Course Report: Log of Activities

Course Title: Computational intelligence
Student Name: Ghisolfo Giorgia
Student ID: s332968
Submission Date: 13th June 2025

---

2. Log of Activities

This section includes a chronological log of the activities carried out during the course, organized by date. It also provides descriptions of the tasks and challenges and discussions about the laboratories.

Week 1: [23 September, 30 September]

- **Activity**:
    - **Lecture**: Introduction to the course and the local search algorithm (hill climber).
    - **Details**: We discussed how to apply the newly introduced concepts by analyzing the "6 Friends Problem" as a practical example. We also discussed the 1-max problem, 2-max problem and knapsack problem.
    - **Challenges**: The main challenge was formulating a solution for the 6 Friends Problem.

Week 2: [30 September, 7 October]

- **Activity**:
    - **Lecture**: Defined the concepts of exploration and exploitation, introduced additional local search algorithms (RMHC, Steepest-step HC, Simulated Annealing, Tabu Search), and introduced Evolutionary Algorithms, brief mention of Iterated Local Search.
    - **Details**: The lecture focused on the balance between **exploration** (searching new areas of the solution space) and **exploitation** (optimizing within the current area). We also explored different local search algorithms such as **Simulated Annealing**, which incorporates randomness to avoid local optima, and **Tabu Search**, which prevents revisiting previously explored solutions. Additionally, we were introduced to the basics of **Evolutionary Algorithms**, which mimic natural selection to evolve solutions over generations.

- **Challenges**: The main challenge was applying these concepts to the **Set Cover Problem**. We needed to ensure that the solution could efficiently cover all elements while minimizing the number of sets.
- **Solutions**: I applied a hybrid approach starting with a **Greedy Algorithm** to quickly generate an initial solution by selecting sets that cover the most uncovered elements. From there, I used **Tabu Search** to escape local optima, iteratively improving the solution by exploring the neighborhood of the current solution and preventing cycling back to previously considered solutions. This approach allowed me to find a more optimal solution while avoiding stagnation in local minimum.

Week 3: [7 October, 14 October]

- **Activity**:
  - **Lecture**: discuss Evolutionary Algorithm and population management model,
  - **Details**: EA terminology is introduced (genotype, phenotype, fitness) along with population management models (generational and steady-state). Genetic operators (crossover, mutation). Additionally, historical and modern EA variants are discussed, including Genetic Algorithms (GA), Evolutionary Strategies (ES), Differential Evolution, and Particle Swarm Optimization.
  - **DEADLINE for the set-cover problem 9 October -> lab01**

Week 4: [14 October, 21 October]

- **Activity**:
  - **Lecture**: Discussion on parent selection strategies, genetic operators, and different types of representation. Introduction to survivor selection strategies.
  - **Details**: The lectures focused on the various **parent selection strategies** used in Evolutionary Algorithms, such as **fitness-proportional selection**, **tournament selection**, and **rank-based selection**. Each strategy has its own method for determining which individuals are chosen for reproduction, influencing the diversity and convergence speed of the population. We also examined **genetic operators**, including **crossover** (recombination of two parents to create offspring) and **mutation** (small changes in the genotype). These operators help explore the solution space, balancing between exploration and exploitation. Different forms of **representation** were introduced, such as **binary strings**. Finally, we covered **survivor selection strategies**, which determine how the next generation is formed. Examples include **elitism**, where

the fittest individuals are preserved, and **steady-state selection**, where offspring compete with parents for survival. These strategies ensure that the population evolves while maintaining diversity.

- **Challenges**: Travelling salesman problem.
- **Solutions/Outcome**: I began analyzing the provided data by initializing the problem, calculating distances between points, and organizing cities into an appropriate data structure for further processing.

Week 5: [21 October, 28 October]

- **Activity**:
  - **Lecture**: Discuss about evolutionary algorithm and its operators. These algorithms employ different representations.
  - **Details**: **Evolutionary algorithms** employ diverse representations such as binary, integer, and floating-point encodings to address a wide range of optimization challenges, from discrete combinatorial problems to continuous numerical domains. Modern variants, like Particle Swarm Optimization and Differential Evolution, further enhance adaptability and performance through innovative selection, mutation, and crossover techniques.
  - **Challenges**: Travelling salesman problem.
  - **Solutions/Outcome**: I continued analyzing the provided data by initializing the problem, calculating distances between points, and organizing cities into an appropriate data structure for further processing.

Week 6: [28 October, 4 November]

- **Activity**:
  - **Lecture**: The lecture covered a range of topics related to evolutionary computation techniques and strategies, focusing on both classical and contemporary methods.
  - **Details**: during the lectures of that week we analyze different concepts, including Evolutionary Programming (EP), which focuses on prediction through mutation of real-valued vectors; Genetic Programming (GP), which evolves tree structures for symbolic tasks like regression; and Learning Classifier Systems (LCS), combining genetic algorithms with rule-based systems. Differential Evolution (DE) and Particle Swarm Optimization (PSO) optimize solutions through differential mutation and social-inspired velocity adjustments, respectively. To

prevent premature convergence and promote diversity, various strategies, like the island model, fitness sharing, and lexicase selection, are employed within these algorithms.

- **Challenges**: Travelling salesman problem.
- **Solutions/Outcome**: The goal of the challenge is to provide two solutions for the TSP problem:
    1. **Fast but Suboptimal Solution**: A method that prioritizes speed, even if the solution is not perfectly optimal. I choose **Greedy Nearest Neighbor**, which quickly generates a solution by iteratively selecting the nearest unvisited city,
    2. **Optimal but Slower Solution**: A method that focuses on achieving a more optimal solution, even at the cost of longer execution time. I choose **Simulated Annealing**, which explores the solution space iteratively and probabilistically, gradually converging to a near-optimal result.
- **DEADLINE TSP problem 3 November -> lab02**

Week 7: [4 November, 11 November]

- **Activity**:
    - **Lecture**: During the lecture we discuss various genetic algorithm (GA) techniques, such as gender-based selection, random immigration, and extinction to promote diversity in populations. It covers different genetic programming (GP) methods, including symbolic regression, and problem-solving through search algorithms, like breadth-first and A* search. It also explores how diversity can be enhanced in evolutionary processes, considering both fitness and entropy. Additionally, it includes insights on different genetic programming strategies like LGP and their computational advantages and challenges.
    - **Details**: During the lecture the professor presents several methods for enhancing genetic diversity in evolutionary algorithms. Techniques like gender-based reproduction and extinction promote diversity by preventing inbreeding and encouraging exploration of new solutions. Random immigrants periodically introduce novel individuals to the population, while extinction events remove optimized individuals to allow the emergence of new solutions. It discusses the application of genetic algorithms (GA) and genetic programming (GP) to problems such as symbolic regression and program synthesis, along with searching techniques like greedy best-first and A* for pathfinding and optimization.

- **Challenges**: flipped lab,
    - Quickly find a smart path from A to B on the Italian map,
    - Consider each city connected with the 5 closest cities.
- **Solutions/Outcome**: I started to create the dataset with the information of the Italian road map respecting the constraint of 5 connections

DEADLINE TSP problem review -> while I was looking at my colleagues' codes to do the review, I realized that I had not provided the correct results for the tls problem i.e. the number of steps and the final cost, so I modified the code to comply with the request.

Week 8: [11 November, 18 November]

- **Activity**:
    - **Lecture**: Provide more information about a* and propose a bonus problem. The main concept of this week was genetic programming.
    - **Details**: Genetic Programming (GP) is a form of evolutionary algorithm where solutions are represented as tree-like structures. It features diverse initialization methods, such as the Full and Grow methods, and selection mechanisms like fitness-proportionate parent selection. Recombination and mutation operators modify the solutions, including subtree exchanges, point mutation, and bloat-reducing strategies. GP's flexibility contrasts with other evolutionary methods (e.g., GA, PSO), as it operates with varying tree structures rather than fixed-length chromosomes.
    - **Challenges**: n - puzzle problem.
    - **Solutions/Outcome**: I started creating random initialization matrix and I did research on the possible approaches to solve the problem.

Week 9: [18 November, 25 November]

- **Activity**:
    - **Lessons**: the lectures focused on Modern genetic programming and other approaches like CGP, LGP, and GE. Parameter adjustment is crucial and can be done through pre-execution tuning or adaptive/self-adaptive control during evolution. The focus is on modularity, efficiency, and automated optimization using evolutionary methods
    - **Details**: Modern genetic programming (GP) includes advanced approaches such as Cartesian Genetic Programming (CGP), which uses graphs with non-coding

genes for flexibility, and Linear Genetic Programming (LGP), which evolves programs as sequences of instructions. Other methods, like Grammatical Evolution (GE), base evolution on grammatical rules. However, challenges like large search spaces and infeasible regions require improvements, such as graphs to preserve structure and semantics. Parameter management is fundamental to balancing performance and optimization. Key strategies include:

- **Parameter Tuning**: testing values pre-execution, limited by time costs and the variation of parameters over time.
- **Parameter Control**: dynamically adjusting parameters during evolution through deterministic, adaptive, or self-adaptive rules.

- **Challenges**: n - puzzle problem.
- **Solutions/Outcome**: To solve the n-puzzle problem (a simplified sliding puzzle) using the A* search algorithm, the key components are:

  1. **State Representation**: Represent the puzzle as a 2D array, where the blank space is denoted by 0.
  2. **Heuristic Function**: Use a heuristic to estimate the cost to the goal. The Manhattan distance is commonly used.
  3. **Actions**: Valid moves for the blank space (up, down, left, right).
  4. **Priority Queue**: Maintain states in a priority queue sorted by f(n) = g(n) + h(n):
     - g(n): Cost to reach the current state.
     - h(n): Estimated cost to reach the goal (heuristic).
  5. **Goal Test**: Check if the state matches the goal configuration.

- **DEADLINE n - puzzle problem 20 November -> lab03**

Week 10: [25 November, 2 December]

- **Activity**:
  - **Lessons**: the lessons focused on Reinforcement Learning (RL). RL focuses on decision-making through interaction with an environment to maximize cumulative rewards. Using Markov Decision Processes (MDPs), RL optimizes policies through methods like value iteration, policy iteration, and Q-learning. Challenges include balancing immediate versus long-term rewards, handling uncertainty, and addressing sparse or delayed rewards.

- **Details**: Reinforcement Learning (RL) aims to solve sequential decision-making problems by optimizing an agent's policy to maximize expected cumulative rewards. Agents interact with their environment, receiving observations and rewards based on their actions. RL is formalized using Markov Decision Processes (MDPs), defined by states, actions, transition probabilities, rewards, and discount factors. Policies, either deterministic or stochastic, guide actions in each state. RL addresses challenges such as sparse or delayed rewards, where long-term planning is crucial. We also discussed:
    - **Model-free method**s (e.g., Q-learning, SARSA) learn directly from trial-and-error,
    - **Model-based methods** use environment models for planning, leveraging predictions of state transitions and rewards.
- **Challenges**: Project work.
- **Solutions/Outcome**: No sufficient information for understanding something about project work.

Week 11: [2 December, 9 December]

- **Activity**:
    - **Lessons**: provide examples of RL code and discuss project work.
    - **Challenges**: Project work.
    - **Solutions/Outcome**: I'm trying to understand how to find a formula given the data. Searching for information about this kind of problem.

Week 12: [9 December, 16 December]

- **Activity**:
    - **Lessons**: the lessons focused on the Model-Free Reinforcement Learning methods like Monte Carlo (MC) and Temporal-Difference (TD) Learning. MC works with complete episodes, producing unbiased high-variance estimates, while TD bootstraps intermediate steps, offering lower-variance but biased estimates. the lessons also explored techniques like SARSA and Q-learning enable on-policy and off-policy learning which drive control algorithms with exploration strategies like ε-greedy and GLIE.
    - **Details**: Model-Free Reinforcement Learning focuses on estimating value functions and improving policies directly from episodes of experience without requiring knowledge of the Markov Decision Process (MDP) model. Monte Carlo

(MC) methods calculate values as the mean return from complete episodes, making them suitable for episodic MDPs but requiring termination for updates. MC methods provide unbiased but high variance estimates and are insensitive to initial conditions. Temporal-Difference (TD) methods, such as TD(0), update values using bootstrapping which allows online learning in continuing environments. TD updates have lower variance but are biased and sensitive to initial values. For control tasks, SARSA enables on-policy learning, while Q-learning supports off-policy learning by using the behavior policy for exploration and the target policy for evaluation. These methods utilize exploration strategies like ε-greedy and GLIE to ensure sufficient exploration while converging to optimal policies. Advanced game-search techniques like Minimax and Alpha-Beta Pruning optimize decision-making in deterministic, two-player games with perfect information, using heuristics and evaluation metrics to manage large search spaces efficiently.

- **Challenges**: Project work
- **Solutions/Outcome**: I started to study the suggested library.

Week 13: [16 December, 23 December]

- **Activity**:
  - **Lessons**: the lessons focused on Stochastic games integrate random events like dice rolls into decision-making models (e.g., Stochastic MinMax), while games with imperfect and non-zero-sum information use strategies based on prediction and equilibrium.
  - **Details**: Stochastic games, like incorporating dice rolls into game trees, introduce computational complexity (e.g., high branching), requiring techniques like Expected MinMax and Monte Carlo simulations. In imperfect information games, unknown states can be treated as either unknown or probabilistic to optimize decisions using MinMax or Expected MinMax strategies. The Iterated Prisoner's Dilemma explores cooperation and conflict with strategies like Tit for Tat.
  - **Challenges**: Project work.
  - **Solutions/Outcome**: I attempted to implement a possible solution using the libraries to obtain a comparison of the MSE value.

---

3. Projects

Symbolic Regression:

- **Objective**: The project focuses on the Symbolic Regression problem, which is a task aimed at discovering mathematical formulas or expressions that best represent a given dataset in particular the goal is to identify a symbolic expression that effectively captures the relationship between input variables and corresponding output values. The problem is defined as follows: given a dataset consisting of input-output pairs, the objective is to find a symbolic expression that closely approximates the relationship between inputs and outputs. In the project, we work with 8 distinct datasets, each containing a varying number of input variables and output values. The challenge is to find the formula that provides the best fit for each dataset.
- **Approach**: The implementation follows a modular approach, breaking down the problem into smaller tasks such as tree creation, selection, mutation, and evolution.

1. **Initialization and Data Loading**

The **SymbolicRegression** class begins by initializing important parameters such as population size, number of generations, mutation and crossover rates, and the path to a file containing the dataset. The dataset is loaded using `numpy`, extracting x (input features) and y (target output) arrays. The number of variables in the dataset is determined by the shape of x, and variable names are dynamically generated to match the number of features.

2. **Tree Creation and Representation**

One of the key components of symbolic regression is the creation of symbolic trees that represent mathematical expressions. Each individual in the population is represented as a tree, where nodes are either or leaf nodes that hold variables (like x0, x1) or constants. The challenge in this implementation is building these trees correctly, especially ensuring the proper usage of binary and unary operators. For binary operators, which require two child nodes (left and right). Unary operators, like sin or sqrt, require only one child node. Therefore, the tree-building function must handle both types of operators properly, ensuring that no illegal tree structures (e.g., unary operators with more than one child) are created. The process starts with recursive calls that build the tree from the root, and at each step, the algorithm decides whether to create a binary operator node or a unary operator node. If a leaf node is needed, it will choose either a constant or a variable. A crucial part of this function is managing the node count to avoid overly deep trees, which can lead to overfitting or computational inefficiency.

3. **Fitness Evaluation**

Fitness evaluation is an essential step to guide the evolution of the population. The fitness of each individual is calculated by measuring how well its corresponding expression fits the data. The fitness function is the Mean Squared Error (MSE) between the predicted and actual output values, and the goal is to minimize this error. Each tree in the population is evaluated by applying the symbolic expression it represents to the input data and calculating the MSE. The fitness value is then assigned to the individual, which will be used in subsequent selection, crossover, and mutation operations. The implementation also includes error handling to ensure

that individuals with invalid expressions (e.g., division by zero or mathematical errors) are assigned high fitness values (poor solutions) to avoid their selection in future generations.

4. **Selection Mechanism**

Selection is a critical aspect of evolutionary algorithms, ensuring that individuals with better fitness values have a higher chance of being selected for reproduction. In this implementation, tournament selection is used. During tournament selection, a random subset of the population is chosen, and the best individual (the one with the lowest fitness value) is selected from this group. This method promotes the survival of the fittest individuals while maintaining diversity in the population. The size of the tournament is dynamically adjusted based on the fitness of the best individual, favoring exploration when the model's error is high and exploitation when the model is performing well.

5. **Crossover - Recombination**

Crossover is the genetic operator that combines two parent individuals to produce offspring. In the project, this involves selecting random subtrees from two parent trees and swapping them to create new offspring. This process is known as subtree crossover. The crossover function ensures that the resulting child trees are valid, they must respect the rules for binary and unary operators. The depth of the trees is also controlled to ensure that the offspring do not grow too deep, which could lead to overfitting or computational inefficiency. If a child tree violates the depth constraints, the algorithm reverts to the parent tree. The challenge here is maintaining the structural integrity of the trees after crossover while ensuring that the resulting offspring can still represent valid mathematical expressions.

6. **Mutation**

Mutation introduces small, random changes to an individual to explore the search space further. In the project, a point mutation is applied to randomly selected nodes in a tree. If the selected node is a leaf node (constant or variable), it is replaced with a new randomly selected leaf. If the node is an operator, it is replaced with a randomly chosen operator of the same type (either unary or binary). Mutation is typically applied with a lower probability compared to crossover, as the goal is to introduce small variations rather than large changes. The mutation rate is adaptive, changing during the evolution process based on the error (MSE) of the best individual. When the error is high, the mutation rate is increased to promote exploration; when the error is low, the mutation rate is decreased to allow for more exploitation of the current best solutions.

7. **Elitism**

Elitism is a mechanism that ensures the best individuals are preserved in each generation. The top elitism_sizeindividuals, based on their fitness values, are carried over to the next generation without any changes. This guarantees that the population does not lose the best solutions and that they can continue to evolve and improve. The elitism size is dynamically determined, with a default of 10% of the population.

8. **Evolutionary Process**

The main loop of the evolutionary process involves iterating through multiple generations, each of which consists of the following steps:

- Evaluate Fitness: Evaluate the fitness of each individual in the population,
- Selection: Select individuals based on their fitness using tournament selection,
- Crossover and Mutation: Apply crossover and mutation to create offspring,
- Elitism: Carry over the best individuals from the current generation,
- Replace Population: Replace the old population with the new generation. The process continues until a stopping condition is met, such as reaching the maximum number of generations.

- **Results**:

Dataset 1

- **MSE value**: 7.125940794232773e-32
- **Best individual**:

```
sin(x0)
```

Dataset 2

- **MSE value**: 6.64606e+14
- **Best individual**:

```
denominator = x[0] + x[0] + x[1] + x[2]
np.sign(-x[0] + (52.4 / denominator)) * 4.79e6
```

Dataset 3

- **MSE value**: 1027.43
- **Best individual**:

```
((((4.9127 * np.abs(1.7364 * x[0])) - (17.737 * 1.5036 * x[1])) - 2.9209 *
1.1914 * x[2]) + 5.0619 * np.cos(-1.3349 * x[0])) + 51.8489 * np.sin(0.5644
* x[1])
```

Dataset 4

- **MSE value**: 3.41247e-05
- **Best individual**:

```
(x[0] * -0.0909) + (np.cos(x[1]) * 7.00) + 3.28
```

Dataset 5

- **MSE value**: 1.78446e-23
- **Best individual**:

```
(x[0] ** x[1] - 16.2) * (-1.00e-11)
```

Dataset 6

- **MSE value:** 2.61502e-05
- **Best individual**:

```
((1.2832 * 1.3207 * x[1]) - 0.4743 * 1.464 * x[0])
```

Dataset 7

- **MSE value**: 34878.3
- **Best individual**:

```
4.44 ** (x[0] * x[1])
```

Dataset 8

- **MSE value**: 4.65604e+07
- **Best individual**:

```
(x[5]**2) * (x[5]**3) * 5.00 - 520
```