Part2 Report

Code Explanation

This program demonstrates and executes the application of Genetic Programming (GP) in Symbolic Regression problems. Symbolic Regression is a machine learning technique aimed at finding the mathematical expressions that best describe a given dataset. This process involves not only the optimization of model parameters but also the search for the model structure. The program utilizes the DEAP (Distributed Evolutionary Algorithms in Python) library to implement various stages of GP, including population initialization, selection, crossover, mutation, and evaluation.

The program has been enhanced and characterized by the following improvements:

1. **Global Parameter Configuration**: A series of configurable parameters, such as problem type (`Problem`), population size (`PopulationSize`), tournament size (`TournamentSize`), crossover rate (`CrossoverRate`), mutation rate (`MutationRate`), etc., are defined through global variables, making the experimental setup more flexible.

2. **Dataset Creation and Processing**: The `create_dataset` function is provided to generate input points (`InputPoints`) and target points (`TargetPoints`) according to the selected problem (`Problem`), preparing data for GP operations.

3. **Protected Division Operation**: The `protectedDiv` function is added to the primitive set to prevent division-by-zero errors during program execution.

4. **Dynamic Folder Creation**: The program automatically creates folders for storing charts, logs, and results based on the current time, facilitating users to track and manage outputs generated by different run instances.

5. **Performance and Size Statistics**: The `create_mstats_object` function sets up a multi-dimensional statistics object (`MStats`) to track and record statistics like fitness and individual size for each generation.

6. **Results Visualization**: Functions such as `plot_fitness_and_size`, `plot_dataset`, and `plot_comparison_Target_Vs_EvolvedSolution` are provided to plot curves of fitness and individual size over generations, show the distribution of datasets, and compare the difference between the GP-found solution and the target function, respectively.

7. **Flexible Command Line Interface**: The program supports specifying the number of generations (`NumGenerations`), population size (`PopulationSize`), and tournament size (`TournamentSize`) directly via command-line arguments or running with default values.

8. **Main Function Logic**: The `main` function orchestrates the entire GP process, from problem setting and population initialization to GP execution and result output, supporting experiments with different combinations of problems, population sizes, and tournament sizes.

I add two problems that the fitness function of the first one is `5*x**5 + 4*x**4 + 3*x**3 + 2*x**2 + x` and the fitness function of the second one is `4*math.sin(5*math.pi*x/4)`. Given the hyper-

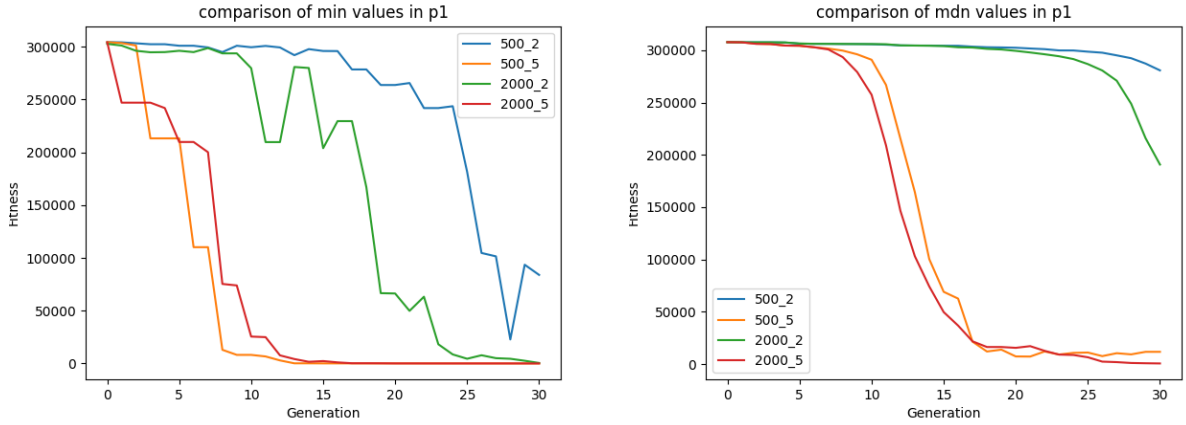
parameters choices, I do four experiments for each problem with two population sizes and two tournament sizes.

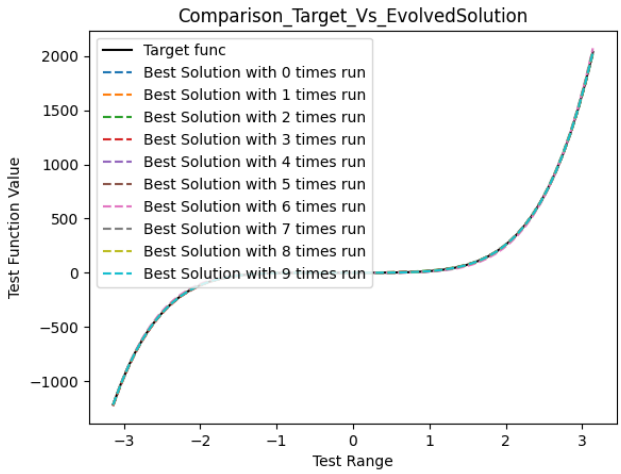
The program firstly sampled `65` points in inside the range `-pi` and `pi` as the target points. Then a primitive set is generated which include basic calculation operators such as `add`, `sub`, `mul`, `neg`, `sin`, `cos`, and `div` which with protection to avoid denominator to be zero. According to the given hyper-parameters, the program then initialized the population composed with primitive trees with the max height `64`. The program then iteratively executed the GP algorithm for the given number of generations, and finally output the best solution (HOF) found.

In the next part, I will show the results of the experiments, and my observation and discussion.

Observation and Discussion:

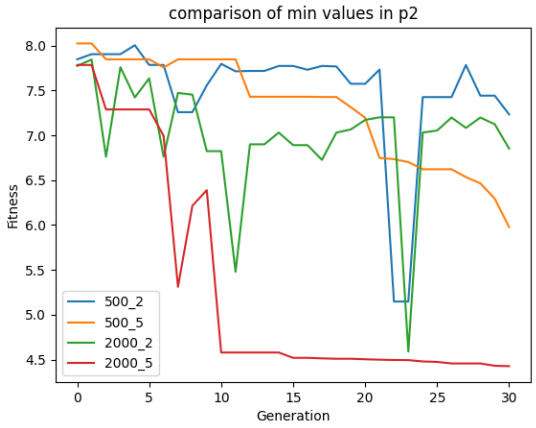
The Genetic Programming (GP) algorithm demonstrated significant changes in behavior as the problem definition, population size, and tournament size were altered. Specifically, for different problems, the shapes of the best solution curves varied considerably, reflecting the GP's ability to adapt its search strategy based on the problem context. The diversity in the generated functions signifies that GP is capable of exploring a wide range of solutions across different parameter settings. This adaptability is crucial for solving complex problems where the landscape of possible solutions is vast and varied. In the following figures, we can see the change of fitness through generation s for different parameter settings.

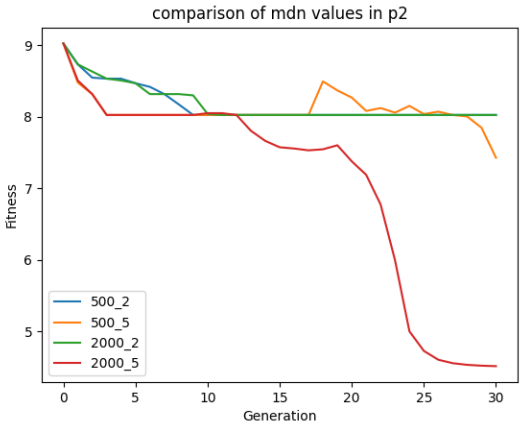


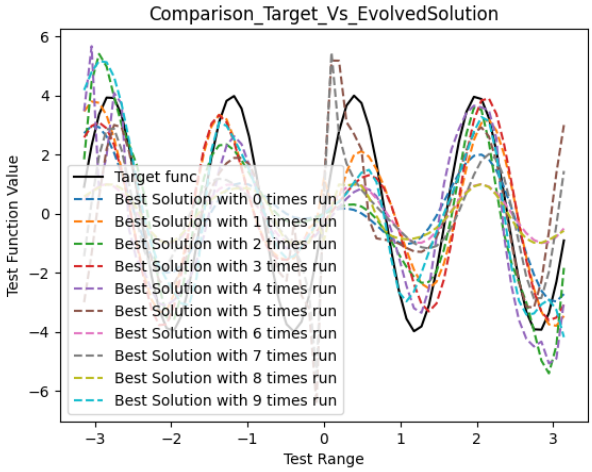


The comparsion of min fitness values and median fitness values for different parameter settings is shown in the above figures. The legend `500_2` means 500 population size and 2 tournament size. The same applies

to others. It is clear to see tournament size play a key role for accelerating the convergence of the best solution. The larger the tournament size, the faster the best solution is found. Besides, the larger the population size, the more diverse the population becomes, which may lead to better solutions.

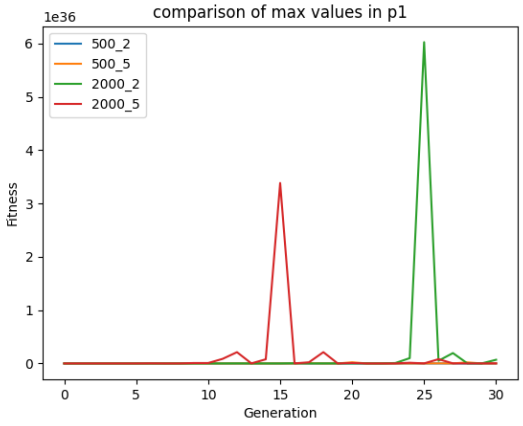


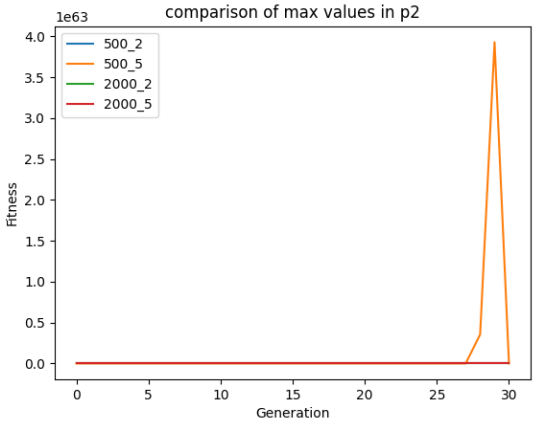


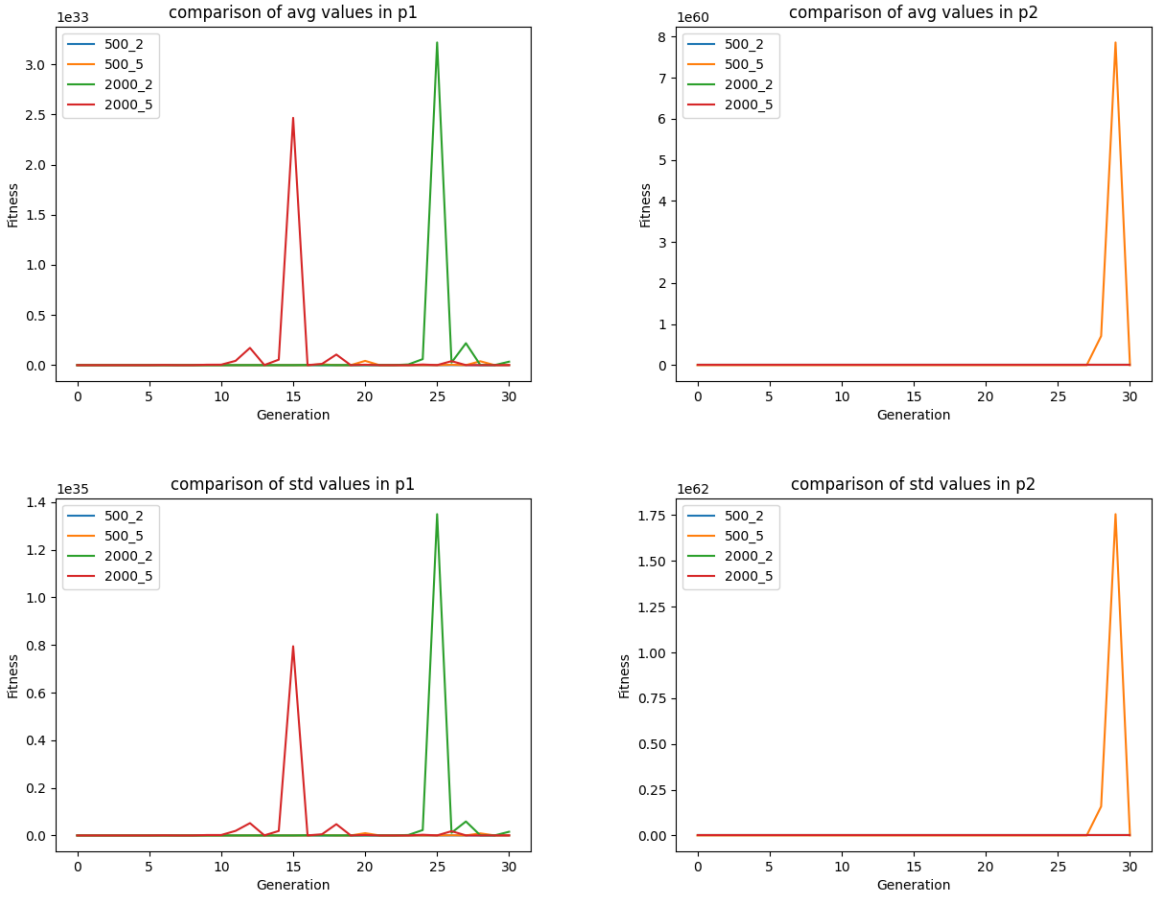


The second problem is more challenging. The fitness landscape is more complex and varied. The above figures show the change of fitness through generation s for different parameter settings. It is clear to see that red dots line perform good, where the hyper-parameter configuration is 2000 population size and 5 tournament size. In this case, population size looks like play a key role for convergence.

I also collect other statistics figures, such as the max fitness value, average fitness value, and standard deviation fitness value. The results are shown in the following figures.



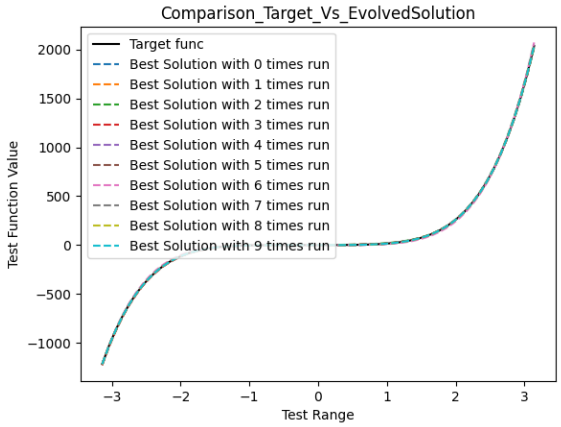


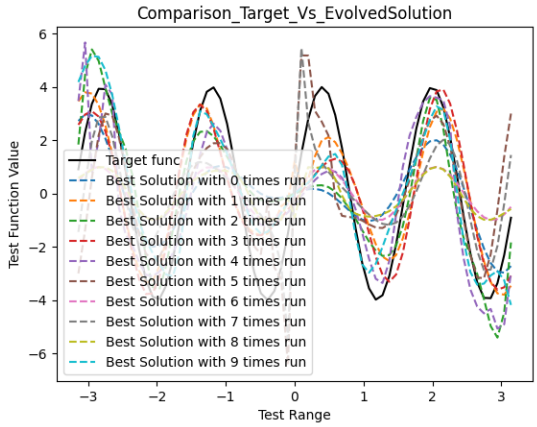


In the left column, you can find the maximum, average, and standard deviation fitness values for the first problem. The right column displays the corresponding values for the second problem. Interestingly, it is noticeable that despite variations in parameter settings, a similar trend seems to emerge. While occasional outliers may be observed in the data, overall, they exhibit consistent fluctuations around a certain level without showing any clear convergence pattern.

The criterion for identifying the best parameter configuration was based on minimizing the square error between the GP-generated solutions and the ground truth. This objective aims to closely match the GP solution to the target function, ensuring accuracy and relevance. By systematically varying the parameter configurations and evaluating the resulting square errors, we identified the configurations that yielded the lowest errors as the optimal settings for each problem. This approach ensures a data-driven, empirical basis for selecting parameter configurations that are most effective for the given problems. From the observation, the best configuration is population size equal to 2000 and tournament size equal to 5.

Using best hyper-parameters configuration, I tried running another 10 times of the program. The results are shown in the following figures.





According to the figures above, using the best configuration looks quiet robust that can fit the target function.

In summary, the findings and insights underscore the versatility and potency of GP as an invaluable tool for addressing symbolic regression challenges. Its capacity to deliver a wide array of adaptive solutions across diverse parameter settings and problem scenarios exemplifies its robustness. However, it is imperative to underscore that the efficacy of GP critically hinges on the meticulous selection of parameter configurations, necessitating comprehensive experimentation and analysis. Furthermore, GP's applicability is contingent upon its adeptness in striking a balance between exploration and exploitation, thereby enabling effective navigation through intricate solution spaces. While GP exhibits tremendous potential, its success in tackling specific problems demands a nuanced grasp of its underlying mechanisms and a strategic approach to parameter optimization.

The following points elucidate why GP is excellently tailored for symbolic regression problems:

- Flexibility: GP's independence from predefined model structures empowers it to explore a broad spectrum of potential solutions, encompassing complex and nonlinear relationships.
- Automatic Feature Construction: GP's capability to autonomously generate new features or variables during evolution facilitates the discovery of intricate data relationships.
- Global Optimization: Leveraging evolutionary principles such as selection, crossover, and mutation, GP diligently seeks optimal solutions within a global context, thereby sidestepping local optima.
- Interpretability: The solutions derived through GP often manifest as mathematical expressions, rendering them interpretable and furnishing insights into the underlying data relationships.
- Adaptability: GP adeptly adjusts to various types of regression problems, effectively handling both univariate and multivariate datasets.

This comprehensive amalgamation of traits solidifies GP's standing as an indispensable and adaptable approach for symbolic regression, offering a compelling blend of flexibility, interpretability, and global optimization capabilities to tackle regression tasks with finesse.

Table of summary

Problem	Population_size	tounament_size	size_median	fitness_median	fitness_min
p1	500	2	54	227,717.51	1,588.43
p1	500	5	45	22,441.73	667.36
p1	2000	2	48	188,272.49	1,728.76
p1	2000	5	70	242.87	3.39
p2	500	2	15	8.02	7.24
p2	500	5	38	6.25	4.33
p2	2000	2	14	8.02	6.57
p2	2000	5	14	5.71	2.07