MEC Day 2 Assignment Results

Danyan Zha

May 22, 2018

Q1 Parrelizing the Jacobian Coordinate Update Algorithm

Here is the result from Parrelizing Method using my own computer:

```
[1] "Number of cores used is 1"
[1] "Jacobi Coordinate Descent Method converged in 21 steps and 0.985999999999999s."
[1] "Precision error is = 6.49126557689387e-07"
[1] "Value of the minimization problem = 191.668911118167"
[1] "Number of cores used is 2"
[1] "Jacobi Coordinate Descent Method converged in 26 steps and 1.18900000000002s."
[1] "Precision error is = 6.72308772579451e-07"
[1] "Value of the minimization problem = 191.66891111818"
[1] "Number of cores used is 3"
[1] "Jacobi Coordinate Descent Method converged in 27 steps and 1.26700000000005s."
[1] "Precision error is = 8.6293577276771e-07"
[1] "Value of the minimization problem = 191.668911118204"
[1] "Number of cores used is 4"
[1] "Jacobi Coordinate Descent Method converged in 28 steps and 1.26599999999996s."
[1] "Precision error is = 7.49009036195554e-07"
[1] "Value of the minimization problem = 191.668911118195"
```

which is actually slower than the non-parrellization code, which is as follows:

```
  [1] "Jacobi Coordinate Descent Method converged in 30 steps and 0.0710000000000264s."
  [1] "Precision error is = 8.38923706543084e-07"
  [1] "The total number of rides demand = 30.0743045856514"
  [1] "The total number of rides supply = 30.0742552882611"
  [1] "The average price of a ride is = 0.844430117608689"
  [1] "Value of the minimization problem = 191.668911118212"
```

In this case, we can write out the closed-form expression of the price update process, which takes away the advantage of the parallel coding. Moreover, there is some fixed cost of assigning more cores, which may explain why giving more cores actually takes longer time.

Matching with NTU, Gale-Shapley

In Q2, I break the tie by giving priority to options with lower coordinates, both for passengers and drivers. Here are the results:

```
[1] "Gale and Shapley converged in 820 steps and 125.85s."
[1] "Total number of passengers is 100; Unmatched passengers are 69.75"
[1] "Total Welfare on the passengers' side is -139.5"
[1] "Total number of drivers is ,30.25; Unmatched drivers are 0"
[1] "Total Welfare on the drivers' side is -9.49617937947586e-17"
[1] "Total matches are 30.25"
```

For Q3, with random disturbances to the payoff matrix, the results are pretty similar to Q2:

```
[1] "Question 3 with tie breakings"
[1] "Gale and Shapley converged in 912 steps and 140.447s."
[1] "Total number of passengers is 100; Unmatched passengers are 69.75"
[1] "Total Welfare on the passengers’ side is -139.336339508261"
[1] "Total number of drivers is ,30.25; Unmatched drivers are 0"
[1] "Total Welfare on the drivers’ side is 0.137101335625071"
[1] "Total matches are 30.25"
```

Matching with NTU, Adachi Algorithm

I am not sure whether we can directly apply Adachi Algorithm to this problem because this algorithm would require each individual is different from each other on one side, while we have types with different densities here. I tried to multiply the density by 100 and hence get an expanded X matrix (10000 rows) and Y matrix (3025 rows), but the dimension seems a bit large for this problem. Hence I adjusted this question by changing all X and Y to same density. Here are the results when there are 100 different passengers and 100 different drivers:

```
[1] "Question 2"
[1] "Adachi converged in 2 steps and 0.0410000000000537s."
[1] "Total Welfare on the passengers’ side is 0"
[1] "Total Welfare on the drivers’ side is 0"
```

In Q2, total welfare is 0 because we would have perfect identical matching in this case, and $\alpha_{xy} = \gamma_{xy} = 0$ when $x = y$

```
[1] "Question 3"
[1] "Adachi converged in 64 steps and 0.127000000000066s."
[1] "Total Welfare on the passengers’ side is 0.899624805161729"
[1] "Total Welfare on the drivers’ side is -15.0572973407013"
```