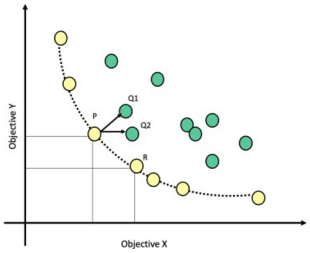
Help Khoury Assign TAs to Labs!
Resource Allocation with Evolutionary Computing

Prof. Rachlin / DS3500





Purpose

- Use evolutionary computing to solve an interesting resource allocation problem
- Help Khoury College faculty with managing TA / lab allocation for large classes
- Explore an important field of AI: Intelligent Decision-Support
- Practice functional programming and test-driven development (TDD)

Overview

Northeastern University's Khoury College offers several large classes supported by dozens of teaching assistants (TAs). These include Intro to Programming with Data, Discrete Structures, and Fundies. These courses have many lab practicums that need to be supported by the course TAs. Allocating TAs to labs is a difficult but interesting resource-allocation challenge. While a simple sign-up sheet might be a solution, the results are often sub-optimal. In general, our goal is to assign TAs to labs that are consistent with their availability and preferences while making sure we assign enough TAs to each lab. Performing this scheduling task by hand takes many hours of the instructor's time and it is unlikely that they are coming up with the best possible solution. Solving this problem would be a great service to Khoury college. In this homework assignment we take on the challenge!

Two tables of data have been provided:

sections.csv: A table of information about each lab/section. There are 17 sections numbered 0-16. The critical columns are min_ta and max_ta, the minimum and maximum number of TAs that need to be assigned to that section. The min/max values are determined by the number of students enrolled in that section.

tas.csv: A table of information about each of our 40 TAs, numbered 0-39, how many sections they are willing to support (max_assigned) and their availability and preferences for each section (U = Unavailable, W = Willing, P = Preferred).

Detailed Instructions

In this assignment, we will use evolutionary computing to assign TAs to specific recitation sections. You should be able to use the evo.py framework developed in class, but with a few enhancements.

a) Define functions for each objective (described below). Each function should be written in a functional style to the extent possible. Leverage the array-based processing capabilities of Pandas data frames and NumPy arrays. Avoid loops and assignments to the extent possible.

b) Implement one unit test for each objective function to verify that your objectives are working correctly on three test solutions provided. Here are their evaluation scores you should get if your objectives are working correctly. (See below for a description of each objective):

Objective	**Test1**	**Test2**	**Test3**
Overallocation	34	37	19
Conflicts	7	5	2
Undersupport	1	0	11
Unavailable	59	57	34
Unpreferred	10	16	17

c) Define agents that modify existing solutions and create new solutions. These can be simple or complex. It is entirely up to you. I recommend having some agents that try to address specific objectives. Code up at least *four* distinct agents.

d) Run your optimizer *for at most 5 minutes*. There is no limit on how many agent invocations you perform or how many solutions you produce. Your only limit is *time*. You will need to modify the evo framework so that the *evolve* method accepts a time limit parameter. (I recommend that your time limit be specified in total seconds, e.g., time_limit=300)

e) Report your final non-dominated set of solutions. You must output the solutions in table format in **exactly** the format specified in the deliverables section below. This requires a second evo framework enhancement – a *summarize* method that converts your population of solutions to a summary table.

Objectives

1. **Minimize overallocation of TAs (*overallocation*):** Each TA specifies how many labs they can support (max_assigned column in tas.csv). If a TA requests at most 2 labs and you assign to them 5 labs, that's an overallocation penalty of 3. Compute the objective by summing the overallocation penalty over all TAs. There is no minimum allocation.

2. **Minimize time conflicts (*conflicts*):** Minimize the number of TAs with one or more time conflicts. A time conflict occurs if you assign a TA to two labs meeting at the same time. If a TA has multiple time conflicts, still count that as one overall time conflict for that TA.

3. **Minimize Under-Support (*undersupport*):** If a section needs at least 3 TAs and you only assign 1, count that as 2 penalty points. Minimize the total penalty score across all sections. There is no penalty for assigning too many TAs. You can never have enough TAs.

4. **Minimize the number of times you allocate a TA to a section they are unavailable to support (*unavailable)***. You could argue this is really a *hard constraint,* but we will treat it as an objective to be minimized instead.

5. **Minimize the number of times you allocate a TA to a section where they said "willing" but not "preferred". (*unpreferred*).** In effect, we are trying to assign TAs to sections that they prefer. But we want to frame every objective a minimization objective. So, if your solution score has unwilling=0 and unpreferred=0, then all TAs are assigned to sections they prefer! Good job!

Deliverables

1. Submit four code files to gradescope. This includes your version of **evo.py**, **profiler.py**, **assignta.py**, and **test_assignta.py**.

2. Submit one summary table (**groupname_summary.csv**) of your final non-dominated Pareto-optimal solutions in comma-delimited value (csv) format. Each row is one *non-dominated Pareto-optimal* solution. Each column is the score for one objective. The SIX columns are: **groupname, overallocation, conflicts, undersupport, unavailable, unpreferred**. The group name can be whatever you like, but please limit to 8 characters, no spaces, and no special characters. The group name will, of course, be the same in every row. If you don't format your summary in **exactly** this format your team is disqualified from the competition! (By the way, this assignment is a friendly competition to see who can come up with the best solutions. I will be participating in the competition, and I hate to lose!)

3. Submit your PyTest unit test report (**groupname_pytest.txt**) to convince the TAs that your objectives are scoring your solutions correctly. (All tests should pass!)

4. Submit a profiling report (**groupname_profile.txt**) proving that your solutions were produced in five minutes or less. At minimum, you should profile your agents, your objectives, and the evo framework *evolve* method. (It is ok if the total run time is a fraction of a second over 300.)

Important Policy

It is the responsibility of your entire group to submit all seven files (4 code, 3 output) deliverables listed above. Forgetting to submit one of the deliverables will earn your group *zero credit* for that portion of the assignment. *All team members must verify the group submission!*