TuneScape: Visualizing Multi-Tuner Parameter Optimization Logs

Donghee Hong* Sungkyunkwan University
Minjong Kim* Sungkyunkwan University
Sooyoung Cha* Sungkyunkwan University
Jaemin Jo*† Sungkyunkwan University

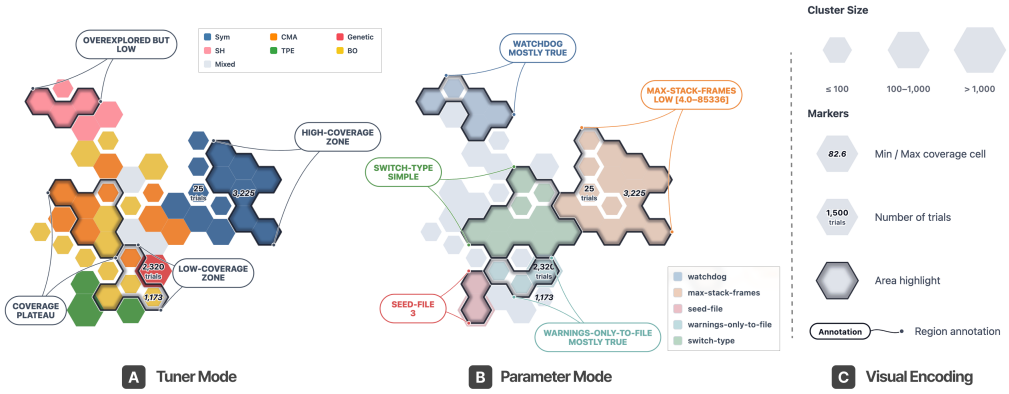

Figure 1: *TuneScape* visualizes MTPOL over a shared hexagonal partition of the parameter space, where each cell groups trials with similar parameter configurations. (A) Tuner mode colors cells by the dominant tuner and annotates regions of interest, such as the high-coverage zone and the overexplored but low region. (B) Parameter mode colors cells by the dominant value of a contrastive parameter, with the top five selected automatically. (C) Legend describing the visual encodings used in the map. The example shows a run on `grep` with six tuners.

Abstract

Tuning symbolic execution engines requires configuring tens of parameters, and researchers often run multiple tuners on one program, producing Multi-Tuner Parameter Optimization Logs (MTPOL). However, existing tools reduce each tuner run to a scalar metric, obscuring where it searches and how its coverage complements other tuners'. We present *TuneScape*, a map visualization that lays out MTPOL over a shared hexagonal partition of the parameter space, allowing researchers who run multiple tuners to contrast them, inspect the parameter values that drive their differences, and find complementary regions. Two use cases, on a KLEE benchmark and a hyperparameter optimization log, illustrate how *TuneScape* shows tuner-specific search regions and the parameter values that characterize them, and helps locate cross-tuner combinations that extend branch coverage beyond any single tuner in these cases.

Index Terms: Visual analytics, parameter tuning, symbolic execution, hyperparameter optimization, comparative visualization.

1 Introduction

Symbolic execution [13, 5] automatically generates test cases by treating its inputs as symbolic variables and exploring execution paths to derive concrete inputs for each path. The resulting test suite is commonly measured by *branch coverage*, the number of conditional branches executed at least once. Modern symbolic execution engines such as KLEE expose dozens of tunable, mixed-type parameters that govern this exploration; for example, the *search strategy* parameter determines which paths are visited first, such as *depth-first* or *random-path*. It is well known that parameter values strongly affect the resulting branch coverage [7, 17, 21].

Parameter tuning searches a high-dimensional, mixed-type space for a configuration that maximizes branch coverage in a given program. Due to the large search space and the nonlinear relationship between parameters and coverage, it is typically delegated to an automated *tuner*. Researchers have proposed both general-purpose black-box tuners [16, 3, 26, 14, 20] and domain-specific tuners [8, 7, 17], each adopting a different search strategy and concentrating its trials on different regions. Because no single tuner covers the entire space well, running multiple tuners on the same program and aggregating their results can yield higher coverage than any alone. Exploiting this, however, requires understanding where each tuner searches, what values characterize those regions, and where one tuner's coverage can complement another's.

We refer to such multi-tuner data as Multi-Tuner Parameter Optimization Logs (MTPOL). Although prior approaches support the analysis of MTPOL, they typically rely on surface-level summaries that obscure deeper insights. For example, tuners are typically compared using a single branch coverage value [8, 7], hiding where each searches, what configurations drive its results, and which branches it leaves uncovered. Such comparisons cannot reveal whether two tuners explore the same region, nor do they expose the cross-tuner combinations that motivate running multiple tuners. Visual analytics systems for hyperparameter optimization in machine learning [24, 28, 9] offer richer views, but focus on a single tuner. A recent visual analytics system for symbolic execution tuning [18] compares user-defined trial groups within a single experiment, yet requires multi-step manual analysis to identify complementary regions and does not compare exploration patterns across tuners.

To address this gap, we present *TuneScape*, a visualization that partitions the parameter space into a shared hexagonal layout, where each cell aggregates trials with similar configurations across tuners. On this layout, *TuneScape* provides two coordinated views, by tuner and by parameter value, that highlight regions of interest and recommend cells complementing the user's working set. We demonstrate *TuneScape* through use cases on a KLEE benchmark program with six tuners and a hyperparameter optimization log.

2 Related Work

2.1 Visual Analytics for Parameter Tuning

Although *TuneScape* targets symbolic execution, an instance of MTPOL has a direct counterpart in hyperparameter optimization

*email: {dh.hong, minjong.kim, sooyoung.cha, jmjo}@skku.edu
†Jaemin Jo is the corresponding author.

(HPO) for machine learning: both consist of a tuner identity, a parameter configuration, the resulting aggregate metric, and a per-instance outcome vector. This structural correspondence motivates us to review visual analytics work on HPO alongside research on symbolic execution tuning.

In HPO, visual analytics systems typically focus on a single tuner: HyperTendril [24] supports user-driven steering of hyperparameter optimization in a model-agnostic setting, ATMSeer [28] exposes the internal behavior of an AutoML pipeline, and VisEvol [9] supports interactive intervention in an evolutionary search over five ML algorithms. Comparative analysis across multiple tuners has been explored in adjacent settings, such as evolutionary multi-objective optimization, where Huang et al. [19] compare algorithms by the similarity of their solution sets in the objective space across generations. However, these systems either focus on a single tuner or compare algorithms in the objective space, leaving exploration patterns over the parameter space unexamined.

Only a few visual analytics systems have been proposed in the context of symbolic execution. SymNav [1] targets execution traces rather than parameter tuning. Symetra [18] provides overviews of parameter effects and branch coverage and supports comparing user-defined trial groups, but operates on a single experiment at a time and treats trials as individual points without spatial aggregation in its overview. While Symetra defines complementarity between groups, identifying complementary groups requires the user to manually construct and pair groups across multiple views.

TuneScape addresses these gaps by embedding the trials of multiple tuners in a shared spatial layout, automatically detecting and labeling regions of interest, and recommending complementary cells against a working set of cells the user has chosen.

2.2 Comparative and Spatial Visualization

Because the core analytical task in MTPOL is comparing multiple tuners over a high-dimensional parameter space, two threads of prior work shaped our work: comparative visualization and spatial layouts of non-spatial data.

For comparing multiple tuners, we adopt a shared spatial layout on which each tuner's trials can be overlaid, contrasted, or aggregated. This corresponds to *superposition* in the comparative visualization taxonomy of Gleicher et al. [12, 11]. Superposition keeps all items in one frame of reference, supporting many comparisons in a single view, but can clutter as the number of items grows [12]. In MTPOL, every tuner samples the same parameter space, and we aggregate trials into cells, so a single layout supports many comparisons without the overplotting that plain superposition incurs.

Realizing this shared layout requires a discrete grid that aggregates many trials into legible units. Hexagon mosaic maps [6] aggregate geographical data over hexagonal tessellations, tile maps [23] display geographical regions as grids of identical tiles, and gridded glyph maps [25] aggregate multivariate measurements over a regular grid, all trading spatial precision for the legibility of summary glyphs. *TuneScape* extends this idea to parameter configurations, which lack an inherent spatial embedding, by projecting them to a 2D layout via dimensionality reduction and aggregating trials over a hexagonal partition.

3 The *TuneScape* Visualization

3.1 Data: Multi-Tuner Parameter Optimization Logs

A tuner run produces an MTPOL instance, consisting of a set of trials, where each trial consists of four elements: (i) a *tuner identity*, indicating which tuner generated the trial, (ii) a *parameter configuration*, a vector of values for the engine's tunable parameters, (iii) a *branch coverage value*, the number of branches executed by the resulting test cases, and (iv) a *branch coverage vector B*, the set of branch indices executed by those test cases. Together, these elements describe both where each tuner searched in the parameter space and which branches its trials covered. An example trial is shown in our supplementary material.

We generate an MTPOL instance by running six tuners to tune the 61 parameters of KLEE on `grep`, a benchmark with 3,403 reachable branches. The 61 parameters comprise 35 boolean, 6 categorical, and 20 numeric values. Each tuner produces 2,200 trials within a 24-hour budget, yielding 13,200 trials in total.

3.2 Tasks

TuneScape targets researchers who study or develop tuners, with practitioners who run tuners on their code as a secondary audience. Over three months, we collaborated with two researchers experienced in tuning symbolic execution through regular discussions and prototype feedback to understand their needs in a multi-tuner setting. From these sessions, we identified three tasks they perform to contrast different tuner behaviors.

T1. Identify where each tuner explores. For each tuner, they locate the regions of the parameter space its trials concentrate in, the branch coverage achieved there, and how these regions differ from other tuners, to characterize each tuner's exploration pattern.

T2. Characterize regions by their dominant parameter values. For each region in **T1**, they identify the parameters whose values are concentrated within it, to interpret what the region represents in the high-dimensional parameter space.

T3. Discover regions that complement a selection. Given regions the user finds promising, they search for additional regions whose trials cover branches that those regions do not, in order to form combinations that extend branch coverage across tuners.

3.3 Design Rationale

To support the tasks above, we adopt three design rationales.

DR1. Spatialize trials over a shared layout. We treat the parameter space as a 2D map on which each tuner traces a path of trials, a metaphor familiar to researchers in symbolic execution and HPO who routinely reason about search behavior in spatial terms. Concretely, we embed trials from multiple tuners in a shared 2D layout where spatial proximity reflects similarity in parameter configurations, so that the user can identify where each tuner concentrates (**T1**) and which parameter values dominate those areas (**T2**). We cluster trials to keep the layout legible when the number of trials is large, providing multiple levels of detail.

DR2. Convey information through modes and overlays. A single layout cannot show all of tuner identity, parameter values, and coverage at once, but juxtaposing one view per attribute breaks the shared frame from **DR1**. We instead layer information on the same layout. A *mode* recolors cells by a primary attribute (tuner identity for **T1**, parameter value for **T2**), and an *overlay* replaces the color with a secondary attribute (coverage, or complementarity against the user's working set for **T3**), so the user moves between perspectives without losing spatial context.

DR3. Annotate regions with contrastive explanations. **DR2** lets the user compare cells once they know where to look, but with hundreds of cells across many tuners, an analyst rarely has a starting point. We therefore detect noteworthy regions automatically and label each with a *contrastive annotation* that states what distinguishes it from the rest, giving the user entry points (**T1**, **T2**) and shortening the path to candidate cells in the complementary search (**T3**).

3.4 System

TuneScape presents MTPOL as a single map visualization (Figure 1). Trials are projected onto a shared 2D layout (**DR1**), regions of interest are highlighted and labeled with contrastive annotations (**DR3**), and the user explores the layout through coordinated modes and overlays (**DR2**).

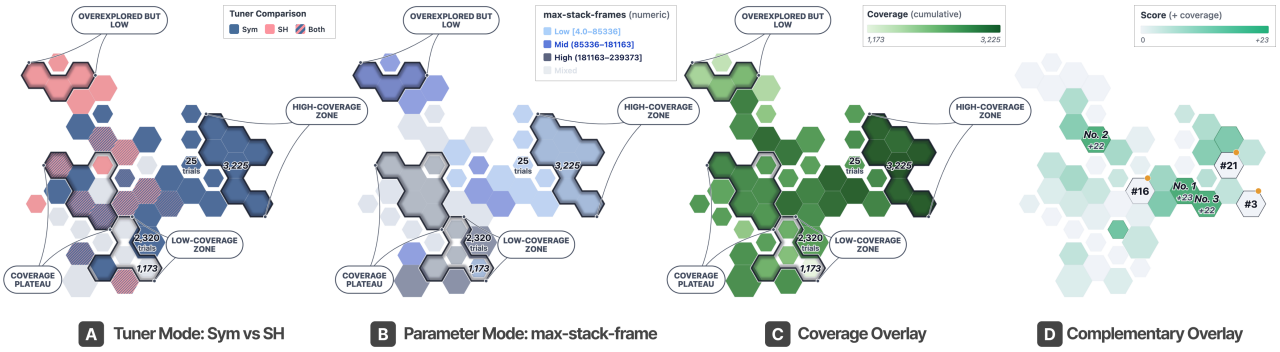

Figure 2: Detail views of the two modes and two overlays applied to the same `grep` run as Figure 1. (A) Sym (blue) and SH (pink) pinned in Tuner mode reveal nearly disjoint exploration territories. (B) Selecting `max-stack-frames` in Parameter mode shows that Sym's high-coverage zone overlaps the low bin and SH's overexplored region overlaps the mid bin, explaining the coverage gap. (C) The coverage overlay recolors cells by cumulative coverage; darker green indicates higher coverage, with the maximum (3,225) in the high-coverage zone. (D) The complementary overlay recolors the remaining cells by branches added to a three-cell working set (#3, #16, #21), with the top three candidates labeled (No. 1–3).

3.4.1 Layout Construction

Trial encoding and projection. To embed trials from multiple tuners in a shared 2D layout (**DR1**), we encode each configuration as a mixed-type vector: numeric parameters are min-max normalized, boolean parameters are mapped to $\{0,1\}$, and categorical parameters are one-hot encoded. Normalizing each type before combining follows Gower's coefficient [15], a standard measure for mixed-type data, so that no parameter dominates the distance through its scale or number of categories. To keep the layout legible when the number of trials is large, we cluster the encoded trials with k-means and project the cluster centroids to two dimensions with multidimensional scaling (MDS), which preserves pairwise distances and so keeps spatial proximity aligned with parameter similarity (**DR1**), providing five levels of detail with $k \in \{12, 24, 50, 100, 200\}$ that the user can switch between. A 2D projection cannot preserve all distances in a high-dimensional space, so we treat the layout as an overview, with cell aggregation absorbing small positional errors.

Hex grid assignment. We greedily assign the projected centroids to the nearest empty cell of a hexagonal grid. The layout preserves coarse neighborhood structure from MDS and yields a regular adjacency graph where each cell has up to six neighbors.

3.4.2 Region Annotation

To direct the user's attention to noteworthy parts of the layout (**DR3**), *TuneScape* annotates regions of interest and labels them with contrastive leader-line annotations (Figure 1).

Region identification. A region is a connected set of hex cells that satisfies a mode-specific predicate. *TuneScape* defines four predicates in Tuner mode and one predicate per parameter in Parameter mode, for the top five by feature importance. Given a predicate and a minimum size, we extract the satisfying cells, run a breadth-first search over the six-neighbor adjacency graph for connected components, and select one component per category by maximizing a category-specific score. Selected cells are excluded from later categories, so each cell belongs to at most one region.

Leader-line placement. For each region, we search outward for a label position that minimizes visual conflict with other regions, labels, and leader lines. The supplementary material details both the region predicates and the leader-line placement algorithm.

3.4.3 Modes and Overlays

To support multiple analytical perspectives on the shared layout (**DR2**), the user can switch between two *modes*, tuner and parameter, that recolor cells by a primary attribute, and toggle two *overlays*, coverage and complementary, that replace the color with a secondary attribute (Figures 1 and 2).

Cell encoding. Cell color depends on the active mode and overlay, while cell size encodes the number of trials in the cell, discretized into three tiers. A cell is marked *Mixed* (rendered in gray) when no value dominates its trials, with mode-specific thresholds given in the supplementary material. The cells with the highest and lowest coverage are annotated with their coverage values, and the densest and sparsest cells with their trial counts.

Tuner mode. In Tuner mode, each cell is colored by its dominant tuner, defined as the tuner with the most trials in the cell, and four region annotations are added (Figure 1A): *overexplored but low*, *high-coverage zone*, *low-coverage zone*, and *coverage plateau*. Hovering or pinning up to two tuners in the toolbar overrides the cell color with each pinned tuner's hue, allowing comparison while keeping the region annotations in place (Figure 2A).

Parameter mode. Parameter mode opens with an overview of the five parameters with the highest feature importance, annotating one region per parameter as the largest non-Mixed bin component (Figure 1B). This overview shows which parameters carry the most distinct regions, a starting point for choosing which one to inspect. When the user selects a specific parameter, each cell is recolored by the dominant value of that parameter, with bins defined per parameter type. The Tuner-mode region annotations remain visible during this inspection, so the user can locate the selected parameter's bins relative to coverage and density patterns (Figure 2B).

Coverage overlay. The coverage overlay can be toggled on top of either mode and recolors the cells by a sequential coverage scale (Figure 2C). The user selects between mean coverage (the average per-trial coverage in a cell) and cumulative coverage (the union of branches covered by all trials in the cell). The region annotations of the active mode remain in place.

Complementary overlay. The user can save cells of interest into a *working set* by clicking them on the map; cells in the working set are outlined with an orange ring. When the working set is non-empty, the complementary overlay recolors each remaining cell by the number of new branches its trials would add beyond those already covered by the working set. Region annotations are replaced by cell-level labels: each working-set member is marked #i with its id, and the top three candidate cells are marked with both their rank (No. 1–3) and their numeric gain (+N) (Figure 2D). The supplementary material gives the precise complementary score.

3.4.4 Implementation

TuneScape is implemented as a React [27]-based web application using d3.js [4] for rendering. The layout, per-cell branch coverage vectors $B(\cdot)$, and parameter importance scores (mean |SHAP| values from a random-forest surrogate) are precomputed in Python and

exported as static JSON; in the browser, only the cell-level overlays are updated as the user assembles a working set, while region annotations are recomputed only on mode or parameter changes. A live demo is available at: https://idclab.skku.edu/TuneScape/.

4 Use Cases

We demonstrate *TuneScape* through two use cases. The first is a detailed analysis of a MTPOL instance from symbolic execution, walking through the three tasks (Section 3.2) on a representative program. The second applies *TuneScape* to a hyperparameter optimization log from a machine learning task, suggesting that the same workflow can apply beyond symbolic execution.

4.1 Comparing and Complementing Tuners on `grep`

For this use case, two of the authors, including an experienced symbolic execution researcher, walked through *TuneScape* to understand how the six tuners differ and whether their outputs complement one another. They are SymTuner [8] (Sym), a domain-specific tuner for symbolic execution, and five general-purpose black-box tuners: CMA-ES [16] (CMA), a genetic algorithm [14] (Genetic), successive halving [20] (SH), the tree-structured Parzen estimator [3] (TPE), and Bayesian optimization [26] (BO).

We open *TuneScape* in Tuner mode (Figure 1A) to see how the six tuners distributed their trials across the parameter space (**T1**). The visualization separates the tuners into distinct territories. Sym concentrates in a high-coverage zone on the right, where one cell reaches 3,225 cumulative branches; its domain-specific design pays off under tight trial budgets. SH occupies an overexplored but low-coverage region on the upper left, where early-promising candidates drew continued sampling that did not translate into high final coverage. CMA forms a coverage plateau in the middle without reaching the highest-coverage cells. A Genetic cell is the densest with over 2,000 trials, reflecting its exploit-heavy resampling.

Sym and SH carry the most prominent region annotations (the high-coverage zone and the overexplored but low region), so we pin them for closer comparison (Figure 2A). We find that the two tuners explore almost disjoint territories, with only a handful of cells overlapping. This contrast between a domain-specific and a general-purpose tuner is informative for practitioners choosing among tuners or designing a tuning strategy.

Switching to Parameter mode (Figure 1B), five contrastive parameter regions are highlighted automatically (**T2**). Among them, `max-stack-frames` stands out because its region overlaps the high-coverage zone identified in Tuner mode. Selecting this parameter for closer inspection (Figure 2B), we confirm the alignment: Sym's high-coverage zone is colored with the lightest blue, indicating the smallest stack-frame bin, while SH's overexplored territory is colored with the darker mid-range blue. According to the researcher's prior knowledge, `max-stack-frames` controls when symbolic execution stops forking new states, and smaller values trigger earlier termination and restart, mitigating path explosion and yielding higher coverage within a fixed time budget. What *TuneScape* added was showing that the stack-frame bins explain the coverage gap between Sym and SH, rather than merely coincide with it.

We then ask whether Sym's coverage can be extended with cells from other tuners (**T3**). We save the highest-coverage Sym cell into the working set, covering 3,225 of 3,403 branches. Since Sym already covers what the general-purpose tuners reach and more, we expected the recommendations to stay within Sym's own cells. The first recommendation is indeed a Sym cell: adding it lifts the union to 3,303. The next, however, is a CMA-dominant cell where Sym holds almost no trials, adding 31 branches no Sym cell had reached and lifting the union to 3,334 (Figure 2D). Even the domain-specific Sym leaves regions uncovered that a general-purpose tuner reaches. This cross-tuner gain is what motivates running several tuners together. This walkthrough illustrates the three steps *TuneScape* supports: observing tuner differences, explaining them through parameter values, and exploiting them through cross-tuner combinations.

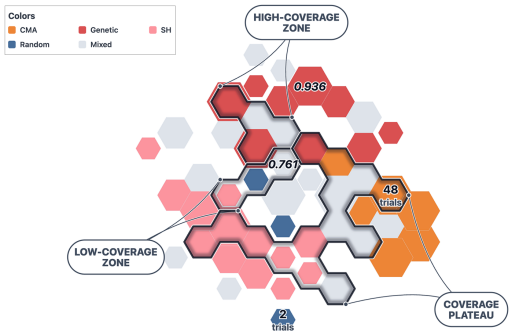

Figure 3: *TuneScape* applied to the HPO log.

4.2 Comparing HPO Algorithms for Machine Learning

Since HPO algorithms also produce logs structured like MTPOL, we apply *TuneScape* to an HPO log to examine whether the same workflow applies beyond symbolic execution. We ran four algorithms (Random, Genetic, CMA, SH) to tune XGBoost [10] on the `adult` dataset [2] over a 9-dimensional hyperparameter space, with 200 trials per algorithm. The per-instance outcome vector B records which validation samples each configuration classifies correctly.

In Tuner mode, Genetic dominates the high-coverage zone, where the top cell reaches accuracy 0.936, while CMA concentrates densely in a single 48-trial cell on the coverage plateau (**T1**, Figure 3). Both are exploit-heavy strategies, but they converge to different regions: Genetic reaches the global optimum, whereas CMA's covariance-based search settles into a locally optimal area without escaping it. In Parameter mode, `max_depth` HIGH (10–12) and `n_estimators` HIGH (206–297) align with the high-coverage zone, while `learning_rate` LOW (0.001–0.02) aligns with the low-coverage zone (**T2**). However, selecting `learning_rate` shows that the high-coverage zone contains both low and mid learning rates, indicating that learning rate alone does not separate high-coverage from low-coverage cells. We then select CMA's densest cell into the working set, which alone covers 0.909 of validation samples; the complementary overlay recommends three Genetic cells, and adding the strongest lifts the union to 0.939, closing samples that CMA's local optimum had missed (**T3**).

5 Conclusion

We present *TuneScape*, a visualization that lays out MTPOL on a shared hexagonal partition, annotates regions with contrastive labels, and recommends complementary cells against a working set. Beyond branch coverage, the complementary overlay extends to the per-instance correctness vectors of HPO trials, pointing toward ensemble construction where the working set becomes the candidate ensemble [22].

The mixed-type layout, both modes, and the complementary score require only that multiple tuners search a shared parameter space and report a per-instance outcome vector, so they carry over to other multivariate settings that meet this condition. The region categories, phrased here for branch coverage, are the part that a new domain would relabel and re-threshold.

Two limitations remain. *TuneScape*'s complementary recommendation is greedy with respect to the working set, so the order in which cells are added influences which complements are recommended; jointly optimizing combinations across all cells is left to future work. A formal user study with symbolic execution researchers, along with a quantitative comparison against baseline visualizations, is also needed to assess whether the observed patterns translate into practical tuning decisions.

Acknowledgments

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (RS-2019-II190421, Artificial Intelligence Graduate School Program (Sungkyunkwan University)) and the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (RS-2026-25497428). This work was also supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2023-00221186).

References

[1] M. Angelini, G. Blasilli, L. Borzacchiello, E. Coppa, D. C. D'Elia, C. Demetrescu, S. Lenti, S. Nicchi, and G. Santucci. Symnav: Visually assisting symbolic execution. In *2019 IEEE Symposium on Visualization for Cyber Security (VizSec)*, pp. 1–11, 2019. doi: 10.1109/VizSec48167.2019.9161524 2

[2] B. Becker and R. Kohavi. Adult. UCI Machine Learning Repository, 1996. doi: https://doi.org/10.24432/C5XW20. 4

[3] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, eds., *Advances in Neural Information Processing Systems*, vol. 24. Curran Associates, Inc., 2011. 1, 4

[4] M. Bostock, V. Ogievetsky, and J. Heer. D3 data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, Dec. 2011. doi: 10.1109/TVCG.2011.185 3

[5] C. Cadar, D. Dunbar, and D. Engler. Klee: unassisted and automatic generation of high-coverage tests for complex systems programs. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, OSDI'08, p. 209–224. USENIX Association, USA, 2008. 1

[6] D. B. Carr, A. R. Olsen, and D. White. Hexagon mosaic maps for display of univariate and bivariate geographical data. *Cartography and Geographic Information Systems*, 19(4):228–236, 1992. doi: 10.1559/152304092783721231 2

[7] S. Cha, S. Hong, J. Bak, J. Kim, J. Lee, and H. Oh. Enhancing dynamic symbolic execution by automatically learning search heuristics. *IEEE Transactions on Software Engineering*, 48(9):3640–3663, 2022. doi: 10.1109/TSE.2021.3101870 1

[8] S. Cha, M. Lee, S. Lee, and H. Oh. Symtuner: maximizing the power of symbolic execution by adaptively tuning external parameters. In *Proceedings of the 44th International Conference on Software Engineering*, ICSE '22, p. 2068–2079. Association for Computing Machinery, New York, NY, USA, 2022. doi: 10.1145/3510003.3510185 1, 4

[9] A. Chatzimparmpas, R. M. Martins, K. Kucher, and A. Kerren. Visevol: Visual analytics to support hyperparameter search through evolutionary optimization. *Computer Graphics Forum*, 40(3):201–214, 2021. doi: 10.1111/cgf.14300 1, 2

[10] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, p. 785–794. Association for Computing Machinery, New York, NY, USA, 2016. doi: 10.1145/2939672.2939785 4

[11] M. Gleicher. Considerations for visualizing comparison. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):413–423, 2018. doi: 10.1109/TVCG.2017.2744199 2

[12] M. Gleicher, D. Albers, R. Walker, I. Jusufi, C. D. Hansen, and J. C. Roberts. Visual comparison for information visualization. *Information Visualization*, 10(4):289–309, Oct. 2011. doi: 10.1177/1473871611416549 2

[13] P. Godefroid, N. Klarlund, and K. Sen. Dart: directed automated random testing. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '05, p. 213–223. Association for Computing Machinery, New York, NY, USA, 2005. doi: 10.1145/1065010.1065036 1

[14] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., USA, 1st ed., 1989. 1, 4

[15] J. C. Gower. A general coefficient of similarity and some of its properties. *Biometrics*, 27(4):857–871, 1971. 3

[16] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001. doi: 10.1162/106365601750190398 1, 4

[17] J. He, G. Sivanrupan, P. Tsankov, and M. Vechev. Learning to explore paths for symbolic execution. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, p. 2526–2540. Association for Computing Machinery, New York, NY, USA, 2021. doi: 10.1145/3460120.3484813 1

[18] D. Hong, M. Kim, S. Cha, and J. Jo. Symetra: Visual analytics for the parameter tuning process of symbolic execution engines. *Computer Graphics Forum*, p. e70461, 2026. doi: 10.1111/cgf.70461 1, 2

[19] Y. Huang, Z. Zhang, A. Jiao, Y. Ma, and R. Cheng. A comparative visual analytics framework for evaluating evolutionary processes in multi-objective optimization. *IEEE Transactions on Visualization and Computer Graphics*, 30(1):661–671, Jan. 2024. doi: 10.1109/TVCG.2023.3326921 2

[20] K. Jamieson and A. Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In A. Gretton and C. C. Robert, eds., *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, vol. 51 of *Proceedings of Machine Learning Research*, pp. 240–248. PMLR, Cadiz, Spain, 09–11 May 2016. 1, 4

[21] T. Kapus, F. Busse, and C. Cadar. Pending constraints in symbolic execution for better exploration and seeding. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, ASE '20, p. 115–126. Association for Computing Machinery, New York, NY, USA, 2021. doi: 10.1145/3324884.3416589 1

[22] L. I. Kuncheva and C. J. Whitaker. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Mach. Learn.*, 51(2):181–207, May 2003. doi: 10.1023/A:1022859003006 4

[23] G. McNeill and S. A. Hale. Generating tile maps. *Computer Graphics Forum*, 36(3):435–445, 2017. doi: 10.1111/cgf.13200 2

[24] H. Park, Y. Nam, J.-H. Kim, and J. Choo. Hypertendril: Visual analytics for user-driven hyperparameter optimization of deep neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):1407–1416, 2021. doi: 10.1109/TVCG.2020.3030380 1, 2

[25] A. Slingsby, R. Reeve, and C. Harris. Gridded glyphmaps for supporting spatial covid-19 modelling. In *2023 IEEE Visualization and Visual Analytics (VIS)*, pp. 1–5, 2023. doi: 10.1109/VIS54172.2023.00009 2

[26] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'12, p. 2951–2959. Curran Associates Inc., Red Hook, NY, USA, 2012. 1, 4

[27] J. Walke. React - the library for web and native user interfaces. `https://react.dev/`, April 2024. Last Accessed 29 November 2024. 3

[28] Q. Wang, Y. Ming, Z. Jin, Q. Shen, D. Liu, M. J. Smith, K. Veeramachaneni, and H. Qu. Atmseer: Increasing transparency and controllability in automated machine learning. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, p. 1–12. Association for Computing Machinery, New York, NY, USA, 2019. doi: 10.1145/3290605.3300911 1, 2