2021 Fall CS454 at KAIST

Assignment 2:

**Solving Minimum-Time Path Problem in a Metro System Involving Multiple Levels of Express Lines, Using Genetic Algorithms**

Gunhee Yi (20180431)

In almost every weekend, I take the line 9 of Seoul Metro. It conveniently connects the Express Bus Terminal and my town, and the 18 minutes that I spend there becomes the last step of the long journey from KAIST in Daejeon to my home in Seoul. 18 minutes, that is the time when I take the express train. There are both express and local trains in line 9. It takes 30 minutes if I take the local train, so it is better to take the next express rather than the local one even if I just missed the express. Though knowing that already, I still check the metro app often, to see the arrival time of the next train. Then one day, as I was staring blankly through the screen, I started wondering about the cases where the departing or arriving stations were not express stations. In these cases, going forward some local stations before/after getting in/out of an express train was a common doing. But what if going BACK a little actually earned you some time? It turns out that it is not the case in line 9. Refer to the following deduction:

Settings: *Station 1* is a local station right after the express *station 0*. There are *n* local stations between *station 0* and another express *station (n+1)*. You are now at *station* *1* and desires to reach *station (n+1)*. It takes constant *l* minutes for a local train to move 1 station. It takes *e* minutes for an express train to move from *station 0* to *station* *(n+1)*.

Going forward for *n* local stations takes *ln* minutes, whereas going back a station and taking the express takes *(l+e)* minutes. If the backward route is to be better, it should be *l+e < ln*, that is, *e < (n-1)l*. Therefore we can conclude that *n* must be at least 3, if we assume that *e* cannot be smaller than *l* (which is always true, if express trains accelerate at least as much as local ones.).

There are not any adjacent pairs of express stations in line 9 that has more than two local stations in-between, so going back should never be an option for the passengers. But further search revealed that there was another line that had expresses. It was line 1, and its expresses passed through well over 10 stations during their operation.

Now that we have found a real-world example that going back would actually benefit us, many path-finding algorithms may fail or struggle to utilize the fact. Take a greedy algorithm for example, which picks a path that is now possible and most decreases the distance to destination at a constant time. It will never try going back because it increases both the distance and time taken, in short sight.

An algorithm that looks through every possible routes, will succeed on finding the optimal path but will still struggle, for two reasons. First, the capability of going back practically doubles the number of edges. Most of the path finding algorithms has a time complexity proportional to this number, so the solving time will double as well. Second, loop is newly introduced. A loop may cause the path finder to iterate in it indefinately and fail eventually, so we should modify the algorithm to detect and exclude loops from its routes.­

This is where I thought the genetic algorithm could come to place. In a problem like this, we expect that suboptimal solutions will be a lot alike with the actual optimal solution. And in alike I mean that their paths are likely to be the same part by part. This is what the genetic algorithm directly looks for. By some predefined methods, my genetic algorithm will try to preserve the optimal part of genes (that is, sub-paths in this case) and try new types in other parts. By doing this, the genetic algorithm exceeds other algorithms in that it retains some memories and only tries solutions that are likely to be good, using prior memories. There is also some factor of “exploration” such that the solver lies not only in local optimums.

One may argue that even though the amount of calculation increases due to the addition of reverse ways, it is still in a manageable range for the traditional algorithms and a modern computer to handle if the number of stations is limited to about the magnitude of a single metro line. I was also convinced by this self-argument, so I decided to complicate the problem in a useful way: adding multiple levels of expresses. There won’t be many metro systems that has multiple levels of expresses in a single line, but we can think of it as an analogy of the whole transportation system that involves not only metros but also buses, trains and even airliners. Level 0 should be the slowest means such as town buses, and the highest level should be the airliner.

This kind of expansion will allow the problem to be applied to a system not only with higher levels, but also with bigger number of “stations” (Think about the number of stations that you would have to pass if you were to travel from Seoul to Busan only on town bus!), where traditional algorithms will begin to fall.

Now that I have pictured the problem, I formulated into a real set of stations and local/express lines. Refer to the following:

Level 0 (2min): 0-1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20

Level 1 (5min): 1---------6----------11-------------16

Level 2 (8min): 0-------------------10----------------------------20

\* Mr. Yi is now at station 2, and wants to reach station 19.

In this setting, there are 21 stations and three level of lines in total. Level 0 line is the local line, where every adjacent stations are connected and takes a 2 minute ride between. There is level 1 line as well, and this connects every (5n+1)th stations with a 5 minute ride. Lastly we have the level 2 line, where every (10n)th stations are connected. It is the sparsest but is the fastest at the same time (10 stations in 8 minutes). Using this metro system, Mr. Yi wants to travel from station 2 to station 19.

Now let’s see through some example solutions:

Sol1. 2-19, 34 minutes (using only level 0)

Sol2. 2-6-16-19, 24 minutes (using level 0 and 1, no reverse direction)

Sol3. 2-1-16**-19**, 23 minutes (using level 0 and 1, 1 reverse direction)

Sol4. 2**-0**-10-11-16-19, 25 minutes (using level 0, 1 and 2, 1 reverse direction)

Sol5. 2**-0**-20**-19**, 22 minutes (using level 0 and 2, 2 reverse directions)

We have 5 solutions in total, each with a point. Solution 1 is the naïve solution, which uses only the local level 0 line. It is the easiest solution, but no useful algorithm will produce this as its answer because of its long time. Solution 2 is the optimal solution if no reverse direction is involved. Many imperfect algorithms such as the greedy algorithm will produce this as the solution. It greatly improved from the naïve solution by 10 minutes, but there is still more to go. Solution 3 and 4 tried involving 1 reverse direction, and one succeeded in improving by 1 minute. Solution 5 involved two reverse direction sub-routes, and succeeded to reach the best solution of 22 minutes. As the optimal solution involved two reverse directions, it is expected for other algorithms to struggle to find this solution. Then what will be the case for genetic algorithms?

As in the prior notation of solutions, we can record any meaningful paths by writing down only the “hub” stations plus the starting and ending stations. I defined the hub stations to be the stations that at least one express line passes for each. In our example, station 0, 1, 6, 10, 11, 16 and 20 are the hub stations. If a direct express line exists between the two written stations, we assume that the route takes the line. If there is no such line, we assume that the route just takes the local line between the stations. As the starting and ending stations were known and recurring information, it was excluded from the final form as well. As a result, an instance of a “Path” class was initialized by a list of interior hub stations.

In the very beginning, the algorithm initializes a pool containing a predefined number (POP variable) of Path organisms. All organisms in the pool are initialized by a random sequence of hub stations. The length of sequence can vary from 0 to the number of hub stations present.

I have written previously that my genetic algorithm will try to preserve the optimal genes and also explore through the gene spaces. This is done by two functions on the Path organisms in the current pool starting from the initial one, respectively.

Preservation is done through the crossover() function. It takes two path organisms as input, and chooses a random subsequence from the first one’s sequence. If there is a subsequence in the second one’s sequence that matches the starting and ending stations, two children is produced by cutting and pasting the subsequence to each other’s sequence. It works even if the starting and end points are reversed (ex. subsequence [1, …, 6] from path 1 and subsequence [6, …, 1] can be interchanged, by reversing the order first), to reflect the spirit of reverse progressing. The crossover() function aims to preserve the optimal part of route and refer to other organisms for a better solution in other parts, in short.

Exploration is done by the mutation() function. It takes a single path organism, and literally mutates its sequence by choosing a random index and replacing the station at that index with a new hub station. It aims to introduce an all-new factor to the pool to encourage it to have solutions for other local optimums. For both functions, as a duplicate value in the path means some stagnation or a loop, a path containing a duplicate value is not added to the pool.

The amount of children and mutants being produced is defined as a certain factor (“RATE\_CROSSOVER” and “RATE\_MUTATION”) to the pool population. After summing the children, mutants and original organisms, the algorithm only includes the fittest few organisms as predefined as the population. This process is repeated by a predefined number (variable “EPOCH”) of times.

As a result, 14 executions out of 20 trials succeeded to find the optimal solution. They all found it within 20 epochs, which is a surprising result because the population and rate of reproduction/mutations were not high as well. 20 epochs with a pool population of 10 and other 10s for additional reproductions and mutations, result in roughly 20\*(10+10+10)=600 paths, including duplicates. It was considerably small compared to one that might be involved for other types of algorithms.

Non-optimal solutions were also not a big failure at all but sub-optimal, since they all ranged from 23to 25 minutes. A sub-optimal solution is acceptable in this kind of problem, so I may say that the use of genetic algorithm on this problem was a success.

The following is an example log of a successful execution.

Initial pool------------------

2-19 (34min)

2-16-19 (34min)

2-11-6-16-19 (39min)

2-11-6-10-20-19 (41min)

2-0-19 (42min)

2-20-10-19 (62min)

2-10-0-11-20-19 (66min)

2-11-16-1-19 (74min)

2-6-10-0-20-11-19 (74min)

2-6-16-11-0-1-10-20-19 (75min)

-------------------------------

The fittest path in epoch 1 is 2-6-11-19 (29min)

The fittest path in epoch 2 is 2-6-11-19 (29min)

The fittest path in epoch 3 is 2-1-6-10-20-19 (25min)

The fittest path in epoch 4 is 2-1-6-10-20-19 (25min)

The fittest path in epoch 5 is 2-1-6-10-20-19 (25min)

The fittest path in epoch 6 is 2-1-6-10-20-19 (25min)

The fittest path in epoch 7 is 2-6-16-19 (24min)

The fittest path in epoch 8 is 2-1-11-10-20-19 (24min)

The fittest path in epoch 9 is 2-1-11-10-20-19 (24min)

The fittest path in epoch 10 is 2-1-11-10-20-19 (24min)

The fittest path in epoch 11 is 2-1-11-10-20-19 (24min)

The fittest path in epoch 12 is 2-1-11-10-20-19 (24min)

The fittest path in epoch 13 is 2-1-0-20-19 (22min)

The fittest path in epoch 14 is 2-1-0-20-19 (22min)

The fittest path in epoch 15 is 2-1-0-20-19 (22min)

The fittest path in epoch 16 is 2-1-0-20-19 (22min)

The fittest path in epoch 17 is 2-1-0-20-19 (22min)

The fittest path in epoch 18 is 2-1-0-20-19 (22min)

The fittest path in epoch 19 is 2-1-0-20-19 (22min)

The fittest path in epoch 20 is 2-1-0-20-19 (22min)