TEAM PROJECT 01 – SEARCH & NATURE-INSPIRED ALGORITHMS

CSC14003 – FUNDAMENTALS OF ARTIFICIAL INTELLIGENCE

1. PROJECT OVERVIEW

This project focuses on **implementing, analyzing, and comparing classical graph search algorithms and nature-inspired optimization algorithms** using only **NumPy** and basic Python for implementing data structures and optimization process from scratch.

The project is divided into two major algorithm families:

1. **Classical Search Algorithms on Graphs**
2. **Nature-Inspired Algorithms**, including:
    1. Evolution-based algorithms
    2. Physics-based algorithms
    3. Biology-based algorithms
    4. Human behavior-based algorithms

Through this project, students will understand how different search paradigms explore state spaces and optimization landscapes, ranging from deterministic graph traversal to stochastic population-based optimization..

2. LEARNING OBJECTIVES

By completing this project, students will:

- Understand the theoretical foundations of **graph search** and **metaheuristic optimization**.

- Distinguish between **exact search algorithms** and **approximate/metaheuristic methods**.
- Implement classical and nature-inspired algorithms **from scratch**.
- Analyze convergence behavior, exploration–exploitation trade-offs, and scalability.
- Apply different algorithm families to **both discrete and continuous problems**.
- Develop skills in visualization, experimentation, and scientific reporting.
- Collaborate effectively in a team-based AI project.

3. ALGORITHM TAXONOMY

Students must implement the following five swarm intelligence algorithms:

3.1 Classical Search Algorithms on Graphs

Category	Algorithms
Uninformed Search	Breadth-First Search (BFS), Depth-First Search (DFS), Uniform Cost Search (UCS)
Informed Search	Greedy Best-First Search, A* Search
Local Search	Hill Climbing (Steepest Ascent), Simulated Annealing

3.2 Nature-Inspired Algorithms

3.2.1 Evolution-Based Algorithms

Inspired by Darwinian evolution: selection, crossover, mutation.

Algorithm	Inspiration	Applications
Genetic Algorithm (GA)	Population evolution via crossover & mutation	Feature selection, scheduling

Differential Evolution (DE)	Vector differences for mutation	Continuous optimization
Evolution Strategies (ES)	Self-adaptive mutation	Engineering optimization

3.2.2 Physics-Based Algorithms

Inspired by physical laws and phenomena.

Algorithm	Inspiration	Applications
Simulated Annealing (SA)	Thermodynamic cooling	Combinatorial optimization
Gravitational Search Algorithm (GSA)	Newtonian gravity	Continuous optimization
Harmony Search (HS)	Musical harmony principles	Parameter tuning

3.2.3 Biology-Based Algorithms

Inspired by collective intelligence in biological systems.

Algorithm	Strengths	Weaknesses	Best Applications
Ant Colony Optimization (ACO) [1]	Effective for combinatorial problems and handles complex discrete spaces well	Computationally intensive and requires fine-tuning	Routing problems, scheduling, and resource allocation
Particle Swarm Optimization (PSO) [2]	Good for continuous optimization and	Can converge to local optima and is less effective for	Hyperparameter tuning, engineering

	simple and easy to implement	discrete problems	design, financial modeling
Artificial Bee Colony (ABC) [3]	Adaptable to large, dynamic problems and balanced exploration and exploitation	Computationally intensive and requires careful parameter tuning	Telecommunications, large-scale optimization, and high-dimensional spaces
Firefly Algorithm (FA) [4]	Excels in multimodal optimization and has strong global search ability	Sensitive to parameter settings and slower convergence	Image processing, engineering design, and multimodal optimization
Cuckoo Search (CS) [5]	Efficient for solving optimization problems and has strong exploration capabilities	May converge prematurely and performance depends on tuning	Scheduling, feature selection, and engineering applications

3.2.4 Human Behavior-Based Algorithms

Inspired by social and cognitive behaviors.

Algorithm	**Inspiration**	**Applications/ Notes**
Teaching–Learning-Based Optimization (TLBO)	Classroom learning	Parameter-free

Social Force Optimization (SFO)	Crowd dynamics	Continuous domains
Cultural Algorithm (CA)	Cultural evolution	Parameter tuning

Students are required to work in group for performing this project. You have to implement:

Category	**Algorithms**
Evolution-based	Genetic Algorithm (GA), Differential Evolution (DE)
Physics-based	Simulated Annealing (SA)
Biology-based	ACO, PSO, ABC, FA, CS
Human-based	TLBO (*optional, bonus*)

Then, compare against at least four traditional search algorithms: Breadth-First Search (BFS), Depth-First Search (DFS), A* Search, Hill Climbing, Simulated Annealing.

Implementation requirements:

- All algorithms must be implemented using **NumPy only** (no scikit-learn, scipy.optimize, or other high-level libraries)
- Code should be modular, well-documented, and follow Python best practices
- Each algorithm should have configurable parameters (population size, iterations, etc.)
- Implementations should handle both continuous and discrete optimization problems. Maybe you can choose suitable problem for each algorithm.

Students are required to create visualizations that demonstrate *convergence ability*, *comparative performance*, *parameter sensitivity analysis*, (advanced) 3D surface plots to show the objective function landscape (for continuous optimization problems). Recommended libraries for visualization: Matplotlib, and Seaborn.

The recommended metrics for comparison are:

- Convergence speed.
- Best / average solution quality
- Computational complexity (time and space).
- Robustness (mean ± std over multiple runs). (*You can use your knowledge in Statistic course to perform hypothesis testing, that would be great*.)
- Scalability (performance with problem size).
- Exploration vs exploitation behavior

For choosing test problems, students should report both on continuous and discrete problems. You can choose at least one for each type of problem as shown in the table below:

Type of problem	**Continuous Optimization**	**Discrete Optimization**
	Sphere function (*convex, unimodal*)	Traveling Salesman Problem (TSP) with **constraints time and cost**, *suitable for testing ACO, GA, SA*
	Rastrigin function (*highly multimodal*)	Knapsack Problem (KP) *suitable for testing GA, ABC*
	Rosenbrock function (*narrow valley*)	Graph coloring (GC) *suitable for testing GA, SA*

	Griewank function (*Regularly distributed minima*) Ackley function (*many local optima*)	Shortest Path, *suitable for testing BFS, DFS, A**

The report must fully give the following sections:

- Member information (student ID, full name, ...)
- Work assignment table, which includes detailed information on each task assigned to team members, along with the completion rate of each member compared to the assigned tasks
- Self-evaluation of the completion rate at each level of the project and other requirements.
- Detailed algorithm description for each level. Although a higher level may encompass the previous ones, presenting the information by levels will enhance clarity for the assessment and commentary process. Illustrative images are encouraged.
- Describe the test cases and the results when run on each of those test cases.
- The report needs to be **well-formatted and exported to PDF**. Note that for editors like Jupyter notebook, the team needs to find a way to format it well before exporting it to PDF.
- If there are figures cut off by the page break, etc., points will be deducted.
- References (if any) with APA format.
- Language usage: **Vietnamese or English**

Recommended report structure as follow:

1. Chapter 1 - Introduction & Algorithm Taxonomy

2. Chapter 2 - Classical Graph Search Algorithms

3. Chapter 3 - Local Search & Physics-Based Algorithms

4. Chapter 4 - Evolution-Based Algorithms

5. Chapter 5 - Swarm & Biology-Based Algorithms

6. Chapter 6 - Human-Inspired Algorithms (if any)

7. Chapter 7 - Discussion & Insights

8. Chapter 8 - Conclusion & Future Work

For submission:

- Your report, source code and test cases must be contributed to the form of a compressed file (**must be .zip**) and named according to the format **<Group_ID>.zip**, for example: *Group_01.zip*.
- Demo videos (recording the running process of your program for each test case) should be uploaded to YouTube or Google Drive, and the public URLs are included in the report.
- If the compressed file is larger than **25MB**, prioritize compressing the report and source code. Test cases may be uploaded to Google Drive and shared via a link.

4. ASSESSMENT

No	Description	Details	Scores
1	Technical Report	1.1 Algorithm descriptions with mathematical formulations 1.2 Implementation details and design decisions 1.3 Experimental methodology and test	40%

		configurations 1.4 Results analysis with tables and charts 1.5 Comparative analysis and discussion 1.6 Conclusions and insights learned 1.7 Minimum 25 pages	
2	Source code	2.1 Well-structured Python implementation of all swarm algorithms 2.2 Implementation of traditional search algorithms for comparison 2.3 README with setup instructions and usage examples. And uploaded it on Github.	40%
3	Demo	Short video demo uploaded on Youtube at least 5 minutes.	20%

5. NOTICES

Please pay attention to the following notices:

- This is a GROUP assignment. Each group has about 3 to 4 members.
- Duration: **about 4 weeks**.
- Any plagiarism, any tricks, or any lie will have a 0 point for the course grade.

6. REFERENCES

[1] Dorigo, M., Birattari, M., & Stutzle, T. (2007). Ant colony optimization. *IEEE computational intelligence magazine*, *1*(4), 28-39.

[2] Wang, D., Tan, D., & Liu, L. (2018). Particle swarm optimization algorithm: an overview. *Soft computing*, *22*(2), 387-408.

[3] Karaboga, D., & Basturk, B. (2007). A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *Journal of global optimization*, *39*(3), 459-471.

[4] Yang, X. S., & He, X. (2013). Firefly algorithm: recent advances and applications. International journal of swarm intelligence, 1(1), 36-50.

[5] Yang, X. S., & Deb, S. (2014). Cuckoo search: recent advances and applications. *Neural Computing and applications*, *24*(1), 169-174.