Semi Supervised Explanation System for CSC 591

Bella Samuelson, insamuel@ncsu.edu
Katherine Mitchell, ksmitch3@ncsu.edu
Rithik Jain, rjain25@ncsu.edu

***Abstract*—The goal of this project is to write a multi objective, semi-supervised explanation system. Software engineers are often faced with the problem of finding the best solution, reducing cost but maintaining quality, minimum and efficient testing. Searching algorithms help to solve these problems. In this paper we optimized a searching algorithm: Sampling Way (Sway). The original SWAY algorithm used Euclidean distance. We optimized SWAY by using Manhattan Distance which takes into account the covariance structure of the data to provide a more accurate and robust measure of distance in non linear and high dimensional space.**

I. Introduction

Search-based software engineering (SBSE) is a sub field of software engineering that uses search-based optimization algorithms to address software engineering problems. SBSE techniques use search algorithms to explore a space of possible solutions to a problem, and select the best solution based on a fitness function that measures the quality of the solution. A search problem consists of a search space (a search tree of possible solutions), start state, and goal state [1]. Initial state is the root of the search tree, actions are the branches and outcomes are the nodes. Search algorithms provide search solutions through a sequence of steps which help the AI move from start state to goal state. It helps to provide best solutions at lowest cost. Search algorithms work in two parts: define the problem and search the problem in the search place. Defining the problem by defining the following: initial state, state space (all possible states that can be achieved), actions, goal state, goal test and path cost. AI performs a goal test for every state achieved to see if it is the goal state and the search algorithm keeps working until the goal state is achieved.

There are various applications of search algorithms like GPS routing to find the optimal path, scheduling system at college, offices, public transportation, indexing of files in a database to retrieve data quickly, manufacturing to find a way to efficiently and economically produce a product. SBSE helps to solve difficult problems or problems with exponentially large datasets. However, there are few disadvantages of SBSE like it requires high computational power which is costly and it is hard to define a fitness function which can accurately measure the quality of a solution. No single search algorithm is best for all search problems. An algorithm may perform best for one problem but worst for another problem.

Many search algorithms are used to solve complex problems. In this paper we will optimize the SWAY algorithm (Sampling Way). SWAY is easy to implement and has fast execution time. However, Euclidean distance used in the SWAY algorithm and has limitations when dealing with non-linear or high dimensional space where distance between two points can be incorrect or even meaningless. In this paper, we propose the use of the Manhattan Distance which takes into account the covariance structure of the data to provide a more accurate and robust measure of distance in non linear and high dimensional space. We evaluated the performance of Manhattan distance in the search algorithm by comparing it to Euclidean distance. Our algorithmic approach has the following steps: define the initial space (Initialization), generate successor state from current state (successor function), check if the current state is goal state (goal test), define a distance metric to measure distance to each generated states (distance metric), maintain a frontier of states that have been generated but not yet explored (frontier), then we use the SWAY algorithm to select states from the frontier for exploration.

II. Research Questions

To do comparison analysis between SWAY1 and SWAY2, we organized our analysis around the following research questions (RQ):

(RQ1) To what extent is SWAY2 faster than SWAY1?
(RQ2) To what extent does Manhattan distance improve the performance of SWAY and other algorithms compared to when using the original Euclidean distance.

III. Related Work

Traditionally, there have been two main ways to approach machine learning: supervised and unsupervised learning.

Unsupervised learning trains a model using unlabeled examples. The model is forced to find patterns in these examples whose true classifications are not known. This is shown in our recursive clustering algorithm we implemented in class. One advantage of unsupervised learning is that it does not require the training data to be labeled, which saves on the time and resources that are required of manual classification. Some disadvantages are that results vary considerably in the presence of outliers and that it only performs classification and not regression tasks [2].

On the other hand, in supervised learning, each example data point fed to a model during training is labeled, meaning its true classification is known. In this case, the model is shown a set of real-world examples to base itself on. TheSummarization and `merge` algorithms we implemented in class are examples of supervised learning. Purely supervised learning can get quite expensive in practice, however: knowing what “classification” or “result” comes as a result of executing on a particular example requires expending a lot of time and resources. For example, if you are looking to buy a new car that has comfortable seats you would have to go test drive many cars to find the perfect one. Therefore, figuring out which car has comfortable seats i.e. knowing the classification is very time consuming.

Semi-supervised learning is a combination of supervised and unsupervised learning. A minimal amount of labeled examples and a large number of unlabeled examples are used to train a model. By doing this, cost can be minimized while also providing the benefits of both supervised and unsupervised learning. SWAY, which will be touched on more in this paper, is a great example of semi-supervised learning.

While looking for different ways to improve SWAY we became curious if there were different alternatives for the distance measure. In the existing implementation of SWAY we use a Euclidean Distance metric. Through our literature review we found some alternatives to Euclidean distance such as the Manhattan Distance and Chebyshev Distance. In the paper “Analysis of Euclidean Distance and Manhattan Distance in the K-Means Algorithm for Variations Number of Centroid K“ [3] the authors were interested in different distance metrics and their effect on the performance of a K-Means Algorithm. The authors stated that the distance matrix is an important factor that will affect the performance of the algorithm. Although Euclidean Distance is the most widely used distance matrix function, the results of the paper show the Manhattan Distance matrix method has better performance than the Euclidean Distance method [3]. The authors clustered on an Iris data set with various centroid values and found that for each centroid value the Number of Iterations using Manhattan Distance K-Means was less than Euclidean Distance K-Means. Taking the previous statements into account, we decided to investigate using Manhattan distance as an alternative to Euclidean Distance and used this change in SWAY2.

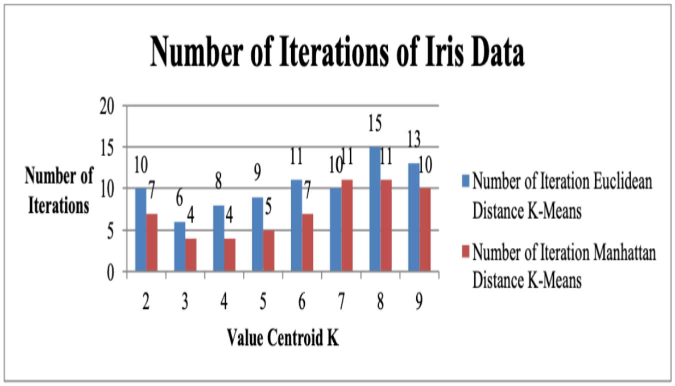
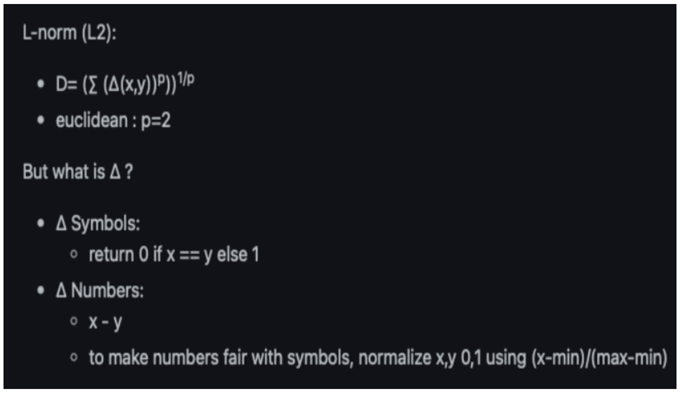


Fig. 1. Graph variation in Centroid Value (K) Iris Dataset [3]

IV. Methods

We started investigating the existing implementation of the dist function and found that it implements the L2 norm, Euclidean Distance shown in the Figure 2 below.



L-norm (L2):

- D= (Σ (Δ(x,y))^{p}))$^{1/p}$
- euclidean : p=2

But what is Δ ?

- Δ Symbols:
  - return 0 if x == y else 1
- Δ Numbers:
  - x - y
  - to make numbers fair with symbols, normalize x,y 0,1 using (x-min)/(max-min)

Fig. 2. Current Distance Formula in SWAY

The general equation for the L-norm (L2) is:

$$D = (\Sigma(\Delta(x,y))^p)^{\frac{1}{p}} \text{ [4]}$$

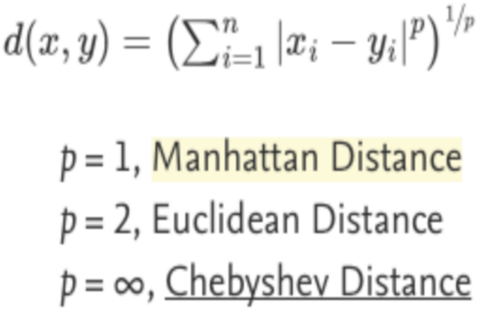
Δ is a function that computes the difference between two values x and y for the L2 norm with Euclidean distance i.e., $\Delta(x,y) = x - y$. If the values being compared are symbols or categories, such as in text or categorical data, the Δ function can be defined as follows:

Δ Symbols:

return 0 if x == y else 1

If two symbols are the same then Δ returns 0 meaning no difference between the two. If they are different then Δ returns 1 meaning difference between the two. If we use numeric data, then we should normalize the data to ensure that values are on the same scale. This improves performance and training stability of the model. We normalize x,y, 0, 1 using the formula:

$$\frac{(x - minValue)}{(maxValue - minValue)}$$



$$d(x,y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p}$$

$p = 1$, Manhattan Distance

$p = 2$, Euclidean Distance

$p = \infty$, Chebyshev Distance

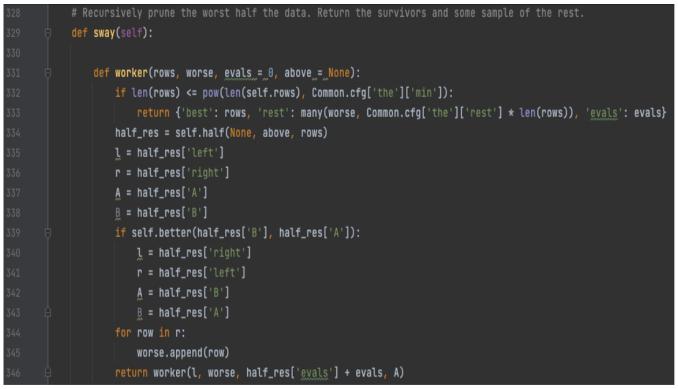
Fig. 3. Generalized Minkowski Distance Equation [5]

The L2 norm function in Figure 2 is used in the current distance function used in SWAY. The L2 norm is the same as this equation in Figure 3, the generalized Minkowski Distance Equation, but with a p = 2 which is Euclidean Distance. We replace p and set it equal to 1 in order to use Manhattan Distance.

V. Algorithms

In this section we will describe the original main algorithms in the project and the new functions we have added.

SWAY



```python
# Recursively prune the worst half the data. Return the survivors and some sample of the rest.
def sway(self):

    def worker(rows, worse, evals = 0, above = None):
        if len(rows) <= pow(len(self.rows), Common.cfg['the']['min']):
            return {'best': rows, 'rest': many(worse, Common.cfg['the']['rest'] * len(rows)), 'evals': evals}
        half_res = self.half(None, above, rows)
        l = half_res['left']
        r = half_res['right']
        A = half_res['A']
        B = half_res['B']
        if self.better(half_res['B'], half_res['A']):
            l = half_res['right']
            r = half_res['left']
            A = half_res['B']
            B = half_res['A']
        for row in r:
            worse.append(row)
        return worker(l, worse, half_res['evals'] + evals, A)
```

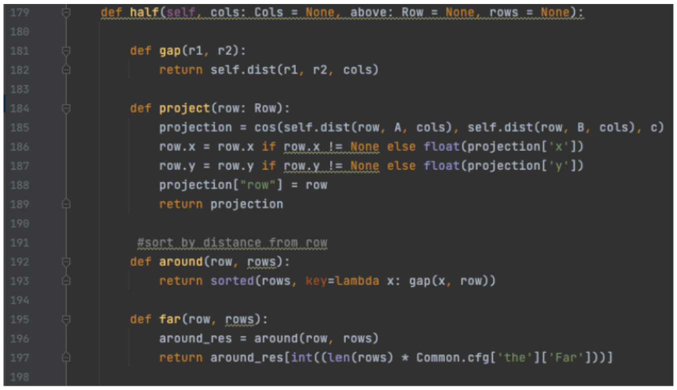
Fig. 4. Current Implementation of SWAY

Short for “Sampling Way”. Used as a baseline algorithm for search based software engineering. SWAY is a semi-supervised machine learning algorithm as it uses a combination of active learning and random sampling to generate training data for model training. It starts with a set of labeled data points and a large set of unlabelled data points. It then uses active learning to find the most important data points in an unlabeled set based on given criteria. Those important points are then labeled and added to the training dataset. It then uses random sampling to add additional data points to reduce bias by active learning. Thus, SWAY is semi supervised as given N examples, we only have a limited budget $B_0 \ll$ N of times, we can access the Y values of any one example.

XPLN

Extreme Projection Learning Network (XPLN) is a type of dimensionality reduction algorithm that uses a nonlinear transformation to project high-dimensional data into a low-dimensional space. XPLN can be used in speech and image recognition, natural language processing, and data visualization.

HALF

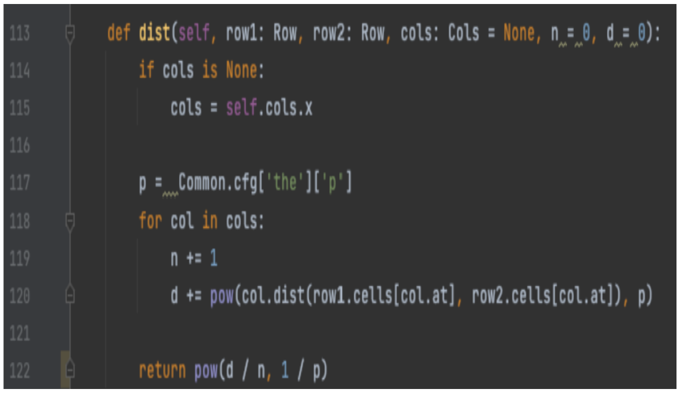


```
def half(self, cols: Cols = None, above: Row = None, rows = None):

    def gap(r1, r2):
        return self.dist(r1, r2, cols)

    def project(row: Row):
        projection = cos(self.dist(row, A, cols), self.dist(row, B, cols), c)
        row.x = row.x if row.x != None else float(projection['x'])
        row.y = row.y if row.y != None else float(projection['y'])
        projection["row"] = row
        return projection

     #sort by distance from row
    def around(row, rows):
        return sorted(rows, key=lambda x: gap(x, row))

    def far(row, rows):
        around_res = around(row, rows)
        return around_res[int((len(rows) * Common.cfg['the']['Far']))]
```

Fig. 5. Current Implementation of HALF

Half returns a tuple of two lists (left and right), two values (A and B), mid row, mid and a number c. It calculates a sample of rows using the many function, and sets A to the first element of the sample or the value of above if provided. B is set to the row from self.around(A, some) that has the largest distance from A. The value of c is the distance between A and B. Sorts the rows by the result of the project function applied to each row and maps it to a list of dictionaries containing the row and its corresponding distance from A and B. The function then iterates over this sorted list, adds each row to either left or right based on its index, and sets mid to the row in the middle. This function is used in SWAY and calls the DIST method to get the distance between two rows.

DIST

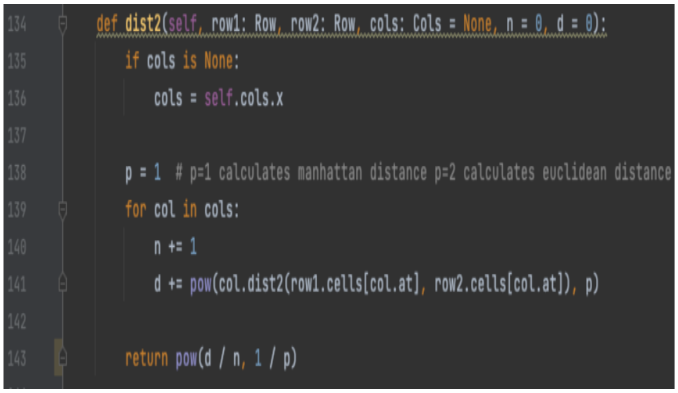


```
113    def dist(self, row1: Row, row2: Row, cols: Cols = None, n = 0, d = 0):
114        if cols is None:
115            cols = self.cols.x
116
117        p = Common.cfg['the']['p']
118        for col in cols:
119            n += 1
120            d += pow(col.dist(row1.cells[col.at], row2.cells[col.at]), p)
121
122        return pow(d / n, 1 / p)
```

Fig. 6. Current Implementation of Euclidean Distance

This is the current implementation of the dist function. This implementation uses the generic Minkowski Distance Formula but with a p of 2 which is Euclidean Distance. This p value is stored in our config file and is accessed on line 117.

DIST-2



```
134    def dist2(self, row1: Row, row2: Row, cols: Cols = None, n = 0, d = 0):
135        if cols is None:
136            cols = self.cols.x
137
138        p = 1  # p=1 calculates manhattan distance p=2 calculates euclidean distance
139        for col in cols:
140            n += 1
141            d += pow(col.dist2(row1.cells[col.at], row2.cells[col.at]), p)
142
143        return pow(d / n, 1 / p)
```

Fig. 7. Current Implementation of Manhattan Distance

Our new implementation of the dist function, dist2, uses the Minkowski Distance Formula but with a p of 1 which is Manhattan Distance. This can be seen on line 138 in Figure 7 above.

SWAY-2 and XPLN-2

SWAY-2 is our implementation of SWAY that calls the dist2 function instead of dist. The sequence of actions is that when SWAY-2 is called instead of calling the half function we call half2 which is exactly the same as half, it just within the method calls dist2 instead of dist. XPLN-2 is the same as XPLN but the input is from SWAY2 instead since our change is a parameter change and it does not affect the structure of the original SWAY.

VI. Data

"Car Design" Data

These datasets originated from the StatLib library which is maintained by Carnegie Mellon University. Each dataset describes cars in terms of attributes such as their weights, MPG values, horsepower, etc.

"Software Project Estimation" Data

Software effort estimation is concerned with predicting the most realistic amount of labor required to develop or maintain software. Since software development in the real world is fraught with uncertainty, tools that can aid in making accurate predictions are greatly valued.

The datasets in this group describe software projects in terms of the COCOMO ontology: 23 variables that define a software project as well as aspects of its developers, platform, and product features. A description of each of these variables are listed below.

In Scale factors: Flex, Pmat, Prec, Resl, Team are all x variables. In Effort multipliers: acap, cplx, data, docu, ltex, pcap, pcon, pvol, rely, ruse, sced, site, stor, time, tool are x variables and aexp, plex are y variables.

“Issue Close Time” Data

These data sets describe the factors surrounding code issue descriptions. In Agile development, making accurate estimations regarding the time a flaw in some code base will take to be resolved is very valuable. Using a data set such as this in order to generate a model that can accurately make such predictions would be very valuable.

In the ”easy” version of this data set (`healthCloseIsses12mths0011-easy.csv`), prec, flex, resl, team, pmat, rely, data, cplx, ruse, docu, time, stor, pvol, acap, pcap, pcon, apex, plex, ltex, tool, site, and sced are input x variables. The goal variables, Kloc+, Effort-, Defects-, Months- are the y varaibles.

In the ”hard” version of this data set (`healthCloseIsses12mths0001-hard.csv`), N-estimators, criterion, Min-sample-leaves, Min-impurity-decrease, and Max-depth are input x variables. The goal variables, MRE-, ACC+, PRED40+ are the y variables.

“Agile Project Management” Data

The POM3 model is a tool for exploring the management challenges of agile development. It aims to balance developer idle times, completion rates, and overall cost (these are its goal variables).

The data sets in this group describe a large number of projects that have been described using the variables below. These variables are either inputs to the POM3 model or the goals that POM3 aims to balance.

In Inputs to POM3, Culture, Criticality, Criticality Modifier, Initial Known, Inter-Dependency, Dynamism, Size, Plan, Team Size are x variables.

In Goals, Cost, Score, Completion, Idle are all y variables.

“Computational Physics” Data [7]

In the modern world, most software systems are configurable, meaning that they can be tuned to achieve a wide range of both functional and non-functional properties (1). Often, there are certain goal outcomes users of these software systems have from their configurations., but understanding the configuration possibilities of a software system in its entirety is difficult and time consuming. The FLASH method is used to make this task easier: the model reflects on past configurations in order to generate a new suggestion. The SSM and SSN data sets cover the configuration space of Trimesh, a library used to manipulate triangle meshes.

Performance measures

Goal variables are variables that are to be either maximized or minimized by a model. In our case, they are the variables we are interested in from the perspective of creating an explanation system. As such, these are the variables that are highlighted in the following discussion.

In order to evaluate the performance of our algorithms, we took the average result of their outputs across 20 runs of execution. These results were automatically reported by our program in the form of a table.

Along with the output of these algorithms, we also kept track of how many evaluations of example points were required. As previously mentioned, evaluating on an example data point can be very expensive in practice. Therefore, limiting the amount of evaluations required to train a model is of the upmost importance.

Output				
	Lbs-	Acc+	Mpg+	Average Evaluation Count
all	4561.4	18.1	10.1	0.0
sway1	3764.2	14.7	20.0	28.2
xpln1	4528.4	15.8	10.1	28.2
sway2	3198.4	15.4	23.2	6.0
xpln2	4297.1	14.6	10.1	6.0
top	4584.1	14.9	10.0	1870.6

Fig. 8. Average output and evaluation count from each algorithm.

In order to determine the ”average” value of a variable’s output per algorithm, we considered all its values over 20 runs and used its central tendency value. For numeric values, this was its mean value. For symbolic values, this was its mode (most common value).

Average evaluations were determined by the amount of times an algorithm had to ”evaluate” an example point. As you can see, the values for `xpln1` and `xpln2` are the same as their corresponding `sway` implementations. This is due to the fact that the `xpln` requires no further evaluations of example data points after `sway` has been evaluated.

Summarization methods

While it would be easy to only look at the number of evaluations required by each algorithm in order to determine which was the ”best”, this would be naive. We ultimately needed to determine if the outputs of these algorithms were statistically insignificantly distinguishable by more than a small effect in order to declare one better than the other. The importance of the performance of an algorithm is greatly diminished if its performance is not also significantly better.

Comparisons			
	Lbs-	Acc+	Mpg+
all to all	=	=	=
all to sway1	≠	≠	≠
all to sway2	≠	≠	≠
sway1 to sway2	≠	≠	≠
sway1 to xpln1	≠	≠	≠
sway2 to xpln2	≠	≠	≠
sway1 to top	≠	≠	≠

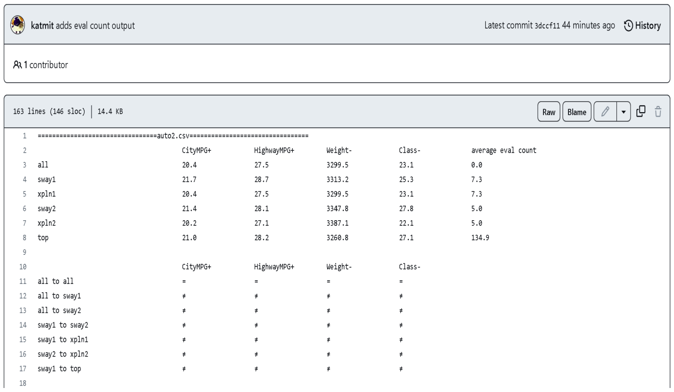
Fig. 9. Algorithm comparison output.

In order to determine if the outputs from each algorithm were statistically insignificantly distinguishable by more than a small effect, we ran each algorithm 20 times, each time with a different random seed value. For each set of 20 outputs, we compared values as described in the above table. In each comparison, we used the conjunction of both an effect-size test and a significance test.

An effect-size test determines if two distributions are different by more than some trivial amount. In our program, we implemented the Cliff’s Delta algorithm as our effect-size test. For our significance test, we used the Bootstrap algorithm.

Each comparison was a conjunction of both our effect-size test and significance test. If any of these comparisons returned false, then the overall result of the comparison was also deemed to be false: meaning the two distributions were not statistically insignificantly distinguishable by more than a small effect.

Perform a ”prudence” study


katmit adds eval count output
Latest commit 3dccf11 44 minutes ago
History
1 contributor
163 lines (146 sloc) | 14.4 KB
Raw
Blame

```
================================auto2.csv================================
                                CityMPG+        HighwayMPG+     Weight-         Class-          average eval count
all                             20.4            27.5            3299.5          23.1            0.0
sway1                           21.7            28.7            3313.2          25.3            7.3
xpln1                           20.4            27.5            3299.5          23.1            7.3
sway2                           21.4            28.1            3347.8          27.8            5.0
xpln2                           20.2            27.1            3387.1          22.1            5.0
top                             21.0            28.2            3260.8          27.1            134.9

                                CityMPG+        HighwayMPG+     Weight-         Class-
all to all                      =               =               =               =
all to sway1                    ≠               ≠               ≠               ≠
all to sway2                    ≠               ≠               ≠               ≠
sway1 to sway2                  ≠               ≠               ≠               ≠
sway1 to xpln1                  ≠               ≠               ≠               ≠
sway2 to xpln2                  ≠               ≠               ≠               ≠
sway1 to top                    ≠               ≠               ≠               ≠

```

Fig. 10. CONJUNCTION of a effect size test and a significance test

The table shows conjucting different tests for comparing different treatments based on given variables. Tests are performed by both effect size and significance tests. The treatment of all which is comparing all results to all should return equal. As, it is comparing the results by itself. $\neq$ shows that the treatment is significantly different from the other treatment. We can see the all to all are not significantly different. While, all to sway1, all to sway2, sway1 to sway2, sway1 to xpln1, sway2 to xpln2, sway1 to top are all significantly different.

VII. Discussion

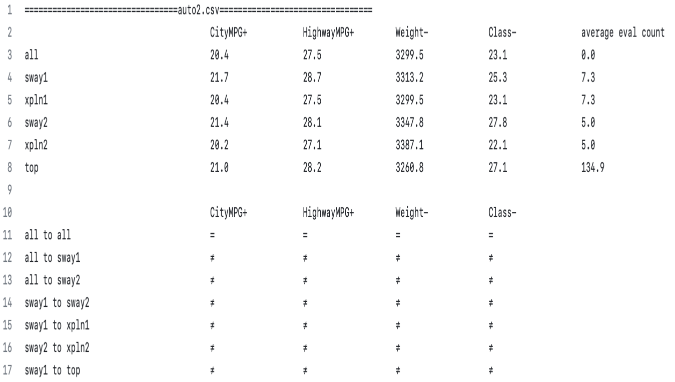
There are various threats to validity in SWAY algorithm like bias in sampling strategy, overfitting to training data, sensitivity to initialization, sensitivity to dataset characteristics. SWAY algorithm relies on sampling strategy to select next sets of values to examine. If sampling strategy is biased, it can lead to inaccurate or poor solution (bias in sampling strategy). It uses a training dataset to learn sampling strategy. If the model is overfitting, then it would generaize to a new dataset thus poor adaptability, robustness and performance (overfitting to training data). SWAY algorithm requires initialization parameters like sampling threshold. If we incorrectly choose these values it will lead to poor performance (sensitivity to initialization). SWAY algorithm are also sensitive to dimenshionality, sparsity and distribution of data. Thus, it would perform good with certain datasets and poorly with others (sensitivity to dataset characteristics).

For future work, there are lot of possibilities. We did not have time to compare our SWAY algorithm using various stats like Scott-Knott, normality or a t-distribution, one-way ANOVA, etc. using sufficient sample size.

VIII. Results

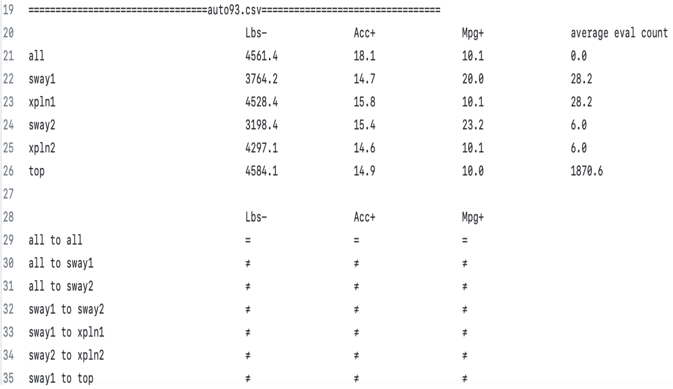
(RQ1) To what extent is SWAY2 faster than SWAY1? We compared the efficiency of each algorithm using their average evaluation counts. By this metric, it is quite faster. Additionally in all algorithm comparisons for each dataset SWAY2 was found to be statistically distinguishable from SWAY1. This was found to be true for the XPLN and XPLN2 algorithms as well.

(RQ2) To what extent does Manhattan distance improve the performance in SWAY and other algorithm when replaced with earlier Euclidean distance. The Manhattan distance metric improved the performance in SWAY and other algorithm by significant fold. A lower eval count is considered better for an algorithm because it means that the algorithm is able to achieve good performance using fewer evaluations of the objective function. As, we can see in the figures below, the eval count of SWAY2 and XPLN2 is more than 4 times smaller than that of SWAY1 and XPLN1 respectively.



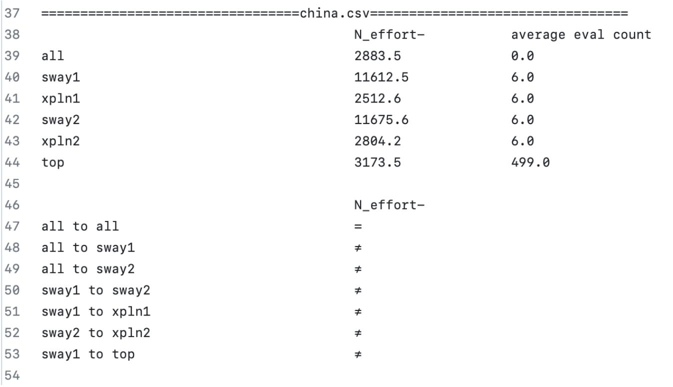
```
================================auto2.csv================================
                                CityMPG+        HighwayMPG+     Weight-         Class-          average eval count
all                             20.4            27.5            3299.5          23.1            0.0
sway1                           21.7            28.7            3313.2          25.3            7.3
xpln1                           20.4            27.5            3299.5          23.1            7.3
sway2                           21.4            28.1            3347.8          27.8            5.0
xpln2                           20.2            27.1            3387.1          22.1            5.0
top                             21.0            28.2            3260.8          27.1            134.9

                                CityMPG+        HighwayMPG+     Weight-         Class-
all to all                      =               =               =               =
all to sway1                    ≠               ≠               ≠               ≠
all to sway2                    ≠               ≠               ≠               ≠
sway1 to sway2                  ≠               ≠               ≠               ≠
sway1 to xpln1                  ≠               ≠               ≠               ≠
sway2 to xpln2                  ≠               ≠               ≠               ≠
sway1 to top                    ≠               ≠               ≠               ≠
```

Fig. 11. auto2.csv Results



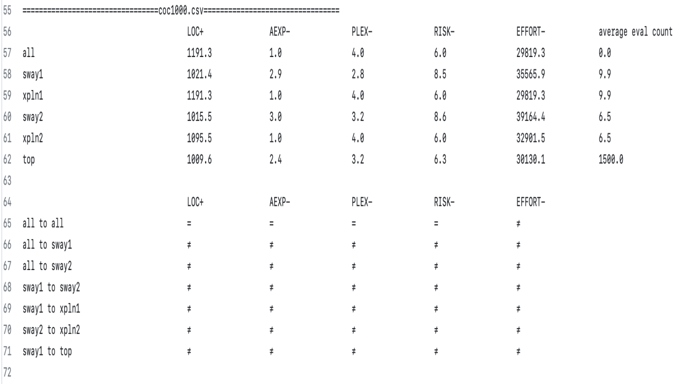
```
================================auto93.csv================================
                                Lbs-            Acc+            Mpg+            average eval count
all                             4561.4          18.1            10.1            0.0
sway1                           3764.2          14.7            20.0            28.2
xpln1                           4528.4          15.8            10.1            28.2
sway2                           3198.4          15.4            23.2            6.0
xpln2                           4297.1          14.6            10.1            6.0
top                             4584.1          14.9            10.0            1870.6

                                Lbs-            Acc+            Mpg+
all to all                      =               =               =
all to sway1                    ≠               ≠               ≠
all to sway2                    ≠               ≠               ≠
sway1 to sway2                  ≠               ≠               ≠
sway1 to xpln1                  ≠               ≠               ≠
sway2 to xpln2                  ≠               ≠               ≠
sway1 to top                    ≠               ≠               ≠
```

Fig. 12. auto93.csv Results



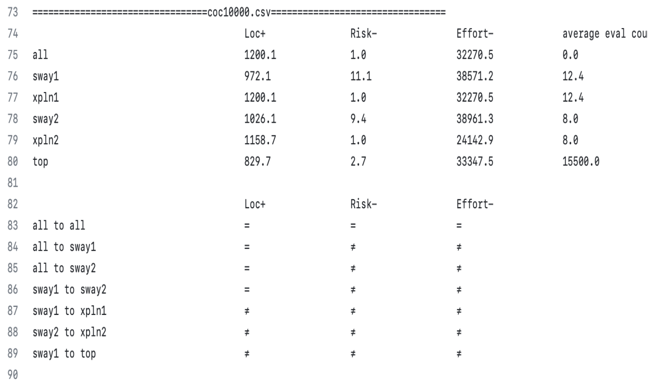
```
================================china.csv================================
                                N_effort-           average eval count
all                             2883.5              0.0
sway1                           11612.5             6.0
xpln1                           2512.6              6.0
sway2                           11675.6             6.0
xpln2                           2804.2              6.0
top                             3173.5              499.0

                                N_effort-
all to all                      =
all to sway1                    ≠
all to sway2                    ≠
sway1 to sway2                  ≠
sway1 to xpln1                  ≠
sway2 to xpln2                  ≠
sway1 to top                    ≠

```

Fig. 13. china.csv Results



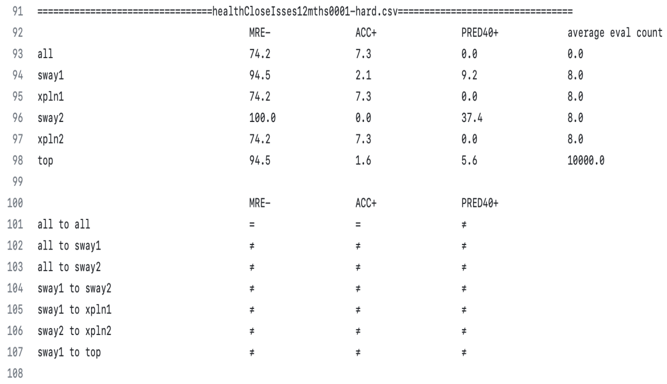
```
================================coc1000.csv================================
                                LOC+        AEXP-       PLEX-       RISK-       EFFORT-     average eval count
all                             1191.3      1.0         4.0         6.0         29819.3     0.0
sway1                           1021.4      2.9         2.8         8.5         35565.9     9.9
xpln1                           1191.3      1.0         4.0         6.0         29819.3     9.9
sway2                           1015.5      3.0         3.2         8.6         39164.4     6.5
xpln2                           1095.5      1.0         4.0         6.0         32901.5     6.5
top                             1009.6      2.4         3.2         6.3         30130.1     1500.0

                                LOC+        AEXP-       PLEX-       RISK-       EFFORT-
all to all                      =           =           =           =           ≠
all to sway1                    ≠           ≠           ≠           ≠           ≠
all to sway2                    ≠           ≠           ≠           ≠           ≠
sway1 to sway2                  ≠           ≠           ≠           ≠           ≠
sway1 to xpln1                  ≠           ≠           ≠           ≠           ≠
sway2 to xpln2                  ≠           ≠           ≠           ≠           ≠
sway1 to top                    ≠           ≠           ≠           ≠           ≠
```

Fig. 14. coc1000.csv Results



```
================================coc10000.csv================================
                                Loc+        Risk-       Effort-     average eval count
all                             1200.1      1.0         32270.5     0.0
sway1                           972.1       11.1        38571.2     12.4
xpln1                           1200.1      1.0         32270.5     12.4
sway2                           1026.1      9.4         38961.3     8.0
xpln2                           1158.7      1.0         24142.9     8.0
top                             829.7       2.7         33347.5     15500.0

                                Loc+        Risk-       Effort-
all to all                      =           =           =
all to sway1                    =           ≠           ≠
all to sway2                    =           ≠           ≠
sway1 to sway2                  =           ≠           ≠
sway1 to xpln1                  ≠           ≠           ≠
sway2 to xpln2                  ≠           ≠           ≠
sway1 to top                    ≠           ≠           ≠
```

Fig. 15. coc10000.csv Results



```
================================healthCloseIsses12mths0001-hard.csv================================
                                MRE-        ACC+        PRED40+     average eval count
all                             74.2        7.3         0.0         0.0
sway1                           94.5        2.1         9.2         8.0
xpln1                           74.2        7.3         0.0         8.0
sway2                           100.0       0.0         37.4        8.0
xpln2                           74.2        7.3         0.0         8.0
top                             94.5        1.6         5.6         10000.0

                                MRE-        ACC+        PRED40+
all to all                      =           =           ≠
all to sway1                    ≠           ≠           ≠
all to sway2                    ≠           ≠           ≠
sway1 to sway2                  ≠           ≠           ≠
sway1 to xpln1                  ≠           ≠           ≠
sway2 to xpln2                  ≠           ≠           ≠
sway1 to top                    ≠           ≠           ≠
```

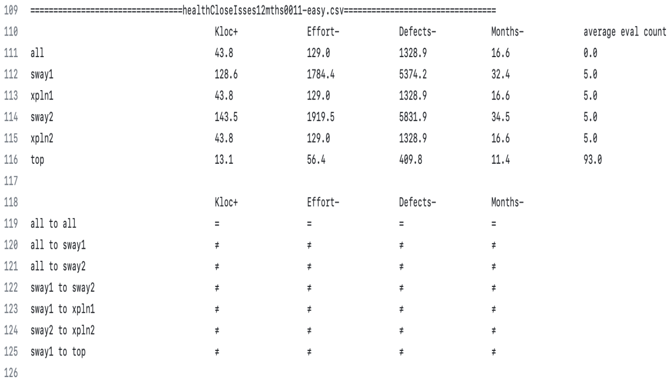
Fig. 16. healthCloseIsses12mths0001-hard.csv Results

IX. Conclusion

In conclusion using a Manhattan Distance metric in SWAY significantly increases the performance on the given datasets. Additionally, sway1 to sway2, sway1 to xpln1, sway2 to xpln2, sway1 to top are all significantly different. Further work could be done using Scott-Knot or one-way ANOVA to more distinctly quantify the magnitude in which Manhattan and Euclidean distance metrics effect the performance of SWAY. Using Chebyshev distance as the distance metric in SWAY is also an avenue that could also be explored.

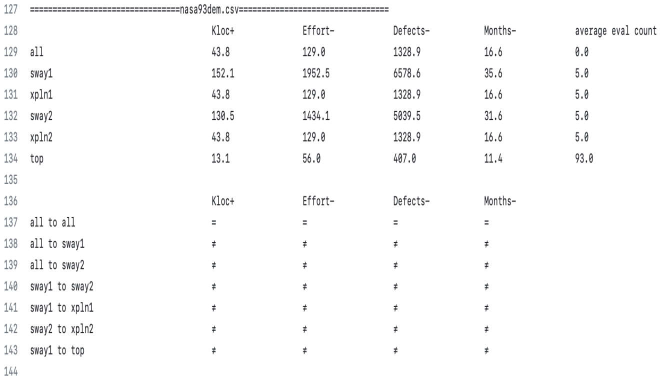
Acknowledgment

We would like to thank our Professor Tim Menzies, Andre Lustosa (TA), Sherry (Xueqi) Yang (TA), Rahul Yedida (TA).



```
================================healthCloseIsses12mths0011-easy.csv================================
                                Kloc+       Effort-     Defects-    Months-     average eval count
all                             43.8        129.0       1328.9      16.6        0.0
sway1                           128.6       1784.4      5374.2      32.4        5.0
xpln1                           43.8        129.0       1328.9      16.6        5.0
sway2                           143.5       1919.5      5831.9      34.5        5.0
xpln2                           43.8        129.0       1328.9      16.6        5.0
top                             13.1        56.4        409.8       11.4        93.0

                                Kloc+       Effort-     Defects-    Months-
all to all                      =           =           =           =
all to sway1                    ≠           ≠           ≠           ≠
all to sway2                    ≠           ≠           ≠           ≠
sway1 to sway2                  ≠           ≠           ≠           ≠
sway1 to xpln1                  ≠           ≠           ≠           ≠
sway2 to xpln2                  ≠           ≠           ≠           ≠
sway1 to top                    ≠           ≠           ≠           ≠
```

Fig. 17. healthCloseIsses12mths0011-easy.csv Results



```
================================nasa93dem.csv================================
                                Kloc+       Effort-     Defects-    Months-     average eval count
all                             43.8        129.0       1328.9      16.6        0.0
sway1                           152.1       1952.5      6578.6      35.6        5.0
xpln1                           43.8        129.0       1328.9      16.6        5.0
sway2                           130.5       1434.1      5039.5      31.6        5.0
xpln2                           43.8        129.0       1328.9      16.6        5.0
top                             13.1        56.0        407.0       11.4        93.0

                                Kloc+       Effort-     Defects-    Months-
all to all                      =           =           =           =
all to sway1                    ≠           ≠           ≠           ≠
all to sway2                    ≠           ≠           ≠           ≠
sway1 to sway2                  ≠           ≠           ≠           ≠
sway1 to xpln1                  ≠           ≠           ≠           ≠
sway2 to xpln2                  ≠           ≠           ≠           ≠
sway1 to top                    ≠           ≠           ≠           ≠
```

Fig. 18. nasa93dem.csv Results

References

[1] Nkurikiyinka, Grace. *"Understanding Search Algorithms in AI."* Engineering Education (EngEd) Program — Section, 16 Dec. 2021, www.section.io/engineering-education/understanding-search-algorithms-in-ai/.

[2] Caballé-Cervigón, Nuria, et al. "Machine Learning Applied to Diagnosis of Human Diseases: A Systematic Review." Applied Sciences, vol. 10, no. 15, 26 July 2020, p. 5135, https://doi.org/10.3390/app10155135. Accessed 19 Nov. 2020.

[3] Suwanda, R, et al. "Analysis of Euclidean Distance and Manhattan Distance in the K-Means Algorithm for Variations Number of Centroid K." IOPScience, 2020, iopscience.iop.org/article/10.1088/1742-6596/1566/1/012058.

[4] Tim Menzies, Tim. "TESTED." GitHub, 13 Feb. 2023, github.com/timm/tested/blob/main/docs/onCluster.md. Accessed 20 Apr. 2023.

[5] Kamble, Vaishali H., and Manisha P. Dale. "Chapter 1 - Machine Learning Approach for Longitudinal Face Recognition of Children." ScienceDirect, Academic Press, 1 Jan. 2022, www.sciencedirect.com/science/article/abs/pii/B9780323852098000110. Accessed 20 Apr. 2023.

[6] Samuelson, Bella, et al. "CSC591-HW-LUA." GitHub, 11 Apr. 2023, github.com/katmit/ASE_Project. Accessed 20 Apr. 2023.

[7] V. Nair, Z. Yu, T. Menzies, N. Siegmund and S. Apel, "Finding Faster Configurations Using FLASH," in IEEE Transactions on Software Engineering, vol. 46, no. 7, pp. 794-811, 1 July 2020, doi: 10.1109/TSE.2018.2870895.