Problem Set 4

刘双铖

521021910904

Problem1

Greedy

Algorithm

When an online vertex v arrives, match v to an unmatched vertex $u \in N(v)$ (if such u exists), which maximizes w_u

Proof

If the Greedy algorithm is not the optimal, we can always find a vertex u that is matched in the optimal offline matching but not in the greedy algorithm. Consider vertex v^* that is matched to u in the optimal matching.

According to the Greedy algorithm, v^* must have been matched to a vertex u' such that $w_u \leq w_{u'}$, so that when v^* arrives, it matched to u' and u remains unmatched. Therefore, we can charge the loss of w_u in the optimal algorithm to u'.

Note that each u' is charged for the loss only once by its optimal partner in the matching, so that the loss in optimal matching is no more than the value of matching output by Greedy algorithm, indicating $\frac{w_{u'}}{w_u+w_{u'}} \geq \frac{1}{2}$. Hence Greedy algorithm achieves a competitive ratio of 0.5

Water-Filling

Background

Consider the online vertices as buyers and the offline vertices as goods to better explain the algorithm. Consider the **Primal**:

- maximize $\sum_{(u,v)\in E} w_v x_{u,v}$
- for each buyer u, $\sum_{v\in N(u)} x_{u,v} \leq 1$
- for each good v, $\sum_{u\in N(v)} x_{u,v} \leq 1$
- $x_{u,v} \geq 0$

为了与课上笔记对应，令 u 代表online vertex (buyer), v 代表 offline vertex (good)

Dual:

- minimize $\sum \alpha_u + \sum \alpha_v$
- $\forall u, v : \alpha_u + \alpha_v \geq w_v$
- $\alpha_u, \alpha_v \geq 0$

Let x be the choice of waterfilling, which is feasible. If we can construct α:

- $obj(\alpha) = obj(x)$
- α is approximately feasible, i.e. $\forall u, v : \alpha_u + \alpha_v \geq (1-\frac{1}{e})w_v$

Then we will have:

$ALG = obj(x) = obj(\alpha) \geq (1 - \frac{1}{e})obj(x^*) = (1 - \frac{1}{e})OPT$

So that algorithm achieves a competitive ratio of $1 - \frac{1}{e}$

Now we need to prove that by waterfilling algorithm, we have $\forall (u, v) \in E, \alpha_u + \alpha_v \geq (1 - \frac{1}{e})w_v$

Set $g(l)$ introduced in our last class as $g(l) := e^{l-1}$ (increasing func), where l denotes the water level.

Algorithm

When u arrives, let $l_v := \sum_{u' \in N(v)} x_{u',v}$ be the total amount of good v that has been sold so far.

To maximize the total utility, the arriving buyer u solves the following program:

- $\max_{(x_{u,v})v \in N(u)} \sum_{v \in N(u)} \int_{l_v}^{l_v + x_{u,v}} w_v(1 - e^{l-1})dl$
- subject to $\sum_{v \in N(u)} x_{u,v} \leq 1$
- $x_{u,v} \geq 0, \forall v \in N(u)$

Where u pays $\int_{l_v}^{l_v + x_{u,v}} w_v e^{l-1} dl$ for each good v. The waterfilling algorithm is that each arriving u computes the optimal solution $x_{u,v}(v \in N(u))$ to the program.

Proof

- Let l^* be the water level of good v at the end of the waterfilling algorithm, then we have $\alpha_v = \int_0^{l^*} w_v e^{l-1} dl$
- Let l_v be the initial water level before u arrives, then α_u is the obj function value of the convex program and $\alpha_u \geq w_v - w_v e^{l_v + x_{u,v} - 1}$, otherwise the solution could be improved by increasing $x_{u,v}$ with some small $\epsilon > 0$, and decreasing $x_{u,v'}$ for some $v' \in N(u)$ to meet the request that $\sum_{j \in N(u)} x_{u,j} \leq 1$

Combining the results above, we have:

$\alpha_u + \alpha_v \geq w_v(\int_0^{l^*} e^{l-1} dl + 1 - e^{l_v + x_{u,v} - 1}) \geq w_v(e^{l^* - 1} - \frac{1}{e} + 1 - e^{l^* - 1}) = (1 - \frac{1}{e})w_v$

Thus the competitive ratio is $1 - \frac{1}{e}$, better than 0.5

Problem 2

From the target function, we can tell the matching is unweighted.

Greedy

Algorithm

Wait until the ddl of each arriving vertex x to make the decision: arbitrarily match x to y, $y \in N(x)$ and y is unmatched (if such y exists).

Proof

If the Greedy algorithm is not the optimal, we can find a vertex u that is matched in the OPT matching but not in the greedy algorithm. Consider vertex v^* that is matched to u in the optimal matching.

According to the Greedy algorithm, v^* must have been matched to a vertex u'. Therefore, we can charge the loss of u in the optimal algorithm to u'. (OPT also obeys the ddl rule, so we do not need to worry that u has not arrived when v^* arrives)

Note that each u' is charged for the loss only once, and when u(matched in OPT) remains unmatched in the Greedy algorithm, the partner of u in OPT(v^*) must be matched in the Greedy algorithm. So that the loss in optimal matching is no more than the value of matching output by Greedy algorithm. Hence Greedy algorithm achieves a competitive ratio of 0.5

Water-Filling

Background

Consider **Primal**:

- maximize $\sum_{(u,v)\in E} x_{u,v}$
- for each buyer u, $\sum_{v\in N(u)} x_{u,v} \leq 1$
- for each good v, $\sum_{u\in N(v)} x_{u,v} \leq 1$
- $x_{u,v} \geq 0$

Dual:

- minimize $\sum \alpha_u + \sum \alpha_v$
- $\forall u, v : \alpha_u + \alpha_v \geq 1$
- $\alpha_u, \alpha_v \geq 0$

We need to prove that by waterfilling algorithm, we have $\forall (u, v) \in E, \alpha_u + \alpha_v \geq 2 - \sqrt{2}$, so that we can prove the algorithm achieves a competitive ratio of $2 - \sqrt{2}$

Set $g(l)$ introduced in our last class as $g(l) := \frac{1}{\sqrt{2}} l + 1 - \frac{1}{\sqrt{2}}$ (increasing func), where l denotes how much of the vertex has been matched so far.

Algorithm

Each vertex u acts as a good until its deadline at which point it acts as a buyer. Accordingly, each buyer computes the optimal solution ($\triangle x_{u,v}(\forall v \in N(u))$) to the following program:

- $\max_{(\triangle x_{u,v})v\in N(u)} \sum_{v\in N(u)} (\triangle x_{u,v} - \int_{l_v}^{l_v+\triangle x_{u,v}} (\frac{1}{\sqrt{2}} l + 1 - \frac{1}{\sqrt{2}}) dl)$
- subject to $\sum_{v\in N(u)} \triangle x_{u,v} \leq 1 - l_u$
- $\triangle x_{u,v} \geq 0, \forall v \in N(u)$

update $x \leftarrow x + \triangle x$

The idea of the algorithm is to maximize the utility when the vertex u acts as buyer and pays $\int_{l_v}^{l_v+\triangle x_{u,v}} (\frac{1}{\sqrt{2}} l + 1 - \frac{1}{\sqrt{2}}) dl$ to all the neighbor $v \in N(u)$

Proof

When the ddl of vertex u arrives, we need to prove that at the end of our waterfilling algorithm, we have $\alpha_u + \alpha_v \geq 2 - \sqrt{2}$, where α_u denotes the gain of the vertex u, which is the sum of profit as a good and utility as a buyer.

After the ddl of u, $\forall (u, v) \in E$, we can tell that either $l_u = 1$ or $l_v = 1$, otherwise u could buy more from v.

- When $l_v = 1$, then we have $\alpha_v = \int_0^1 (\frac{1}{\sqrt{2}} l + 1 - \frac{1}{\sqrt{2}}) dl = 1 - \frac{\sqrt{2}}{4} > 2 - \sqrt{2}$

- When $l_u = 1$, then we have $l_v < 1$. Let l'_u denote how much of u was bought when acting as a good before its ddl. So we have that vertex u gains $\int_0^{l'_u}(\frac{1}{\sqrt{2}}l + 1 - \frac{1}{\sqrt{2}})dl$ from previous matchings. Also u gains at least $1 - g(l_v)$ per unit as it could always buy from v. So we have: $\alpha_u \geq \int_0^{l'_u}(\frac{1}{\sqrt{2}}l + 1 - \frac{1}{\sqrt{2}})dl + (1 - l'_u)(1 - (\frac{1}{\sqrt{2}}l_v + 1 - \frac{1}{\sqrt{2}}))$

Since $\alpha_v = \int_0^{l_v}(\frac{1}{\sqrt{2}}l + 1 - \frac{1}{\sqrt{2}})dl,$ we have:

$$\begin{aligned}\alpha_u + \alpha_v &\geq \int_0^{l'_u}(\frac{1}{\sqrt{2}}l + 1 - \frac{1}{\sqrt{2}})dl + (1 - l'_u)(1 - (\frac{1}{\sqrt{2}}l_v + 1 - \frac{1}{\sqrt{2}})) + \int_0^{l_v}(\frac{1}{\sqrt{2}}l + 1 - \frac{1}{\sqrt{2}})dl \\ &\geq \frac{1}{2\sqrt{2}}(l'_u + l_v - 2 + \sqrt{2})^2 + 2 - \sqrt{2} \geq 2 - \sqrt{2}\end{aligned}$$

Thus the competitive ratio is $2 - \sqrt{2}$ (not tight) , better than 0.5

Gurobi Solver

Use the Gurobi Solver to solve the LP, I can get the following result:

```
a.py  ×
C: > Users > 86188 > Desktop > a.py > ...
Lp.Params.LogToConsole=True
Lp.optimize()

问题    输出    调试控制台    终端    端口

Barrier statistics:
 AA' NZ     : 5.999e+06
 Factor NZ  : 7.998e+06 (roughly 1.7 GB of memory)
 Factor Ops : 2.133e+10 (less than 1 second per iteration)
 Threads    : 12

                  Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0   6.27238558e+03 -9.73263666e-03  2.00e+02 1.49e-03  1.00e-02    19s
   1   2.38555662e+03 -1.44039162e-03  7.60e+01 3.08e-02  3.87e-03    20s
   2   3.38058559e+02  1.91322102e-03  1.08e+01 1.27e-02  5.51e-04    20s
   3   8.40160419e+01  4.36852983e-03  2.67e+00 4.66e-03  1.38e-04    21s
   4   4.17211744e+01  9.29346412e-03  1.33e+00 1.73e-03  7.05e-05    22s
   5   9.81655480e+00  1.17705640e-02  3.12e-01 1.10e-03  1.66e-05    23s
   6   5.09759523e+00  2.58100501e-02  1.62e-01 4.72e-04  8.74e-06    23s
   7   4.01957109e+00  3.35667432e-02  1.28e-01 4.12e-04  6.91e-06    24s
   8   3.05986320e+00  4.96072641e-02  9.72e-02 3.41e-04  5.27e-06    24s
   9   2.25713433e+00  9.54025984e-02  7.14e-02 2.51e-04  3.90e-06    25s
  10   2.03463223e+00  9.87595085e-02  6.42e-02 2.46e-04  3.49e-06    25s
  11   1.43434662e+00  1.44243794e-01  4.45e-02 2.16e-04  2.38e-06    26s
  12   1.26107516e+00  1.61416752e-01  3.83e-02 2.07e-04  2.06e-06    27s
  13   1.09661206e+00  1.69789217e-01  3.17e-02 2.04e-04  1.80e-06    28s
  14   9.30442956e-01  1.98774562e-01  2.43e-02 1.91e-04  1.54e-06    29s
  15   8.33682424e-01  2.97528631e-01  1.97e-02 1.41e-04  1.23e-06    30s
  16   7.40911686e-01  4.12705017e-01  1.43e-02 1.06e-04  9.14e-07    31s
  17   6.67658572e-01  5.02252825e-01  9.17e-03 7.96e-05  6.54e-07    31s
  18   6.28960962e-01  5.54271542e-01  4.97e-03 4.48e-05  3.89e-07    32s
  19   6.07870457e-01  5.71940786e-01  1.87e-03 2.08e-05  1.82e-07    33s
  20   5.95847691e-01  5.83714127e-01  6.17e-04 3.52e-06  5.80e-08    34s
  21   5.89632174e-01  5.85130993e-01  1.91e-04 1.14e-06  2.06e-08    35s
  22   5.88062959e-01  5.85376487e-01  1.14e-04 4.92e-07  1.22e-08    35s
  23   5.87854287e-01  5.85387571e-01  1.04e-04 4.41e-07  1.12e-08    36s
  24   5.86264994e-01  5.85430573e-01  3.32e-05 1.32e-07  3.76e-09    36s
  25   5.85707257e-01  5.85470106e-01  9.01e-06 3.27e-08  1.06e-09    37s
  26   5.85555425e-01  5.85472172e-01  3.01e-06 1.56e-08  3.73e-10    37s
  27   5.85533569e-01  5.85472465e-01  2.16e-06 1.58e-08  2.74e-10    38s

Barrier performed 27 iterations in 37.89 seconds (44.36 work units)
Barrier solve interrupted - model solved by another algorithm


Solved with primal simplex
Iteration    Objective       Primal Inf.    Dual Inf.      Time
   50796    5.8547638e-01   0.000000e+00   0.000000e+00     40s

Solved in 50796 iterations and 40.00 seconds (48.98 work units)
Optimal objective  5.854763810e-01
PS C:\Users\86188>
```

The competitive ratio reaches 0.585476, smaller than $2-\sqrt{2}$ but larger than 0.5, we are supposed to find some competitive ratio better than $2-\sqrt{2}$, but the solver can not run more iterations due to memory restriction.

C: > Users > 86188 > Desktop > a.py > ...

```python
import gurobipy as gp
from gurobipy import GRB
Lp = gp.Model('OBM')
n = 5000

ratio = Lp.addVar(lb=0)
Lp.setObjective(ratio, GRB.MAXIMIZE)
```

问题 2 输出 调试控制台 终端 端口

```
Optimal objective  5.851675228e-01
PS C:\Users\86188> & C:/Users/86188/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/86188/Desktop/a.py
Set parameter Username
Academic license - for non-commercial use only - expires 2025-01-14
Gurobi Optimizer version 11.0.0 build v11.0.0rc2 (win64 - Windows 11.0 (22621.2))

CPU model: 12th Gen Intel(R) Core(TM) i7-12700H, instruction set [SSE2|AVX|AVX2]
Thread count: 14 physical cores, 20 logical processors, using up to 20 threads

Optimize a model with 25005002 rows, 10003 columns and 100000003 nonzeros
Model fingerprint: 0x39709771
Coefficient statistics:
  Matrix range     [2e-04, 2e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e-04, 1e+00]
Presolve removed 0 rows and 0 columns (presolve time = 7s) ...
Presolve removed 4 rows and 3 columns (presolve time = 12s) ...
Presolve removed 4 rows and 3 columns (presolve time = 17s) ...
Presolve removed 6 rows and 5 columns (presolve time = 21s) ...
Presolve removed 6 rows and 5 columns (presolve time = 30s) ...
Presolve removed 6 rows and 5 columns (presolve time = 36s) ...
Presolve removed 6 rows and 5 columns (presolve time = 40s) ...
Presolve removed 6 rows and 5 columns (presolve time = 45s) ...
Presolve removed 6 rows and 5 columns
Presolve time: 118.22s
Presolved: 9999 rows, 25004998 columns, 99989996 nonzeros

Concurrent LP optimizer: primal simplex, dual simplex, and barrier
Showing barrier log only...

Warning: Possible non-determinism after error

Out of memory
Traceback (most recent call last):
  File "c:\Users\86188\Desktop\a.py", line 32, in <module>
    Lp.optimize()
  File "src\\gurobipy\\model.pxi", line 893, in gurobipy.Model.optimize
gurobipy.GurobiError: Out of memory
PS C:\Users\86188> 
```