Artificial Intelligence Programming Assignment 2

Viresh Gupta | Roll No. - 2016118

12 October 2018

Q1. User vs Computer TicTacToe with min-max and alpha-beta pruning

Methodology:

Both min-max and alpha-beta pruning have been implemented with no randomisation. i.e if at any point, there are two states with the same utility value, the algorithm will choose the state that occurs first.

No memoization is done, so the computer calculates all steps everytime.

By default, the computer uses only min-max strategy without alpha-beta pruning. To use alpha-beta pruning, pass the `--alphabeta` switch.

Usage:

To use the program and play against it, type:

```
python PA2_2016118_Viresh_Gupta_minmax.py
```

To give input from a file instead of choosing steps interactively:

```
python PA2_2016118_Viresh_Gupta_minmax.py < user_moves.txt
```

To use alpha-beta pruning, use:

```
python PA2_2016118_Viresh_Gupta_minmax.py --alphabeta
```

Experiments:

I calculated the time taken for each algorithm, by using the unix `time` command. e.g

```
time python PA2_2016118_Viresh_Gupta_minmax.py < user_moves.txt
time python PA2_2016118_Viresh_Gupta_minmax.py --alphabeta < user_moves.txt
```

Observations:

time	Min-Max	Alpha-Beta
real	0m 1.293s	0m 0.132s
user	0m 1.280s	0m 0.109s
sys	0m 0.012s	0m 0.020s

Inference:

Thus, we can observe that the time taken when using alpha-beta pruning strategy for making a decision is considerably smaller, and the alpha-beta strategy is giving a big boost to the decision making (~ 10 times).

Q2. Course time scheduling using GA, MA and CSP

Assumptions:

I have taken the following assumption:

- A class fits completely into one time slot.
- No class needs to span for more than one slot.

A clash is indicated when a professor is teaching two times in a given time slot. Similarly a venue clash occurs when two or more courses are being taught in the same venue at the same day-slot combination.

a. Logical constraints on the given problem:

Various logical constraints that I have taken on the problem are:

- No professor is teaching at two places at a time.
- Only one course is taught at a venue at a time.
- A course has class only once a week, in one of the given time slots
- The institute is a *basic* CS institute, where each professor has basic knowledge of each CS domain, so they can teach any course floated at the institute.

b. Implementations using Genetic and Memetic learning

Methodology:

I take up five parameters for a gene:

- Day of week (1-5)
- Slot of day (1-8)
- Hall number (1-N)
- Course number (1-M)
- Professor id (1-P)

A gene completely determines a course's slot in that week, since there is only one class per course in a week.

A chromosome is thus comprised of M genes, one for each course, and this determines the schedule for the week.

Usage:

The code for GA can be run as:

```
python PA2_2016118_VireshGupta_GA_MA.py
```

The code for MA can be run as:

```
python PA2_2016118_VireshGupta_GA_MA.py --ma
```

To see a full experiment successfully converging, the parameters for the number of days / slots and courses, instructors and halls can be changed in the python file and number of iterations increased.

After changing the default params:

```
python PA2_2016118_VireshGupta_GA_MA.py -i 40 --ma
```

Experiments and Observation:

I tested the implementations with 50 courses, 10 professors and 10 halls. Thus with 50 ($> 8 * 5 = 40$) courses, the initial population is expected to have some small number of clashes, since naively putting every course in every slot won't work.

Crossovers are single point rectangular crossovers, (one cut in the horizontal direction, swap and join, and then one cut in the vertical direction, then swap).

Mutations are completely eradicatory, a chromosome that undergoes a mutation can be disrupted and replaced with a fully randomised chromosome, or just the (hall, professor, course) combination gets randomised in a chromosome.

For memetic algorithm, the hillclimbing permutes the chromosome genes across properties to evaluate a nearby chromosome, thus leading to better parent selection before crossovers.
This has significant benefits over GAs in terms of faster convergence to a valid schedule.

A sample run of my GA implementation:

```
generation #0 census 15 Best of this generation  0.3332555736994701 clash val 6
generation #1 census 24 Best of this generation  0.4998000799680128 clash val 4
generation #2 census 33 Best of this generation  0.4998000799680128 clash val 4
generation #3 census 42 Best of this generation  0.4998000799680128 clash val 4
generation #4 census 51 Best of this generation  0.4998000799680128 clash val 4
generation #5 census 19 Best of this generation  0.4998000799680128 clash val 4
generation #6 census 28 Best of this generation  0.4998000799680128 clash val 4
generation #7 census 37 Best of this generation  0.4998000799680128 clash val 4
generation #8 census 46 Best of this generation  0.4998000799680128 clash val 4
generation #9 census 55 Best of this generation  0.4998000799680128 clash val 4
generation #10 census 19 Best of this generation  0.4998000799680128 clash val 4
generation #11 census 28 Best of this generation  0.4998000799680128 clash val 4
generation #12 census 37 Best of this generation  0.4998000799680128 clash val 4
generation #13 census 46 Best of this generation  0.4998000799680128 clash val 4
generation #14 census 55 Best of this generation  0.4998000799680128 clash val 4
generation #15 census 19 Best of this generation  1250.0 clash val 0
generation #16 census 28 Best of this generation  1250.0 clash val 0
generation #17 census 37 Best of this generation  1250.0 clash val 0
generation #18 census 46 Best of this generation  1666.6666666666667 clash val 0
generation #19 census 55 Best of this generation  1666.6666666666667 clash val 0
```

A sample run of my MA implementation:

```
generation #0 census 15 Best of this generation  0.49972515116685823 clash val 4
generation #1 census 24 Best of this generation  0.4998000799680128 clash val 4
generation #2 census 33 Best of this generation  0.4998000799680128 clash val 4
generation #3 census 42 Best of this generation  0.4998000799680128 clash val 4
generation #4 census 51 Best of this generation  0.9993004896572399 clash val 2
generation #5 census 19 Best of this generation  0.9993004896572399 clash val 2
generation #6 census 28 Best of this generation  1000.0 clash val 0
generation #7 census 37 Best of this generation  1000.0 clash val 0
generation #8 census 46 Best of this generation  1000.0 clash val 0
generation #9 census 55 Best of this generation  1000.0 clash val 0
generation #10 census 19 Best of this generation  1000.0 clash val 0
generation #11 census 28 Best of this generation  1000.0 clash val 0
generation #12 census 37 Best of this generation  1000.0 clash val 0
generation #13 census 46 Best of this generation  1000.0 clash val 0
generation #14 census 55 Best of this generation  1000.0 clash val 0
generation #15 census 19 Best of this generation  1000.0 clash val 0
generation #16 census 28 Best of this generation  1000.0 clash val 0
generation #17 census 37 Best of this generation  1250.0 clash val 0
generation #18 census 46 Best of this generation  1250.0 clash val 0
generation #19 census 55 Best of this generation  1250.0 clash val 0
```

Observations:

The initial population has around 6 clashes in the most fit schedule, and with generations passing by, we get better results.
By 15^{th} generation, we get 0 clashes and by 19^{th} generation we have improvised upon this schedule even further.

Inference:

We can infer that GAs often suffer from the problem that the population is not improving after each generation, it may happen that a single most chromosome is the best one out of all the rest and thus it persists for a lot of generations before a mutation occurs and leads to a better solution.

Whereas for MAs, this problem is not that significant unless a local maxima is encountered.

c. Comparision between GA and MA

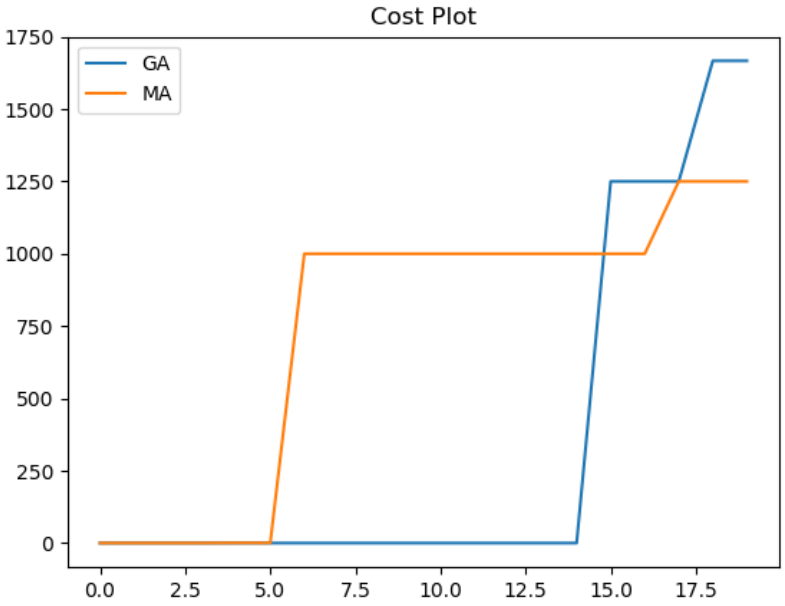
In my implementations, achieving a no clash schedule is not the only objective that the algorithm needs to achieve.
In addition to clashes, the fitness function also takes into account:
- Free slots (the less free slots, the better) - Free professors (the less free professors, the better)

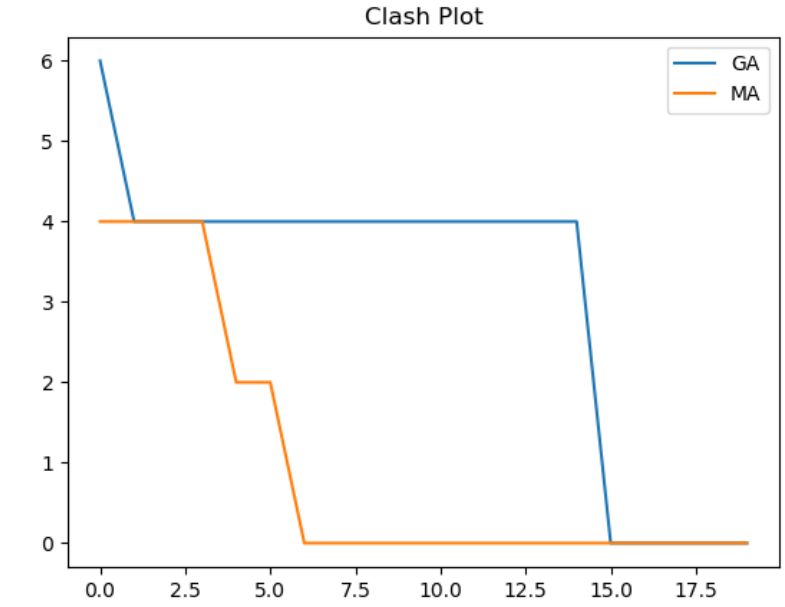
Thus even after achieving a no clash schedule, the algorithm can evolve further so that the load is evenly distributed across professors and the classes are not all conducted in the same day, thus providing sufficient time for repair and maintenance of the institute by having classes spread over venues on different days and time.

GA and MA both are able to reach no collision schedule, although once MA has achieved no clash, the evolution in that schedules happens very slowly, which most likely can be attributed to a local maxima, due to which it needs to wait for a favorable mutation.
Whereas in a GA, once we have reached a no clash schedule, the evolution happens at the same pace, and we often are able to achieve a more fit solution, although it takes more iterations.

MA is able to find a solution faster than GA, because of the inherent improvement in the parent selection process.

Plots for the same:
(Can be replicated using : `python plotter.py`)


Cost Plot
GA
MA
1750
1500
1250
1000
750
500
250
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Clash Plot
GA
MA
6
5
4
3
2
1
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

e. Solution using Constraint Satisfaction Problem Approach

For this approach, I use the same logical constraints as defined in part a). This helps in comparing the two approaches.
For the CSP algorithm, since only one class per course exists in a week's schedule, I am representing the schedule again as a 5xM matrix, with column i (0-indexed) representing course i+1.

For the main algorithm, I put a valid combination one-by-one for each course (i.e one column at a time). In case the addition of the column makes the schedule invalid, I take a step back and using the type of clash returned from a clash calculation function, I choose what action to take:
- Whether to replace the time slot - Whether to replace the professor - Whether to replace the venue

All these decisions are taken intelligently. Thus we obtain a valid solution once all M courses have been placed successfully and the algorithm terminates. If at any point of time, it so happens that a course cannot be put without violating one of the constraints, there is no schedule possible, and my algorithm terminates with an indication that no such schedule can exist.

Usage:

```
python PA2_2016118_Viresh_Gupta_CSP.py
```

Inference:

The runtime of finding a solution using CSP is much much lesser than the one found by GA, and by choosing an appropriate replacement strategy for clashes, we can also develop a spread-out schedule. Thus CSP are more useful in finding a solution for a shorter instance of the problem, but they fail terribly when there is no clash-free solution.

In such a case we have to utilize GA/MA approaches to get a schedule that is not completely clash-free, but has minimum clashes.