Collective Intelligence – Assignment 2

How to use:

Startup python

Import the main file
```
>>>import scheduler
```

run the normal scheduler taking only single flights into account
```
>>>scheduler.bestschedule()
```

run the edited scheduler for retour flights
```
>>>scheduler.bestretourschedule()
```

run the retour scheduler using the algorithm with a dynamic mutation rate
```
>>>scheduler.bestretourschedule2()
```

run the brute force algorithms (takes almost forever)
```
>>>scheduler.bruteforce()
```

How it works:

The retour functionality:
First of all I made a new class RetourTicket which can be found in the general.py file. Objects of this class would be used to store the price of the ticket and the two flights it combined into a round-trip flight. I also created a loadRetourTickets function which would read retourtickets from a file and return a list filled with RetourTicket objects.
I created a new function retourschedulecost, which is the same as the schedulecost function except it checks if the solution can profit from the new retour prices, if so, it overwrites the previous ticketcost, which was based on two single flights. I wanted to keep the old schedulecost function for comparing the two later on. The last thing that needed editing was the printschedule function, so it would print the flightcost.

The algorithm with dynamic mutation:
The desired behavior of the algorithm is that it starts with a lot of mutations and with each iteration the number of mutations decrease. I did some estimations and came up with his formula:

$$\frac{mutationprobability * 2}{(i + maxiterations \div 5) \div (maxiterations \div 5)}$$

With a mutation probability of 0.2 the result of this formula lies between 0,4 and 0,067. Starting high, but with each iteration the chance of a mutation becomes smaller. In the original file the mutation probability was a static 0.2.

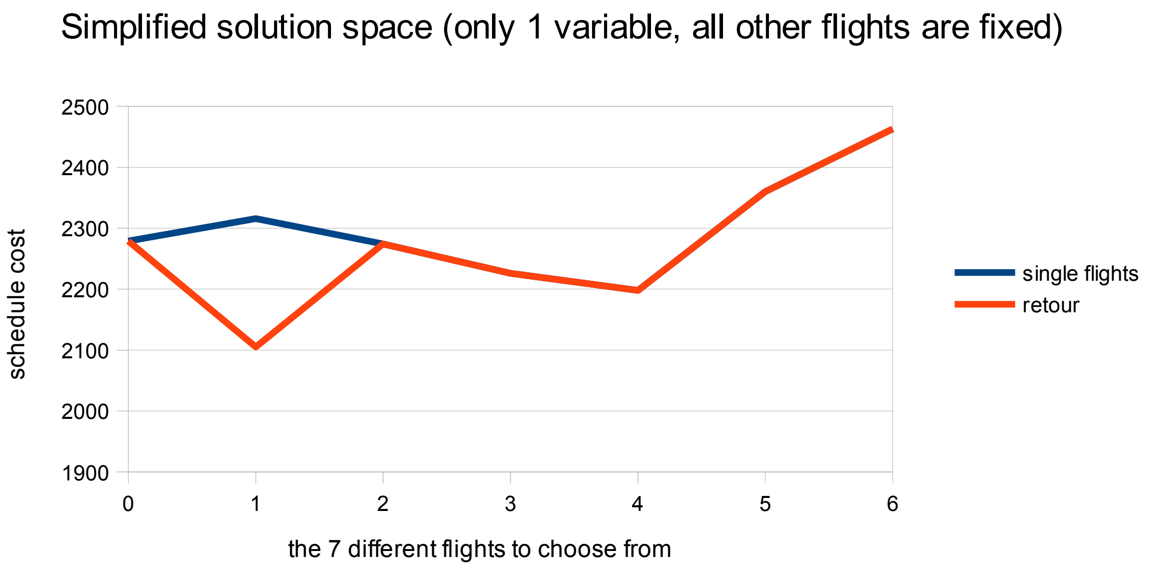
The brute force algorithm:
I also created a brute force algorithm, it's very easy, just a lot of for loops. When I executed it, I soon found out it would take a very long time, and would never get a result before the deadline of this assignment.

Testing:

After adding the new retourschedulecost function, I was of course very curious how much this would improve the schedule cost. To test this I ran the algorithms 100 times, and averaged the scores to compare them (I'll use this method for all tests).

n (number of runs)	min:	max:	avg:
geneticoptimize,popsize=100,step=1,mutprob=0.2,elite=0.25,maxiter=100,**schedulecost**			
100	2075	2331	2179,68
geneticoptimize,popsize=100,step=1,mutprob=0.2,elite=0.25,maxiter=100,**retourschedulecost**			
100	1901	2359	2072,34

As expected, adding the retour prices created some cheaper solutions (in price and time). You could actually see the retour price as some kind of discount, and that is exactly what the cost function does. When two flights from the solution appear in the retour tickets file, it replaces their price with a lower one. When you look at it this way, and try to imagine what the solution space looks like, you'll notice the problem might get harder to solve. Let me explain this with a picture.



When you're only using single flights, there aren't any big differences in price (if the origin and destination are the same). When you add retour tickets, a few tickets would be much cheaper than others. This creates big 'unpredictable' minima in the solution space and increases the chance of the algorithm getting stuck at a local minimum and not being able to find the optimal solution.

The step size is used when a solution mutates and determines how big the mutation is. It might be interesting to test if the algorithm will be able to find the optimal solution better by increasing this. Doing so, will increase the chance of hopping over a 'bump' (maximum) and discovering the minimum, but it will also increase the chance of completely jumping over this minimum.

n (number of runs)	min:	max:	avg:
geneticoptimize,popsize=100,**step=1**,mutprob=0.2,elite=0.25,maxiter=100,retourschedulecost			
100	1901	2359	2072,34
geneticoptimize,popsize=100,**step=2**,mutprob=0.2,elite=0.25,maxiter=100,retourschedulecost			
100	1900	2332	2067,44
geneticoptimize,popsize=100,**step=3**,mutprob=0.2,elite=0.25,maxiter=100,retourschedulecost			
100	1900	2334	2122,32

By increasing the stepsize from 1 to 2 the average cost decreases a bit, but not enough to call it a significant improvement. When increasing the stepsize to 3 the performance decreases again.

n (number of runs)	min:	max:	avg:
geneticoptimize,popsize=100,step=1,**mutprob=0.05**,elite=0.25,maxiter=500,retourschedulecost			
100	1891	2391	2091,43
geneticoptimize,popsize=100,step=1,**mutprob=0.1**,elite=0.25,maxiter=500,retourschedulecost			
100	1891	2259	2068,95
geneticoptimize,popsize=100,step=1,**mutprob=0.2**,elite=0.25,maxiter=100,retourschedulecost			
100	1901	2359	2072,34
geneticoptimize,popsize=100,step=1,**mutprob=0.3**,elite=0.25,maxiter=500,retourschedulecost			
100	1891	2261	2078,61

The amount of mutations, by increasing their probability, is also something to experiment with. I didn't know what a good setting would be so I just tried some different values. Apparently 0,05 is too low and 0,3 is too high. It doesn't make really big differences though, settings like 0,1 and 0,2 seem to work fine.

n (number of runs)	min:	max:	avg:
geneticoptimize,**popsize=100**,step=1,mutprob=0.2,elite=0.25,**maxiter=100**,retourschedulecost			
100	1901	2359	2072,34
geneticoptimize,**popsize=100**,step=1,mutprob=0.2,elite=0.25,**maxiter=500**,retourschedulecost			
100	1891	2272	2071,63
geneticoptimize,**popsize=500**,step=1,mutprob=0.2,elite=0.25,**maxiter=100**,retourschedulecost			
100	1891	2077	1933,25

I expected a lot from these adjustments, by increasing the population or the maximum iterations the algorithm gets to make more calculations and therefore give better results. Increasing the maximum number of iterations did lower the maximum (worst solution) quite a bit, but on average it didn't perform any better while taking much longer to complete than the standard maximum of 100 iterations. Increasing the population size did have dramatic effects! An average score of 1933,25 is really good.
The geneticoptimize function uses the survival-of-the-fittest-idea, the top 25% of a population will survive every iteration. So when the population is 5 times bigger, what happens if I only let the top 5% of the population live, will they start a super population and thus return the optimal solution every time? Probably not, but let's find out:

n (number of runs)	min:	max:	avg:
geneticoptimize,**popsize=500**,step=1,mutprob=0.2,**elite=0.05**,maxiter=100,retourschedulecost			
100	1891	2125	1987,54

The results are not that impressive, you're better of with the top 25%.

Now it was time to check out the other algorithms including my tweaked version of geneticoptimize, proudly named geneticoptimize2.

n (number of runs)	min:	max:	avg:
geneticoptimize,popsize=100,step=1,mutprob=0.2,elite=0.25,maxiter=100,retourschedulecost			
100	1901	2359	2072,34
randomoptimize, retourschedulecost			
100	2253	2702	2570
hillclimb, retourschedulecost			
100	2069	2986	2501,29
annealingoptimize, retourschedulecost			
100	2209	3093	2584,58
geneticoptimize2,popsize=100,step=1,mutprob=0.2,elite=0.25,maxiter=100,retourschedulecost			
100	1900	2270	2037,65

My edited version of geneticoptimize, with it's dynamic mutation probability, has the best results. The other algorithms, didn't work very well. I expected more from annealingoptimize.

Two pages ago I made a point that adding the retour tickets would make it more likely for the algorithms to get stuck at local minima. This should be especially true for the hillclimbing algorithm.

n (number of runs)	min:	max:	avg:
hillclimb, **schedulecost**			
100	2110	3024	2559,38
hillclimb, **retourschedulecost**			
100	2069	2986	2501,29

As you can see the hillclimbing algorithm scored better with the added retour tickets, but only with about 60 points on average. How does that compare to the geneticoptimize algorithm?

n (number of runs)	min:	max:	avg:
geneticoptimize,popsize=100,step=1,mutprob=0.2,elite=0.25,maxiter=100,**schedulecost**			
100	2075	2331	2179,68
geneticoptimize,popsize=100,step=1,mutprob=0.2,elite=0.25,maxiter=100,**retourschedulecost**			
100	1901	2359	2072,34

The geneticoptimize function scored almost 110 points lower with the added retour tickets. That's a much bigger difference. I didn't have the time to run my brute force algorithm but I expect the best score to be 1891 (solution: [1, 2, 4, 8, 7, 2, 4, 2, 10, 5, 8, 3]). I ran the algorithms a lot times and this seems to be the best I can get.
So geneticoptimize improved a lot by adding the retour tickets, even though its solutions were already near the optimal solution. Hillclimbing's solutions improved only a little, while there was so much room for improvement...
So it seems like my expectations and the way I visualized the solutions in my head were correct! :)