CSC591 Final Project Report

Connor Smith
Email: cpsmith6@ncsu.edu
Unity ID: cpsmith6

Ashvin Gaonkar
Email: agaonka2@ncsu.edu
Unity ID: agaonka2

Liwen "Cynthia" Du
Email: ldu2@ncsu.edu
Unity ID: ldu2

1. Introduction

Learning to draw conclusions from large sets of data has been one of the leading problems in the modern computer science industry. Many different algorithms have been developed for a variety of different datasets with the hope to draw meaningful connections between different points of data in the least expensive manner possible. There are a number of techniques used to address these problems, but some of the more popular solutions are: Genetic Algorithms, Neural Networks, and Support Vector Machines (SVMs).

However, these machine learning algorithms are designed to work best in specific situations, and when a dataset is not idea for the algorithm, optimality of our results begin to suffer. With Genetic Algorithms, the randomness of mutations could cause the evolution of our generations to settle on a less than ideal solution if it is unable to get out of a minima or maxima. Neural Networks have a very difficult time learning over time when working with a limited dataset. SVMs can have drastically different results based on the hyperparameter values you use, which can lead them to be unreliable.

Each one of these techniques has pros and cons, but that is where SWAY[7] comes in. SWAY attempts to cluster data efficiently and effectively, working with datasets of a variety of sizes to make a best-effort prediction of the optimal set of data points from a dataset. Working with this clustering algorithm yielded great results throughout this past semester, but it isn't without it's issues. This issues mainly boil down to how it handles missing data points and potential issues when using Zitzler's Domination Predicate to determine which half is better. The XPLN algorithm is an effective explanation algorithm to compliment the SWAY clustering algorithm, offering a reliable way to generate a rule set based off resulting cluster data to offer a significantly faster way to try and find the best data points from a dataset, albeit with an inevitable sacrifice in reliability and accuracy. In exploring this algorithm, we found that the results of this algorithm were already quite high, and that there was not much room for getting more accurate results. Our focus instead was on improving the efficiency of the algorithm instead. For both algorithms, we also wanted to explore if changing the hyperparameters yielded improvements as well.

Rather than developing a new algorithm from scratch, we sought to make optimizations to the SWAY and XPLN algorithms with the hopes these changes would be significant enough to yield algorithms better than the original implementations. In regards to SWAY, in the original implementation, if there were missing values, we skipped over these data points.

A. Structure

Approaching this project, we had two main goals: "Can we make optimizations to SWAY that yield more optimal results than the original implementation?" and "Can we make optimizations to XPLN that yield faster runtimes with no sacrifice to accuracy in the worst case, and possible some improvements in accuracy in the best case?" With regards to our optimizations to SWAY, we were and were not successful in accomplishing our goal. Our improvements to SWAY did yield more optimal results across all 11 datasets for almost every objective within these datasets, but there was not a statistically significant difference between sway2 (our version of SWAY) and sway1 (SWAY from Chen *et. all*[7]). Our optimizations of XPLN yielded similar results. We were able to reduce the runtime of XPLN quite considerably in some instances, but normally by not a statistically significant difference though.

For our study, we did not use solely the best row for each algorithm for each of the 20 runs we performed. We understand this is different than what was proposed before through discussions with instructors, but we believe that taking the average of the best cluster would yield more meaningful results. The analogy of a clothes store was given by TA Rahul Yedida about a clothes store where if we are looking to buy something we would want to know what the very best articles of clothing, not the average of a pool of best articles of clothing. While we certainly understand, and can agree to an extent, this philosophy we believe that it is quite situational to observe only the singular best row our algorithm returns. There could be the very realistic situation where the single best row we find is truly an ideal row, but everything else we find in our best cluster are horribly optimized results. The analogy of the clothes store breaks down as soon as we consider buying more than one article of clothing, and now you are in the situation we have presented. By using the mean of the best cluster, we are finding the average of the best, whereas the average of the best row from each run is the average of the best of the best. The average of the best can offer stronger insights into how the algorithm performs overall from an optimality standpoint versus the performance being based off 1 singular data point. There is room to argue that basing our averages off of one single data point per run would defeat to purpose of clustering in the first place as well. If we only care about the best row, why not just find the very best row? Sure it is more expensive,

but the very best row is our metric for statistical analysis, so why cluster? The best row is still contained within the cluster, so while the values from this row are being somewhat masked by the values of the other rows in best, it still is contributing to the overall strength of the algorithm's results. For explanations sake, and to demonstrate this is not simply just "choosing the path of least resistance", you could update the existing code to determine the normalized Utopia point (1 for an objective if you want to maximize an objective, and a 0 for an objective we want to minimize), find the normalized position of our data points in best from the entire data set, then sort our rows based off their distance to the Utopia point, and then take the first row, which should have the smallest distance to Utopia. This is our ideal point on the Pareto Frontier. This wouldn't involve much new code at all, as it leverages code and processes that are already being used elsewhere for SWAY, but we once again believe this is not as accurate a measure of performance of our algorithms as the average of the best cluster for each run would yield.

Our other possible major point of contention could be why we sought to make changes to SWAY and XPLN instead of implementing other clustering/optimization/explanation algorithms? The answer for this was simply because many other methodologies are already well-documented, that could have yielded more significantly different results, but probably would have already been in-line with what is expected from those techniques already. SWAY and XPLN are proprietary machine learning algorithms, so potential results for changes to these algorithms would be more novel than simply swapping over to a K-means clustering algorithm or something similar.

2. Related Work

In deciding our approach, we examined past work to decide whether changes to SWAY or implementing an entirely different algorithm would yield more useful results. Exploring semi-supervised machine learning algorithms, we sought to see if there were viable options that only need a portion of the data to be labeled to find optimal data points from potentially large datasets. This was also done with the hope that the optimization process would be significantly more efficient since there are less attributes of the dataset that need to be taken into consideration when searching for the best rows[2].

One option we looked at were Genetic Algorithms. These are algorithms that use generation improvements to seek out optimal solutions[3]. Using this type of algorithm for semi-supervised explanations, we would iteratively select a subset of our current data that maximizes our fitness function as much as currently possible. SWAY does something similar by selecting a subset of the data on each iteration based off a distance metric to a chosen point of data. However, genetic algorithms have a drawback that can be hard to reconcile. They can get stuck in local minima/maxima depending on how we are optimizing a particular objective[1]. This can lead to inefficient exploration of our data, making it an unideal solution.

Another implementation choice was Neural Networks. This is a powerful machine learning tool that works well for semi-supervised learning. Neural Networks would be able to take some input and the labeled points of the data can be used to predict the values of the unlabeled data[4]. SWAY also does this, albeit, in a much different fashion. We felt this still was not a good option for a couple of reasons. Firstly, if there is not a lot of labeled data your neural network has access to, your predictions could be quite inaccurate. We had datasets of very varied sizes, so this didn't seem like a good option to work with the 11 datasets we were utilizing. Secondly, none of our group members had worked with neural networks in a capacity that would be required for an effective implementation, so the tech skill curve was too great to overcome with the time given.

The last option we were considering was SVMs. SVMs work well for trying to classify data by splitting data iteratively that creates the most distinction between classes[5]. The original implementation of SWAY uses a similar technique by selecting the A and B points from the recursively splitting half function and using these point in Zitzler's Domination Predicate as representatives of each half to determine which is better. SVMs are very sensitive to the hyperparameter values[6], so it would have required a lot of experimentation to come up with an optimal set of hyperparameter values and since there were a lot of datasets, some of which were very large, being run 20 times using all 6 techniques on each run, would lead to a lot of waiting time trying to generate results. This would have likely been the next best option.

However, it is worth noting that SWAY takes a lot of the strong points from these popular machine learning methodologies that yields an effective and efficient solution that works for a wider variety of datasets. This examination resulted in the decision to seek improvements to SWAY, since it was unlikely we would get better results with a good portion of the datasets using another option.

3. Methods

A. Algorithms

1) all: This is the simple average of each objective column in a dataset.

2) sway1: This is the original SWAY[7] algorithm. This algorithm works by selecting a row, determining the 95% (by default) farthest points from this data point, sorting all other rows based off their distance to the randomly selected row, then using Zitzler's Domination Predicate to determine which half of the data is better by using a representative for each half, and recursively following this process until we reach clusters within a certain size. For this algorithm to work properly for all 11 datasets, we needed to update our Python implementation to handle rare divide by 0 errors, properly converting string values to floats when they are read in from the CSV files, and make sure proper behavior is followed when "?" values were encountered (we skip these values).

3) sway2: This works exactly the same as sway1, with a few differences. First, whenever a missing value ("?") is encountered, instead of skipping the value, we assume that

value is actually whatever the current average for the column currently is. This is done because the row may otherwise be a very ideal data point, but missing 1 value now excludes it from being a potential contender. By using the current average of the column, we make a best-effort guess for what the data point actually is. If a value is a symbol, we use the mode instead of the average. Additionally, we do not use the resulting A and B points from the half function as representatives for each half. We instead create a new temporary data point that represents the true center of the half. Our A and B points lie at the edges of each half, and while they do make a clear distinction between the halves, they are not accurate representations of each half, which could lead to the wrong half being selecting by Zitzler's Domination Predicate.

4) xpln1: This is the original XPLN algorithm. This algorithm works by taking the data from the resulting cluster from SWAY (best and rest), organizing the data into bins, generating ranges from the bins and score them by finding the weighted average of the objective columns. We then sort all these ranges based off the quality score in descending order, and filtering out all ranges that have a quality score of 0.05 or below and ranges that are less than 1/10 the quality score of the best range. We then iterate through the remaining ranges, and see if a rule can be generated for each of the ranges. If a rule is not generated for a range, it is because the features found through the pruning process are all found to be unnecessary, which means the range itself does not yield any useful rule. We then select the rows the satisfy the rule from the best and rest rows to determine the number of positives/negatives and false positives/negatives. We ideally want the rule to select everything in best and nothing in rest. We then determine a quality score of our newly selected best/rest rows based off our current rule. If the quality score is better than our current best know quality score for a rule, our new rule becomes the current rule. This process repeats for all of our ranges and we return the rule and quality score of the best rule we found in our ranges. While not a part of XPLN explicitly, we can then use this rule to select all the rows from the original data set that match this rule. We can then use these selected rows and compare them to the quality of the rows from SWAY. This will allow us to see how optimal our results are using the explanation algorithm, a cheap operation, versus an expensive clustering/optimization algorithm.

5) xpln2: This works exactly the same as xpln1, but the filtering process of the ranges is differently. In xpln1, only ranges with a score of 0.05 or below and ranges less than 1/10 of the range with the highest quality score are filtered out. While this is done to keep the widest range of possibilities open, we feel this is an extreme breadth of options being left open. We instead filter all ranges with a score of 0.5 or below. We felt that if a range wasn't in the top half of possible scorings, that the potential rule it generated would be so poor that it isn't even worth the effort of exploring. This was done with the hope that by not having to examine ranges that were almost certainly not going to yield useful results, we could save time by not wasting time exploring their results.

6) top: Simply finds the true best N rows.

B. Data

Our algorithms were applied to the following dataset domains:

1) Car Design: auto2.csv and auto93.csv belong to car design domain. This dataset has various attributes such as "Weight" (The weight of the car in pounds), "CityMPG+" (The estimated miles per gallon the car can achieve in city driving conditions), "Class-" (The class of the car). This data can be used to understand the correlation between multiple attributes of car design. This can help us answer some of the questions such as: Which car models are the most fuel efficient in city and/or highway driving conditions? Is there a correlation between a car's weight and its fuel efficiency?

2) Software Project Estimation: china.csv, coc1000.csv, coc10000.csv belong to software project estimation, From this data, insights can be extracted about the relationship between the size and complexity of a software project, the number of resources required, and the effort required to complete the project. There are various attributes in this dataset, for example, "N_effort-" (The estimated effort required to complete the project in person-hours), "Duration" (The estimated duration of the project in days). The columns can be grouped into several categories: LOC+, CPLX, DATA, and STOR relate to the size and complexity of the software being developed. ACAP, AEXP-, PCAP, PLEX-, TEAM, and TOOL relate to the capability, experience, and resources of the development team. ARCH, DOCU, LTEX, and PMAT relate to the level of software engineering best practices being used in the project.

3) Issue Close Time: healthCloseIsses12mths0001-hard.csv, healthCloseIsses12mths0001-easy.csv, belong to issue close time domain. This data is used to predict the time it takes to close an issue in a software project. There are various attributes in this dataset, for example, "MRE" (Mean Relative Error of the model), "ACC+" (Accuracy of the model), "PRED40+" (Percentage of predictions within 40% of the actual close time).

4) Agile Project Management: pom.csv, belongs to agile project management domain. This data is used to assess and manage various factors that could impact the success of an agile project.

5) Computational Physics: ssm.csv and ssn.csv, belong to the computational physics domain. This data is used to evaluate and optimize numerical solution methods and algorithms for solving problems in computational physics. The columns represent different aspects of the solution process, such as the algorithm used, the type of cycle, and various parameters used to control the convergence rate.

C. Analysis Measurements

The summarization methods we are using will be the original and updated versions of sway/xpln, the all method, and the top method as explained in section 2A. As for how we are evaluating the performance of these algorithms, we are using a few different metrics. First, we are examining the

average values for the objective columns in our best cluster (where clustering is used) and then finding the average of the columns over 20 repeated runs using a new seed value for each run. See section 1A for our justification on this approach. This resulting data is then used to perform Scott-Knott for each objective column for each dataset, which ranks the data and displays their distributions. We are also using our average data to find the sampling and explanation taxes between our updated algorithms and the original algorithms to have enough metric for improvement. For both versions of xpln, we are also monitoring the runtime of the algorithm and then also using the average runtimes to also apply Scott-Knott to those as well to see if there is a statistical difference in runtimes between our two explanation algorithms.

4. Results

Through testing the SWAY and XPLN algorithms, we were able to make interesting conclusions from the data we received. All of the raw output can be viewed here. A lot of data discussed will not have LaTeXtables created for space/duplication purposes. The data included in the repository shows the raw averages for each objective column over 20 rules for each file, Scott-Knott percentile plots, ranks, and quartile breakdowns for each objective for each file, a table displaying whether objective column results equal the outputs of other algorithms, the average runtimes for xpln1 and xpln2 for both XPLN algorithms, a Scott-Knott result table for the XPLN runtimes similar to the objective columns Scott-Knott table, and sampling and explanation taxes for all files.

A. SWAY Results

First, we examined what happened when no changes were made to the hyperparameters from what we have been working on previously throughout the semester. We grouped the datasets by size: small (¡=1000 data points), medium (1000-10000 data points), and large (¿10000 data points). We first examined how the small data sets performed.

Just from observing the raw average results from table 1, we can see that for the most part there appear to be slightly more optimized results from sway2 compared to sway1, but we have more variability with xpln2 and xpln1. This variability is expected with xpln in general since this is generally going to be a less accurate explanation solution compared to using a clustering/optimization algorithm like SWAY. When looking at SWAY though, it is clear without even looking at Scott-Knott results that there is not a statistically significant difference between sway1 and sway2 for Acc+. However, we did find that there were statistically significant differences between sway1 and sway2 for Lbs- and Mpg+, with sway2 being ranked better than sway1 for both of these objectives. This type of pattern was also consistent when observing another smaller dataset, coc1000.csv. Looking at table 2, we can see the positive results for this file were even more profound than for auto93.csv. Once again, for all objectives sway2 yielded more optimal results, but with even greater statistical significance this time.

auto93.csv Objective Averages (No Hyperparameter Changes)			
	Lbs-	Acc+	Mpg+
all	2970.42	15.57	23.84
sway1	2159.9	16.97	30.53
xpln1	2332.09	16.64	29.12
sway2	2038.04	16.71	35.88
xpln2	2211.02	16.19	31.58
top	2005.08	19.81	40.77

TABLE 1. Shows the average values of the best cluster for each objective in auto93.csv over 20 runs for each explanation/clustering technique

coc1000.csv Objective Averages (No Hyperparameter Changes)					
	LOC+	AEXP-	PLEX-	RISK-	EFFORT-
all	1011.78	2.97	3.04	6.66	30923.04
sway1	1029.64	2.96	2.98	5.17	27703.54
xpln1	981.55	2.97	3.04	6.70	29600.32
sway2	1056.82	2.81	2.91	3.70	22305.61
xpln2	984.85	2.96	3.05	6.21	26974.40
top	1591.61	1.62	1.31	4.94	35220.62

TABLE 2. Shows the average values of the best cluster for each objective in coc1000.csv over 20 runs for each explanation/clustering technique

For objective LOC+, sway2 was ranked 4 out of 5 using Scott-Knott, while sway1 was only ranked at a 2. For objective RISK-, sway2 was ranked 1 whereas sway1 was ranked 2. The same pattern also occurred for EFFORT-. Not shown in charts here, similar trends also occurred in the auto2.csv and nasa93dem.csv files where one or more objective columns had sway2 ranked better than sway1, and almost every other column had sway2 as more optimal than sway1, but not by a statistically significant amount. There was one anomaly found in the data. Kloc+ in the nasa93dem.csv file had sway2 ranked 1 with sway1 ranked 3. This was the only instance in any objective column where sway1 performed better than sway2. While the reason isn't certain, it could be that because this was a largely symbolic dataset, the changes we made to how the representatives were selected for Zitzler's Domination Predicate don't perform well for symbolic data. This would suggest that at least for smaller datasets with 1000 or less data points, sway2 yields better results almost all the time and some of the time we also get a statistically significant improvement over sway1.

For our medium datasets (1000-10000 data points), we wanted to see if we saw similar positive benefits using sway2 over sway1.

Already from looking at table 3, we can see there are no different patterns on display here that were not present in the smaller datasets. For all objectives, sway2 was more optimal than sway1, with only the Loc+ objective yielding there was a statistically significant difference in results between sway1 and sway2. Sway2 was ranked 4, whereas sway1 was ranked 1, a pretty drastic difference in ranking. For

coc10000.csv Objective Averages (No Hyperparameter Changes)			
	Loc+	Risk-	Effort-
all	1010.25	6.71	30941.02
sway1	1003.85	3.92	21865.54
xpln1	999.88	6.76	28050.14
sway2	1062.09	2.71	18709.24
xpln2	1004.24	5.93	24694.24
top	1941.94	0.4	19642.31

TABLE 3. Shows the average values of the best cluster for each objective in coc10000.csv over 20 runs for each explanation/clustering technique

SSM.csv Objective Averages (No Hyperparameter Changes)		
	NUMBERITERATIONS-	TIMETOSOLUTION-
all	33.06	512.96
sway1	7.04	160.14
xpln1	8.35	164.86
sway2	4.78	116.65
xpln2	8.59	149.20
top	3.66	62.34

TABLE 4. Shows the average values of the best cluster for each objective in SSM.csv over 20 runs for each explanation/clustering technique

SSN.csv Objective Averages (No Hyperparameter Changes)		
	PSNR-	Energy-
all	44.43	1673.84
sway1	44.38	1098.41
xpln1	44.33	1298.83
sway2	44.71	674.33
xpln2	44.92	901.37
top	25.93	501.45

TABLE 5. Shows the average values of the best cluster for each objective in SSN.csv over 20 runs for each explanation/clustering technique

all the other medium datasets (healthCloseIsses12mths0001-hard.csv, healthCloseIsses12mths0001-easy.csv, and pom.csv), this exact same pattern occurred. All objective columns were more optimal, though with these there was no more than column that had a statistically significant improvement from sway1 to sway2. The only file this did not apply to was healthCloseIsses12mths0001-easy.csv, which had 0 columns where the optimization to sway2 yielded a statistically significant improvement when compared to sway1. The only noteworthy anomaly was in pom.csv. For objective Completion+, sway2 yielded the least optimal result of all methodologies. The result from sway2 was even less optimal than the result from just taking the raw average of all the values in the Completion+ column from the original dataset. Despite this performing worse, it was not by a statistically significant amount where top was the only approach that was statistically different than any of the others.

The ultimate test is to see how sway2 performs against sway1 with our two large datasets, SSM.csv and SSN.csv.

It is harder to analyze the trends in the larger datasets because there are not only fewer files compared to the small and medium datasets, but there are also only two objectives per file in the large datasets as well. Therefore, an analysis can only be performed off of the results we have. For SSM.csv, sway2 was more optimal than sway1 for both objectives, but neither were statistically significant. For SSN.csv, sway2 was slightly less optimal than sway1 for objective PSNR-, but the difference in sampling tax between sway2 and sway1 was only -0.3, which for double digit numbers with values typically in the mid-40s, is clearly a very minuscule difference. For objective Energy-sway2 had a statistically significant improvement over sway1. For large datasets, these results make sense given the changes we made. For very large datasets (particularly SSM.csv with 240k data points), the impact missing values are going to have is very small and there are so many data points that finding the true center of each half to use as a representative is not necessary, so results don't change much. However, with the large datasets we did see improvements in the amount of variability you get for the averages between runs. The quartile plots for the NUMBERITERATIONS- objective for SSM.csv had a range of values from 4.22 - 19.22 for sway1, whereas it was only 4.02 - 5.66 for sway2. TIMETOSOLUTION- also had a similar variance decrease from 90.30 - 356.15 for sway1 and 86.98 - 181.73 for sway2. SSN.csv did see variance improvements, but there were nowhere near as pronounced as they were for SSM.csv.

1) Notable Hyperparameter Changes: We tried manipulating the bins, cliff, min, and rest hyperparameters, along with manipulating how the seed is generated before each run to see if this would yield different results than the default values we have been using this semester. Changing min was the only thing that had a significant difference on the performance of our algorithm. As you can see from table 6, compared to table 2, the results are more optimal when we decrease min to 0.25, which effectively decreases the size of our resulting cluster. While the results for sway2 become more optimal, the results for sway1 do as well. When these statistics were generated, we did not store the values of each individual run in order to do a proper Scott-Knott analysis to determine if the difference between the improvements from decreasing min from 0.5 to 0.25 between sway1 and sway2 were statistically

coc1000.csv Objective Averages (min decreased from 0.5 to 0.25)					
	LOC+	AEXP-	PLEX-	RISK-	EFFORT-
all	1011.78	2.97	3.04	6.66	30923.04
sway1	1003.59	2.71	2.79	4.51	27920.84
xpln1	1020.31	2.94	3.05	6.20	29807.33
sway2	1156.12	2.67	2.38	2.95	21064.41
xpln2	1018.09	2.95	3.07	5.95	26700.77
top	1557.75	1.25	1.50	2.25	27871.50

TABLE 6. Shows the average values of the best cluster for each objective in coc1000.csv over 20 runs for each explanation/clustering technique

significant or not. In order words, did sway2 see more, less, or the same degree of improvement when compared to sway1? Because of this, examining the percent difference between the objectives and methods is the best analysis that can be performed. While the one file is a small sample, we can see

coc1000.csv Objective Averages % Differences (min 0.5 to 0.25)					
	LOC+	AEXP-	PLEX-	RISK-	EFFORT-
sway1	-2.53%	-8.45%	-6.38%	-12.77%	0.78%
sway2	9.40%	-4.98%	-18.21%	-20.27%	-5.56%

TABLE 7. Shows the percent difference between average values of the best cluster for each objective in coc1000.csv over 20 runs for both SWAY algorithms

that table 7 demonstrates that purely from a percent different standpoint, there does initially appear to be quite a bit of an improvement that sway2 receives from the min hyperparameter being reduced when compared to sway1. This may be due to sway2 doing a better job at handling random noise than sway1 does. This pattern is observed across other files with varying degrees of improvement, but reducing the size of the cluster through min appears to improve the quality of the resulting cluster for both SWAY algorithms, with the possibility of this improvement being more significant for sway2 than sway1.

To further confirm that the decrease of min from 0.5 to 0.25 was truly the source of the improvement, we also examined the results of increasing min as well. Doing this resulted in larger clusters, which means more data points that are less ideal, so we found that for sway1 and sway2, we got less optimal results than the default value and certainly the 0.25 value we experimented with as well.

B. XPLN Results

Our other objective was to see if we could decrease the runtime of xpln1 without sacrificing accuracy in the worst case. Similarly with SWAY, we first explored what happened with the default hyperparameters.

To first address differences in accuracy between xpln1 and xpln2, there were virtually no differences in accuracy between xpln1 and xpln2. According to Scott-Knott, whether xpln2 was more optimal than xpln1 was a near even coin toss across all files of all dataset sizes. There were only a couple of instances where xpln2 was ranked better than xpln1, but this was not a common trend for any dataset. The accuracy results were so unremarkable, this is where the analysis of accuracy will end. The full percentile and quartile plot breakdowns are available on the project repository here if you want to see more details of the results yourself.

With great success in our accuracy objective, we can now see how much runtimes improved. We broke this down, similarly with the SWAY analysis, by dataset size.

For the small datasets, there was a negligible runtime difference between xpln1 and xpln2 from a raw numerical difference and statistical significance perspective. Across all the small datasets, there was only a 0.75ms difference between xpln2 and xpln1, with xpln2 being faster. All the

auto2.csv XPLN Average Runtimes	
xpln1	7.69ms
xpln2	6.95ms

TABLE 8. Shows the average runtime of the XPLN algorithms for auto2.csv

Scott-Knott results yielded there was of course no statistically significant difference between the runtimes. With the smaller datasets, we must be hitting the minimum required runtime of the algorithm, so any improvements will have a small impact than when working with larger datasets, and then begins to become apparent with the medium datasets. Once

coc10000.csv XPLN Average Runtimes	
xpln1	59.8ms
xpln2	28.02ms

TABLE 9. Shows the average runtime of the XPLN algorithms for coc10000.csv

we get to somewhat larger datasets, we really begin to see the benefits of the improvements of xpln2 over xpln1. For coc10000.csv, xpln2 runs in half the amount of time xpln1 takes. According to Scott-Knott, this still is not a significantly different result, which is somewhat surprising. For examining runtime, Scott-Knott analysis is not as important as just the straight runtime benefit. When examining runtime, time is our only objective, whereas examining our SWAY results, the quality of the answer matters a lot more since SWAY is a more expensive action than XPLN, so we still view this as a great result. This carried throughout the other medium datasets as well. For coc1000.csv xpln1 took 18.27ms on average and xpln2 took 8.18ms on average, healthCloseIsses12mths0001-hard and healthCloseIsses12mths0001-easy both had xpln1 take 14ms on average and xpln2 took 8ms on average, and pom.csv had xpln1 take 26.92ms on average and xpln2 took 13.12ms on average. Despite none of these being statistically significantly different, from raw numerical averages, there is a pretty drastic difference in runtime.

SSM.csv XPLN Average Runtimes	
xpln1	156.96ms
xpln2	96.65ms

TABLE 10. Shows the average runtime of the XPLN algorithms for SSM.csv

SSN.csv XPLN Average Runtimes	
xpln1	83.91ms
xpln2	64.50ms

TABLE 11. Shows the average runtime of the XPLN algorithms for SSN.csv

For large datasets, we see less of an improvement for xpln2 versus xpln1. There is still some improvement, but it is not by a magnitude of two like was present for the medium datasets. It could be that the number of ranges that need to be tested in large datasets take longer to comb through, even when we filter out the likely bad options. This is similar to the reason why there are negligible improvements for the smaller datasets, but

on the opposite end of the spectrum. There is a minimum amount of time that will be required to simply comb through all the options and evaluate the quality of the ranges from our bins. Scott-Knott also determined there were no statistically significant differences between the average runtimes for the large datasets as well.

1) Notable Hyperparameter Changes: We tried changing all the same hyperparameters as we did in section 3A-1. Interestingly, it was common for the xpln1 and xpln2 runtimes to be near identical with only 1ms difference between them. For most of the hyperparameters adjustments (increasing/decreasing all of the hyperparameters from section 3A-1), we were unable to determine the reason for this. The only hyperparameter we were able to properly analyze was once again the min hyperparameter. By reducing min to 0.25 like we did for SWAY, the accuracy for xpln2 improved, albeit, at the cost of the increased efficiency over xpln1. The runtimes between xpln1 and xpln2 being more similar makes sense because if our resulting cluster from SWAY is smaller, then there are fewer things that xpln1 and xpln2 are likely to filter, so they are examining a similar number of range options, causing similar runtimes since the other mechanics of the algorithm are similar. The accuracy improvements are strange though. We were unable to explain why simultaneously the accuracy of xpln2 could improve, but the runtimes of xpln1 and xpln2 became more similar. If the runtimes are becoming similar, then that would mean that xpln1 and xpln2 have a similar set of ranges they are exploring for rules options, which means they should have similar resulting objective values, yet xpln2 consistently outperformed xpln1 with this hyperparameter changed. It was more common for xpln2 to be statistically significantly better when compared to xpln1, according to our Scott-Knott rankings.

5. Discussion

While our results were somewhat a mixed bag, we do feel that are fairly conclusive. A possible point of argument from our results/methodology is the use of the means of the best cluster, versus just using the best row from the best cluster from each methodology on each run. This is a valid criticism that was largely addressed in section 1A, but an improvement to our results could have been exploring if there were more statistically significant differences between the algorithms when just using the best row from the best cluster for each algorithm on each run. To compensate for the loss of data, 30 or 40 runs could have been performed instead of just 20 to ensure more stable results. Another argument could be the seemingly drastic move to drop all ranges in xpln2 that don't have a quality score greater than 0.5. It is entirely possible that all ranges don't have a score greater than 0.5, so no rule can be generated at all. While this is an extreme change from xpln1, we feel that if there aren't any ranges that have a quality score greater than 0.5, then one of two things must be true: there are fairly loose associations among attributes in your best cluster, or your clustering/optimization algorithm does a poor job at finding the optimal data points in the dataset. If no range exists that has a quality score over 0.5, then there probably shouldn't be a rule for the cluster in the first place.

There are places to go from the work we've done we simply didn't think of at the time of development, or didn't have the time/experience to properly implement. The first being that instead of simply selecting the average of each half when recursively splitting to determine the representatives for Zitzler's Domination Predicate, maybe select 5 data points at random, find their average, and use that as your representative and repeat for the other half. This would prevent potential issues in a situation where when the data is split in half, you have a large cluster of data, a gap, and then a smaller cluster of data all in the same half. Taking the raw average could lead to a data point that is in the gap, which is not a good representation of there exists no data that would occupy that space otherwise. Using 5 random rows and taking their average increases your chances that you are finding the center of the main body of data in your half. Another place open for exploration is using different distance metrics or different scoring metrics. For distance, we use simple Euclidean distance, but different results may have arisen if we used something like Manhattan distance. The same applies for the use of a simple weighted average when calculating the quality score. Lastly, from our results, we found that sway and xpln are very sensitive to the hyperparameter values, similar to a SVM. There were a variety of datasets that yielded quite different results from changing the min, and to a lesser degree, rest hyperparameter values, as detailed in our Results section. This can cause problems because it now creates the issues of trying to figure out the ideal set of hyperparameters for each dataset. There are two ways to address this: either use a different clustering/optimization/explanation algorithm altogether that is less sensitive to hyperparameter values (like Genetic Algorithms or Neural Networks), or implement an algorithm that would optimize the hyperparameter values automatically for each file. Both of these options have pros and cons. Using a different algorithm entirely would involve trade-offs that cannot be fine tuned at all as discussed in the Related Work section, and attempting to optimize the hyperparameters for each file could be a very expensive operation in terms of time and processing power, but would allow for the continued use of the SWAY and XPLN algorithms.

6. Conclusion

While our two algorithms fell into the category of being more of a micro-optimization than a sweeping improvement, we still had very promising results that our changes to SWAY and XPLN did truly yield better results than the original implementation. There is plenty of room left to continue making more improvements from what we have implemented. As Wolpert and Macready said, there is no machine learning algorithm that will be perfect for all situations[8], but there is a lot of promise in the SWAY and XPLN algorithms that this gap can be shortened more in the future.

REFERENCES

[1] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, Apr. 2002, doi: 10.1109/4235.996017.

[2] X. Zhu, A. B. Goldberg, and L. Li, "Semi-supervised learning literature survey," Technical Report, University of Wisconsin-Madison, Department of Computer Sciences, 2009.

[3] K. Deb, *Multi-objective optimization using evolutionary algorithms*. John Wiley & Sons, 2001.

[4] S. Laine, G. Azzopardi, and D. Hiemstra, "Neural ranking models with weak supervision," in *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2019, pp. 345–354, doi: 10.1145/3331184.3331376.

[5] O. Chapelle, B. Schölkopf, and A. Zien, *Semi-supervised learning*. MIT Press, 2006.

[6] C. W. Hsu, C. C. Chang, and C. J. Lin, *A practical guide to support vector classification*. Tech. Rep., Department of Computer Science, National Taiwan University, 2002.

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

[8] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Trans. Evol. Comp.*, vol. 1, no. 1, pp. 67-82, April 1997.