Assignment #2

Due date: December 9th, 2022 14:00

In our scenario, we are a vendor that has several items in our stock as given in the following table:

Items	A	B	C	D	E
Stock ($\mathbf{s}$)	30	40	20	40	20

We want to sell these items in five city markets. The cities will be denoted as x_j, where $j \in \{1, \ldots, 5\}$. Each city market has a different purchase price per item. These prices are given in the following table:

$\mathbf{P} = \{p_{i,j}\}$	x_1	x_2	x_3	x_4	x_5
A	1	4	6	4	4
B	3	8	2	5	15
C	3	12	3	5	5
D	2	6	10	2	4
E	10	5	12	6	3

We want to find a match (or assignment) matrix, $\mathbf{M} = \{m_{i,j}\}$, between individual item counts and city markets that will maximize the total revenue, f, according to some constraints. The matches should be from the set of natural numbers, as an individual item will not be fractional: $m_{i,j} \in \mathbb{N}$.

The base revenue gained by selling the items will be denoted with

$$f_{base} = \sum_i \sum_j p_{i,j} m_{i,j}$$

where $p_{i,j}$ is the revenue from selling one of i^{th} item to j^{th} city market, and $m_{i,j}$ is the count of the i^{th} item sold in j^{th} city market.

Each constraint affects the total revenue to a degree. If the conditions do not satisfy a hard constraint, the total revenue, f, will be zero, regardless of the other bonuses.

Hard Constraints

- The match matrix will be composed of the natural numbers.
- Exactly 150 items will be sold, that is to say, the sum of the match matrix should be equal to 150. $\sum_{i,j} m_{i,j} = 150$.
- Sold item amounts must be equal to the stock. $\sum_j m_j = \mathbf{s}$, where $\mathbf{s}$ is the stock vector.

Soft Constraints

- If all cities are visited, the vendor gains a bonus of 100, $f_1 = 100$.
- If a city receives a balaced amount of items, the vendor can gain bonuses up to the 20% of the revenue gained from that city. If the vendor sells the same amount of items in a city, we call that transaction as balanced. The

bonus falls gradually as the balance is broken. If there is a difference of 20 between the maximum and the minimum items sold in that city, the vendor does not gain any bonus from that city. The bonus will not be negative. The bonus, f_2, will be the sum of all the bonuses gained in all the cities.

$$f_2 = \sum_j \left[\frac{1}{100} \max(20 - (\max_i m_{i,j} - \min_i m_{i,j}), 0) \sum_i p_{i,j} m_{i,j} \right]$$

- The previous soft constraint is similar for the cities. If all the cities receive a balanced amount of items, the vendor can gain a bonus up to the 20% of the base revenue. The gradual fall of the bonus, f_3, applies similarly as in the previous constraint.

  $$f_3 = \frac{1}{100} \max(20 - (\max_j \mu_j - \min_j \mu_j), 0) f_{base}$$

  where $\mu_j = \sum_i m_{i,j}$, is the sum of all items sold in the j^{th} city market, x_j.

The total revenue, f, is calculated as in the following:

$$f = f_{base} + f_1 + f_2 + f_3$$

Assignment: You are supposed to search for the best STATE vector (or matrix), **M**, that maximizes the fitness function using evolutionary algorithm design of your choice. The total revenue, f, will serve as the fitness function in the evolutionary algorithms context. You need to implement an iterative evolutionary algorithm tailored to your needs. The details of the problem were discussed in the course. You are free to choose any version of "population", "initialization", "cross-over", "mutation", "selection" and "stop condition" options in your implementation.

Submission:

- Submit the code that searches for the best match matrix possible.
- Submit the screen shot or output of one of the algorithm runs.
- A small report that includes the following information:
  - The data structure of the chromosome. How you implemented the matrix **M** in your code.
  - The size of the population.
  - How you initialize the algorithm.
  - How you implemented the cross-over and why.
  - How you implemented the mutation and why.
  - How you implemented the selection and why.
  - The maximum revenue you reached, and the corresponding match matrix.
  - At what condition your algorithm stops. For how many iterations your algorithm runs (on average).