Scheduling K classes of jobs with a deterministic oracle

November 20, 2024

1 Job model

- n jobs
- K classes of jobs
- jobs in class k have processing time $p(k)$
- assume that $p(1) \leq p(2) \leq \cdots \leq p(K)$
- no preemption

2 Oracle model

Assume we do not know in which class a given job is. We consider we have access to a cheap oracle, giving the class of a job. The oracle is not perfect, i.e., it may give a wrong class for a given job. The oracle is deterministic, i.e., it always gives the same answer for a given job, no matter if its prediction is correct or not.

We may represent this kind of oracle as a $K \times K$ matrix E, such that the entry e_{ij} on the i-th row and the j-th column represents the number of jobs that the oracle believes to be in class i, whereas they are in reality in class j.

Example 1. *Consider the following oracle for $K = 3$ classes of jobs:*

$$E = \begin{pmatrix} 1 & 1 & 3 \\ 2 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix}.$$

The oracle believes that:

- *5 jobs are in class 1, among which 1 job is actually in class 2, and 3 jobs are in class 3;*
- *5 jobs are in class 2, among which 2 jobs are actually in class 1, and 1 job is in class 3;*

- *4 jobs are in class 3, among which 1 job is actually in class 1, and 1 job is in class 2.*

We note J_{ij} the set of jobs that the oracle believes to be in class i but are actually in class j (thus, $e_{ij} = |J_{ij}|$). We note A_k (resp. B_k) the set of jobs that are actually in class k (resp. the set of jobs that the oracle believes to be in class k), i.e., $A_k = \bigcup_{i=1}^{K} J_{ik}$ and $B_k = \bigcup_{j=1}^{K} J_{kj}$. We also note a_k (resp. b_k) the number of jobs in A_k (resp. B_k), i.e., $a_k = |A_k| = \sum_{i=1}^{K} e_{ik}$ and $b_k = |B_k| = \sum_{j=1}^{K} e_{kj}$.

2.1 Bloom filter model

For 2 classes of jobs, the oracle may consist in a probabilistic data structure called a Bloom filter, which represents a set. Assume a Bloom filter represents the set of *large* jobs. One property of Bloom filters is that they may produce false positives (i.e., the BF believes that a job is large while it is small), but never produce false negatives.

Hence, a Bloom filter oracle can always be represented by a matrix of the following form:

$$\begin{pmatrix} a & 0 \\ b & c \end{pmatrix}.$$

3 Minimizing the sum of completion times

Consider the problem $1||\sum C_j$.

3.1 Clairvoyant model

The optimal algorithm in the clairvoyant model is SPT.

Let Δ_{ij} be the minimum starting time of jobs in J_{ij}, i.e.,

$$\Delta_{ij} = \sum_{i'=1}^{K} \sum_{j'=1}^{j-1} e_{i'j'} \cdot p(j') + \sum_{i'=1}^{i-1} e_{i'j} \cdot p(j).$$

Hence,

$$\text{SPT} = \sum C_j = \sum_{i=1}^{K} \sum_{j=1}^{K} e_{ij} \cdot \Delta_{ij} + \frac{e_{ij}(e_{ij}+1)}{2} \cdot p(j).$$

3.2 Non-clairvoyant model

In the non-clairvoyant model, simply shuffling jobs yields

$$\begin{aligned}\text{SHUF} = E(\sum C_j) &= E(P_1 + (P_1 + P_2) + \cdots + (P_1 + \cdots + P_n)) \\ &= nE(P_1) + (n-1)E(P_2) + \cdots + E(P_n) \\ &= \frac{n(n+1)}{2} E(P_i) \\ &= \frac{n+1}{2} \sum p_j,\end{aligned}$$

where each P_i is a random variable denoting the processing time of the i-th scheduled job (we have $P(P_i = p(k)) = \frac{a_k}{n}$ for any i, k, hence $E(P_i) = \sum_{k=1}^{K} \frac{a_k p(k)}{n}$ for any i).

3.3 Using the oracle

Let us see what is the worst possible objective if we fully trust the oracle, i.e., we try to apply the SPT algorithm with the given predictions.

This time, we have

$$\Delta_{ij} = \sum_{i'=1}^{i-1} \sum_{j'=1}^{K} e_{i'j'} \cdot p(j') + \sum_{j'=j+1}^{K} e_{ij'} \cdot p(j'),$$

and

$$\sum C_j \leq \sum_{i=1}^{K} \sum_{j=1}^{K} e_{ij} \cdot \Delta_{ij} + \frac{e_{ij}(e_{ij}+1)}{2} \cdot p(j).$$

Another possibility is to shuffle each set B_k independently, in order to try to avoid putting large jobs first.

Let Δ_i denote the minimum starting time of each job in B_i, i.e.,

$$\Delta_i = \sum_{i'=1}^{i-1} \sum_{j=1}^{K} e_{i'j} \cdot p(j).$$

We have $E(\sum C_j) = E(\sum_{j \in B_1} C_j + \sum_{j \in B_2} C_j + \cdots + \sum_{j \in B_K} C_j) = \sum_{i=1}^{K} E(\sum_{j \in B_i} C_j)$.

Moreover,

$$\begin{aligned}E(\sum_{j \in B_i} C_j) &= b_i \cdot \Delta_i + E(P_1 + (P_1 + P_2) + \cdots + (P_1 + \cdots + P_{b_i})) \\ &= b_i \cdot \Delta_i + \frac{b_i(b_i+1)}{2} \cdot \sum_{k=1}^{K} \frac{e_{ik}}{b_i} \cdot p(k) \\ &= b_i \cdot \Delta_i + \frac{b_i+1}{2} \sum_{j \in B_i} p_j.\end{aligned}$$

Hence,

$$\text{SHUFOR} = E(\sum C_j) = \sum_{i=1}^{K} b_i \cdot \Delta_i + \frac{b_i + 1}{2} \sum_{j \in B_i} p_j.$$

Note that this algorithm is not robust: consider the extreme case where the oracle is completely wrong, i.e., it is represented by an anti-diagonal matrix. The algorithm will schedule jobs in non-increasing order of processing times. In this case, it would have been better to shuffle all jobs together.

Question: which class of oracles yields SHUFOR $\leq$ SHUF?

A better algorithm could consists in picking e_{i1} jobs randomly in each set B_i, and schedule them first. Then, pick e_{i2} jobs randomly in each B_i, and schedule them in second. Repeat the procedure until the last column.

TODO: give the expression of $E(\sum C_j)$.

4 Some algorithms

4.1 A DP

Find the best possible value of $\mathbb{E}(\sum_{j\in \text{Jobs}} Flowtime_j)$. Let be E the state :

$$E = \begin{pmatrix} a_{1\ 1} & 0 & ... & 0 & 0 \\ a_{2\ 1} & a_{2\ 2} & ... & 0 & 0 \\ ... & ... & ... & ... & ... \\ a_{m-1\ 1} & a_{m-2\ 2} & ... & a_{m-1\ m-1} & 0 \\ a_{m\ 1} & a_{m\ 2} & ... & a_{m\ m-1} & a_{m\ m} \end{pmatrix}.$$

The DP choose a task from non-null line k that minimises g(k), such as :

$$g(k) = \sum_{i\in[0,m]} \left(\frac{a_{ki}}{\sum_{j\in[0,m]} a_{kj}} \cdot \left(\sum_{\ell<i} \left((p_i - p_\ell) \sum_{q\in[1,m]} a_{q\ell} \right) + \text{Solve} \begin{pmatrix} a_{1\ 1} & 0 & ... & 0 & 0 \\ ... & ... & ... & ... & ... \\ a_{k1} & ... & a_{ki}-1 & ... & 0 \\ ... & ... & ... & ... & ... \\ a_{m\ 1} & a_{m\ 2} & ... & a_{m\ m-1} & a_{m\ m} \end{pmatrix} \right) \right)$$

On an example.

$$\text{S}\begin{pmatrix} 0 & & \\ 0 & 1 & \\ 4 & 0 & 2 \end{pmatrix} = \min \begin{cases} \text{I can't choose a job from line 1.} \\ \text{If I choose a job from the line 2, i will pay :} \\ \quad \frac{1}{1} \cdot \left(4 \cdot (p_2 - p_1) + \text{S}\begin{pmatrix} 0 & & \\ 0 & 0 & \\ 4 & 0 & 2 \end{pmatrix} \right) \\ \text{If I choose a job from line 3, i will pay :} \\ \quad \frac{4}{4+2} \left(0 + \text{S}\begin{pmatrix} 0 & & \\ 0 & 1 & \\ 3 & 0 & 2 \end{pmatrix} \right) + \frac{2}{4+2} \left(4 \cdot (p_3 - p_1) + 1 \cdot (p_3 - p_2) + \text{S}\begin{pmatrix} 0 & & \\ 0 & 1 & \\ 4 & 0 & 1 \end{pmatrix} \right) \end{cases}$$

And solving a matrix with no uncertainty gives 0.

Complexity

Let n the number of jobs and m the number of class for a job. The number of different matrices the dynamic program have to compute is $c = \prod_{i\in[1,m]} \prod_{j\in[1,m]} (a_{ij} + 1)$, where $\sum a_{ij} = n$. Nash equilibrium says the biggest value of c is achieved where all values of a_{ij} are equal. The number of relevant values (that are not forced to be equal to zero) in the matrix is $\frac{m(m+1)}{2}$. Since we are in integer domain, we can define an instance where n is divible by $\frac{m(m+1)}{2}$, so all values of a_{ij} are equal. So number of matrices the DP have to compute is near $\frac{n}{0.5m(m+1)}^{0.5m(m+1)}$). This is not polynomial, even if we impose hypothesis like number of class m is

bounded by $log(n)$.
(We can argue when there is no uncertainty remaining, we can directly say S(matrix)=0, and maybe consider $\frac{1}{2}m(m-1)$ instead of $\frac{1}{2}m(m+1)$?)

4.2 A naive heuristic

Choose the line with less pondarate average.

Counter example : $p_1 = 1.1$, $p_2 = 3.01$ and $p_3 = 5.001$.

For this matrix $\begin{pmatrix} 0 & & \\ 0 & 1 & \\ 1 & 0 & 1 \end{pmatrix}$, the average cost of choosing a task from line 2 is $p_2 = 3.01$, and the average cost from line 3 is $\frac{p_1+p_3}{2} = 3.0505$. This heuristic choose line 2 while the DP will find the minimal expected cost with choosing line 3.

$$\mathrm{S}\begin{pmatrix} 0 & & \\ 0 & 0 & \\ 1 & 0 & 1 \end{pmatrix} = \frac{1}{2}\left(0 + \mathrm{S}\begin{pmatrix} 0 & & \\ 0 & 0 & \\ 0 & 0 & 1 \end{pmatrix}\right) + \frac{1}{2}\left((5.001 - 1.1) + \mathrm{S}\begin{pmatrix} 0 & & \\ 0 & 0 & \\ 0 & 0 & 1 \end{pmatrix}\right) = 1.9505$$

$$\mathrm{S}\begin{pmatrix} 0 & & \\ 0 & 1 & \\ 1 & 0 & 1 \end{pmatrix} = \min \begin{cases} \frac{1}{1} \cdot \left((3.01 - 1.1) + \mathrm{S}\begin{pmatrix} 0 & & \\ 0 & 0 & \\ 1 & 0 & 1 \end{pmatrix}\right) = 3.8605 \\ \frac{1}{2}\left(0 + \mathrm{S}\begin{pmatrix} 0 & & \\ 0 & 1 & \\ 0 & 0 & 1 \end{pmatrix}\right) + \frac{1}{2}\left((5.001 - 1.1) + (5.001 - 3.01) + \mathrm{S}\begin{pmatrix} 0 & & \\ 0 & 1 & \\ 1 & 0 & 0 \end{pmatrix}\right) = 2.946 \end{cases}$$