AI430 COMPUTATIONAL INTELLIGENCE SPRING 2025

COURSE PROJECT INSTRUCTIONS

Instructions to Students:

- This is a group work project. Each group consists of **Five to Six students** (*the Teaching Assistant must approve group members through registration)*. Each group must develop the idea assigned to them using Python.
    - **Project Objectives:** The objectives of this project can be summarized as applying the main ideas, fundamental concepts, and basic algorithms in the field of Evolutionary Algorithms & Computing.
- **Submission:** Submission is done according to the following schedule:
    - **Week 13: Submission and Discussion of the** (1) **Project** and (2) **Documentation**. *The report should include the following: (1) Project idea in detail, (2) Main functionalities, (3) Similar applications in the market, (4) A literature review of Academic publications (papers) relevant to the idea (at least 4 to 6 papers, as per the number of team members), (5) the Dataset employed (preferably a publicly available dataset), (6) Details of the algorithm(s)/approach(es) used and the results of the experiments, and (7) Development platform.*
- **Assessment:** Assessment will be on the reports, code submitted, and discussions with team members. All the team members must contribute to all the phases, and the role of each member must be clearly stated in each report.
    - The Project will be assessed based on the following criteria:
        - The complexity of the problem, & the correctness of the algorithms employed.
        - The quality/comprehensiveness of your experiments & documentation.
        - The correctness of your analysis and design diagrams.
        - Implementation correctness.
- **Feedback:** If requested, further details and feedback could be provided for each group through discussions with the teaching assistant(s) during the weekly-labs/office-hours.
- You can only submit your work. Any student suspected of plagiarism will be subject to the procedures set out by the Faculty/University (including failing the course entirely).
    - **Academic Integrity:** The University's policies on academic integrity will be enforced on students who violate University standards of academic integrity. Examples of behaviour that is not allowed are:
        - Copying all or part of someone else's work and submitting it as your own;
        - Giving another student in the class a copy of your work and
        - Copying parts from the internet, textbooks, etc.
        - If you have any questions concerning what is allowed, please don't hesitate to discuss them with me.
    - **N.B., We understand that you might positively refer to AI tools to support your research. Please note that, in case there are any concerns about AI-generated submission content, you will be invited for an opportunity to verify and defend your work.**

AI430 Computational Intelligence Spring 2025

Course Project Descriptions (10 Ideas)

Table of Contents

[1] Capstone-Driven Optimisation via One CI Algorithm Family 4
[2] Eco-Route Planner for Emergency Services 7
[3] Job Scheduling in Cloud Environments 8
[4] Clustering-Based Customer Segmentation with CI/EC Algorithms 9
[5] Dimensionality Reduction for Data Visualisation Using Nature-Inspired Algorithms 10
[6] Hybrid Evolutionary Portfolio Optimiser 11
[7] Smart Irrigation Scheduler Using CI and EC Strategies 12
[8] Nature-Inspired Pattern Recognition for Classification Problems 13
[9] Intelligent Resource Allocation for Disaster Relief 14
[10] Heterogeneous Island-Model Evolution for Multimodal Benchmark Optimisation 15

Important Guidelines for ALL Projects to follow:

a) At least five varying CI approaches are expected to be implemented for teams of 5 members, 6 approaches for teams of 6 members, and 7 approaches for teams of 7 members.
b) Clearly define and formalise your problem as one of the problem types we've studied throughout the EA module: Optimisation, Modelling, Simulation, Constraint Satisfaction, Free Optimisation, Constrained Optimisation, etc.
c) If your selected idea mandates Constraint Handling, clearly implement an approach to handling constraints (i.e., Penalty Functions, Repair Functions, Restricting Search to the Feasible Region, Decoder Functions, etc.).
d) If your selected idea mandates Coevolution, clearly implement a coevolutionary approach (cooperative or competitive).
e) Clearly define the Components of your CI algorithm: For instance, in the case of a genetic algorithm, clearly define the Representation (Definition of Individuals), Evaluation Function (Fitness Function), Population, Parent Selection Mechanism, Variation Operators (Mutation and Recombination), Survivor Selection Mechanism (Replacement), Initialisation, and Termination Condition(s).
f) Study the effects of varying settings for your chosen algorithms. For instance, for a GA try out [multiple parent selection techniques (each independently) / multiple recombination techniques (each independently) / multiple mutation techniques (each independently) / multiple population-management-models/survivor-selection (each independently)] and report the results for each.
g) Clearly describe and implement approaches to control/tune the parameters.
h) Describe and implement a suitable approach for preserving diversity.
i) Incorporate a functional user interface demonstrating the algorithm, parameters, inputs, and results.
j) The students may be awarded bonus marks in the following cases (only if the experiments are carried out properly and the results/performance were measured, reported, and analysed adequately:
    - Contributions (experiments and results) with high publication potential.
    - Hybrid approaches (employing together more than a single EAs/SI approach).
    - Employing SOTA novel variants of the EAs/SI approaches rather than the traditional implementations.
k) Report the results (independently) for each setting (e.g., a choice of a mutation operator, a recombination operator, a representation, a parent selection approach, a survivor selection approach, an initialisation, an approach for preserving diversity, .. etc.).
    - An experiment should be carried out multiple times (optimally, 30 runs per setting).
    - The list of seeds (used to initialise the random number generator before each run) should be stored & provided.

[1] Capstone-Driven Optimisation via One CI Algorithm Family

Problem Description

- Choose your own optimisation problem. Teams pick a practical sub-problem from their Graduation Project (or a standard benchmark like scheduling, routing, feature selection).
- This flexibility lets you work on something you care about while still exploring advanced CI techniques.

Algorithm Family & Variants: Select one sub-category of Computational Intelligence and implement at least five, six, or seven of its variants or extensions (based on your team size). For example:

- Differential Evolution (DE):
  - Differential Evolution Algorithm with Strategy Adaptation (SaDE),
  - Self-adapting Control Parameters in Differential Evolution (jDE),
  - Success-History-Based Parameter Adaptation for Differential Evolution (SHADE),
  - Improving the Search Performance of SHADE Using Linear Population Size Reduction (LSHADE),
  - An Ensemble Sinusoidal Parameter Adaptation Incorporated with L-SHADE (LSHADE-EpSin),
  - Ensemble Sinusoidal Differential Covariance Matrix Adaptation with Euclidean Neighbourhood (LSHADE-cnEpSin),
  - LSHADE with Semi-parameter Adaptation Hybrid with CMA-ES (LSHADE-SPACMA),
  - Enhanced LSHADE-SPACMA Algorithm (ELSHADE-SPACMA).
- Ant Colony Optimisation (ACO):
  - Ant system,
  - Ant colony system,
  - Elitist ant system,
  - Max-min ant system,
  - Rank-based ant system,
  - Parallel ant colony optimisation,
  - Continuous orthogonal ant colony,
  - Recursive ant colony optimisation.

- The set of algorithms inspired by different honeybees' behavioural patterns:
    - Artificial bee colony "ABC" algorithm,
    - Honeybees mating optimisation "HBMO" algorithm,
    - Artificial beehive algorithm "ABHA",
    - Bee colony optimisation "BCO" algorithm,
    - Bee colony inspired algorithm "BCiA",
    - Bee swarm optimisation "BSO" algorithm,
    - Bee system "BS" algorithm,
    - BeeHive algorithm,
    - Bees algorithm "BeA",
    - Bees life algorithm "BLA",
    - Bumblebees algorithm,
    - Honeybee social foraging "HBSF" algorithm,
    - OptBees algorithm,
    - Simulated bee colony "SBC" algorithm,
    - Virtual bees algorithm "VBA",
    - Wasp swarm optimisation "WSO" algorithm.
- Immune Algorithms:
    - Clonal Selection Algorithm,
    - Negative Selection Algorithm,
    - Artificial Immune Recognition System,
    - Immune Network Algorithm,
    - Dendritic Cell Algorithm.

N.B., Pick one family only (whether from the algorithms mentioned above or not); your novelty will come from how you compare, hybridise or adapt these variants.

Implementation Guidelines

1. **Problem Formalisation**
    - Write the objective function clearly.

- List any constraints.
- Choose or generate any data needed (public dataset or synthetic).

2. Experimental Setup

- Runs & Seeds: Perform at least 30 independent runs per variant, storing random seeds.
- Test Cases: Use multiple problem instances (e.g., different sizes or dynamic scenarios).

3. Performance Analysis

- Collect metrics: best-found value, convergence speed (iterations to reach a threshold), computational time.
- Use relevant statistical tests to compare variants rigorously.

4. Novelty Pathway

- Hybridisation: Combine two variant operators (e.g., jDE mutation + SHADE memory).
- Adaptive Control: Design a rule that changes parameters (e.g., mutation factor or pheromone decay) during the run based on feedback (e.g., population diversity).
- New Operator: Propose one small change to a variant (e.g., a novel recombination method) and test its impact.

5. Deliverables

- Code Repository: Well-documented, modular Python code.
- Written Report:
  - Problem definition and mathematical model.
  - Description of each variant and your novelty.
  - Experimental results with tables/plots.
  - Statistical analysis and discussion of which variant or hybrid works best and why.

[2] Eco-Route Planner for Emergency Services

Guidelines:

- Research existing routing methods and emergency response needs.
- Define an objective to minimize travel time while considering road conditions.
- Develop and compare multiple algorithms to optimise route segments.
- Simulate routes using map data (real or synthesized).
- Compare performance against a simple baseline (e.g., shortest distance).

Detailed Description:

Context and Problem Statement: Emergency responders (such as ambulances or fire trucks) need the fastest route to an emergency. This project will explore whether CI algorithms can plan routes that account for variables like road congestion or blocked roads.

Key Terms and Concepts:

- **Fitness Function:** A measure to evaluate how good a route is (e.g., time, distance, or safety).

Requirements/Deliverables:

- A Python implementation of a route-planning algorithm using CI algorithms.
- Simulation of multiple routes on a mapped network.
- Comparative performance analysis (e.g., average travel time).
- A clear report with diagrams, simulation screenshots, and discussion.

[3] Job Scheduling in Cloud Environments

Guidelines:

- Study CI methods and basic cloud scheduling.
- Define scheduling criteria (e.g., balancing computational load and minimising wait time).
- Develop a simulation of task queues and available cloud nodes.
- Apply CI algorithms to optimise task assignments.
- Evaluate and compare simulation results with standard scheduling approaches.

Detailed Description:

Context and Problem Statement:
With the growth of cloud computing, managing how tasks are assigned to different servers is critical. This project applies CI approaches to improving job scheduling.

Key Terms and Concepts:

- **Job Scheduling:** The process of assigning jobs to resources (computers) efficiently.

Requirements/Deliverables:

- A Python-based simulation of a cloud environment and workload.
- Implementation of CI algorithms to assign tasks.
- Comparative data on scheduling efficiency (e.g., response times).
- Documentation including simulation parameters, graphs, and analysis.

[4] Clustering-Based Customer Segmentation with CI/EC Algorithms

Guidelines:

- Collect or simulate a simple customer dataset with numerical features.
- Review basic clustering methods and understand the role of feature selection.
- Implement CI/EC Algorithms that evolve cluster centres.
- Compare results with standard techniques like k-means.
- Present visualisations (scatter plots, cluster boundaries).

Detailed Description:

Context and Problem Statement:
Businesses often need to segment their customers into groups for targeted marketing. Using CI/EC Algorithms to optimise cluster centres may provide novel insights.

Key Terms and Concepts:

- **Clustering:** Grouping data points based on similarity.

Requirements/Deliverables:

- A dataset (or synthetic data) representing customer characteristics.
- Python code implementing CI/EC algorithms to determine optimal cluster centroids.
- Comparative analysis between EC-based segmentation and traditional clustering.
- Visual outputs such as graphs and cluster diagrams, plus a written report.

[5] Dimensionality Reduction for Data Visualisation Using Nature-Inspired Algorithms

Guidelines:

- Study methods of dimensionality reduction (e.g., PCA, t-SNE) and their limitations.
- Research simple nature-based algorithms (e.g., SOM) to create lower-dimensional representations.
- Use a high-dimensional public dataset (e.g., from UCI repositories).
- Implement the algorithms to reduce dimensions and visualise data.
- Compare visual outputs with traditional methods.

Detailed Description:

Context and Problem Statement:
High-dimensional datasets can be difficult to interpret. This project applies nature-inspired algorithms to transform data into 2D or 3D space for easier visualisation, offering a potential novel contribution in data interpretation.

Key Terms and Concepts:

- **Dimensionality Reduction**: Techniques that simplify data by reducing the number of variables.
- **Self-Organizing Map (SOM):** A neural-network-based approach that visualises high-dimensional data in a low-dimensional space.

Requirements/Deliverables:

- Implement a nature-inspired dimensionality reduction method in Python.
- Use a public dataset (e.g., from UCI) and compare with PCA/t-SNE results.
- Provide visual outputs (scatter plots) and performance metrics.
- A detailed report discussing methodology, visual improvements, and feasibility.

[6] Hybrid Evolutionary Portfolio Optimiser

Guidelines:

- Introduce the basic concepts of financial portfolio optimisation (risk vs. return).
- Combine multiple CI/EC methods to select an optimal portfolio.
- Use simulated market data or historical price data (open source).
- Establish and test objective functions (maximise return, minimise risk).
- Compare the hybrid approach with standard methods (like Modern Portfolio Theory).

Detailed Description:

Context and Problem Statement:
Investors seek to balance risk and reward. The project aims to produce a novel, robust method for portfolio optimisation by hybridising CI and evolutionary strategies.

Key Terms and Concepts:

- **Portfolio Optimisation:** The process of choosing investments to balance risk and returns.

Requirements/Deliverables:

- A Python program that combines multiple CI/EC approaches for portfolio selection.
- Simulation experiments using either synthetic or historical data.
- Analysis of risk and return metrics and a detailed comparison with standard approaches.
- Visualisations and comprehensive documentation.

[7] Smart Irrigation Scheduler Using CI and EC Strategies

Guidelines:

- Define a simulated field with varying water needs for crops.
- Develop an objective function that minimises water consumption while maximising crop yield.
- Implement CI/EC algorithms to adjust irrigation schedules.
- Incorporate additional evolutionary operators to handle constraints like water availability.
- Run experiments and compare different parameter settings.

Detailed Description:

Context and Problem Statement:
Water is a critical resource in agriculture. This project aims to optimise irrigation schedules by taking inspiration from the collective behaviour of swarms and evolutionary strategies, potentially reducing water waste.

Key Terms and Concepts:

- **Irrigation Scheduling:** Planning the timing and amount of water delivery to fields.

Requirements/Deliverables:

- A simulation in Python to model a field and water usage.
- An objective function that evaluates water usage vs. crop yield.
- Comparisons of CI/EC results with simple heuristics.
- A written report with graphs, parameter studies, and visual simulations.

[8] Nature-Inspired Pattern Recognition for Classification Problems

Guidelines:

- Research nature-based optimisation methods (e.g., Bacterial Foraging Optimisation) to tune a simple classifier's parameters.
- Implement a baseline neural network or support vector machine for classification.
- Apply nature-inspired algorithms for parameter tuning.
- Compare results with traditional parameter tuning methods.

Detailed Description:

Context and Problem Statement:
Choosing a classification problem that is a standard benchmark in machine learning. Explore whether a natural process-inspired algorithm can successfully optimise the parameters of a simple classifier to improve accuracy.

Key Terms and Concepts:

- **Bacterial Foraging Optimisation (BFO):** An algorithm inspired by how bacteria move and swarm to find nutrients, applied here for optimising parameters.
- **Classification:** The task of assigning labels to input data.

Requirements/Deliverables:

- A Python-based classifier trained on a publicly available classification dataset.
- Implementation of CI/EC algorithms to adjust classifier parameters.
- A detailed comparison of performance metrics (accuracy, training time) against standard methods.
- Visualisations of learning curves and error rates.

[9] Intelligent Resource Allocation for Disaster Relief

Guidelines:

- Define a simulated scenario where relief resources (food, medicine, etc.) must be distributed.
- Develop an objective function that balances delivery speed and resource utilisation.
- Use a hybrid algorithm combining natural-process-inspired approaches with traditional evolutionary methods.
- Simulate various disaster conditions (varying demand patterns) and test the allocation.
- Analyse robustness and efficiency based on key performance measures.

Detailed Description:

Context and Problem Statement:
In disaster relief efforts, making quick, effective decisions on resource allocation can save lives. This project explores advanced optimisation techniques inspired by natural processes to allocate scarce resources efficiently.

Key Terms and Concepts:

- **Resource Allocation:** Distributing available resources to meet demands optimally.

Requirements/Deliverables:

- A Python simulation that models disaster scenarios and resource constraints.
- Implementation of hybrid EC/CI algorithms with clearly defined fitness functions.
- Experimental results showing how allocation efficiency varies with different scenarios.
- A detailed final report with performance comparisons, visual maps of resource distribution, and insights on parameter tuning.

[10] Heterogeneous Island-Model Evolution for Multimodal Benchmark Optimisation

Problem Description: Many real-world optimisation tasks feature rugged, multimodal landscapes where different search strategies excel in different regions. Using a coarse-grain (island-model) EA, each "island" runs a distinct algorithm—rather than just varying operators—to foster complementary exploration and exploitation. This project tackles high-dimensional, multimodal benchmark functions to study how heterogeneous islands interact and improve global performance.

Implementation Steps:

1. **Problem Formalisation:** Select 3–5 standard multimodal functions.
2. **Island-Model Framework**
   - Build a Python framework supporting multiple sub-populations ("islands") that:
     - Run in parallel.
     - Exchange migrants every T generations.
   - Choose a migration topology (e.g., ring or random graph) and policy (best-only, random, or diversity-based migrants).
3. **Algorithm Implementations**
   - Code modular versions of varying CI algorithms so they can plug into the island framework.
   - Ensure all islands share the same interface for population size, fitness evaluation, and migration.
4. **Experimental Protocol**
   - **Runs & Seeds**: Perform ≥30 independent runs for each configuration (heterogeneous vs. homogeneous islands vs. single-algorithm baseline).
   - **Configurations to Compare:**
     1. Heterogeneous islands (for instance, GA+CA+MA).
     2. Homogeneous islands (for instance, all GA, all DE, all PSO).
     3. Single-algorithm non-island runs.
   - **Metrics:** Best-found fitness, convergence speed, inter-island diversity, computational cost.
   - Perform relevant statistical tests to compare configurations.

Deliverables:

- **Codebase:** Modular Python implementation with clear documentation on adding new islands or algorithms.
- **Report:**
  - Formal problem definition and island-model design.
  - Detailed descriptions of the varying CI islands.
  - Experimental results with tables/plots comparing all configurations.
  - Statistical analysis demonstrating the benefits (or trade-offs) of heterogeneity.
  - Discussion of adaptive migration or topology strategies and their impact.