COMPUTATIONAL INTELLIGENCE FOR OPTIMIZATION

24/25 PROJECT STATEMENT

Contact:

Inês Magessi

imagessi@novaimsunl.pt

Project Statement

The goal of this project is to apply Genetic Algorithms to solve an optimization problem. You can choose from one of the optimization problems suggested in this statement or you can propose your own. If you decide to propose your own problem, the idea must be approved. For that, you must write a descriptive email proposing the optimization problem and send it to imagessi@novaims.unl.pt until the **4th of April**.

It is highly recommended that you use the code base implementation done during the practical classes and extend that code to fit your optimization problem. However, if you prefer to do all the code from scratch, you're also free to do so.

Optimization Problem Ideas

1. Wedding Seating Optimization

The goal is to optimize the seating arrangement of **guests across tables** to maximize guest happiness and minimize conflicts based on their relationships.

Each solution represents a complete seating arrangement, specifying which guests are seated at each table. These are the constraints that **must** be verified in every solution of the search space (no object is considered a solution if it doesn't comply with these):

- Each guest must be assigned to exactly one table.
- Table placement matters only in terms of grouping. **The arrangement within a table is irrelevant**.

Impossible Configurations: Any arrangement where a guest is seated at multiple tables or left unassigned is not part of the search space and is considered impossible. It is forbidden to generate such an arrangement during evolution.

Here you can find a pairwise relationship matrix of all **64 attendees**, who need to be distributed across **8 tables**. You can also consult the chart of the guests and their relationships, as well as a table explaining relationship values.

The goal is to maximize the overall relationship score by optimizing guest seatings within each table and then aggregating these scores across all tables.

2. Sports League Optimization

In a fantasy sports league, the objective is to assign **players to teams** in a way that ensures a balanced distribution of talent while staying within salary caps.

Each player is defined by the following attributes:

- Skill rating: Represents the player's ability.
- Cost: The player's salary.
- Position: One of four roles: Goalkeeper (GK), Defender (DEF), Midfielder (MID), or Forward (FWD).

A solution is a complete league configuration, specifying the team assignment for each player. These are the constraints that **must** be verified in every solution of the search space (no object is considered a solution if it doesn't comply with these):

- Each team must consist of: 1 Goalkeeper, 2 Defenders, 2 Midfielders and 2 Forwards.
- Each player is assigned to exactly one team.

Impossible Configurations: Teams that do not follow this exact structure (e.g., a team with 2 goalkeepers, or a team where the same defender is assigned twice) are not part of the search space and are not considered solutions. It is forbidden to generate such an arrangement during evolution.

Besides that, each team should not exceed a **750€ million** total budget. If it does, it is not a valid solution and the fitness value should reflect that.

The objective is to create a balanced league that complies with the constraints. A balanced league a is a league where the average skill rating of the players is roughly the same among the teams. This can be measured by the standard deviation of the average skill rating of the teams.

Here you can find a dataset of players with their names, position, skill rating and salary (in million €). These players should be distributed across **5 teams** of **7 players** each.

3. Music Festival Lineup Optimization

The objective is to design the optimal festival lineup by scheduling artists across **stages and time slots** while maximizing **prime slot popularity**, ensuring **genre diversity** among stages and minimizing **fan conflicts** at each time slot.

An artist is characterized by a popularity score (from 0 to 100), a genre, and fan base conflicts with other artists (score from 0 to 1). Conflicts arise when artists with overlapping fan bases perform at the same time but on different stages, which can negatively impact attendance.

Each individual represents a festival lineup, specifying which artist plays on which stage and at which time slot. These are the constraints that **must** be verified in every solution of the search space (no object is considered a solution if it doesn't comply with these):

- Each artist is assigned to exactly one stage and slot
- All time slots start and end at the same time.
- All stages have the same amount of slots

Impossible Configurations: Any lineup where an artist is assigned to multiple time slots or left unassigned is not part of the search space and is not considered a solution. It is forbidden to generate such an arrangement during evolution.

The quality of a festival lineup is determined by balancing **three equally important objectives**, each contributing to an overall score. Since these objectives have different scales, they must be **normalized** to the same range (between 0 and 1) to ensure they contribute equally to the final fitness score. These are the objectives:

- **Prime Slot Popularity**: The most popular artists should be scheduled in the prime slots (the last time slot on each stage). This score is calculated by normalizing the total popularity of artists in prime slots against the maximum possible total popularity (e.g. if only most popular artists - score 100 - were scheduled on the prime slot).
- **Genre Diversity**: A diverse distribution of genres among stages in each time slot improves the festival experience. This score is determined by normalizing the number of unique genres in each slot relative to the maximum possible number of unique genres (e.g. if only different genres were scheduled in that time slot) . Then you take the average among time slots.
- **Conflict Penalty**: Fan conflicts occur when artists with overlapping audiences perform at the same time on different stages. This score is calculated by normalizing the total conflict value in each slot against the worst possible conflict scenario (e.g. where all artists with maximum conflict are scheduled together). Then you take the average among time slots. Since conflicts negatively impact the lineup, this is a **penalization** score.

Here you can find the dataset of artists, and here the pairwise matrix of conflict scores. These artists should be distributed across **5 stages** with **7 time slots each**,

4. Your Own Idea

If you want to suggest your own optimization problem, you are free to do so! The difficulty should roughly match the suggested examples in terms of constraints and objectives. For inspiration, consider topics like:

- **Event Scheduling** (e.g., conference talks).
- **Course Allocation** (e.g., balancing class sizes).
- **Warehouse Logistics** (e.g., optimizing item placement).
- **Travel Route Optimization** (e.g., delivery routes).
- **Resource Allocation** (e.g., assigning limited resources to maximize efficiency).
- **Job Scheduling** (e.g., minimizing delays or optimizing task order).

Ensure your problem has clear constraints, objectives, and a fitness function.

Group rules

This is a group project, as such you are encouraged to do the project in a group. The maximum number of members for a group is 4. Groups of 3-4 people are encouraged, but not mandatory. If you are having trouble finding a group, please send an email. If you can

reasonably justify that you are unable to do the project in a group, send an email and you can only proceed after receiving a prior authorization.

To enrol in the project, you should choose a group in moodle. You must be a member of a group by the **4th of April** to participate in the project. For any students that are not enrolled in a group by the 4th of April, it will be assumed that those students have opted to not participate in the project and **will receive an automatic grade of 0**.

NOTE: You can pass the module without undertaking the project, as it only counts for 35% of your grade. However, this is not encouraged as you will forfeit 35% of your grade.

Deliverables

- Code implementation (github repo)
- Report (pdf)

The project code must be delivered via a GitHub repository hosted on one of the group members' GitHub accounts. All projects will be cloned precisely at 23:56, one minute after the submission deadline. Any commits or changes made to the repository after the deadline will not be considered for evaluation.

Upload your report to Moodle before the deadline with the **link to the git repository** on the **first page**. Do not forget to check if the link on the report is really working. If your repository is private and it is not possible to clone it, it will not be possible to grade your project and you will receive 0 points. If you're not familiar with using Git or unsure how to submit your code via GitHub, feel free to ask during the practical classes or by email.

The report should be delivered as a PDF. Your report should not exceed **5 pages of text**. The cover page and any appendix with images/tables does not count for the page limit. In this module we trust you enough to allow you to choose font and font size yourself. Just keep it easy to read and relatively professional.

Deadline

The report must be submitted via moodle until the **23rd of May at 23:55**. Given the possibility of passing this course without undertaking the project, there is no ability to submit projects late, as any student who has not delivered theirs by the deadline will be considered as having opted to pursue solely the exam. Thereby, forfeiting 35% of their grade.

Oral Defense

There will be a **mandatory oral defense** during the week of **June 2nd to June 6th**. Groups will have the opportunity to select a time slot from a predefined schedule that will be shared closer to that week.

You are **not required to prepare a presentation** for the defense. However, **all group members must be present**, as each student will be asked questions regarding various aspects of the project, including code implementation, experimental setup, and results.

In cases where there are significant inconsistencies between the submitted code, the report, and the oral defense (or if any concerns about plagiarism arise) the defense may be extended, with the additional presence of the lecturer responsible for the theoretical component of the course.

Code Requirements

For your project to be successful you must implement:

- ➔ the fitness function
- ➔ at least 3 mutation operators
- ➔ at least 2 crossover operators
- ➔ at least 2 selection mechanisms

All of these must be different from the ones implemented in class, or at least adapted to your chosen optimization problem, and should respect the problem's constraints.

Additionally, you may also choose to compare Genetic Algorithms (GA) with other optimization techniques, such as hill climbing, simulated annealing etc. While this is entirely optional and not required for the project, it could add value to your final analysis. If you have the time and curiosity, this comparison might deepen your understanding by highlighting the strengths and weaknesses of different approaches, enriching your overall conclusions.

All code should be commented and well-documented.

Some general recommendations: To keep your code organized and maintain clarity, we recommend writing the core logic (such as classes, functions etc.) in separate Python files. Jupyter notebooks are preferred for tasks like performing evaluations, and reporting results, although you may also use Python files for these purposes if you prefer. If you choose to build on the library developed during the practical classes, you should import its code into your own Python files rather than copying it directly. While none of these are mandatory, following these suggestions will help improve the modularity and readability of your project, aspects that will be taken into account during the evaluation.

Report Requirements

Your report should explain your approach and include the following elements:

- **Formal Definition of the Problem:** Clearly define the optimization problem, describe how an individual is represented, define the search space and explain the fitness function.
- **Detailed Description of Implemented Selection and Genetic Operators:** Description of selection mechanisms, mutation and crossover operators. Explanations accompanied by useful illustrations are a plus.

- **Performance Analysis:** Comparison of different implementations and analysis of how they affect the performance.
- **Justification of Decisions**
    - Why did you choose this representation?
    - How did you design the fitness function?
    - Which configurations performed best? How many did you test? How did you measure success?
    - How do different operators influence GA convergence?
    - Have you implemented elitism? What impact does it have?
    - Did your approach yield good results? What improvements could be made?

A strong report should be clear, structured, and easy to follow, with meaningful illustrations where applicable. Your analysis should be thorough and well-supported by results data.

Evaluation Criteria

- 25% Code Functionality: Assessment of the functionality of the code, the correct implementation of the GA techniques and interaction of the various components. While you will work on your projects in a group, you should ensure code is well-structured and works well in conjunction.
- 10% Code Structure: Assessment of the code quality and structure, the code should be clear and well-commented to be inspectable and understandable by anyone.
- 40% Report Content: Assessment of the contents of the report. The report should illustrate your project and not contain vast literature reviews (this is a report not an essay). A good report will justify choices undertaken in the project as well as demonstrate findings and results.
- 25% Project Defense: Assessment of the project defense. All group members must join the defense and present knowledge about every aspect of the project.

*Keep in mind that we have no tolerance for plagiarism. In NOVA IMS, plagiarism is punished with the failure of both Epoch 1 and Epoch 2 of the exam (failure of the whole course). Reports will also be checked against projects from previous years for plagiarism.