Computational Intelligence Report

Luca Ianniello s327313

January 20, 2025

Abstract

This report covers all the laboratories and the project done during the Computation Intelligence course. In details, the laboratories describe the implementation and solution of three problem: Set Cover problem, the Traveling Salesman Problem and the n-puzzle problem. The project is about the solution of the Symbolic regression problem, considering eight different give problems. The report describes the methodology used to solve the problems, the results obtained and the discussion about the results. Each section will be assigned to a particular problem. The report will end with a conclusion about the work done and the results obtained.

Contents

1 Problem 1: Set Cover Problem

1.1 Description of the problem

The first laboratory done was about the Set Cover Problem. The Set Cover Problem is a classical problem in computer science and is considered a NP-hard problem. The problem is defined as follows: given a set of elements $U = \{1, 2, ..., n\}$ and a set of subsets $S = \{S_1, S_2, ..., S_m\}$, find the smallest subset of S that covers all the elements of U. The analysis of the problem was done considering 6 different instances, where the number of elements n, the number of subsets m and the density are different. The instances are the following:

- Instance 1: $n = 100$, $m = 10$, density $= 0.2$
- Instance 2: $n = 1000$, $m = 100$, density $= 0.2$
- Instance 3: $n = 10000$, $m = 1000$, density $= 0.2$
- Instance 4: $n = 100000$, $m = 10000$, density $= 0.1$
- Instance 5: $n = 100000$, $m = 10000$, density $= 0.2$
- Instance 6: $n = 100000$, $m = 10000$, density $= 0.3$

1.2 Implementation

The implemented algorithm is a Hill climbing algorithm. The algorithm begins by generating an initial random solution, where each set is either included or excluded based on a random threshold. For this implementation, the threshold is set to 1, meaning all sets are initially included. The fitness of a solution is evaluated using two main criteria. The first is validity, which checks whether the solution covers all elements in the universe. The second is cost, which calculates the total cost of the sets included in the solution. Lower costs indicate better solutions, and the algorithm aims to minimize this value.

To explore neighboring solutions, a mutation strategy is employed. A proportion of sets in the solution is randomly flipped, either included or excluded, based on a predefined mutation strength parameter. This mutation strength is adjusted dynamically during the optimization process, depending on the coverage achieved by the new solution compared to the previous one. Adjusting the mutation strength ensures a balance between exploration and exploitation in the search space.

The algorithm operates iteratively over a specified number of steps, set to $1,000$ in this case. In each iteration, the coverage of the current solution is evaluated. A new solution is then generated by mutating the current one, and its fitness is assessed. If the new solution demonstrates better performance by achieving higher fitness, it replaces the current solution. The mutation strength is then updated based on the change in coverage, allowing the algorithm to adapt its exploration efforts dynamically.

Throughout the optimization process, the algorithm records the costs of solutions across iterations. These logs are used to generate a plot that visualizes the progress of the optimization, highlighting the history of the best costs encountered. Upon completion, the algorithm outputs the fitness of the best solution found and the number of steps taken to reach this result. This information provides insights into the algorithm's effectiveness and the quality of the final solution.

1.3 Results

The results of the Set Cover Problem are shown in the following table:

Instance	Cost of Solution	Number of Cost Calls	Number of Steps until the best solution
Instance 1	274.4903145957521	1003	0
Instance 2	6447.837166630783	1079	151
Instance 3	1139467.6919141859	1535	986
Instance 4	111223621.17491701	2001	990
Instance 5	237250569.35082552	1977	989
Instance 6	366376527.5306066	1965	995

Table 1: Results of the Set Cover Problem for different instances

As we can see from these results, the algorithm was able to find the best solution for all instances. The cost of the solution increases with the number of elements and subsets, as expected. The number of cost calls and steps taken to reach the best solution also increase with the complexity of the instances. The density has a significant impact on the cost of the solution, with higher densities leading to higher costs. This relationship is evident in the results for instances 4, 5, and 6, where the density is 0.1, 0.2, and 0.3, respectively. The algorithm performs well in finding optimal solutions for all instances, demonstrating its effectiveness in solving the Set Cover Problem.

1.4 Obtained reviews

The results and the code done was reviews by two collegues of the course. In the following section I will report the obtained revies:

- Review 1 done by AlesCarl: The chosen algorithm, Hill Climbing with multiple mutations, is implemented correctly. However, while it performs well, it may be limited in some situations, especially in the presence of local optima. To address this, it would be interesting to explore the use of other algorithms, such as Simulated Annealing, which could be more effective in contexts with large search spaces and situations where greater global exploration is needed. Additionally, for larger instances (4, 5, 6), introducing an early stopping criterion could enhance efficiency by preventing unnecessary iterations when no significant progress is made. That said, you did a great job!

- Review 2 done by SamueleVanini: The proposed algorithm (Hill Climbing with multiple mutations) is well implemented, and the addition of the README file is a great touch. I do have a couple of suggestions that might enhance both the structure and performance of the solution:

  Reduce Redundancy: the algorithm is duplicated across different test instances, which could make debugging more time-consuming for both you and future readers. Consider wrapping the main logic into one or more functions, which you can then call from different blocks. This will improve maintainability and makes the code easier to modify of extend.

  Experiment with Fitness Functions: while your use of Hill Climbing with multiple mutations is solid, you could potentially improve the solution by experimenting with different types of fitness evaluations. For example, instead of simply classifying a solution as valid or invalid, it could be beneficial to assign a range of values to invalid solutions. This would help guide the algorithm more effectively, reducing the possibility of plateaus (mesas).

2 Problem 2: Traveling Salesman Problem

2.1 Description of the problem

The second laboratory done was about the Traveling Salesman Problem. The TSP problem talks about a salesman that has to visit a set of cities and return to the starting city. The goal is to find the shortest path that visits all cities exactly once and returns to the starting city. The problem is defined as follows: given a set of cities and the distances between them, find the shortest path that visits all cities exactly once and returns to the starting city. The analysis of the problem was done considering 5 different countries and the distances between the cities. The countries are the following:

- Vanuatu
- Italy
- China
- Russia
- United States of America

2.2 Implementation

To solve this problem, two different kind of solution were done.

The first solution is a simple greedy approach, where, starting from the first city of the set, the algorithm selects the nearest city and moves to it. This process is repeated until all cities are visited, and the salesman returns to the starting city. The algorithm calculates the total distance traveled and outputs the path taken by the salesman. This approach provides a quick and straightforward solution to the TSP problem, although it may not always yield the optimal path.

The second solution is an evolutionary algorithm. The algorithm begins with an initialization phase, where a population of candidate solutions is created. The starting population can be the best path found the greedy approach or a random path. The evolutionary process is driven by two key operations: mutation and crossover.

The mutation strategy is implemented using an inverted mutation algorithm, which modifies the sequence of cities in a tour to explore alternative paths. The mutation rate is adaptive, decreasing over time to promote convergence. When a mutation occurs, a local search refinement is applied to the mutated paths, further optimizing the solution. If the fitness of the mutated population improves, the best population and its associated fitness are updated. Conversely, if no improvement is observed, the mutation rate is adjusted to enhance exploration, and the number of swaps in the mutation increases to diversify the search.

The crossover strategy utilizes tournament selection to choose parent tours from the current population. These parents are combined using the `inv_crossover` function, which produces a child tour that inherits characteristics from both parents. The child may undergo mutation based on a probabilistic threshold. To ensure progress, the child is only retained in the next generation if it exhibits better fitness than its parents; otherwise, a parent is randomly retained.

The evolutionary algorithm progresses over a specified number of generations. At each generation, the population evolves through the combined effects of crossover, mutation, and selection. Fitness evaluations are performed using the TSP cost function, with periodic reporting of the best fitness achieved. By the end of the process, the algorithm returns the best tour found, representing the shortest path identified.

2.3 Results

The results of the Traveling Salesman Problem are shown in the following tables:

Table 2: Vanuatu

Method	Cost	Steps
Greedy + Mutation	1475.53	10093
Greedy + Crossover	1345.54	196369 (500 gen)
Random + Mutation	1345.54	10303
Random + Crossover	1345.54	196369 (500 gen)

Table 3: Italy

Method	Cost	Steps
Greedy + Mutation	4436.03	17571
Greedy + Crossover	4306.79	1074245 (500 gen)
Random + Mutation	4976.60	19463
Random + Crossover	4291.87	1074245

Table 4: Russia

Method	Cost	Steps
Greedy + Mutation	35805.38	442963
Greedy + Crossover	34145.42	2333203 (1000 gen)
Random + Mutation	36656.11	145303
Random + Crossover	34665.52	2333203 (1000 gen)

Table 5: US

Method	Cost	Steps
Greedy + Mutation	40540.23	742567
Greedy + Crossover	40275.63	2333203 (1000 gen)
Random + Mutation	40759.67	3568171
Random + Crossover	41067.56	5832853 (2500 gen)

Table 6: China

Method	Cost	Steps
Greedy + Mutation	54786.93	4721071
Greedy + Crossover	55809.75	6999403 (3000 gen)
Random + Mutation	965247.20	10003
Random + Crossover	61403.21	13998703 (6000 gen)

The results presented in the tables highlight the performance differences between various methods for solving the Traveling Salesman Problem (TSP). In general, methods incorporating

crossover consistently achieved lower costs compared to mutation-based approaches, albeit at the expense of significantly higher computational steps. For smaller problem instances, such as Vanuatu and Italy, the difference in cost between methods was relatively small, with crossover marginally outperforming mutation. However, for larger problem instances like Russia, the US, and China, the advantages of crossover became more apparent, achieving significantly better solutions. Notably, the random initialization combined with mutation performed poorly for the China dataset, resulting in an exceptionally high cost compared to other methods. Furthermore, while the greedy initialization provided a solid baseline for mutation-based methods, crossover consistently improved upon it, especially when allowed sufficient generations. These results underscore the trade-offs between solution quality and computational cost, with crossover being more effective for larger datasets at the cost of increased computational effort.

2.4 Obtained Reviews

The results and the code done was reviews by two collegues of the course. In the following section I will report the obtained revies:

- Review 1 done by AlesCarl: The implementation of the evolutionary algorithm for the TSP is well-structured and effectively solves the problem. The use of mutation and crossover operations provides a comprehensive exploration of the solution space, leading to high-quality results. However, the computational complexity of the algorithm may be a limiting factor for larger datasets, such as China. To address this, consider optimizing the mutation and crossover strategies to reduce the number of steps required to converge to a solution. Additionally, exploring parallelization techniques could enhance the algorithm's performance by distributing the computational load across multiple cores or processors. Overall, you've done an excellent job in tackling the TSP with an evolutionary approach.
- Review 2 done by SamueleVanini: The implementation of the TSP evolutionary algorithm is well-executed and demonstrates a thorough understanding of the problem. The use of mutation and crossover operations effectively explores the solution space, leading to high-quality results. However, the computational complexity of the algorithm may be a limiting factor for larger datasets, such as China. To address this, consider optimizing the mutation and crossover strategies to reduce the number of steps required to converge to a solution. Additionally, exploring parallelization techniques could enhance the algorithm's performance by distributing the computational load across multiple cores or processors. Overall, you've done an excellent job in tackling the TSP with an evolutionary approach.

3 Problem 3: n-puzzle problem

3.1 Description of the problem

The third laboratory done was about the n-puzzle problem. The n-puzzle problem is a classical problem in artificial intelligence and consists of a board with $n^2 - 1$ tiles, numbered from 1 to $n^2 - 1$, and one empty space. The goal is to rearrange the tiles from a given initial state to a goal state by sliding them into the empty space. The problem is defined as follows: given an initial state of the n-puzzle board and a goal state, find the sequence of moves that transforms the initial state into the goal state.

3.2 Implementation

To solve this problem, two different kind of solution were done. In both solution, it was used the Manhattan distance as heuristic. The Manhattan distance is the sum of the absolute differences

in the x and y coordinates of two points on a grid. It is a common heuristic for the n-puzzle problem.

The first one is the A* algorithm. The A* algorithm is a pathfinding algorithm that efficiently solves puzzles by combining the path cost (number of moves) with a heuristic (Manhattan distance). It begins by adding the initial puzzle state to an open list with a cost based on its Manhattan distance. The algorithm iteratively selects the state with the lowest total estimated cost, calculated as the sum of moves made and the Manhattan distance, and generates possible neighbors by sliding tiles into the empty space. The Manhattan distance heuristic, calculated as the sum of the horizontal and vertical distances of each tile from its goal position, ensures efficient prioritization by reducing the number of expanded nodes. As an admissible heuristic, it guarantees finding the shortest solution path. The algorithm continues until the goal state is reached, returning the solution path, the number of moves, and the total nodes evaluated. With features like support for custom puzzle dimensions, quality and cost tracking, and random initial state generation using a configurable `RANDOMIZE_STEPS` parameter, the A* algorithm provides a robust approach for solving sliding puzzles.

The second one is a Beam Search algorithm. Beam Search is a heuristic-guided search algorithm that explores the most promising states at each level while discarding others, striking a balance between computational efficiency and solution quality by limiting the number of states explored at each depth using a fixed *beam size*. The algorithm starts by adding the initial puzzle state to a priority queue, with its priority determined by a heuristic that combines the Manhattan Distance (an estimate of the moves required to reach the goal) and the Difference from Goal (counting tiles out of place). At each step, it selects the top *beam size* states with the lowest heuristic values and generates possible moves by sliding tiles into the empty space. The combined priority is calculated as $0.8 \times \text{difference\_from\_goal} + 0.2 \times \text{manhattan\_distance}$, ensuring prioritization of states closer to the solution. The search halts upon reaching the goal state or terminates if the search space is exhausted. Beam Search is particularly suited for solving n-puzzles when speed and reduced memory usage are crucial, as it significantly prunes less promising paths, though it sacrifices optimality for efficiency.

3.3 Results

The result obtained for the A* algorithm is that the algorithm esolves efficiently the 3x3 matrix in less tha 2 minutes, but for greater values it can take too much time. A 4x4 was obtained after 25 minutes and this is not optimal for our task.

The result obtained for the Beam Search algorithm is that the 3x3 and 4x4 puzzle is solved in less than a minute with quite good cost. Sometimes the correct path cannot be guaranteed, but in this case we focus mainly on speed. The 5x5 puzzle is solved in less than 3 minutes.

As we can understand from this results, the Beam Search algorithm is more efficient than the A* algorithm for this kind of problem. The A* algorithm is more precise, but it takes too much time to solve the problem. The Beam Search algorithm is faster and can solve the problem in a reasonable time.

3.4 Obtained Reviews

The results and the code done was reviews by two collegues of the course. In the following section I will report the obtained revies:

- Review 1 done by jannu99: Excellent work! The provided code uses intelligent strategies tailored to the specific problem, achieving great results even for 5x5 puzzles with a high number of randomized steps. Nothing to add, well done!
- Review 2 done by lorenzo-ll : The solution is very smart, i really like how you combined the heuristics in order to achieve a good formula for the priority. The only thing that i

can suggest is to dynamically adapt BEAM_SIZE based on the complexity of a problem, instead of having it hardcoded. Other than that i think that this is a brilliant take on the problem, good job!

4 Project: Symbolic Regression

The project work was done together to my colleague Antonio Sirica, s326811.

4.1 Description of the problem

The project was about the Symbolic Regression problem. The Symbolic Regression problem is a machine learning task that aims to discover mathematical expressions that best fit a given dataset. The goal is to find a symbolic expression that accurately models the relationship between input variables and output values, allowing for predictions and insights into the underlying data. The problem is defined as follows: given a dataset of input-output pairs, find a symbolic expression that best approximates the relationship between the inputs and outputs. In this case we have 8 different datasets, each one with a different number of input variables and output values. The aim is to find the formula that best fits the data.

4.2 Implementation

The problem was solved using a kind of evolutionary algorithm. To better represents formulas, we have used a Tree structure and a Node structure. The Tree structure is a binary tree that represents the formula, while the Node structure is a node of the tree. The Node structure contains the value of the node, the type of the node (operator or variable), and the left and right children of the node. The Tree structure contains the root of the tree. Each individual is defined as an object of an Individual class, that contains the tree of the individual and the fitness value. The fitness of the tree is calculated as the mean squared error between the predicted values and the actual values of the dataset.

The operators used are the following: addition, subtraction, multiplication, division, sine, cosine, and exponential. The variables are represented as x1, x2, ..., xn, where n is the number of input variables in the dataset. The constants are randomly generated within a specified range between 0.5 and 10 to avoid issues with the 0 values.

The algorithm begins by generating an initial population of trees, where each tree represents a candidate solution. The trees are initialized with random formulas, combining operators and variables to create diverse expressions. The algorithm then evaluates the fitness of each tree based on the mean squared error.

The parent selection is a tournament selection where only the best 5 individuals are taken. Between them, 2 are selected to be the parent of a child. The child is obtained with a crossover operation, where a random subtree of one parent is exchanged with a random subtree of the other parent. In this way, two childs are generated. The choice between one child and the other is done randomly due to computational efficiency reasons.

The child can be mutated with a percentage of 50%. The mutation is done by changing a random subtree of the child with a new generated random subtree, to introduce diversity in the population and to avoid local optimas. The generation of the new random subtree is controlled considering the depth of the tree, to avoid too deep trees and to avoid too simple trees.

The child not mutated and the mutated child are added in a offspring buffer. At the end of the iteration to obtain all the offsprings from the parent population, the offsprings and the previous population are merged. The next population is obtained doing elitism over the merged population, taking the best 5% of the individuals. The remaining 95% of the population is obtained with a roulette wheel mechanism, where the probability of being selected

is proportional to the fitness of the individual. During this process, all duplicated individuals are removed.

The algorithm continues to evolve the population over a specified number of generations (50), with each iteration involving parent selection, crossover, mutation, and population update. The best individual found is retained as the final solution, representing the symbolic expression that best fits the dataset.

4.3 Results

The algorithm described was tested on 8 different problems. For each of them, the final formula, the final MSE on validation set, composed by fitness and a coefficient of complexity, and a plot of the source data and the validation data are shown. The plot allows us to understand the distribution of the original dataset and the distribution created by the algorithm to better understand if a problem is feasible or not.

4.4 Problem 1

The final formula found for the first problem is the following:

$$sin(x0)$$

The corresponding final MSE is 8.851574024403736e-34 The plot of the source data and the validation data is shown in the following figure:

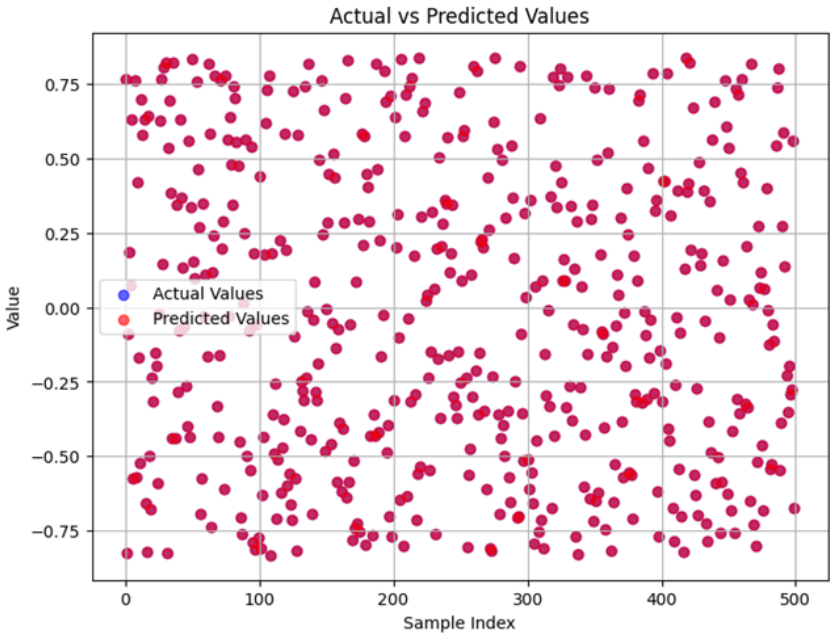

Figure 1: Problem 1

As we can see from the result obtained, the algorithm was able to find the best formula for the problem. The final MSE is very low, indicating that the formula accurately models the relationship between the input and output values. The plot shows a clear correlation between the source data and the validation data, confirming the effectiveness of the algorithm in solving the problem.

The algorithm requires around 30 minutes to compute the all 50 generations.

4.5 Problem 2

The final formula found for the second problem is the following:

$$51.9353 \cdot x_0 \cdot \left(23500.21 + \frac{x_0}{4.8542} \cdot \sqrt{4.0625 - x_1} \cdot 423.0609\right)$$

The corresponding final MSE is 19018888275550.824

The plot of the source data and the validation data is shown in the following figure:

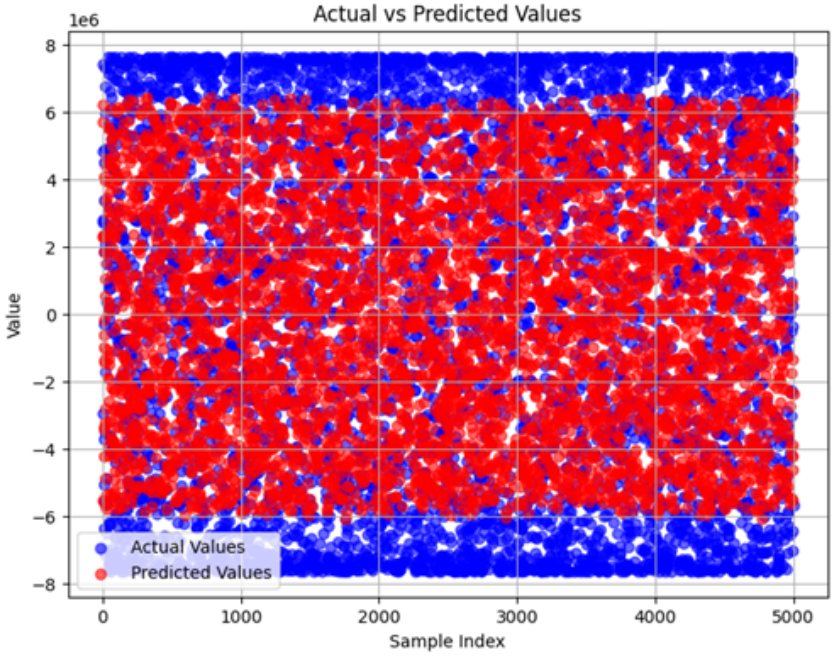

Figure 2: Problem 2

As we can see from the result obtained, the algorithm was not able to find the best formula for the problem. The final MSE is very high, indicating that the formula does not accurately model the relationship between the input and output values. The plot shows a poor correlation between the source data and the validation data, suggesting that the algorithm struggled to find a suitable solution for this problem.

Analysing the distribution of the data, we can see that the data is not linearly separable, and this could be the reason why the algorithm was not able to find the best formula.

The algorithm requires around 100 minutes to compute the all 50 generations.

4.6 Problem 3

The final formula found for the third problem is the following:

$$(22.0833 - 8.625 \cdot x_1 - 4.0625 \cdot \sqrt{x_2}) - \left[\tan\left(\sin\left(\sin\left(\frac{x_2}{10}\right) - \sin\left(3.6667 + (7.2292 - x_0)\right)\right)\right) + 6.0417 \cdot x_1\right]$$

The corresponding final MSE is 615.3642375335544

The plot of the source data and the validation data is shown in the following figure:

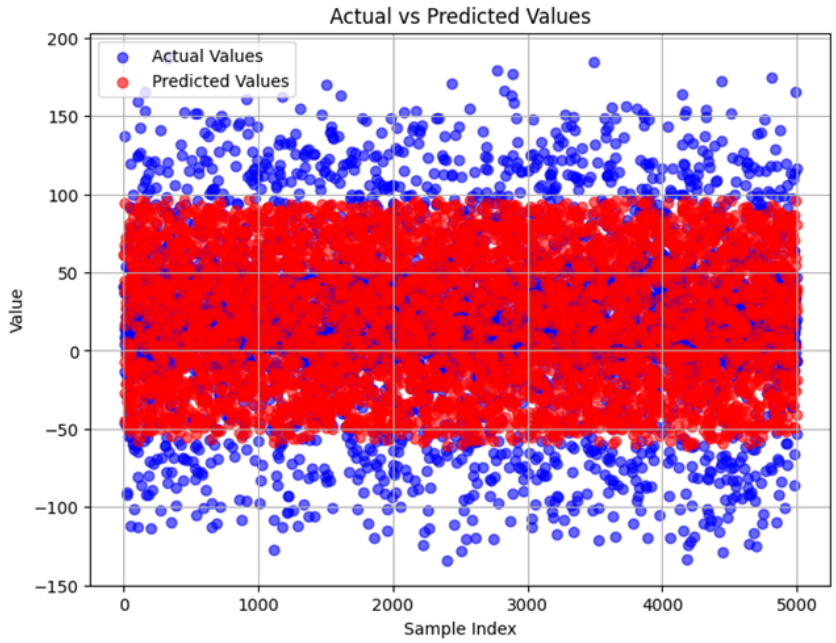

Figure 3: Problem 3

As we can see from the result obtained, the algorithm was not able to find the best formula for the problem. The final MSE is high, indicating that the formula does not accurately model the relationship between the input and output values. The plot shows a poor correlation between the source data and the validation data, suggesting that the algorithm struggled to find a suitable solution for this problem. The algorithm was not able to model the sparse data in the plot and this could be the reason why the algorithm was not able to find the best formula.

The algorithm requires around 110 minutes to compute the all 50 generations.

4.7 Problem 4

The final formula found for the fourth problem is the following:

$$\exp\left(\text{safe\_arcsin}\left(0.6702 + \cos(x_1)\right) + \cos(x_1)\right)$$

The corresponding final MSE is: 3.4654237160276846

The plot of the source data and the validation data is shown in the following figure:

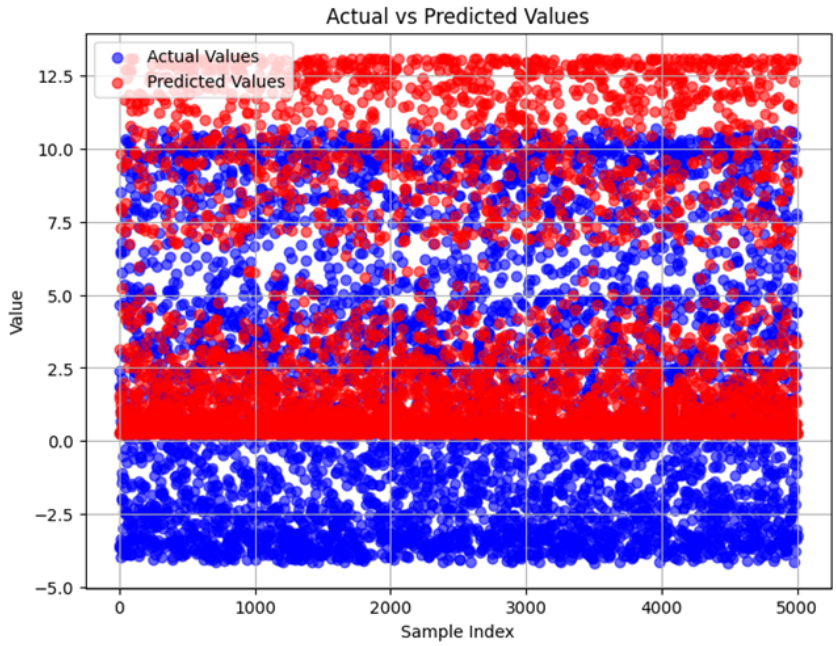

Figure 4: Problem 4

As we can see from the results, the algorithm was able to find a good formula for the problem. The final MSE is low, indicating that the formula accurately models the relationship between the input and output values. We have seen that the final MSE value is between the maximum and the minimum value of the distribution The plot shows a clear correlation between the source data and the validation data, confirming the effectiveness of the algorithm in solving the problem. However, it seems that the algorithm was able to model the sparse data in the plot with a shift in the results because the distribution created by the formula is 3.0 points in high than the baseline of the sparse data. Probably it is only need to shift the final formula to obtain a more correct result.

The algorithm requires around 100 minutes to compute the all 50 generations.

4.8 Problem 5

The final formula found for the fifth problem is the following:

$$(\log(\cos(\text{safe\_arccos}(\text{safe\_arcsin}(\sin(\text{safe\_log}(2-x_1)))))))\cdot \left(\text{safe\_tan}\left(\left(\text{safe\_tan}(\cos(\text{safe\_arcsin}(4.25)))^{0.875}\right)^{0.875}\right)^{0.875}\right)$$

The corresponding final MSE is: 5.081384631747552e-18

The plot of the source data and the validation data is shown in the following figure:

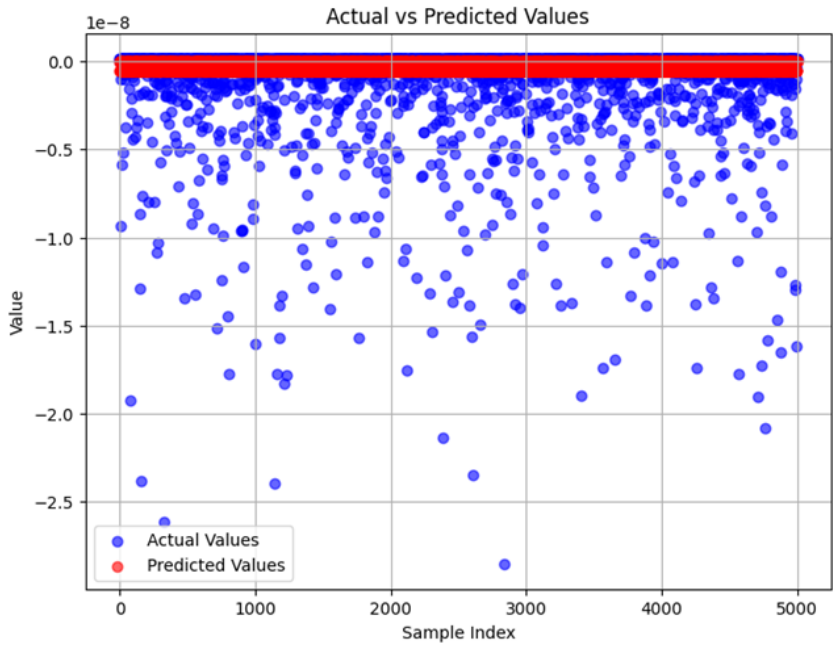

Figure 5: Problem 5

As we can see from the results, the algorithm was able to find a good formula for the problem. The final MSE is very low, indicating that the formula accurately models the relationship between the input and output values. However, this value is obtaine because the formula was able to model correctly the high density area of the source data. As we can see from the plot, there are few sparse data in the source dataset that are not correctly modelled by the algorithm.

The algorithm requires around 140 minutes to compute the all 50 generations.

4.9 Problem 6

The final formula found for the sixth problem is:

$$x_1+\Big(x_1+\Big(\text{safe\_arctan}\Big(\text{safe\_sqrt}\Big(\text{safe\_exp}\Big(\Big(\frac{x_0}{2.1379310344827585}\Big)-x_1-1.1551724137931034\Big)\Big)\Big)-x_0\Big)\Big)$$

The corresponding final MSE is: 0.28204828072700494

The plot of the source data and the validation data is shown in the following figure:

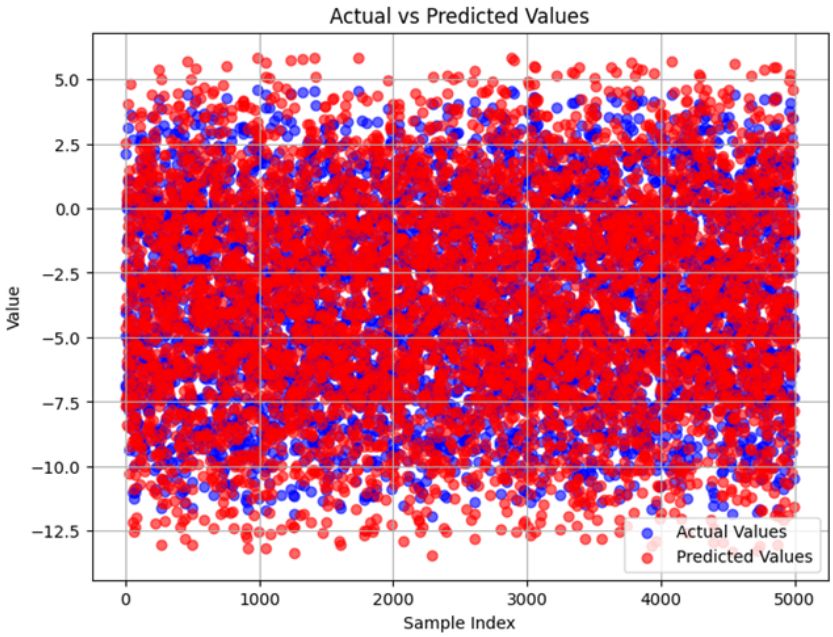

Figure 6: Problem 6

As we can see from the results, the algorithm was able to find a good formula for the problem. The final MSE is low, indicating that the formula accurately models the relationship between the input and output values. The plot shows a clear correlation between the source data and the validation data, confirming the effectiveness of the algorithm in solving the problem. The algorithm does not produce a distribution that cover precisely the source data, but it is able to model the data in a good way.

The algorithm requires around 180 minutes to compute the all 50 generations.

4.10 Problem 7

The final formula found for the seventh problem is:

$$\text{safe\_arctan}(x_1 - 4.25) + 2 \cdot \exp\left((x_0 \cdot x_1) + 0.875\right)$$

The corresponding final MSE is: 330.58545575979997

The plot of the source data and the validation data is shown in the following figure:

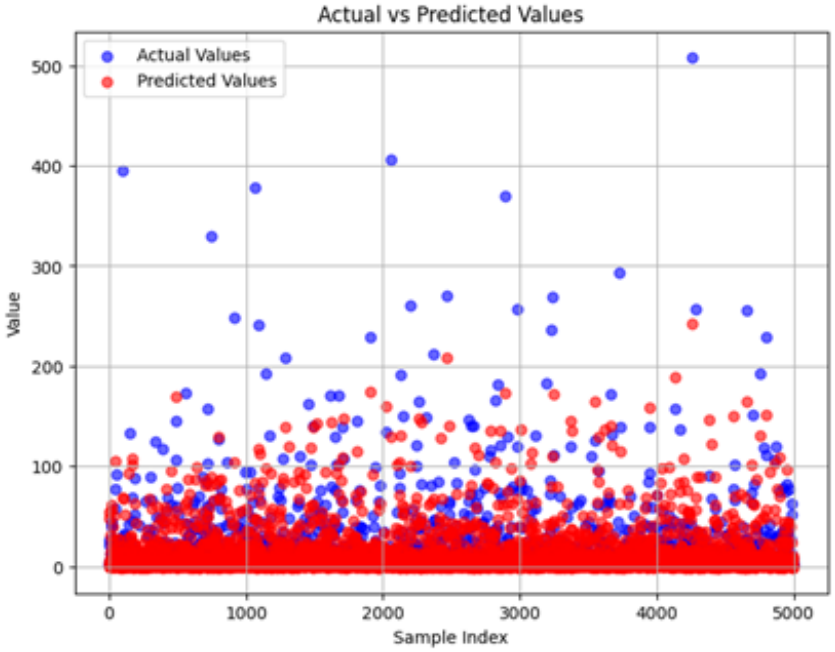

Figure 7: Problem 7

As we can see from the results, the algorithm was not able to find the best formula for the problem. The final MSE is high, indicating that the formula does not accurately model the relationship between the input and output values. The plot shows a poor correlation between the source data and the validation data, suggesting that the algorithm struggled to find a suitable solution for this problem. As in problem 5 The algorithm was not able to model the sparse data in the plot and this could be the reason why the algorithm was not able to find the best formula.

4.11 Problem 8

The problem was too complex for our resources and after 300 minutes the algorithm have computed only 14 generation. The following picture shows the behaviour of our algorithm:

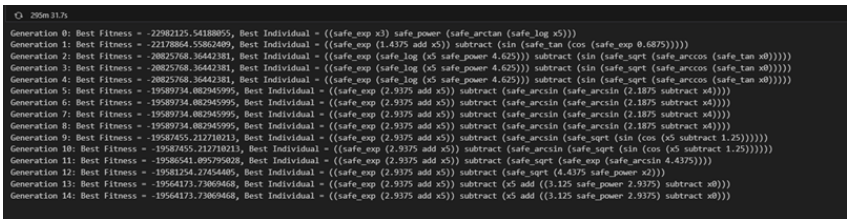

Figure 8: Problem 8

After 14 generation, the obtained formula is

$$\exp(2.9375 + x_5) - \left(x_5 + \left(3.125^{2.9375} - x_0\right)\right)$$

The corresponding final MSE is: 19564173.730069468

As we can see from the results, the algorithm was not able to find the best formula for the problem. The final MSE is very high, indicating that the formula does not accurately model the relationship between the input and output values.

5 Presentation

In the day 09/01/2025, I have presented a short introduction about Deep Q-Learning. The presentation was about the main concepts of Deep Q-Learning, the main differences between Q-Learning and Deep Q-Learning, and the main applications of Deep Q-Learning. The presententation file was sent to the professor Squillero and shared on telegram channel.

6 Conclusions

The laboratory and the project work done during the course were very interesting and challenging. The problems proposed were complex and required a deep understanding of the underlying concepts to develop effective solutions. The Set Cover Problem, Traveling Salesman Problem, n-puzzle problem, and Symbolic Regression problem each presented unique challenges that tested our problem-solving skills and algorithmic knowledge. The implementation of the algorithms, such as Hill Climbing, A*, Beam Search, and evolutionary algorithms, required careful consideration of the problem constraints and solution strategies. The reviews provided by colleagues were insightful and helped identify areas for improvement in the code and algorithm design. Overall, the course provided valuable hands-on experience in solving real-world optimization problems and developing efficient algorithms. The project work on Symbolic Regression was particularly engaging, as it involved exploring complex datasets and developing symbolic expressions to model the underlying relationships. The results obtained from the project highlighted the effectiveness of evolutionary algorithms in solving symbolic regression problems and the importance of understanding the data distribution and problem complexity. The feedback received from colleagues and the professor was valuable in refining the solutions and enhancing the overall learning experience. I look forward to applying the knowledge gained from this course to future projects and research endeavors.