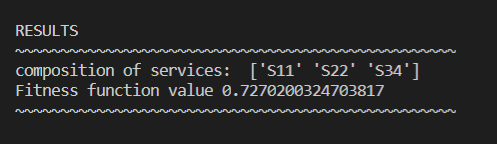
**Information for Grading Purposes**

The run function in solve.py takes the sample input file, which should be a csv, with a structure mirroring Data\_saved\_3.csv as input

To run the program, type python solve.py into the terminal inside the GA folder

The results will be displayed as shown below:



**Explanation of fitness function and how it is calculated:**

The fitness function is calculated as:

Fitness = fc \* cost\_weight + fr \* reliability\_weight + fp \* performance\_weight + fa \* availability\_weight, where the weights represent the importance of cost, reliability, performance, and availability in determining the overall fitness value for a certain service combination.

With our given input file, and specified weights, this equation is:

Fitness = fc \* 35% + fr \* 10%+ fp \* 20% + fa \* 35%

The weights can be adjusted if you decide that, for example, more importance should be placed on the cost and less on availability, in which case you might make cost\_weight equal to 50% and make availability\_weight equal 20%. The percentiles must always sum to 100%.

With the fitness equation determined, the values of fc, fr, fp, and fa must be determined.

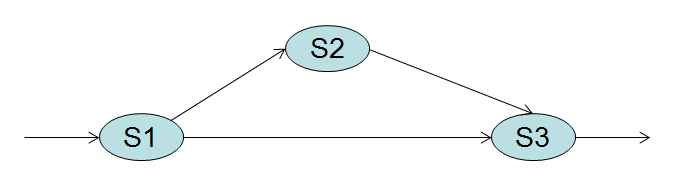
Fc = 1 – (the sum of the cost for each service in the current combination) / (the sum of the maximum cost of a service from each category)

Fp, similarly is calculated as:

Fp = 1 – (the sum of the time for each service in the current combination) / (the sum of the maximum time of a service from each category)

To calculate the values fr and fa, the reliability and availability values should be used as decimals. For example, a reliability of 98% would be used in the calculation as 0.98

I have included the diagram for our workflow below:



The path taken could be S1 -> S2 -> S3 or S1 -> S3

To calculate fr, we find fr(s1 -> s3) = fr(s1) \* fr(s3)

We also calculate fr(s1 -> s2 -> s3) = fr (s1) \* fr(s2) \* fr(s3)

We want fr to be the minimum of these two calculated values

Fr = min(fr(s1 -> s3), fr(s1 -> s2 -> s3) ).

We calculate fa in the same way that we calculated fr.

Once we have calculated fc, fr, fp, and fa, we can finally determine the fitness value for a certain combination of services with the equation below:

Fitness = fc \* 0.35 + fr \* 0.10% + fp \* 0.20 + fa \* 0.35

The fitness value should fall in [0,1] and we want to choose the combination of services with the maximum fitness value. As the Genetic Algorithm runs, and more and more epochs are run, the fitness value should converge and w should have a good combination of services to recommend.

**Explanation of how we built the mapping from business question into the digital world in chromosomes and genes**

The business question concerns 3 different service categories with which you must select a service from each category to create a service combination which should be a good overall combination in terms of 4 metrics – cost, reliability, time, and availability.

To map this into a problem that fits Genetic Algorithm, each chromosome, which represents a combination of services, has 3 genes. One gene corresponds to a service from category 1, a second gene corresponds to a service from category 3, and a third gene corresponds to a service from category 3. The 3 genes are restricted to be integer values, and we further restricted each gene to have a value in the possible range for that category. For example, for category 1, the options are S11 through S15. Hence the gene corresponding to service category 1 can only have a value in [1,5].

With the business question mapped into a digital world, the parameters for the genetic algorithm had to be specified. For these parameters, we specified the maximum number of iterations as 100. We set the population size to be 50. This means there are 50 chromosomes in each generation. The mutation probability was set to 10% so that our algorithm did not get stuck in a local maximum instead of a global maximum.

Our elite ratio was 2%. This means that one elite is kept in the population consistently. The crossover probability we used was 50%. Additionally, the parents\_portion was set to 30%. This means that 70% of the population is replaced each time. We used uniform crossover as our crossover type. With uniform crossover, you treat each gene separately. For example, for gene 1, we flip a coin. If we get Heads, the first gene of child 1 is the first gene of parent 1 and the first gene of child 2 is the first gene of parent 2. On the other hand, if we get Tails, the first gene of child 1 is the first gene of parent 2 and the first gene of child 2 is the first gene of parent 1.

The fitness function value our genetic algorithm tended to converge to was 0.727… with a service combination of S11, S22, S34