A., Xu, L., Hoos, H., Leyton-Brown, K.: SATenstein: Automatically building local search sat solvers from components. In Boutilier, C., ed.: Procs. IJCAI'09, AAAI Press/The MIT Press (2009) 517–524
3. Tompkins, D., Balint, A., Hoos, H.: Captain Jack – New Variable Selection Heuristics in Local Search for SAT. In Sakallah, K., Simon, L., eds.: Procs. SAT'11. Volume 6695 of LNCS, Springer-Verlag (2011) 302–316
4. Hutter, F., Hoos, H., Leyton-Brown, K.: Automated configuration of mixed integer programming solvers. In Lodi, A., Milano, M., Toth, P., eds.: Procs. CPAIOR'10. Volume 6140 of LNCS, Springer-Verlag (2010) 186–202
5. Vallati, M., Fawcett, C., Gerevini, A., Hoos, H., Saetti, A.: Generating fast domain-specic planners by automatically conguring a generic parameterised planner. In: Procs. PAL2011. (2011) 21–27
6. Hoos, H.: Programming by optimisation. Communications of the ACM **to appear** (2012)
7. Ansótegui, C., Sellmann, M., Tierney, K.: A Gender-Based Genetic Algorithm for the Automatic Configuration of Algorithms. In Gent, I., ed.: Procs. CP'09. Volume 5732 of LNCS, Springer-Verlag (2009) 142–157
8. Birattari, M., Stützle, T., Paquete, L., Varrentrapp, K.: A racing algorithm for configuring metaheuristics. In Kaufmann, M., ed.: Procs. GECCO'09, ACM Press (2002) 11–18
9. Hutter, F., Hoos, H., Leyton-Brown, K., Stützle, T.: ParamILS: An automatic algorithm configuration framework. Journal of Artificial Intelligence Research **36** (2009) 267–306
10. Hutter, F., Hoos, H., Stützle, T.: Automatic algorithm configuration based on local search. In: Procs. AAAI'07, AAAI Press (2007) 1152–1157
11. Hutter, F., Hoos, H., Leyton-Brown, K.: Tradeoffs in the Empirical Evaluation of Competing Algorithm Designs. Annals of Mathematics and Artificial Intelligenc (AMAI), Special Issue on Learning and Intelligent Optimization **60**(1) (2011) 65–89
12. Kadioglu, S., Malitsky, Y., Sellmann, M., Tierney, K.: ISAC - Instance-Specific Algorithm Configuration. In Coelho, H., Studer, R., Wooldridge, M., eds.: Procs. ECAI'08, IOS Press (2010) 751–756
13. Lindawati, L., Lau, H., Lo, D.: Instance-based Parameter Tuning via Search Trajectory Similarity Clustering. Procs. LION'11. Volume 6683 of LNCS, Springer-Verlag (2011) 131–145

14. Gomes, C., Selman, B.: Algorithm portfolios. Journal of Artificial Intelligence **126**(1-2) (2001) 43–62
15. 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
16. Hutter, F., Hamadi, Y., Hoos, H.H., Leyton-Brown, K.: Performance prediction and automated tuning of randomized and parametric algorithms. In Benhamou, F., ed.: Procs. CP'06. Volume 4204 of LNCS, Springer-Verlag (2006) 213–228
17. Xu, L., Hoos, H., Leyton-Brown, K.: Hydra: Automatically configuring algorithms for portfolio-based selection. In Fox, M., Poole, D., eds.: Procs. AAAI'10, AAAI Press (2010) 210–216
18. Rice, J.: The algorithm selection problem. Advances in Computers **15** (1976) 65–118
19. Xu, L., Hoos, H., Leyton-Brown, K.: Hierarchical Hardness Models for SAT. In Bessiere, C., ed.: Procs. CP'07. Volume 4741 of LNCS, Springer-Verlag (2007) 696–711
20. Leyton-Brown, K., Nudelman, E., Shoham, Y.: Empirical hardness models: Methodology and a case study on combinatorial auctions. Journal of the ACM **56**(4) (2009) 1–52
21. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T., Schneider, M., Ziller, S.: A portfolio solver for answer set programming: Preliminary report. In Delgrande, J., Faber, W., eds.: Procs. LPNMR'11. Volume 6645 of LNCS, Springer-Verlag (2011) 352–357
22. Hutter, F., Hoos, H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. Procs. LION'11. Volume 6683 of LNCS, Springer-Verlag (2011) 507–523
23. Smith-Miles, K.: Cross-disciplinary perspectives on meta-learning for algorithm selection. ACM Computing Surveys **41**(1) (2008) 6:1–6:25
24. Guerri, A., Milano, M.: Learning Techniques for Automatic Algorithm Portfolio Selection. In de Mántaras, R.L., Saitta, L., eds.: Procs. ECAI'04, IOS Press (2004) 475–479
25. Kotthoff, L., Gent, I., Miguel, I.: A Preliminary Evaluation of Machine Learning in Algorithm Selection for Search Problems. In Borrajo, D., Likhachev, M., López, C., eds.: Procs. SoCS'11, AAAI Press (2011) 84–91
26. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: Conflict-driven answer set solving. In Veloso, M., ed.: Procs. IJCAI'07, AAAI Press/The MIT Press (2007) 386–392
27. MacQueen, J.B.: Some methods for classification and analysis of multivariate observations. In Cam, L., Neyman, J., eds.: Procs. Mathematical Statistics and Probability. Volume 1., University of California Press (1967) 281–297
28. Bishop, C.: Pattern Recognition and Machine Learning (Information Science and Statistics). 1st ed. 2006. corr. 2nd printing edn. Springer (2007)
29. Ward, J.: Hierarchical Grouping to Optimize an Objective Function. Journal of the American Statistical Association **58**(301) (1963) 236–244
30. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press (2003)
31. Zarpas, E.: Benchmarking SAT Solvers for Bounded Model Checking. In Bacchus, F., Walsh, T., eds.: Procs. SAT'05. Volume 3569 of LNCS, Springer-Verlag (2005) 340354
32. Hamerly, G., Elkan, C.: Learning the k in k-means. In: Procs. NIPS'03, Cambridge, MA, MIT Press (2003)
33. Hamadi, Y., Jabbour, S., Sais, L.: ManySAT: a parallel SAT solver. Journal on Satisfiability, Boolean Modeling and Computation **6** (2009) 245–262
34. Streeter, M., Golovin, D., Smith, S.: Combining Multiple Heuristics Online. In: Procs. AAAI'07, AAAI Press (2007) 1197–1203
35. Kadioglu, S., Malitsky, Y., Sabharwal, A., Samulowitz, H., Sellmann, M.: Algorithm Selection and Scheduling. In Lee, J., ed.: CP'10. Volume 6876 of LNCS, Springer-Verlag (2011) 454–469