SLA 461 AI
SP 2023

Program 2 – Genetic Algorithm

For this program, you will use a genetic algorithm to attempt to build a space utilization schedule.

A client agency has been attempting to develop a better method of scheduling their activities. There are many competing factors in scheduling, so a single ideal schedule may not be possible. (This is a simplified version of the real problem.)

- All of these activities are scheduled for MWF, for 50 minutes, so we need only determine the time slot, room, and facilitator for each activity.
- The Sophisticated Learning Association (SLA) has several activities available and has full control of those rooms and can put activities into any time slot they wish. (Typically this is not the case in real world scheduling for potentially conflicting departments or agencies.)
- Each facilitator tends to work best overseeing a small group of activities, but the same activity isn't always provided by the same facilitator, and the same facilitator doesn't always teach the same activity. Thus, each activity has a list of a few facilitators who are preferred for overseeing that activity, and a few others who are able to "cross over" to that activity. Any activity can also be overseen by any facilitator, but this is a stopgap if **no other** facilitators are available.
  - You are given a list of all facilitators. Use this to **randomly assign activity assignments** and for **random substitution** (mutation). Use the preferred/other listing for scoring purposes, **not** for **selecting faculty**. (This keeps the generation/substitution process simple, and allows a broader exploration of the state space.)
- Each activities has an expected enrollment. Any activities should be in a room big enough to hold the expected enrollment. Likewise, a room more than 3 times the expected enrollment can be used, but is too big, and there is a smaller penalty in that case.
- There are 2 sections (A and B) of some activity.
- Your program will assign, for each activity:
  - Room
  - Time
  - Facilitator
- Initially, this assignment will be random. You'll create a population of random possible schedules (N >= 500) and then apply a genetic algorithm to improve it.

Fitness function:

- For each activity, fitness starts at 0.
- Activity is scheduled at the same time in the same room as another of the activities: **-0.5**
- Room size:
  - Activities is in a room too small for its expected enrollment**: -0.5**
  - Activities is in a room with capacity > 3 times expected enrollment**: -0.2**
  - Activities is in a room with capacity > 6 times expected enrollment**: -0.4**
  - Otherwise **+ 0.3**
- Activities is overseen by a preferred facilitator: **+ 0.5**
- Activities is overseen by another facilitator listed for that activity: **+0.2**
- Activities is overseen by some other facilitator: **-0.1**
- Facilitator load:
  - Activity facilitator is scheduled for only 1 activity in this time slot**: + 0.2**
  - Activity facilitator is scheduled for more than one activity at the same time**: - 0.2**
  - Facilitator is scheduled to oversee more than 4 activities total**: -0.5**

- Facilitator is scheduled to oversee 1 or 2 activities*: **-0.4**
  - Exception: Dr. Tyler is committee chair and has other demands on his time. *No penalty if he's ***only required to oversee*** < 2 activities.
- If any facilitator scheduled for *consecutive* time slots: Same rules as for SLA 191 and SLA 101 in consecutive time slots—see below.

Activity-specific adjustments:

- The 2 sections of SLA 101 are more than 4 hours apart: **+ 0.5**
- Both sections of SLA 101 are in the same time slot: **-0.5**
- The 2 sections of SLA 191 are more than 4 hours apart: **+ 0.5**
- Both sections of SLA 191 are in the same time slot: **-0.5**
- A section of SLA 191 and a section of SLA 101 are overseen in consecutive time slots (e.g., 10 AM & 11 AM): **+0.5**
  - In this case **only** (consecutive time slots), one of the activities is in Roman or Beach, and the other isn't: **-0.4**
    - It's fine if neither is in one of those buildings, of activity; we just want to avoid having consecutive activities being widely separated.
    -
- A section of SLA 191 and a section of SLA 101 are taught separated by 1 hour (e.g., 10 AM & 12:00 Noon): + 0.25
- A section of SLA 191 and a section of SLA 101 are taught in the same time slot: -0.25

Computing the fitness function:

- The fitness function for a schedule is the sum of the fitness of the individual activities in it (i.e. add up the scores for each activities using the above rubric, and sum the activities).
- Use **softmax*** normalization to convert fitness scores into a probability distribution.[1] ***(the link is an example of a library implementation for Python's SciPy but this is a common function in other libraries)**
- As you select pairs for reproduction, whether you produce 1 offspring per pairing or both possible offspring is an implementation decision.
  - Programming note: Access to the parent population is read-only; copy 2 parents, produce some offspring, add them to a next-generation pool. This ***can*** parallelize nicely.
- Start with a mutation rate of **0.01**(λ); if an item is selected for mutation, use random selection.
  - Note: This value might need to be adjusted (see note below$^{(\Delta)}$).
- Run **at least 100** generations (***G***). Continue until the improvement in average fitness for generation ***G*** is ***less than 1%*** improvement over generation G_{100}. Report the best fitness from the final generation and print the schedule to an output file. (Don't worry unduly about output formatting but please do try to make it ***legible***.)

  $^{(\Delta)}$Note on rate: Once your program is running, note the fitness you get, and then cut the mutation ($\lambda/2$) rate in half. As long as your results continue to improve, continue cutting the mutation rate in half until things appear to be stable.

1 Alternatively, rather than computing a probability distribution, you can keep your schedules in a minheap. The item at the top is the least-fit individual and is selected for removal. Select any 2 schedules at random from the heap as parents for a replacement. Find the fitness of the new offspring, put it at the top of the heap, and restore the heap property. This ensures we are always removing the least-fit individual. A more-fit individual will stay farther down in the heap longer, and thus have more opportunities to reproduce, than a lower-fitness individual which will find itself migrating to the top of the heap as breeding continues and the population gradually improves. In this case, a generation is N replacements, where N is the size of the population. (If you have a heap of size 500, then 500 remove/replace cycles constitute 1 generation.) This doesn't parallelize as nicely as the other method, but bypasses the need for normalization completely.

Other programming notes:

- Your chosen programing environment's core or library supplied random number generator should be fine for this project; there's not likely to be any significant improvement in performance when implementing any more complex generator. (The generators in early compilers did have some definite weaknesses, but almost all modern languages now use the Mersenne Twister or a cryptographic or hash-based RNG, more than strong enough for our purposes.)

Turn in:

- Your source code: Python, C++, C#, Java
- Sample output (screen captures / generated text)
- **A short report** (probably a page or so) in .doc/.docx/.pdf format, discussing:

i) What were the challenges in writing the program? Or did it seem to go smoothly from the beginning?

ii) What do you think of the schedule your program produced? Does it have anything that still looks odd or out of place?

iii) How would you improve the program, or change the fitness function?

iv) Anything else you feel like discussing, asking about, bragging about, etc.

Appendix: **10 Activities**, **7 Rooms**, and **6 Times**

Facilitators: Lock, Glen, Banks, Richards, Shaw, Singer, Uther, Tyler, Numen, Zeldin

SLA100A, SLA100B:
Expected enrollment: 50
Preferred facilitators: Glen, Lock, Banks, Zeldin Other
facilitators: Numen, Richards

SLA191A, SLA191B:
Expected enrollment: 50
Preferred facilitators: Glen, Lock, Banks, Zeldin Other
facilitators: Numen, Richards

SLA201:
Expected enrollment: 50
Preferred facilitators: Glen, Banks, Zeldin, Shaw Other
facilitators: Numen, Richards, Singer

SLA291:
Expected enrollment: 50
Preferred facilitators: Lock, Banks, Zeldin, Singer
Other facilitators: Numen, Richards, Shaw, Tyler

SLA303:
Expected enrollment: 60
Preferred facilitators: Glen, Zeldin, Banks Other
facilitators: Numen, Singer, Shaw

SLA304:
Expected enrollment: 25
Preferred facilitators: Glen, Banks, Tyler
Other facilitators: Numen, Singer, Shaw, Richards, Uther, Zeldin

SLA394:
Expected enrollment: 20
Preferred facilitators: Tyler,
Singer
Other facilitators: Richards, Zeldin,

SLA449:
Expected enrollment: 60
Preferred facilitators: Tyler, Singer, Shaw
Other facilitators: Zeldin, Uther

SLA451:
Expected enrollment: 100
Preferred facilitators: Tyler, Singer, Shaw
Other facilitators: Zeldin, Uther, Richards, Banks

Room	Capacity
Slater 003	45
Roman 216	30
Loft 206	75
Roman 201	50
Loft 310	108
Beach 201	60
Beach 301	75
Logos 325	450
Frank 119	60

All rooms are available for the following time slots.

Times:
10 AM
11 AM
12 PM
1 PM
2 PM
3 PM