On the Effectiveness of SBSE Techniques
Through Instance Space Analysis

Aldeida Aleti(✉)

Faculty of Information Technology, Monash University, Melbourne, Australia
aldeida.aleti@monash.edu

Abstract. Search-Based Software Engineering is now a mature area with numerous techniques developed to tackle some of the most challenging software engineering problems, from requirements to design, testing, fault localisation, and automated program repair. SBSE techniques have shown promising results, giving us hope that one day it will be possible for the tedious and labour intensive parts of software development to be completely automated, or at least semi-automated. In this talk, I will focus on the problem of objective performance evaluation of SBSE techniques. To this end, I will introduce Instance Space Analysis (ISA), which is an approach to identify features of SBSE problems that explain why a particular instance is difficult for an SBSE technique. ISA can be used to examine the diversity and quality of the benchmark datasets used by most researchers, and analyse the strengths and weaknesses of existing SBSE techniques. The instance space is constructed to reveal areas of hard and easy problems, and enables the strengths and weaknesses of the different SBSE techniques to be identified. I will present on how ISA enabled us to identify the strengths and weaknesses of SBSE techniques in two areas: Search-Based Software Testing and Automated Program Repair. Finally, I will end my talk with potential future directions of the objective assessment of SBSE techniques.

Keywords: Search-based software engineering · Instance space analysis

1 Instance Space Analysis for SBSE

Instance Space Analysis (ISA) has two main goals:

– to help designers of SBSE techniques (SBSET) gain insight into why some techniques are more or less suited to solve certain SBSE problems, thus devising new and better techniques that address any challenging areas, and
– to help software developers select the most effective SBSET for their software system.

ISA provides a way for objective assessment of the effectiveness of SBSE techniques. It has been applied to Search-Based Software Testing [5,6], Search-Based

Program Repair [1], machine learning [3], and optimisation [7]. The concept of instance space analysis was first introduced by Smith-Miles in her seminal work looking at the strengths and weaknesses of optimisation problems [7]. Understanding the effectiveness of an SBSE technique is critical for selecting the most suitable technique for a particular SBSE problem, thus avoiding trial and error application of SBSE techniques.

An overview of the ISA for SBSE is presented in Fig. 1. It starts with a set of problems $p \in P$ and a portfolio of SBSE techniques $t \in T$. For example, P can be a set of buggy programs and T can be a portfolio of Search-Based Program Repair techniques. The performance of each Search-Based Software Engineering Technique (SBSET) is measured for each problem instance as $y(t, p)$. For example, y can indicate whether a plausible patch has been found for a program p by the Search-Based Program Repair Technique t. The first step of ISA is to identify the significant features of problem instances ($f(p) \in F$) that have an impact on how easy or hard they are for a particular SBSET. A feature can be the complexity of code, as measured by code-based complexity metrics. An example is the coupling between object classes, which is a count of the number of other classes to which the current class is coupled [2]. In Search-Based Program Repair, features may include those that are related to the prediction of source code transformations on buggy code [10] and detection of incorrect patches [9].

Next, ISA constructs the footprints $(g(f(P))) \in R^2$ which indicate the area of strength for each SBSET. Finally, ISA applies machine learning techniques on the most significant features to learn a model that can be used for SBSET selection for future application.

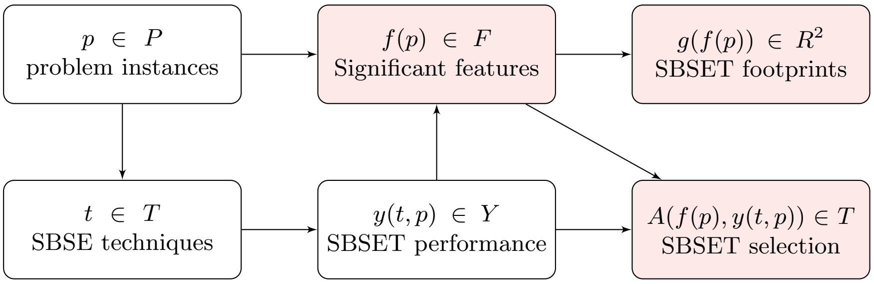

Fig. 1. An overview of instance space analysis for analysing the effectiveness of SBSE techqniques.

The features included in the feature space ($\mathcal{F}$) should be diverse and predictive of the performance of at least one algorithm. Hence, their selection requires careful consideration. They are domain specific and thus developing $\mathcal{F}$ requires significant domain knowledge. Features should be highly correlated with algorithm performance, not highly correlated with other features, and cheaper to compute compared to the runtime of the algorithm. Features should also be capable of explaining the similarities and differences between instances, and should be interpretable by humans [3,4].

The portfolio of SBSE Techniques is constructed by selecting a set of methods, each one with its unique biases, capable of solving the given problem. The more diverse the SBSETs, the higher the chance of finding the most suitable SBSET for a given instance.

The SBSET performance space $\mathcal{Y}$ includes the measures to report the performance of the SBSETs when solving the problem instances. Common performance measures for automated testing are coverage, length/size of the test suite and mutation score [8]. For Search Based Program Repair, a common measure is whether the techique produced a plausaible patch for a buggy program [1].

A critical step of ISA is identifying features of problem instances instances $f(p) \in F$ that have an impact on the effectiveness of SBSE techniques. Features are problem dependent and must be chosen such that the varying complexities of the buggy program instances are exposed, any known structural properties of the software systems are captured, and any known advantages and limitations of the different SBSETs are related to features.

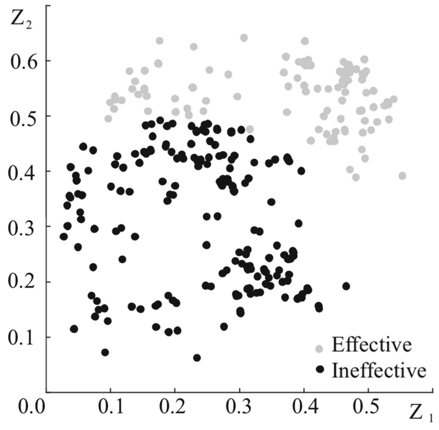

Fig. 2. SBSET footprint.

ISA applies a Genetic Algorithm to select the set of features which result in an instance space – as defined by the 2-dimensional projection of the subset of features through Principal Component Analysis – with problem instances that show similar performance of SBSETs closer to each other in this 2D space (as shown in Fig. 2). The best subset of features is the one that can best discriminate between easy and hard buggy program instances for SBSE techniques. The subset of features that have large coefficients and therefore contribute significantly to the variance of each Principal Component (PC), will be identified as the significant features. Rather than reporting algorithm performance averaged across a chosen set of problem instances, ISA reports the SBSET's footprint, which is the performance of a technique generalised across a diverse set of instances.

In the final step, ISA applies Machine Learning to predict the most effective SBSET for solving a particular SBSE problem. ISA uses the most significant features as an input to learn the relationship between the instance features and SBSET performance. A variety of machine learning algorithms can be used for this purpose, such as decision trees, or support vector machines for binary labels (effective/ineffective), or statistical prediction methods, such as regression algorithms or neural networks for continuous labels (e.g., time complexity of the approach). Previous work has reported Random Forest Classifier as the best performing method for Search-Based Program Repair [1], and Decisin Tree for Search-Based Software Testing [5].

References

1. Aleti, A., Martinez, M.: E-APR: mapping the effectiveness of automated program repair techniques. Empir. Softw. Eng. **26**(5), 1–30 (2021)
2. Chidamber, S.R., Kemerer, C.F.: A metrics suite for object oriented design. IEEE Trans. Softw. Eng. **20**(6), 476–493 (1994)
3. Muñoz, M.A., Villanova, L., Baatar, D., Smith-Miles, K.: Instance spaces for machine learning classification. Mach. Learn. **107**(1), 109–147 (2017). https://doi.org/10.1007/s10994-017-5629-5
4. Muñoz, M.A., et al.: An instance space analysis of regression problems. ACM Trans. Knowl. Discov. Data (TKDD) **15**(2), 1–25 (2021)
5. Oliveira, C., Aleti, A., Grunske, L., Smith-Miles, K.: Mapping the effectiveness of automated test suite generation techniques. IEEE Trans. Reliab. **67**(3), 771–785 (2018)
6. Oliveira, C., Aleti, A., Li, Y.-F., Abdelrazek, M.: Footprints of fitness functions in search-based software testing. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1399–1407 (2019)
7. Smith-Miles, K., Tan, T.T.: Measuring algorithm footprints in instance space. In: 2012 IEEE Congress on Evolutionary Computation, pp. 1–8. IEEE (2012)
8. Tengeri, D., et al.: Relating code coverage, mutation score and test suite reducibility to defect density. In: 2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pp. 174–179. IEEE (2016)
9. Ye, H., Gu, J., Martinez, M., Durieux, T., Monperrus, M.: Automated classification of overfitting patches with statically extracted code features. Technical report 1910.12057, arXiv (2019)
10. Yu, Z., Martinez, M., Bissyandé, T.F., Monperrus, M.: Learning the relation between code features and code transforms with structured prediction. Technical report 1907.09282, arXiv (2019)