Fishing the Fishing Tool

Manoj Ayyappan Email: mayyapp@ncsu.edu
Pradeep Patil Email: papatil@ncsu.edu
Rupin Talwar Email: rtalwar@ncsu.edu

***Abstract*—This project involved optimizing a baseline fishing tool that clusters data on x values and checks y values infrequently. The tool recursively divides the data in half based on distance to a line connecting two distant points and performs hierarchical clustering to select the best cluster. The tool then selects the best attributes to explain the selection of that cluster. The objective was to optimize the baseline tool itself, using it to perform hyper-parameter optimization on different data sets to find the best sway and xpln values. The approach used involved performing more evaluations than the original tool but using all leaf nodes, rather than splitting in half and selecting the best half, resulting in a slightly slower implementation. With a lot of extra evaluations, the results were slightly better. We then did an experimental study proving that our tool could be more accurate than the original tool when going down the best half rather than using all leaf nodes with some extra evaluations.**

I. Introduction

The aim of this project is to enhance the fishing algorithm provided to us as part of our coursework in the current semester. The current algorithm has some limitations that we have identified, and we plan to address these limitations to potentially produce better outcomes. To achieve this goal, we have identified three key components of the existing algorithm: hyperparameter optimization, Zitzler's predicate, and the clustering algorithm.

Hyperparameter optimization is a crucial aspect of machine learning algorithms, including the fishing algorithm. The current algorithm uses a fixed set of hyperparameters, which could be improved by optimizing them. To accomplish this, we attempted to perform hyperparameter optimization using the original fishing tool. However, we realized later that this approach led to an increased number of evaluations since we had to perform a certain number of evaluations in each iteration to find better hyperparameters. We clustered a list of hyperparameters based on distance, selected one from each cluster, and ran sway on all 130 leaf nodes to find the best one. We then conducted a study to go down the best half of the hyperparameter list to optimize our algorithm, thereby creating less than 10 leaf nodes.

We also explored alternatives to Zitzler's predicate, such as implementing Boolean domination as a solution. Furthermore, we made modifications to the "better" and "sway" functions to improve their performance. Specifically, we added three different types of implementations to the "better" function and two different types of implementations to the "sway" function. These modifications allowed us to explore different ways to compare rows and different clustering techniques to select the one that best suited the data. One key point is that we found that the run time for DBSCAN was too long for our study and proved to be innefficient.

Our main focus was on hyperparameter optimization. We created a .csv file containing a range of hyperparameters by generating different values using a for loop. We clustered the hyperparameters to reduce the search space and identify the best settings. Finally, we used the best hyperparameter settings in the fishing tool.

Overall, these modifications and optimization techniques helped us improve the performance of the fishing algorithm and produce more accurate and reliable results. However, the only downside was that we had to conduct more evaluations than the original tool which results in a valiant effort but a failure to beat the original project in the case of using all 130 leaf nodes. Our preliminary study of going down the best half of hyper parameters showed us that it is possible our new tool may be better than the baseline system.

Some research questions that can be asked include by how much did the optimized tool beat the baseline tool or how efficient is the optimized system. We can answer those by saying that we did not beat the original algorithm considering the number of evaluations but we did obtain better results for 10 and only worse on 2. Also, each data-set works on specific values of the hyperparameters. And if we use those specific values, they are far more efficient.

Some caveats would be the absence of a February study and an ablation study. These techniques would definitely yield better results and are missing in our implementation.

II. Related Work

Fishing algorithms are based on the understanding that there is a lot of data in the real world but only a small fraction of that data is useful. They are usually contained in a very small sample of the humongous dataset and are hard to find. A good fishing algorithm tries to find those attributes which give us the most impact on the goal variables thus eliminating the need to process all data.
As the case with all fishing algorithms, we know that in order for them to work in a certain manner, there are some crucial engineering decisions being made which affect the functioning of this algorithm. This gave us the first clue on how to improve on our homeworks. By changing the constants and tweaking the hyperparameters. And this is what we followed for our final project.
The standard practice for comparing nodes has been the zitzler's predicate [2]. But we think because we have varied datasets, the number of goals or y values may be low or high in which case, boolean domination may outperform Zitzler's predicate in certain scenarios. This is because Boolean domination works very well for a small number of dependent variables. Thus, that was a key part of our project as well.

In a paper by A.Agrawal Et al. [1], they state that using the DODGE method (i.e., simply steering way from settings that lead to similar conclusions) is the wiser approach to deploying complex hyperparameter optimizers that require considerable CPU. However, during our research we found that this method may not always prove to be the better approach. Using techniques like Sway and Boolean domination predicate, we were able to go down the hard path and with a very few number of evaluations, find the optimum set of hyperparameters which work wonders for our specific dataset.

In a paper by J. Chen Et al. [3], they discuss using a very fast way for search based optimization techniques. They call it the sampling way or in short "sway". As mentioned in the paper, it performs very well by partitioning the data and going down the better half every single time. This greatly reduces the number of evaluations and thereby yields better results. We have been influenced very much by this paper which implements sway and combines it with the potential of hyperparameter optimization.

III. Methods

A. Algorithms

1) DBSCAN: DBSCAN is a clustering algorithm that identifies clusters based on the density of points in a given region of space. It assumes that clusters are composed of dense areas separated by areas with lower density. In order to enhance the fishing algorithm's performance, we made an initial effort to utilize DBSCAN as it is known for its ability to handle outliers effectively. Instead of employing an existing library, we opted to develop our own implementation. DBSCAN (Python code):

```
def dbscan(self, rows, eps, min_pts):
  clusters = []
```

```
    visited = set()
    noise = set()
    for i, row in rows.items():
        if i in visited:
            continue
        visited.add(i)
        neighbors = []
        for j, other_row in rows.items():
            if j != i and self.dist(row,
                other_row) <= eps:
                neighbors.append(j)
        if len(neighbors) < min_pts:
            noise.add(rows[i])
        else:
            cluster = set()
            self.expand_cluster(rows, i,
                neighbors, cluster, visited, eps,
                min_pts)

            clusters.append(list(cluster))
    return clusters, list(noise)
```

SWAY-DBSCAN (Python code):

```
def sway_dbscan(self, eps= 0.05, min_pts=
5):
    eps = g.the["eps"]
    min_pts = g.the["minpts"]
    clusters, noise = self.dbscan(self.rows,
        eps, min_pts)
    best, rest, evals = self.find_best_
    cluster(clusters)
    rest = lists.many(rest, g.the["rest"]*
        len(best))
    return self.clone(best), self.clone(rest)
        , evals
```

We soon realized that our self-implemented DBSCAN algorithm was excessively resource-intensive and could not be executed efficiently. As a result, we abandoned the idea.

2) Boolean Domination: According to the concept of Boolean domination, a point is considered good only if it surpasses all other points in at least one attribute, without being inferior in any other attribute. B-dom possesses a characteristic where it becomes increasingly challenging to determine a point's superiority over others as the goal space expands. However, in cases where the goal space is relatively small, this approach can yield meaningful outcomes. As a result, we decided to maintain this implementation alongside Zitzler's domination predicate. Our implementation of Boolean domination can be found below.

Boolean Domination (Python code):

```
def better_bdom(self, row1, row2, ys=None):
    is_row1_dominant =
        self.is_dominant_bdom(row1, row2,
        ys=ys)
    is_row2_dominant =
        self.is_dominant_bdom(row2, row1,
        ys=ys)
    if is_row1_dominant and not
        is_row2_dominant:
        return True
    else:
        return False

def is_dominant_bdom(self, row1, row2,
    ys=None):
    ys = self.cols.y if ys is None else ys

    dominates = False
    for n, col in ys.items():
        x = col.norm(row1.cells[col.at]) *
            col.w * -1
        y = col.norm(row2.cells[col.at]) *
            col.w * -1
        if x > y:
            return False
        elif x < y:
            dominates = True
    return dominates
```

3) Hypervolume: Hypervolume is a performance metric used to evaluate the quality of a set of solutions in multi-objective optimization problems. It measures the volume of the objective space dominated by a set of solutions. The hypervolume indicator is widely used to compare different multi-objective optimization algorithms and to assess the performance of the obtained solution set. Due to a lack of expertise in the field and a deadline to complete the implementation, we decided to skip this part and focus on other small things which have impactful results on our end goals.

4) Hyperparameter Optimization: Hyperparameter optimization is the process of finding the best combination of hyperparameters for a machine learning algorithm to achieve the highest possible performance on a given task. For the implementation, we started by generating a .csv file that contains hyperparameters by using a for loop. This came out to be a list of 34,000 hyperparameters. We then clustered these down to 130 leaf nodes using the "tree" function from the original tool. We then ran through this list of 130 hyperparameters to find the best hyper parameters for the fishing tool. Since running the model for all 130 sets of hyper parameters is a very expensive operation we modified the sway method to check if a set of hyperparameters is better at run time and cached results. This means we only went down the better half rather than checking all leaf nodes.

B. Data

In this paper, we analyze several data files containing information related to various aspects of automobiles, health, and software development. The first file, "auto2" [7], contains data related to automobiles, such as engine size, horsepower, and fuel tank capacity. We intend to maximize City and highway mileage, and minimize weight. The second file, "auto93," contains information on attributes such as fuel efficiency and weight. We intend to minimize weight, maximize acceleration and miles per gallon. The data set "china" [8] includes various features related to software development, and we are trying to minimize effort. The fourth and fifth files, "coc1000" [8] and "coc10000" [8], contain data related to software complexity, such as code size, number of defects, and development time. We attempt to maximize lines of code, and minimize effort, experience, complexity and risk.

The next two files are health data sets which contain features relevant to classification in machine learning for health data. It is data from the hyperparameter optimization of random forests. We intend to minimize mean relative error, and maximize accuracy and precision at 40%. "Nasa93dem" [8], contains information on software development, such as reliability and number of iterations. We intend to maximize lines of code while minimizing effort, defects and months. The file "pom" [3] contains data related to project management, including dynamism and team size. We intend to minimize cost and idle time. We want to maximize Completion. The

two tables, "SSM" and "SSN" [9] relate to computational physics. We want to minimize the time to solution and number of iterations in SSM. We want to minimize energy and Peak Signal-to-Noise Ratio in SSN.

C. Summarization Methods

We used GitHub actions to run the optimized fishing tool that we created and the original fishing tool on all data sets. Data was collected for all files. An example of the data collected it down below. The first table for each data file is the central tendency of the list of values found by the relevant algorithm for every y value in the list of y values. The second table shows us a disjunction of bootstrap and cliff's delta run on the data to see if they are equivalent or not. '!' means that they are not equivalent. sway1 represents the original tool. sway2 represents our optimized tool. All represents the data with nothing done to find the optimized solution. Top represents the best row when sorting is done. The below tables is just a sample table of our tool running. We created these two tables for all 10 datasets and summarized the results.

TABLE I

	Cost-	Completion+	Idle-	N
all	331.13	0.89	0.24	10000
sway1	485.27	0.89	0.24	78
xpln1	262.17	0.89	0.25	4998
top	331.6	0.92	0.18	78
sway2	197.55	0.88	0.08	78
xpln2	269.07	0.88	0.13	2703

TABLE II

	Cost-	Completion+	Idle-
All to All	=	=	=
All to Sway1	!	=	=
All to Sway2	!	!	=
Sway1 to Sway2	!	=	=
Sway1 to Xpln1	!	=	=
Sway2 to Xpln2	!	=	=
Sway1 to Top	!	=	=
Sway2 to Top	!	!	=

D. Statistical Methods

Each of the attributes in the data sets can be split into two types of values- numerical values and symbolical ones. Since each of the attributes contain data of different magnitudes, utilizing their raw values in distance measures leads to some attributes being preferred over others. To counteract this effect, measures of central tendency and diversity are calculated for each attribute, to which the numerical data was then normalized to. For the numerical data, the mean and standard deviation were used. In the case of symbolic data, we used the mode as a measure of central tendency and entropy as a measure of diversity.

After running the tool on a data set, measures of central tendency and spread are collected for each of the objectives using the best representatives in the data set. These best representatives are guaranteed to be close together since the sway algorithm recursively removes half of representatives that are far from the chosen best. To compare the performance of our tool with the baseline, we utilized two types of statistical tests- an effect size test and a significance test.

As a measure of effect size, we utilized the non-parametric cliff's delta test. This test takes all pairs of values from two distributions and counts the number of times g a value from the first distribution is greater than the second and the number of times it is less l. To determine whether the distributions are the same, the absolute difference $|g - l|$ must be less than a predefined epsilon value ϵ. For our purposes, we utilized an $\epsilon = 0.4$ which allows for medium to larger effect size differences [10]. The cliffs delta test checks whether the following equation is true:

$$|\Sigma_{x_i \in D_1} \Sigma_{y_j \in D_2} \begin{cases} 1, & \text{if } x_i > y_j \\ -1, & \text{if } x_i < y_j \end{cases}| < \epsilon$$

In order to determine the difference between two distributions through a significance test, the bootstrap procedure was utilized. Bootstrap is a non-parametric method which finds the mean values of two distributions μ_i and mu_j as well as the mean μ_k of the combined distributions. It also finds a delta Δ value when determining the difference between two distributions.

$$\Delta = \frac{abs(\mu_i - \mu_j)}{\frac{\sigma_i}{n_i} - \frac{\sigma_j}{n_j}}$$

The bootstrap procedure finds a delta value Δ_{obs} using distributions D_1 and D_2. The values of the two distributions are shifted to have the same mean μ_k, yielding distributions $\hat{D_1}$ and $\hat{D_2}$. Bootstrap takes several samples of these adjusted distributions and calculates another delta value Δ_{sample} for each sample. It finds the ratio of occurrences where $\Delta_{sample} > \Delta_{obs}$ and checks if that value is greater than a confidence threshold c. This method is also used in the Scott Knott implementation due to it being a non-parametric statistical test.

The non-parametric Scott Knott method is a top down hierarchical clustering method used to partition the treatments into statistically distinct groups. It recursively splits the data into two groups and checks whether they are statistically the different using the bootstrap and cliffs delta methods. If so, the data is split and the recursion continues. This method utilizes Cohen's D to ensure that the cuts do not split data by an insignificant amount. This method is preferred because it clusters the data into groups with only $O(log_2 N)$ statistical tests.

IV. Results

Research questions

A. RQ1: How often did the optimized tool beat the baseline tool?

Each file we worked with contained a list of y values that needed to be optimized. In the case of the auto93 dataset, we had to optimize for three different attributes: weight, acceleration, and miles per gallon. Our goal was to find the most optimal values for these attributes.

We used Cliff's Delta and Bootstrap, which allowed us to compare the records that we found using our implementation of sway to those found by the original sway. By analyzing the differences between these records, we were able to determine whether our implementation was producing equivalent, better, or worse results to the original algorithm.

We also compared our sway results to those generated by top, which sorts all the records and chooses the best ones. To evaluate the effectiveness of our optimization approach, we counted the number of attributes for which our sway outperformed sway1, as well as the number of attributes for which our sway outperformed top. We then tallied these results to obtain a total count, which we plotted on a graph. This approach allowed us to compare the performance of our optimization algorithm to other widely used methods and gauge its overall effectiveness.

Looking at the graph in Figure 1, we can see that just based on counts, we did not have any improvement to the professors tool. However, the graph doesn't show us the extent to which we beat the baseline system when we do better. It also doesn't show us the extent to which the baseline system beats ours for attributes on which we do worse. For example, as we can clearly observe in Table 3, in the case of pom for the Cost attribute, our method gave us a value of 197.55 whereas the baseline algorithm gave a value of 486.27. We were able to beat top which has a value of 331.6 by a very significant amount. This can be seen in table 3. The difference is roughly 1.5 standard deviations which is significant. Due to a lack of time we could not do a comprehensive study to check all instances of the extent to which we did better or worse. The above graph shows us the results when we use all the leaf nodes. We realized this would lead to a high number of evaluations and then decided to conduct an experiment to see if going down the best half of hyperparameters would be more efficient.

TABLE III

	Cost-
Sway1	485.27
Sway2	197.55
top	331.6
Sway1 to Sway2	Not Equal
Sway2 to Top	Not Equal

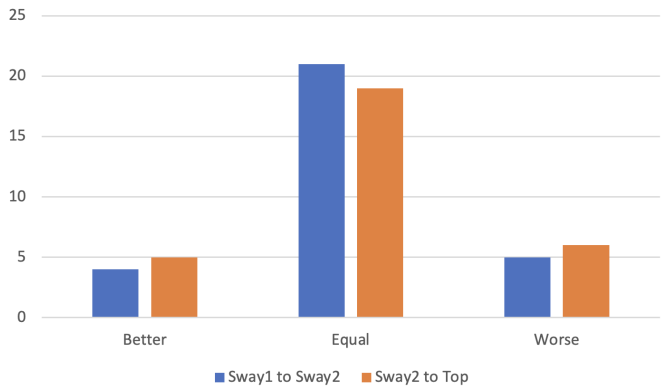

Fig. 1. Going through all nodes

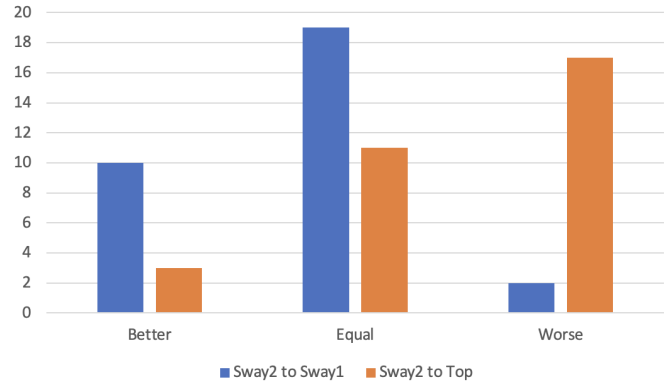

Fig. 2. Going down the better half

When we did this we found that our tool beats the baseline tool or performs as good as it most of the time specifically when comparing sway1 and sway2. The results from the graph in figure 2 are the most significant findings from our experiment. It is key to look at the columns for better in the graph where we see that when comparing sway1 to sway2, when we didn't perform as good as the base system, we have a higher chance of doing better as 10 out of 12 times our system has beat the baseline system.

B. RQ2: How efficient is the optimized system?

It was observed that each dataset required its own set of hyperparameters for efficient functioning. This lack of unanimity in hyperparameters between datasets meant that hyperparameter optimization could only be done for a specific dataset and not for the fishing tool in general.

However, assuming that the hyperparameters are set beforehand for a specific dataset and our tool is not run on multiple datasets, it was found that our tool is as efficient as the baseline tool. The number of evaluations on the y variables remains the same as that of the baseline tool, and accuracy is also occasionally improved.

However, if hyperparameter optimization needs to be performed for different files, the number of leaf nodes generated by the tool would play a significant role. In our case, we first generated 130 leaf nodes from a list of 34,000, which means our first tool ran 130 times the original number of evaluations. This evaluation cost becomes a constant C * (original number of evaluations). Therefore, to consider using our tool, it is necessary to minimize the value of C. Our tool would have been extremely inefficient for large values of C. This one finding made us decide that instead of using all leaf nodes, we would go down the best half of hyperparameters just as the baseline system implemented sway. Our updated tool brought C down to less than 10. So if the original tool used 6 evaluations, our tool would do as good or even better with C * 6 evaluations. Assuming C is 8, this means we would have 48 evaluations to have a better solution.

C. RQ3: What is the best implementation for a sway algorithm?

We explored two different sway algorithms for clustering. The first one used a hierarchical clustering algorithm, which was the original clustering algorithm provided to us. We also created our own implementation of DBSCAN, which is a density-based clustering algorithm.

However, we found that the DBSCAN algorithm was not efficient enough for larger data sets. In particular, its time complexity is $O(N^2)$. This means that as the size of the data set grows, the time required to run the algorithm becomes extremely large. We found this out when we ran DBSCAN on the SSM data set.

As a result, we excluded the DBSCAN algorithm from our study and focused on using the original clustering algorithm instead. Our trials suggest that using DBSCAN defeats the purpose of our project, which is to create a more accurate and efficient fishing tool.

D. RQ4: What is the best implementation for a better algorithm?

In our project, we observed that when given a choice between BDOM and Zitzler's predicate, BDOM was chosen more frequently than Zitzler's predicate. This was mainly due to the fact that most of our data-sets had less number of goals, and BDOM performs well with less number of goals. However, it is important to note that BDOM falls apart when we have more number of goals, and Zitzler's predicate might be a better choice in such cases.

Despite its limitations, we found that BDOM has several advantages that make it a favorable choice in many cases. One major advantage of BDOM is its simplicity and ease of implementation. It is relatively easier to implement and requires less computational resources compared to other methods. Moreover, BDOM performs

really well when bet against a known choice such as Zitzler's predicate.

We believe that BDOM deserves to be explored more in future research work in the field of Automated Software Engineering, especially for problems with less number of goals. However, for problems with more number of goals, it might be necessary to explore other techniques such as Zitzler's predicate or Hypervolume. Overall, our experiments highlight the importance of carefully selecting the appropriate algorithm based on the characteristics of the problem being solved. It is important to note that the reuslts achieved in figure 3 are coming from the solution where we used all 130 leaf nodes.

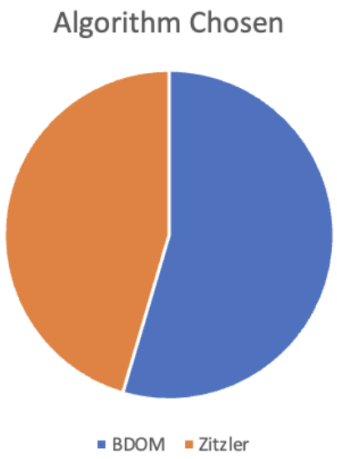

Fig. 3. BDOM vs. Zitzler

V. Discussion

A. Threats to Validity

1) Measurement Bias: The metrics used to evaluate the effectiveness of the optimized tool may not accurately capture its true performance. For example, we used Cliff's Delta and Bootstrap using which we could not tell how much we outperformed the original code. Also, the metrics may be too narrow and not consider all relevant aspects of fishing, such as ease of use, durability, and cost-effectiveness. Additionally, the measurements may be subjective and could be influenced by personal biases and preferences.

2) Internal Bias: Randomness in our system could lead to different results. To mitigate this threat, we repeated our experiments with different random number seeds.

B. Future Work

There are several potential improvements that can be made to the hyperparameter optimization (HPO) process used in our implementation. One such improvement is to do an ablation study. Ablation study involves taking out one node at a time to see when it breaks. This method will help us to greatly improve the algorithm and make it more efficient.

Another potential improvement involves conducting a February study to determine if hyper parameter optimization can be done on a smaller sample of data using the same fishing tool. This study could help determine if the accuracy of the hyper parameter optimization tool can be improved while still maintaining the same number of evaluations on other data.

Overall, these improvements have the potential to significantly improve the efficiency and accuracy of our hyper parameter optimization tool, and should be considered in future iterations of the tool.

VI. Conclusion

Our tool outperformed the baseline tool assuming hyperparameters were set in advance. This is useful in cases where the dataset is expanding and we may need to run these evaluations several times in the future. However, our research has shown that in more complex scenarios, a fixed set of hyperparameters cannot be used and self-optimization through hyperparameter tuning can significantly impact runtime. Overall, our research has provided valuable insights into the use of the fishing tool for automated software engineering and identified areas for future improvements. Since we conducted hyper parameter optimization for each dataset, we did not beat the baseline tool in terms of number of evaluations. However, by going down the best half for hyperparameter clustering, we were able to achieve better results than the original sway using a slightly higher number of evaluations.

We also attempted to use DBSCAN as the clustering algorithm for sway, but the high runtime complexity proved to be a limiting factor. We speculate that using our tool in a February study could provide further insights into its performance compared to the baseline tool. This is because a small number of evaluations could potentially create a classifier with similar output. Therefore, an exploration of using a February study could be a valuable direction for future research.

Overall, our research has provided valuable insights into the use of the fishing tool for self-optimization and identified areas for future improvements. We hope that our findings will inspire further research in this area and contribute to the development of more efficient and effective automated software engineering tools.

Acknowledgment

We would like to express our sincere gratitude towards the Department of Computer Science at North Carolina State University and Professor Dr. Menzies for their invaluable guidance and support in the optimization of our fishing tool using machine learning techniques.

Dr. Menzies' expertise and insights into the field of automated software engineering have been instrumental in the successful completion of this project. We are grateful for the opportunity to work with such a knowledgeable and dedicated mentor.

We would also like to thank the staff and faculty at the Department of Computer Science for providing us with the resources and support necessary to carry out this research.

This project would not have been possible without the contributions and efforts of everyone involved, and we are proud to have had the opportunity to work alongside such talented individuals. Thank you for your valuable assistance and guidance.

References

[1] A. Agrawal, X. Yang, R. Agrawal, R. Yedida, X. Shen and T. Menzies, "Simpler Hyperparameter Optimization for Software Analytics: Why, How, When?," in IEEE Transactions on Software Engineering, vol. 48, no. 8, pp. 2939-2954, 1 Aug. 2022, doi: 10.1109/TSE.2021.3073242.

[2] Zitzler, E., Künzli, S. (2004). Indicator-Based Selection in Multiobjective Search, In: , et al. Parallel Problem Solving from Nature - PPSN VIII. PPSN 2004. Lecture Notes in Computer Science, vol 3242. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-30217-9_84

[3] J. Chen, V. Nair, R. Krishna and T. Menzies, "'Sampling" as a Baseline Optimizer for Search-Based Software Engineering," in IEEE Transactions on Software Engineering, vol. 45, no. 6, pp. 597-614, 1 June 2019, doi: 10.1109/TSE.2018.2790925.

[4] R. Yedida, R. Krishna, A. Kalia, T. Menzies, J. Xiao and M. Vukovic, "Lessons learned from hyper-parameter tuning for microservice candidate identification," 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), Melbourne, Australia, 2021, pp. 1141-1145, doi: 10.1109/ASE51524.2021.9678704.

[5] Abdel Salam Sayyad, Tim Menzies, and Hany Ammar. 2013. On the Value of User Preferences in Search-based Software Engineering: A Case Study in Software Product Lines. In Proceedings of the 2013 ICSE (San Francisco, CA, USA) (ICSE '13). IEEE Press, Piscataway, NJ, USA, 492.
[6] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. 2002. SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization. In Evolutionary Methods for Design, Optimisation, and Control. CIMNE, Barcelona, Spain, 95–1
[7] Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.
[8] Tim Menzies, Ye Yang, George Mathew, Barry Boehm and Jairus Hihn. Negative Results for Software Effort Estimation, 2016; arXiv:1609.05563.
[9] Vivek Nair, Zhe Yu, Tim Menzies, Norbert Siegmund and Sven Apel. Finding Faster Configurations using FLASH, 2018; arXiv:1801.02175.
[10] Romano, J., Kromrey, J. D., Coraggio, J., Skowronek, J. (2006, February). Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen'sd for evaluating group differences on the NSSE and other surveys. In annual meeting of the Florida Association of Institutional Research (Vol. 177, p. 34).