Application of contextual bandit algorithms on Donor data

Sandeep Kumar Gangarapu

November 15, 2018

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

## Loading required package: lattice
```

We apply LinUCB algorithm on Donor data which was the output of randomized field experiment.

In order to simulate contextual bandit procedure and compare it with A/B testing, we do the following trick.

1. Shuffle the data
2. Randomly sample one feature vector(person)
3. Ask the algorithm where the person should be allocated
4. Peek at the actual group and Check whether the suggested group is same as the actual group.
5. If yes, we call this an optimal allocation and simulate pulling the lever by observing the reward(peeking at the value of firm gain)
6. If no, discard the datapoint and move on to the next one

The above process of filtering data points that follow the orders of contextual bandit algorithm simulates the real process of contextual bandits.

We could also store all the discarded points and call them suoptimal and simulate pulling the lever by observing the reward(peeking at the value of firm gain)

We expect the contextual bandit (optimal) allocation and subsequent aggregate reward to be better than random allocation and suboptimal

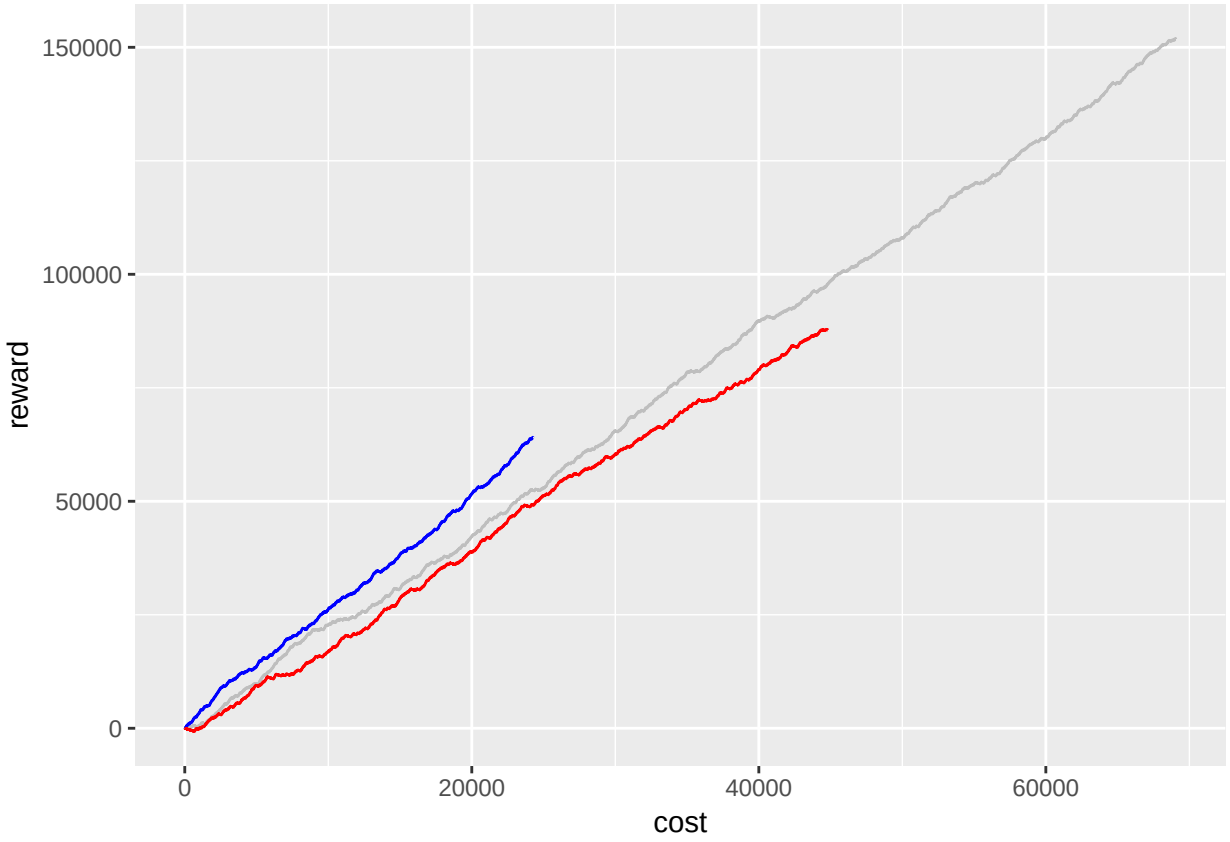
The below graphs demonstrate whether this is the case

We observe this for different values of alpha which is a parameter that gives weight to the upper confidence bound. The higher the value of alpha the more exploration the algorithm does.

for all the graphs red line represents suboptimal, grey line represents random and green line represents optimal

alpha = 1

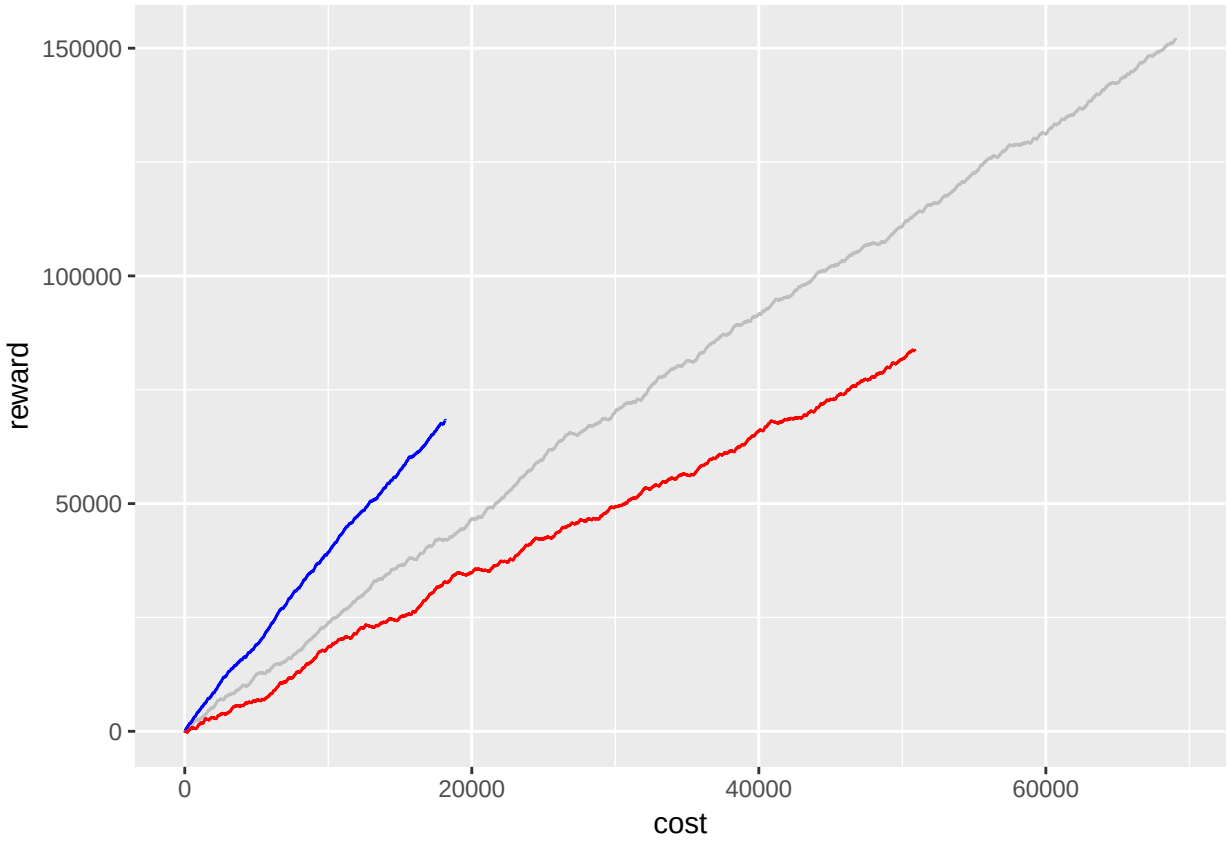
iteration = 1



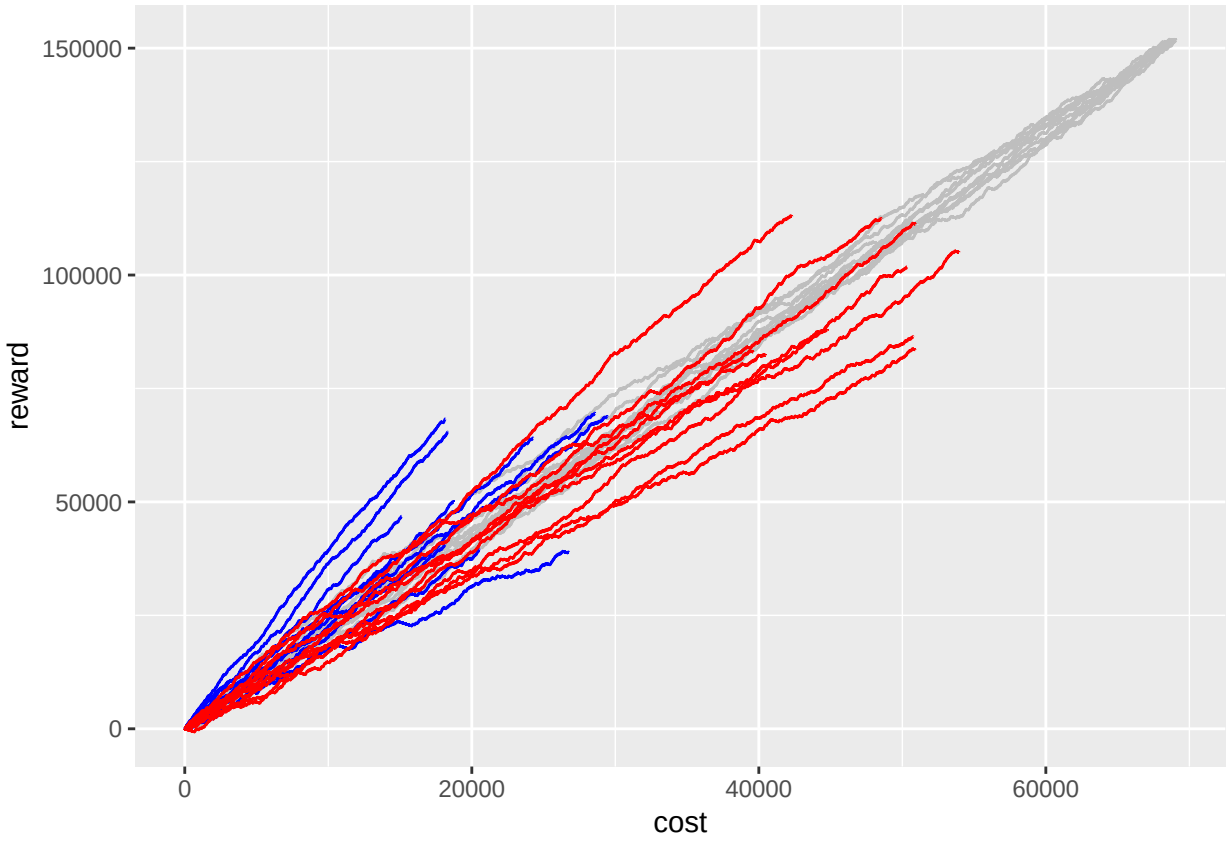
We suspect this may be because of random order of arrival of persons (random sampling of datapoints), So we check again

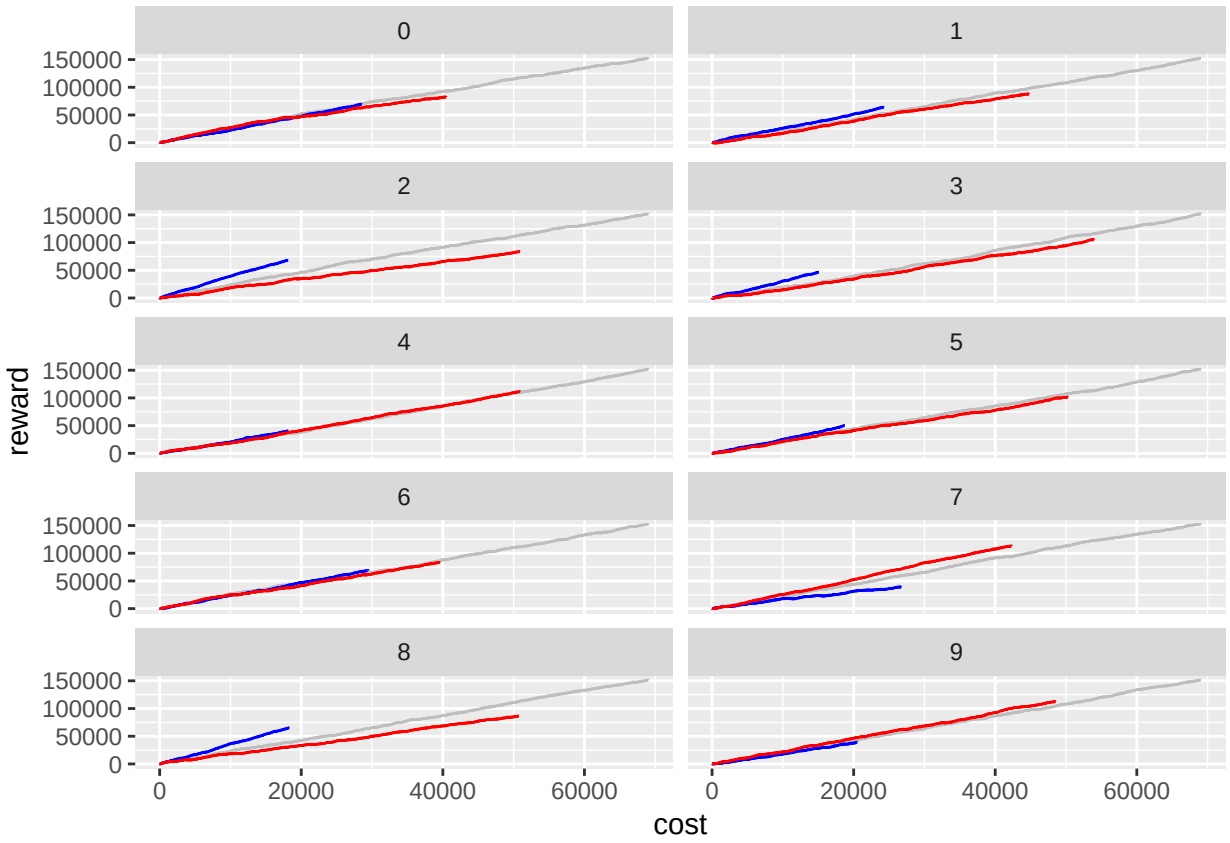
alpha = 1

iteration = 2



Just to be sure, we do it 10 different times

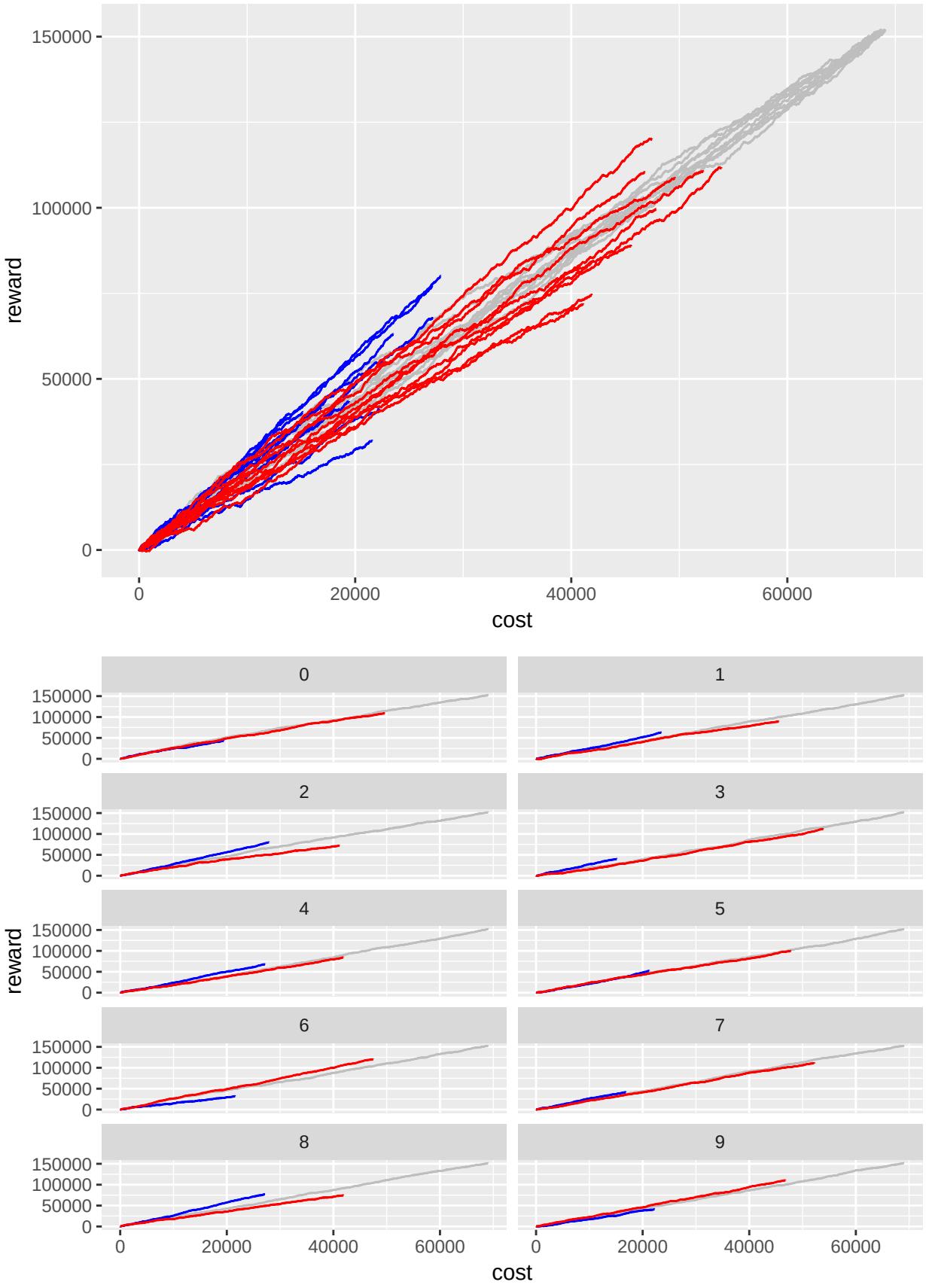

150000
100000
50000
0
reward
0
20000
40000
60000
cost



Its consistent but some iterations have improper order

We can check it for other alphas

alpha=10


reward
cost
0
20000
40000
60000
50000
100000
150000
0
1
2
3
4
5
6
7
8
9

This could be because the data is sparse and only small percentage of datapoints have non zero rewards

Var1	Freq
0	51542
220	41
330	86
440	417

So, the pattern of arrival of these datapoints heavily affect the learning process of the model. This should probably go away if techniques like upsampling or downsampling are used to get balanced data.

This could also be because of unbalanced treatment classes. There are more people allocated to control group than other treatment groups.

Var1	Freq
0	9092
1	14199
2	28795

We check the behavior for both.

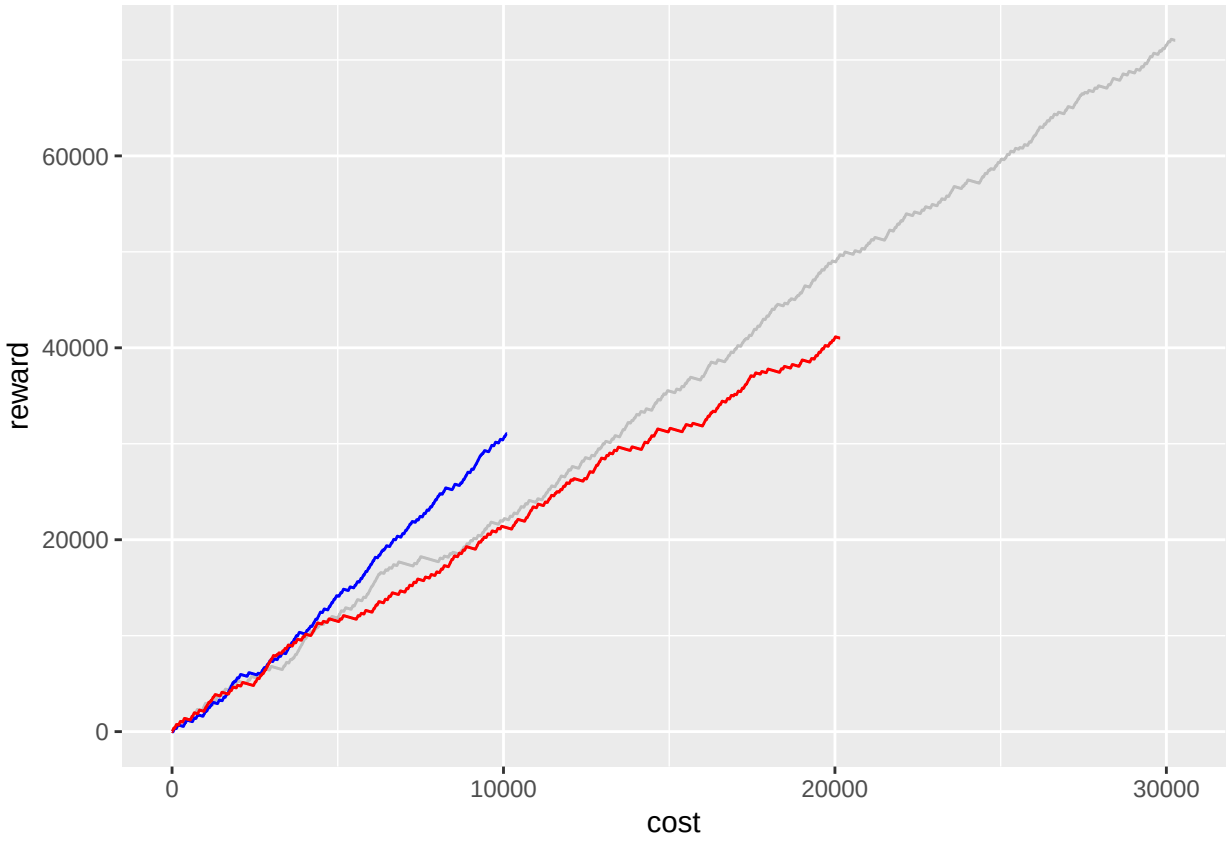
Downsampling treatment groups

Before

Var1	Freq
0	9092
1	14199
2	28795

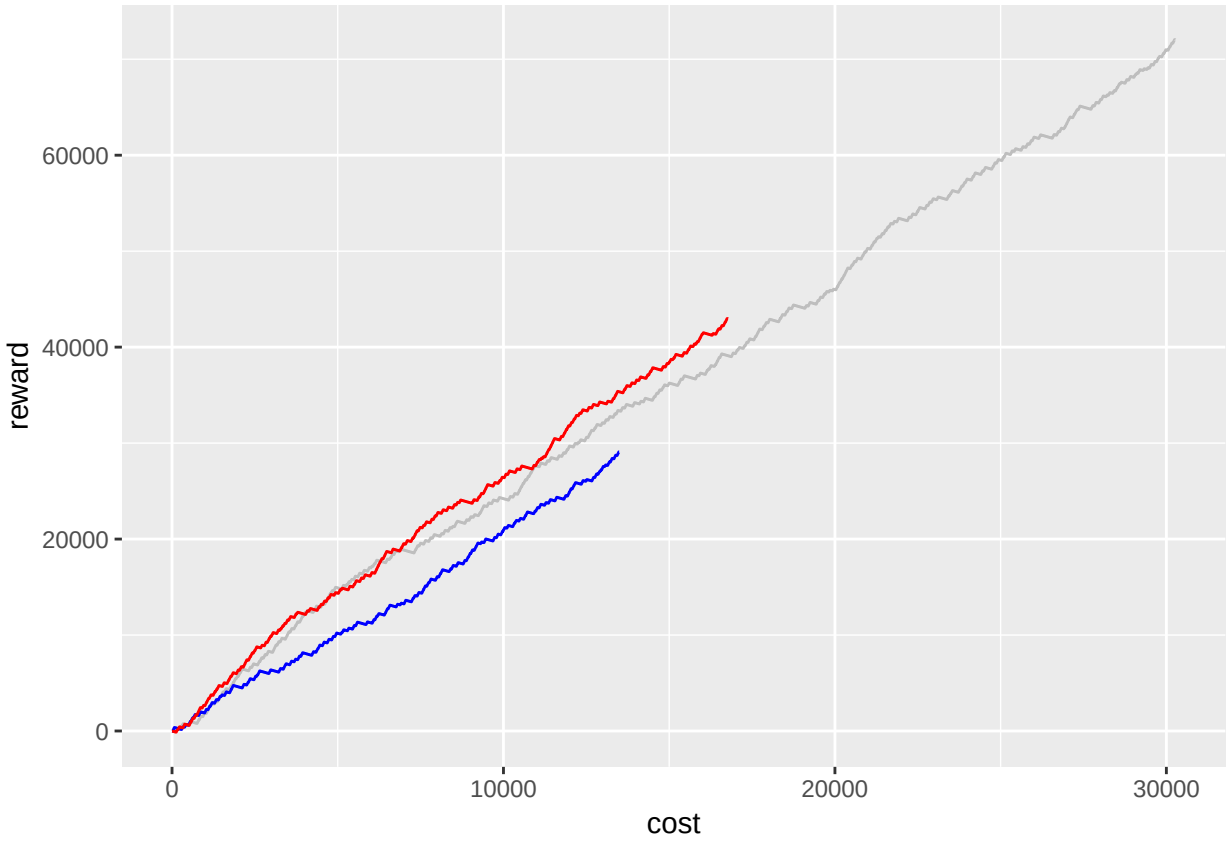
After

Var1	Freq
0	9092
1	9092
2	9092

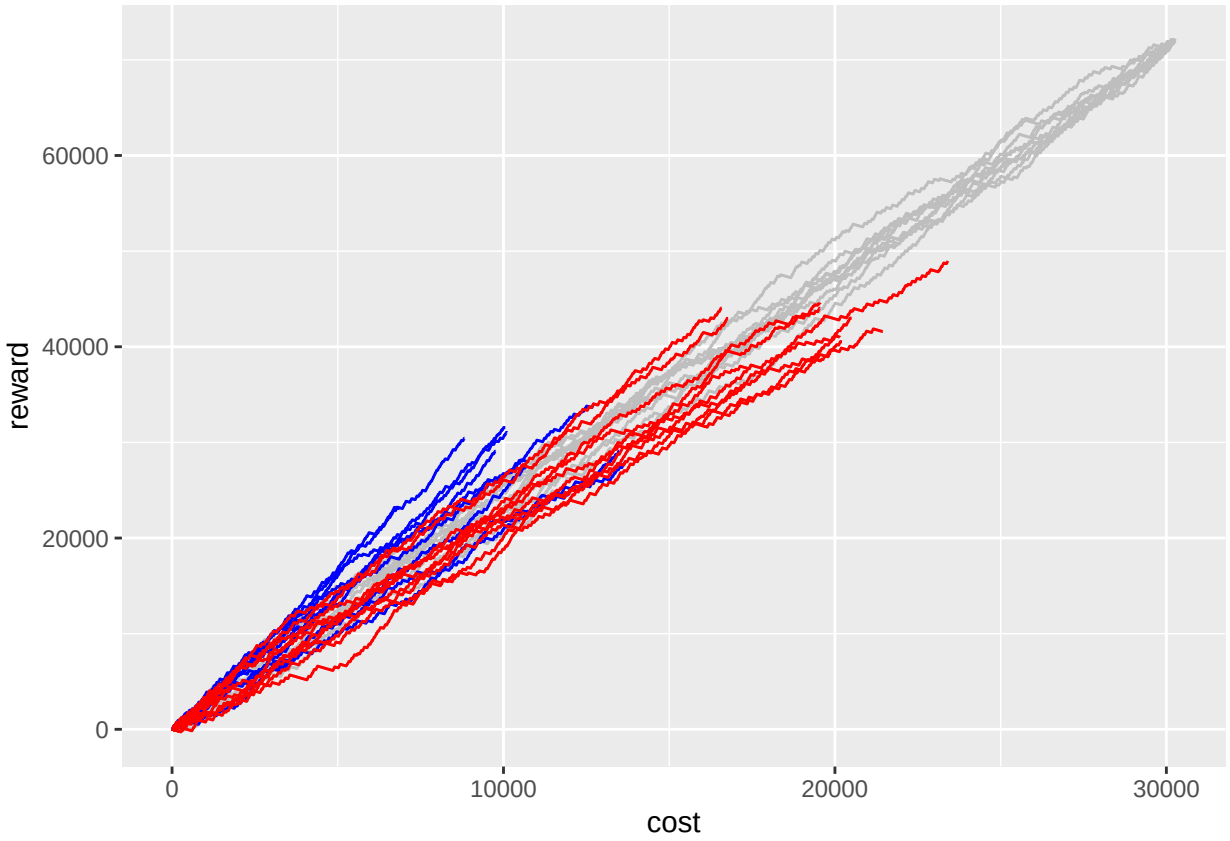

reward
60000
40000
20000
0
0
10000
20000
30000
cost

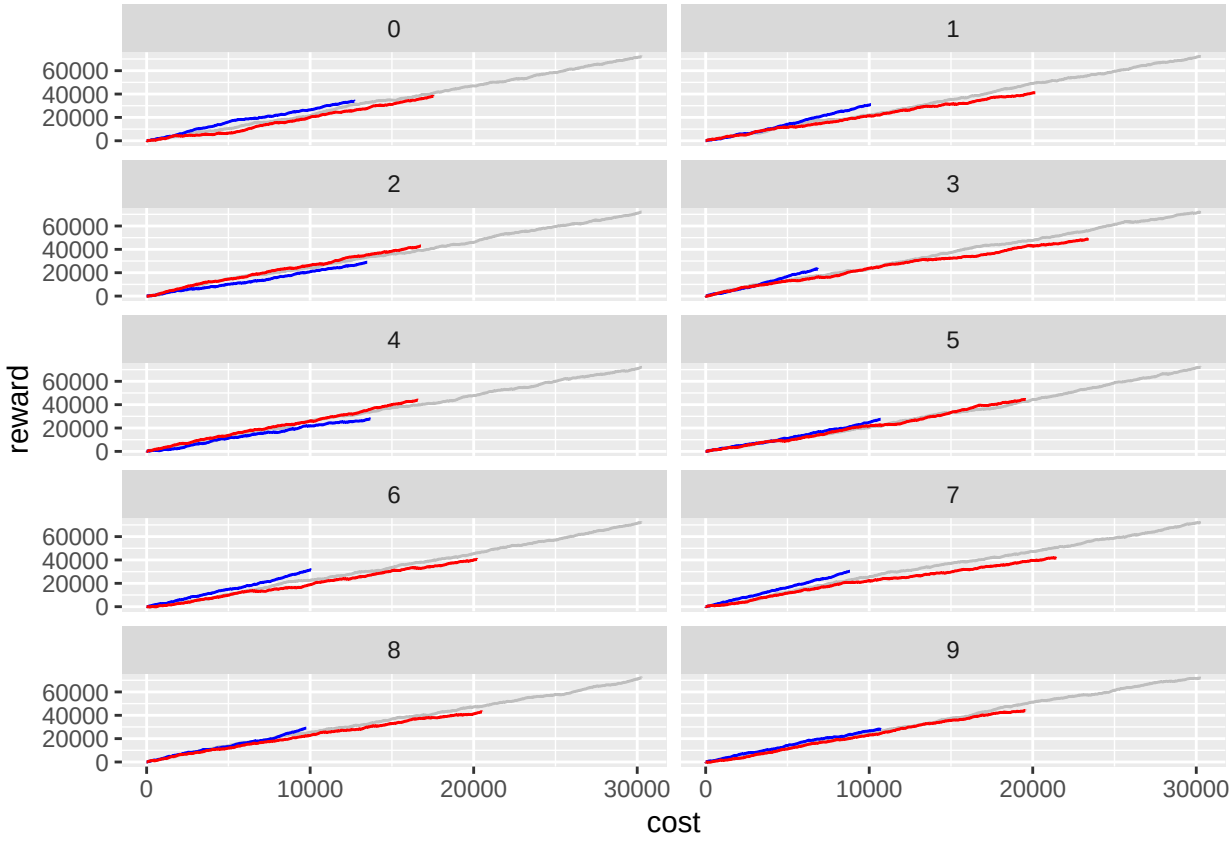
alpha = 1

iteration = 2



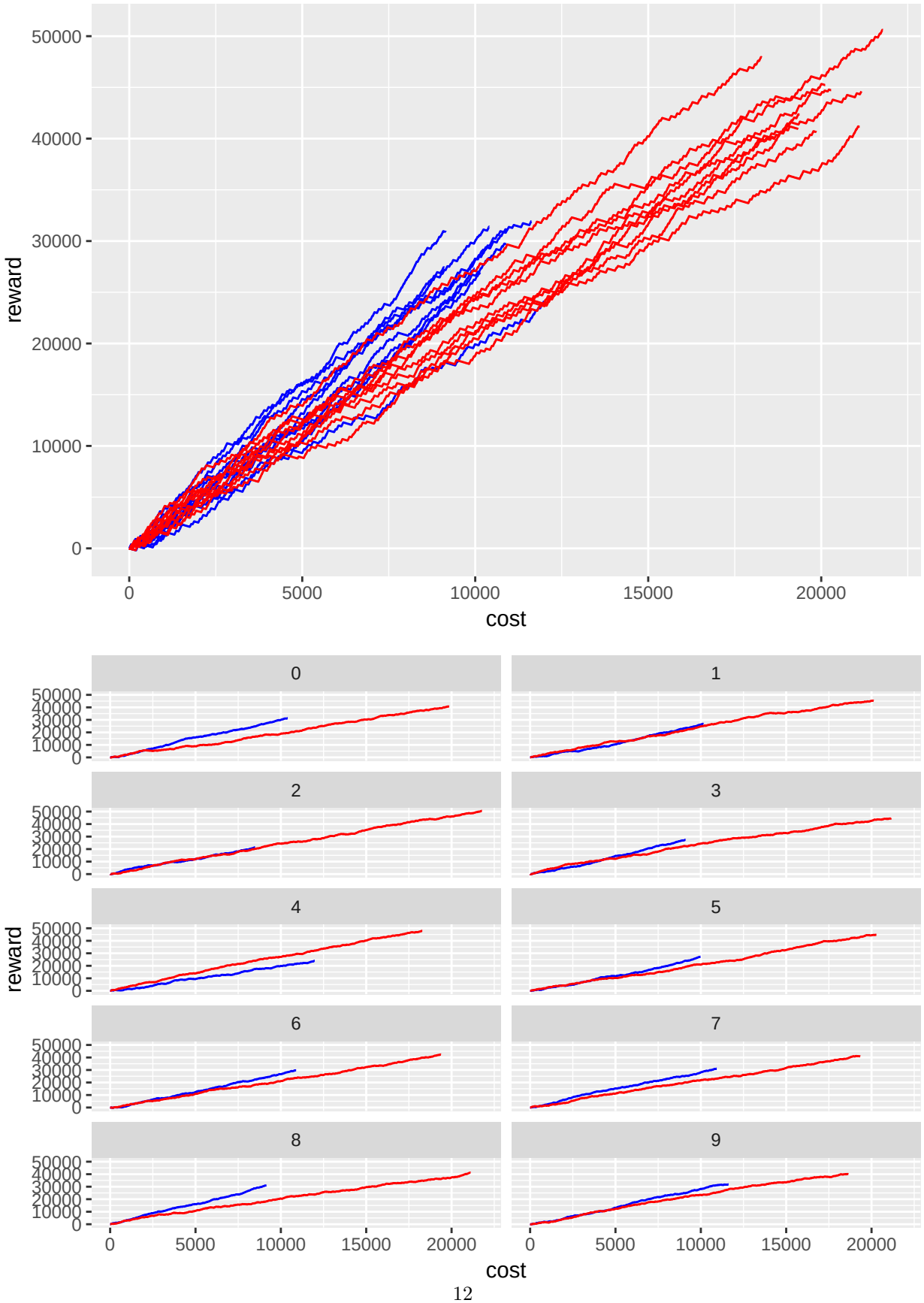
Just to be sure, we do it 10 different times


reward
60000
40000
20000
0
0
10000
20000
30000
cost



We can check it for other alphas

alpha=15


reward
cost
0
5000
10000
15000
20000
0
10000
20000
30000
40000
50000
0
1
2
3
4
5
6
7
8
9

Its consistent but some iterations again have improper order

So the problem is not so serious as only 20% of iterations behave this way and its not going away with balances treatment groups.

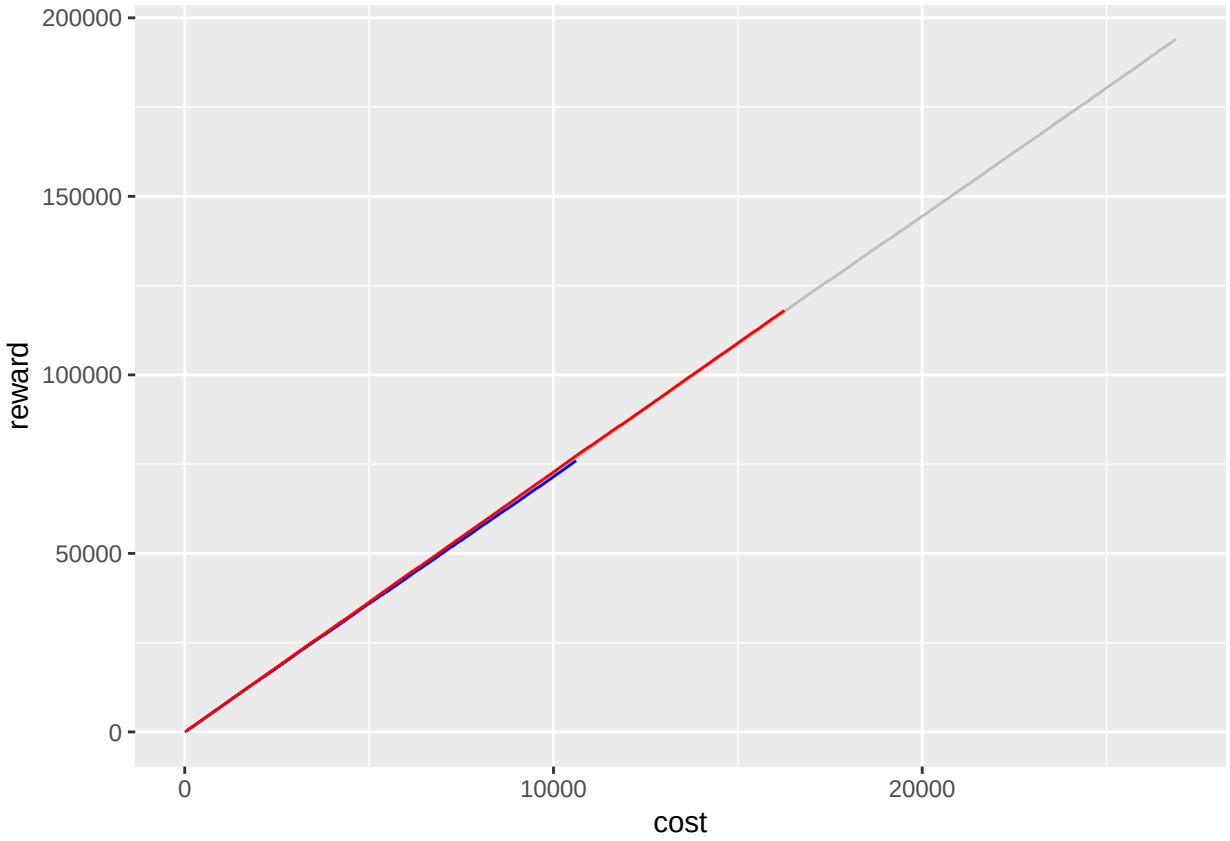
Downsampling zero firm_gain

Before

Var1	Freq
0	51542
220	41
330	86
440	417

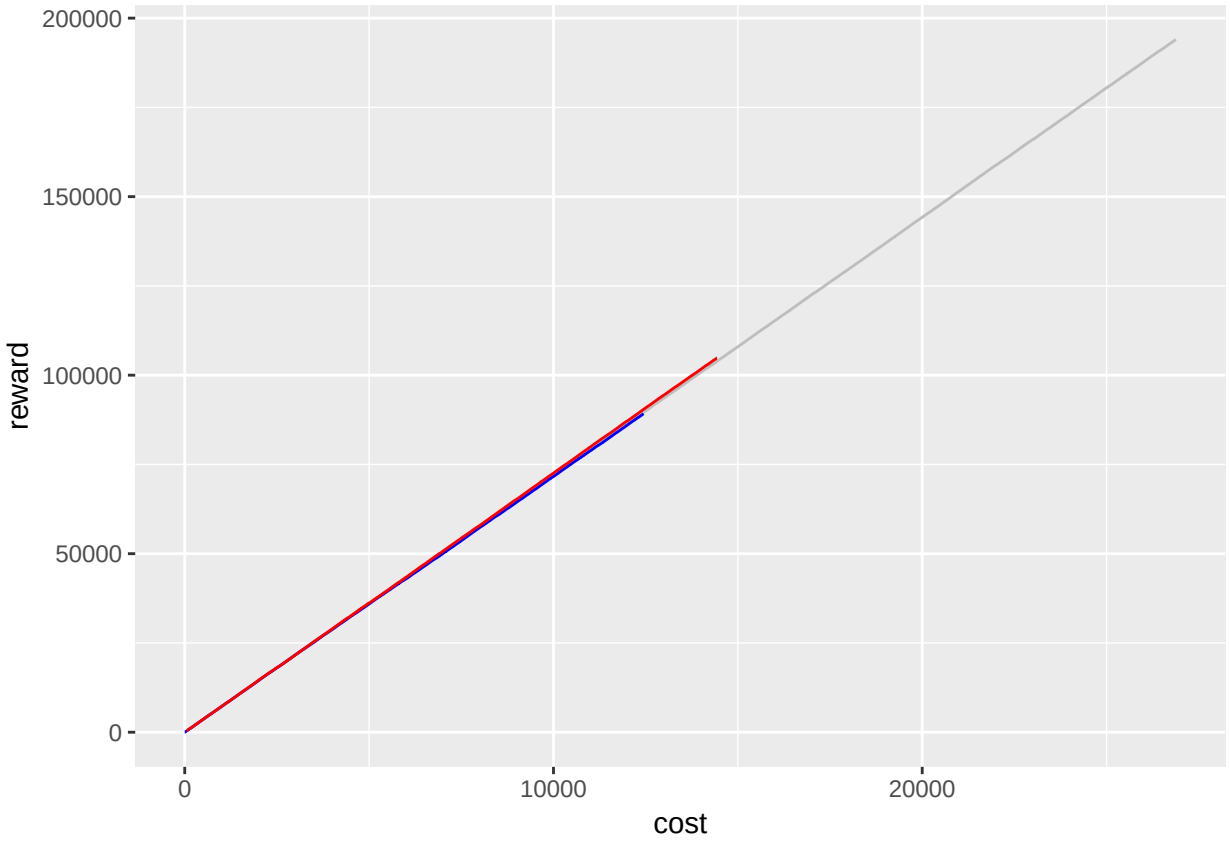
After

Var1	Freq
0	417
220	41
330	86
440	417

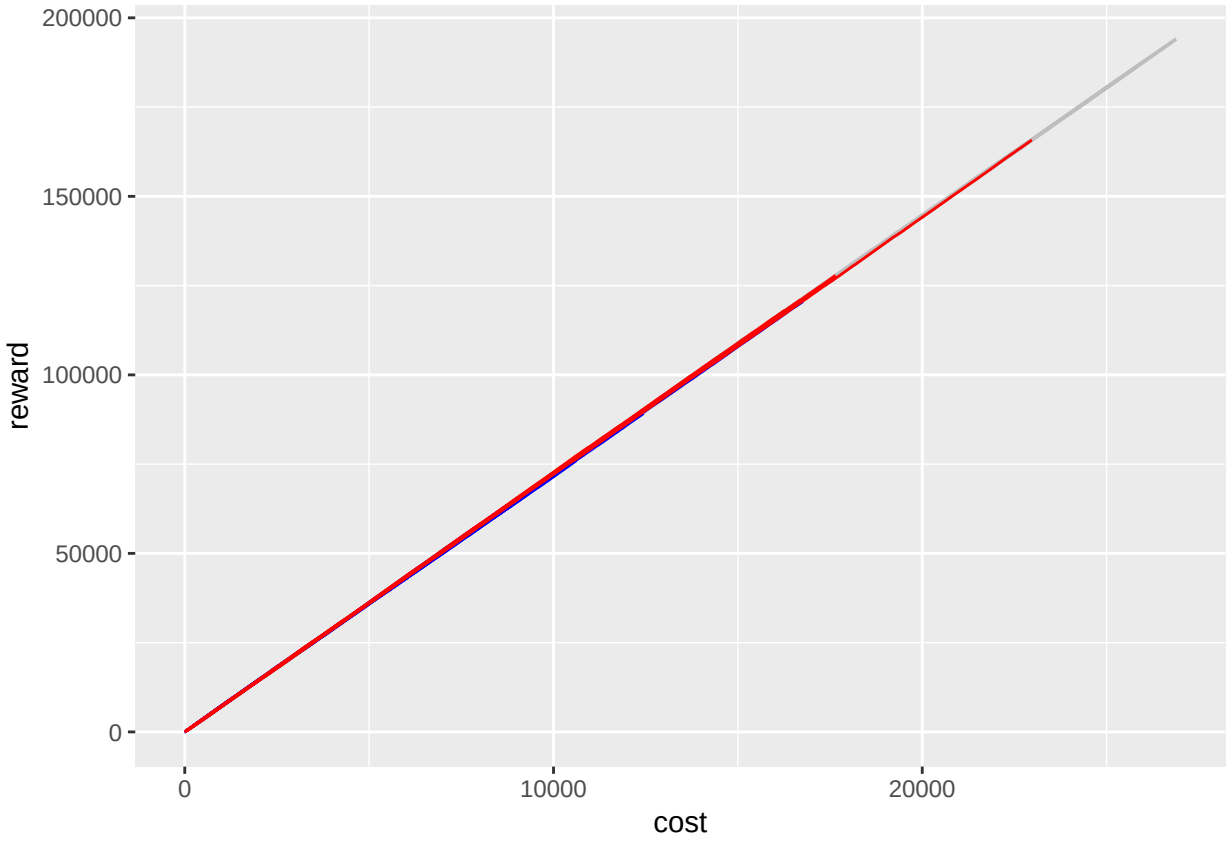

200000
150000
100000
50000
0
reward
0
10000
20000
cost

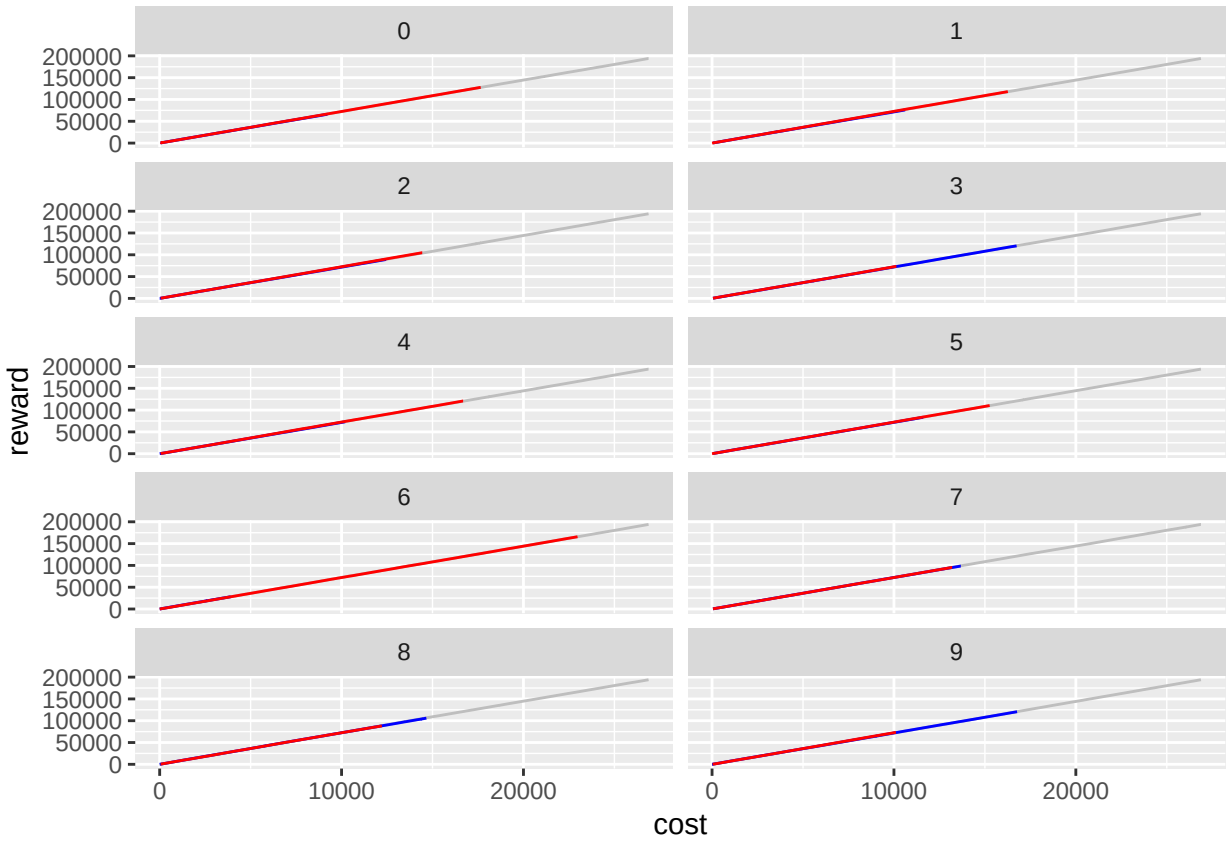
alpha = 1

iteration = 2



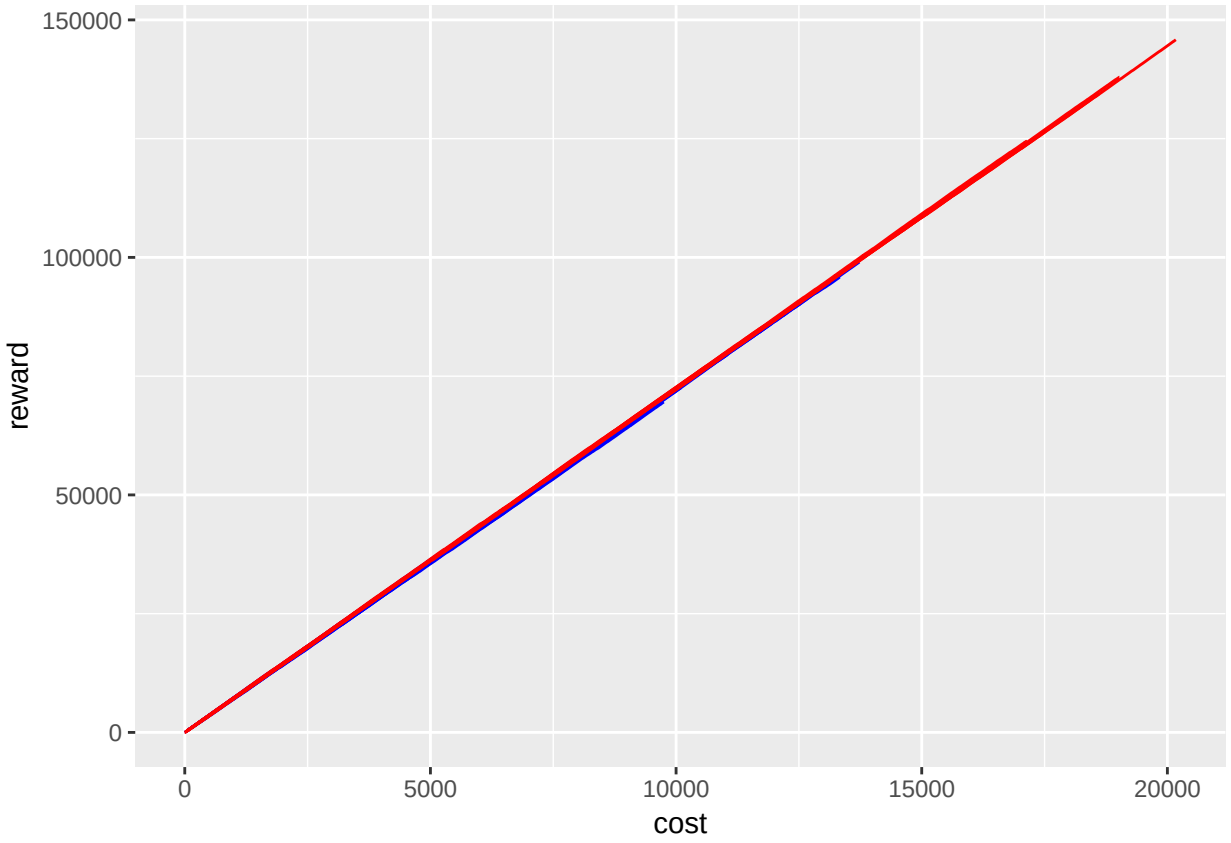
Just to be sure, we do it 10 different times

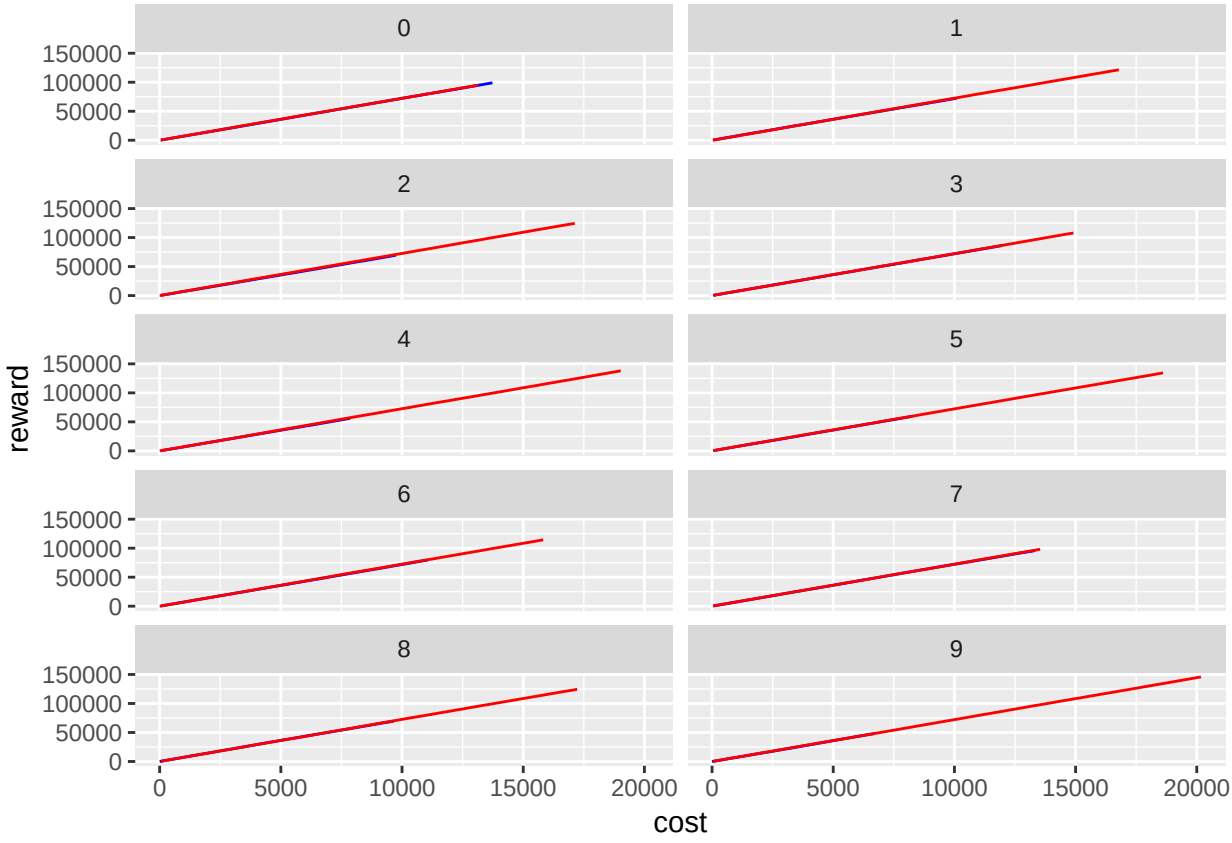

200000
150000
100000
50000
0
reward
0
10000
20000
cost



We can check it for other alphas

alpha=15


150000
100000
50000
0
reward
0
5000
10000
15000
20000
cost


0
1
2
3
4
5
6
7
8
9
150000
100000
50000
0
reward
0
5000
10000
15000
20000
cost