Team name: KPMGComeGetUs

Name: Sng Hong Yao **Student ID:** 17205050
Name: Ankish Raj Prajapati **Student ID:** 17202456
Name: Ronan Mascarenhas **Student ID:** 17379773

NOTE

1. We are not too sure how you will be marking this. Therefore we did not write anything in main, but a launch JavaFX application.
2. The graph runs Simulated Annealing using everything required in this sprint (i.e. hard/soft constraints, random change to the random candidate solution, calculation for satisfaction and energy).
3. Do note that the graph may have some bugs (if it freezes (animating a line chart on different threads is new to us), just close the application and `./run.sh` again to play the graph again, until you get a graph like the one below), and you cannot specify parameters for the Simulated Annealing. The default parameters is:
    - Starting Temperature = 100.0
    - Cooling Rate = 0.001
    - Minimum Temperature = 0.01
    - Max Iterations = 10000

Setting up Environment

1. Open a terminal and navigate to the file where run.sh is.
2. Execute `./run.sh` (Uses maven to build the JAR, then Java to run the JAR)

Meeting Demands

Hard Constraints Implemented

- **Each student is assigned exactly one of their preferred projects.**
  [@src/main/java/ie/ucd/objects/Student.java, preferenceSatisfaction() method]
    - Check if the project assigned to the student is the student's preference list, if it is, we assign FIXED_PROFIT upfront, then deduct it based on the project's position/index in the preference list (i.e. FIXED_PROFIT - FIXED_COST * POSITION).

- **Each student is assigned projects with the same stream as the student.**
  [@src/main/java/ie/ucd/objects/Student.java, streamSatisfaction() method]
    - Check if the project and student have compatible streams, if so, assign FIXED_PROFIT , else FIXED_COST.

- **Each student is assigned exactly one project.** *[@src/main/java/ie/ucd/objects/Student.java, assignmentSatisfaction() method]*
    - Check if "projects assigned to the student"'s list size is equal to 1, if so assign FIXED_PROFIT, else FIXED_COST.

- **Each project is assigned to at most one student.**
  [@src/main/java/ie/ucd/objects/Project.java, assignmentSatisfaction() method]
    - Check if the number of students this project is assigned to is equal to 1, if so assign FIXED_PROFIT, else FIXED_COST.

Soft Constraints Implemented

- Projects are more-or-less equally distributed across supervisors.
  [@src/main/java/ie/ucd/objects/CandidateSolution.java, projectDistributionToSupervisorsSatisfaction() method]
    - For each staff member, we check if the number of research activities proposed is greater than the number of average projects proposed plus 2, if so add 1 to the number of violations. Once all calculated, we return the number of violations * FIXED_COST.
- Students with higher GPAs tend to get higher preferences than those with lower GPAs, or higher GPA means a greater chance of getting one's preferred projects.
  [@src/main/java/ie/ucd/objects/Student.java, gpaSatisfaction() method]
    - This basically means, if a student's GPA is high, and the project assigned is highly preferred by the student, then there is a higher profit.

Design and implement classes to do the above (assuming taking care of constraints, calculating satisfaction, and calculating energy)

- These requirements were implemented as functions/methods prior to this sprint.
- So, no classes were added to implement constraints, calculate satisfaction, or calculate energy.
- Constraints are calculated inside each student and each project, by calling calculateSatisfaction(). This also means calculateFitness(), depending on your preference.
- Since energy is only required and is only used by SimulatedAnnealing, a calculateEnergy() function was already implemented in that class at /src/main/java/ie/ucd/solvers/SimulatedAnnealing.java. For the purpose of this sprint, it is added to the CandidateSolution class for easier testing in this sprint.
- All-in-all, constraints and LocalSatisfaction are taken care of in Student.java and Project.java. GlobalSatisfaction by CandidateSolution.java. Energy calculation by SimulatedAnnealing.java.

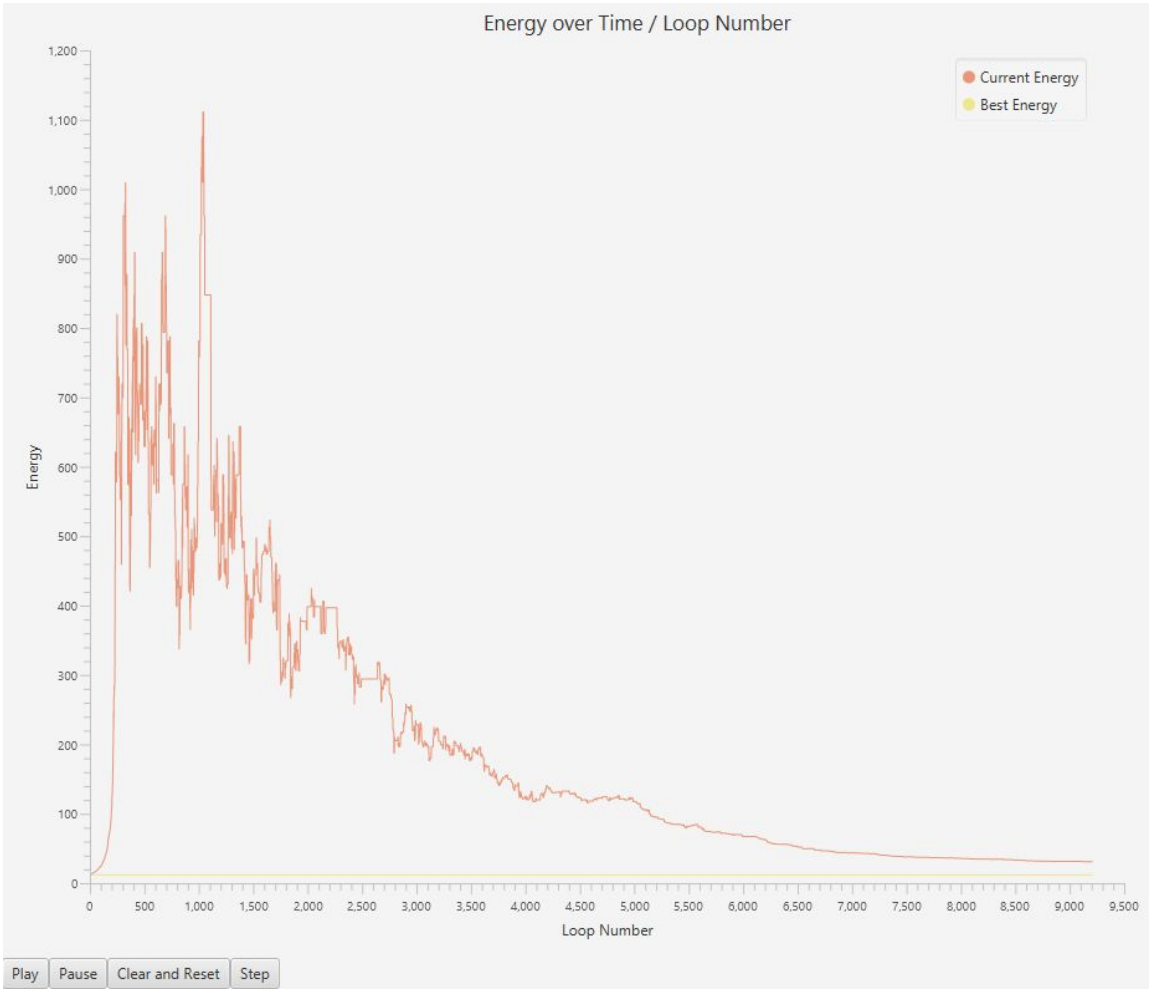
Provide "change" functionality that allows the system to introduce random changes into a given candidate solution

- This random change is similar to that which will be used in Simulated Annealing. Three variants of random change are already implemented in our SimulatedAnnealing.java. The most preferred out of the three is ported to CandidateSolution.java for easier testing.
- Our random change is picking two random students, then student 1 will be assigned a random project in student 2's preference list, and vice versa.

Determine when such a change makes things better or worse

- So we implemented a few random changes, three variants to be precise. The above was chosen because it is not too biased that it will make the calculated fitness too good to be true, at the same time, it is vague and random enough for getting a big jump in difference of fitness.
- Basically the above random change is chosen because its fitness range potential is good, and it works. The below shows Simulated Annealing running with this random change, at

the start, as expected, we accept high risk moves, as temperature lowers, we take more calculated risks to eventually arrive at a good enough global optimum.



Test your code so that your functions are indeed measuring what they are supposed to; the rest of the project depends crucially on them

- Run `mvn test`
- All related tests for this sprint are added in src/test/java/CandidateSolutionTest.java.
- Most importantly, we tested if our functions work if
    - if energy(X) > energy(Y), then fitness(X) < fitness(Y), or
    - if energy(X) < energy(Y), then fitness(X) > fitness(Y)