Online Linear Programming and Resource Allocation

Xiao Chen, Bella Shi, Shuyi Yin

1 Introduction

The context of the subsequent problems can be found at `https://web.stanford.edu/class/msande311/307project2018-4.pdf`

The resource allocation problem considers the following linear programming problem:

$$\max_x \sum_{j=1}^{n} \pi_j x_j \tag{1}$$

$$s.t. \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \qquad \forall i = 1, ..., m \tag{2}$$

$$0 \leq x_j \leq 1, \qquad \forall j = 1, ..., n \tag{3}$$

where π_j is the revenue to allocate a combination of resources to bidder j, a_{ij} is the requested quantity of resource i by bidder j, and b_i is the total known available quantity of resource i. The decision variable x_j can be interpreted as how much portion of the j-th bid should be allocated.

2 Convex Parimutuel Call Auction Mechanism

[This section is mainly in response to question1]

The dual price may not be unique when solving the original problem (1)-(3). One approach to get a unique price is to add an additional term in the objective function when solving the off-line model. We particularly analyze the model called *Convex Parimutuel Call Auction Mechanism* (CPCAM) that is defined as follows

$$\max_{\mathbf{x},\mathbf{s}} \sum_j \pi_j x_j + u(\mathbf{s}) \tag{4}$$

$$s.t. \quad \sum_j a_{ij} x_j + s_i = b_i, \quad \forall i = 1, \ldots, m \tag{5}$$

$$x_j \geq 0, \quad \forall j = 1, \ldots, n \tag{6}$$

$$x_j \leq 1, \quad \forall j = 1, \ldots, n \tag{7}$$

$$s_i \geq 0, \quad \forall i = 1, \ldots, m \tag{8}$$

where $u(\mathbf{s}) = u(s_1, \ldots, s_m)$ is increasing and strictly concave. The gradient entry $\frac{\partial u}{\partial s_i}$ is sufficiently large for all i.

Solution:
We first write out the KKT conditions for the CPCAM problem. Denote the dual variables λ for constraints (5), dual variables $\mu \geq 0, \nu \geq 0, w \geq 0$ for constraints (6) - (8) respectively, we have the KKT conditions [see details in

Submitted CME 307/MS&E311 (markcx@stanford.edu), Stanford, CA.

appendix 7.1] as follows

$$x_j(\pi_j - \sum_i \lambda_i a_{ij} - \nu_j) = 0, \quad \forall j \tag{9}$$

$$s_i \frac{\partial u(\mathbf{s})}{\partial s_i} - \lambda_i s_i = 0, \quad \forall i \tag{10}$$

$$\sum_j a_{ij} x_j + s_i - b_i = 0, \quad \forall i \tag{11}$$

$$\nu_j(x_j - 1) = 0, \quad \forall j \tag{12}$$

$$x_j \geq 0, \quad \forall j \tag{13}$$

$$x_j \leq 1, \quad \forall j \tag{14}$$

$$\lambda \text{ is free} \tag{15}$$

$$s_i \geq 0, \quad \forall j \tag{16}$$

$$\nu_j \geq 0, \quad \forall j \tag{17}$$

$$\nabla u - \boldsymbol{\lambda} \leq 0 \tag{18}$$

$$\boldsymbol{\pi} - A^T \boldsymbol{\lambda} - \boldsymbol{\nu} \leq 0 \tag{19}$$

Since the objective (4) is strictly concave and the feasible set (5)-(8) is convex. It automatically indicates that the KKT conditions are sufficient to yield the optimal solution. Now we prove that the CPCAM has unique price $\boldsymbol{\lambda}$ (which is also the multiplier for the equality constraints (5)).

Theorem 1 *If the CPCAM problem is feasible, it has a unique price λ.*

Proof of Theorem 1: To show the uniqueness of λ, we equivalently show the CPCAM has a unique maximizer $(\mathbf{x}^*, \mathbf{s}^*)$. We demonstrate it through contradiction procedure. Define function $g(\mathbf{x}, \mathbf{s}) = \boldsymbol{\pi}^T \mathbf{x} + u(\mathbf{s})$ which is known to be strictly concave. We denote two maximizer (x_1, s_1) and (x_2, s_2) which give same maximum value $g(x_1, s_1) = g(x_2, s_2)$. Given a scaler $0 < \alpha < 1$, we have

$$g(x_1, s_1) = \alpha g(x_1, s_1) + (1-\alpha) g(x_1, s_1) = \alpha g(x_1, s_1) + (1-\alpha) g(x_2, s_2) \tag{20}$$

$$< g\big(\alpha x_1 + (1-\alpha) x_2, \alpha s_1 + (1-\alpha) s_2\big) \tag{21}$$

This violates the assumption that $g(x_1, s_1)$ holds the maximum value. Thus the CPCAM has a unique maximizer (x^*, s^*)

The interpretation of s and $u(\mathbf{s})$ can be considered as follows. s_i is the remaining resource i after clearing the n bids in the market. Function $u(\mathbf{s})$ captures the "future value" of these remaining resources (collected from s_1 to s_m).

3 Online Model - Sequential Convex Parimutuel Mechanism

[This section is mainly in response to question2]

The disadvantage of the offline model is that it can't tell the bidders whether their bids are accepted or not until the market closes. This is undesirable since sometimes the bidders want to know the results to their bids immediately so that they can modify their bids and submit again. Therefore, in practice, the market is usually implemented in an online version. One popular model called Sequential Convex Parimutuel Mechanism (SCPM) is defined as follows. Whenever k-th bidder submits a bid, the market maker solves the following optimization problem:

$$\max_{x_k, \mathbf{s}} \pi_k x_k + u(\mathbf{s}) \tag{22}$$

$$s.t. \quad a_{ik} x_k + s_i = b_i - q_i^{k-1}, \quad \forall i = 1, \ldots, m \tag{23}$$

$$0 \leq x_k \leq 1 \tag{24}$$

$$s_i \geq 0, \quad \forall i = 1, \ldots, m \tag{25}$$

where $q_i^{k-1} = \sum_{j=1}^{k-1} a_{ij} \bar{x}_j$ is the resource i that is already allocated before the k-th bidder arrives.

Solution:
We express the KKT conditions for the online model as follows [see appendix 7.2 for details], given

$\mu \geq 0, \nu \geq 0, w \geq 0$.

$$\pi_k - \sum_{i=1}^{m} \lambda_i a_{ik} + \mu_k - \nu_k = 0 \tag{26}$$

$$\frac{\partial u(\mathbf{s})}{\partial s_i} - \lambda_i + w_i = 0, \quad \forall i = 1, \ldots, m \tag{27}$$

$$a_{ik} x_k + s_i + q_i^{k-1} = b_i, \quad \forall i = 1, \ldots, m \tag{28}$$

$$\mu_k x_k = 0 \tag{29}$$

$$\nu_k (x_k - 1) = 0 \tag{30}$$

$$w_i s_i = 0, \quad \forall i = 1, \ldots, m \tag{31}$$

In the online model, the number of variables (x_k, $\mathbf{s}$, $\boldsymbol{\lambda}$, $\mu_k, \nu_k, \mathbf{w}$) are $3+3m$ with the corresponding $3+3m$ equations from KKT conditions at k-th step. Hence upto k-th bid, it solves $3k + 3km$ variables with $3k + 3mk$ equations. On the other hand, the original offline model has $3n + 3m$ variables $(\mathbf{x}, \mathbf{s}, \boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu})$ from $3n + 3m$ equations according to the KKT conditions in equation (36) - (41) in appendix 7.1. In the worst case when k approaches n, if the remaining resources are still available to be allocated, the online model may accumulatively solve $3n + 3mn$ variables with $3n + 3mn$ equations in total. However one subtlety for online model is that when $q_i^{k-1} = b_i$, i.e. the remaining resource i is not enough to satisfy future requests at step k and afterwards, then $x_t = 0$ with $a_{it} > 0$ for all $t \geq k$, and corresponding $s_i = 0$. Hence we can automatically determine the variables afterwards without solving the KKT system.

Proposition 1 *The online model would be more efficient than offline model given that the market maker clear up the resources at k-th bid, with the following condition that*

$$k \leq \frac{n+m}{1+m} \tag{32}$$

Proof of Proposition 1: This automatically follows the aforementioned analysis. Since the total cumulative computation task for online model upto k-th bid is $3k + 3mk$, and the offline model is $3n + 3m$. When $3k + 3mk \leq 3n + 3m$, the online model is more efficient than offline model, which completes the proof.

To further understand how x_k are determined through this online approach, we investigate the KKT optimality conditions in a reverse manner. Specifically we precondition x_k and check the corresponding variables and parameter relationships as follows.

1. If $x_k = 0$, we have $\nu_k = 0$. The first order gradient optimality can be reduced to $\pi_k - \sum_i \lambda_i a_{ik} - \mu_k = 0$. Since $\mu_k \geq 0$, we have $\pi_k - \sum_{i=1}^{m} \lambda_i a_{ik} \leq 0$
2. If $x_k = 1$, we have $\mu_k = 0$. It indicates $\pi_k - \sum_i \lambda_i a_{ik} + \nu_k = 0$. Since $\nu_k \geq 0$, we get $\pi_k - \sum_{i=1}^{m} \lambda_i a_{ik} \geq 0$
3. If $0 < x_k < 1$, then $\mu_k = 0, \nu_k = 0$. Moreover $\pi_k = \sum_{i=1}^{m} \lambda_i a_{ik}$

In addition, on one hand when $s_i > 0$, we have $\lambda_i = \frac{\partial u(\mathbf{s})}{s_i}$ and $w_i = 0$. On the other hand if $s_i = 0$, we have $b_i - q_i^{k-1} - a_{ik} x_k = b_i - q_i^k = 0$, and $\lambda_i \geq \frac{\partial u(\mathbf{s})}{s_i}$.

4 Online Model (SCPM)- Specific Utility Function

[This section is mainly in response to question3]

In this section, we implement two specific utility functions:

$$u_1(\mathbf{s}) = \frac{w}{m} \sum_i \log s_i \tag{33}$$

$$u_2(\mathbf{s}) = \frac{w}{m} \sum_i (1 - e^{-s_i}) \tag{34}$$

where $w = 1$ and $w = 10$, $m = 10$. We are interested in the following subquestions: a) Does the state price $\boldsymbol{\lambda}$ yielded by online SCPM converges to the true price $\bar{\mathbf{p}}$? b) Does the choice of w make a difference? c) Is there significant difference between using functions u_1 and u_2?

Solution:
To answer the first subquestion, we use the relative price gap between $\boldsymbol{\lambda}$ and $\bar{\mathbf{p}}$ that is $\frac{\|\lambda-\bar{p}\|}{\|\bar{p}\|}$ as a metric. Before directly giving the answer, we propose two algorithms to calculate the SCPM. One is using KKT condition updates and gradient decent type algorithm. The other is using existing CVX solver.

Algorithm 1 (SCPM) Using KKT updates

initialize $x_j = 0$, $\lambda_i = \frac{\partial u(\mathbf{b})}{\partial b_i} = \nabla u(\mathbf{b})_i$, $q_i = 0, \forall i = 1, \ldots, m$, $j = 1, \ldots n$ when $k = 0$.
Thus $\mathbf{q}^0 = \mathbf{0}, \boldsymbol{\lambda}^0 = \nabla u(\mathbf{b})$
for $k = 1$ to n **do**
  if $\pi_k - \nabla u(\mathbf{b} - \mathbf{q}^k)^T \mathbf{a}_k < 0$ **then**
    $x_k = 0$, $\boldsymbol{\lambda}^k = \boldsymbol{\lambda}^{k-1}$, $\mathbf{q}^k = \mathbf{q}^{k-1}$
  else if $\pi_k - \nabla u(\mathbf{b} - \mathbf{q})^T \mathbf{a}_k > 0$ **then**
    $x_k = 1$, $\boldsymbol{\lambda}^k = \nabla u(\mathbf{b} - \mathbf{q}^{k-1} - \mathbf{a}_k)$, $\mathbf{q}^k = \mathbf{q}^{k-1} + \mathbf{a}_k$
  else
    solve x_k through equation $\pi_k - \nabla u(\mathbf{b} - \mathbf{q}^{k-1} - x_k \mathbf{a}_k)^T \mathbf{a}_k = 0$ where $x_k \in (0, 1)$
    $\boldsymbol{\lambda}^k = \nabla u(\mathbf{b} - \mathbf{q}^{k-1} - x_k \mathbf{a}_k)$
  end if
end for

Algorithm 2 (SCPM) Using CVX

initialize $x_j = 0$, $\lambda_i = \frac{\partial u(\mathbf{b})}{\partial b_i} = \nabla u(\mathbf{b})_i$, $q_i = 0, \forall i = 1, \ldots, m$, $j = 1, \ldots n$ when $k = 0$.
Thus $\mathbf{q}^0 = \mathbf{0}, \boldsymbol{\lambda}^0 = \nabla u(\mathbf{b})$
for $k = 1$ to n **do**
  if $(\mathbf{q} \geq \mathbf{b})$ **then**
    Break
  else
    Get x_k, $\mathbf{s}^k$, $\boldsymbol{\lambda}^k$ from solving the SCPM problem in equation (22)-(25) by CVX.
  end if
  $\mathbf{q}^k = \mathbf{q}^{k-1} + \mathbf{a}_k x_k$
end for

The Figure 1 and Figure 2 show that the price converge to some point but not necessarily the true price. To have a better understanding about the convergence of price, we plot out the zoom-in level price gap in Figure 2. Regarding to the second and third subquestions, we describe the consequence of different choice of weights w and different choice of u in the caption of Figure 2. Yet we address the main discovery here again. One observed result is that higher value of w do give a faster convergence speed. Another observation is that different choice of utility function u won't affect the convergence of price too much until the market clear up the resource. Then different utility function could drive different price behavior.

5 Sequential Linear Parimutuel Mechanism (SLPM)

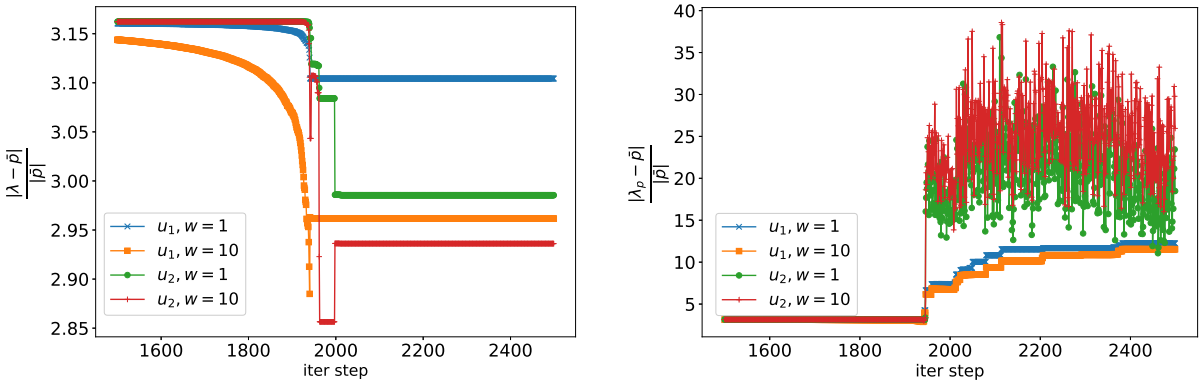
[This section is mainly in response to question4]

Suppose there is a good estimate of total n bidders in the market. Then we wait for the first k bidders arrived and solve the revealed partial linear program

$$\begin{array}{lll} \text{(SLPM):} & \text{maximize}_{x_1,\ldots,x_k} & \sum_{j=1}^{k} \pi_j x_j \\ & \text{s.t.} & \sum_{j=1}^{k} a_{ik} x_k \leq \frac{k}{n} b_i, \ \forall\, i = 1, 2, ..., m, \\ & & 0 \leq x_j \leq 1, \ \forall\, j = 1, ..., k. \end{array}$$

and then use the *dual prices*, say $\bar{\mathbf{y}}^k$ of the partial LP for future online decisions: $x_j = 1$, if $\pi_j > \mathbf{a}_j^T \bar{\mathbf{y}}^k$; and $x_j = 0$, otherwise.

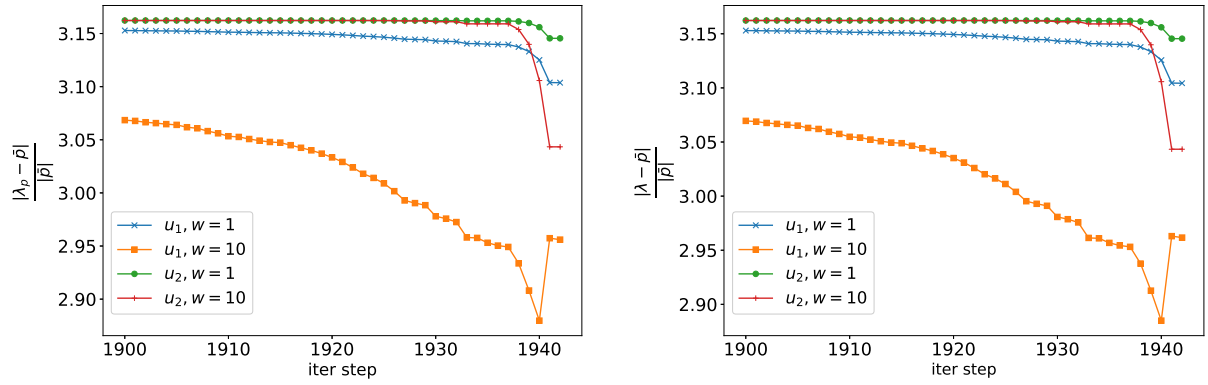
Solution:
Considering k is the look-ahead horizon, we would like to understand a) how the various k affects the dual price convergence? b) How sensitive it is in terms of cumulative revenue when setting different k values? Figure 3 explains the aforementioned questions. In panel(a) of figure 3, we observe that dual price converges to true price with the



(a) Price gap of KKT updates between iteration $k = 1500$ and $k = 2500$

(b) Price gap of CVX updates between iterations $k = 1500$ and $k = 2500$

Figure 1: A general view of dual price convergence with respect to true price. It hits the maximum allocatable resources at around step $k = 1943$. Panel (a) shows the KKT updates. When the market clear up the resources, we set the current step dual price as the same as last differentiable dual price (i.e. when $s_i > 0$); Panel (b) shows the CVX updates, when the market clear up the resource. Certain type of resource price could be high, given the situation that π_k is large. Thus the price going up after step $k = 1943$ also makes sense.

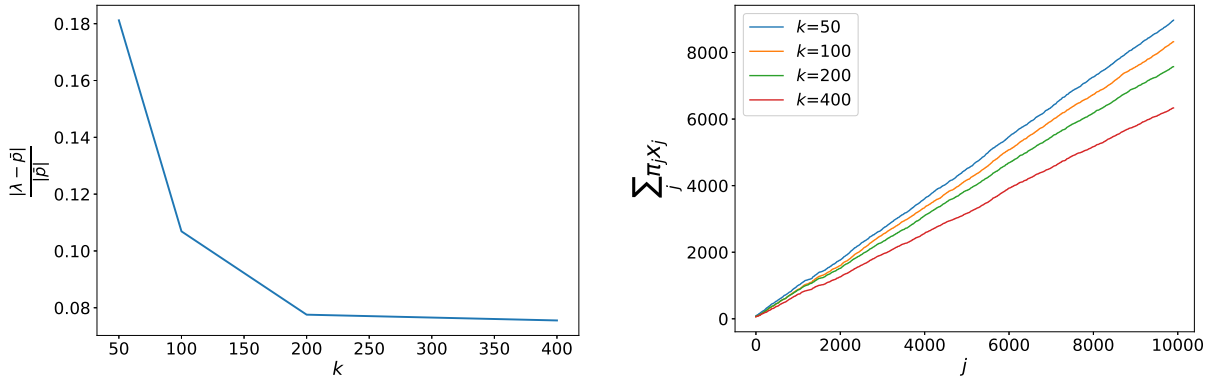


(a) KKT updates

(b) CVX updates

Figure 2: Both KKT and CVX method yield same consequence before hitting the constraints bound (i.e. $s_i > 0$). We note that higher value of w indeed contribute the convergence speed of the price gap to some extent. Namely $(u_1, w = 10)$ converges faster than $(u_1, w = 1)$, and $(u_2, w = 10)$ converges faster than $(u_2, w = 1)$. We don't see the significant difference between the choice of utility function u until the resource constraints are binding. Then different utility function may drive quite different marginal change of the objective value, which automatically reflects the price, i.e. dual state price. That's why in panel (b) of Figure 1, we have quite different price behavior for different utility functions when constraints are binding.

increased k. The relative gap saturates around 0.08 when the sample observed bids are more than 200. The panel(b) of figure 3 shows the higher value of k has slower speed to accumulate the revenue. Small k leads to a "greedy" trend to accumulate the revenue.



(a) price convergence with various look-ahead k step

(b) cumulative revenue upto j-th bid with different look ahead k

Figure 3: The left panel (a) shows the dual price converges to true price with the increased k. (i.e. $\frac{\|\lambda-p\|}{\|p\|}$ decreases as k increases.) The right penal (b) shows the higher value of k has slower speed to accumulate the revenue. Small k leads to a "greedy" trend to accumulate the revenue.

6 Dynamic update using SLPM and combining SCPM & SLPM

[This section is mainly in response to question5]

Now let us dynamically update the dual prices at time point $k = 50, 100, 200, 400, 800...$ and use the prices to make decision for the immediate subsequent period. Compare the results of SCPM and SLPM. Be careful to use the same data generated in Q3. After comparing SCPM and SLPM, think about combining SCPM and SLPM and comment on how does the new algorithm perform.

Solution:
The update time of dual prices are $k = 50, 100, 200, 400, 800, 1600, 3200, 6400, 10000$, and notice the upper limit is $10,000$ instead of $12,800$ because we only have $10,000$ bids. Updating dual prices dynamically, we still follow the general idea of SLPM and solve optimization as size of problem grows. When 50 bids arrive, we solve the first optimization, use the dual prices $\bar{\mathbf{y}}^k$ and $\mathbf{a}_j^T$ to make decisions for bids 51 to 100; then we have 100 bid information and solve the optimization problem of size 100, use the updated dual prices at this point to make decisions for bids 100-200. Following this logic, we constantly make decision based on what we have for the next period.

When comparing the two online methods, we already know SCPM has larger price gap than dynamic SLPM. Figure 1 and Figure 2, we've seen allocation reaches resource limit at around bid $k = 1943$ and the ultimate relative price gap is around 3 for SCPM. In Figure 4, we saw optimizing for first k bids and then make decisions for the future generally give us price gap less than 0.2. As for generating revenue, we observe that SCPM is fast in reaching resource allocation limit and thus, grows revenue a lot faster in the first 2000 bids, but after reaching the limits, it stops growing. At the same time, dynamic SLPM constantly and steadily grows its revenue and surpasses SCPM around iteration 9000.

A simple combination of SCPM and SLPM is the following algorithm: S-SCPM. We solve SLPM k-step horizon upto current episode (i.e. each episode consists k bids) and then use the *dual prices*, say $\bar{\mathbf{y}}^k$ of the partial LP for future online decisions: $x_j = 1$, if $\pi_j > \mathbf{a}_j^T \bar{\mathbf{y}}^k$; and $x_j = 0$, otherwise. With different combinations of parameter w and utility function $u(s)$, we have four choices of S-SCPM algorithms. In Figure 5, we compare the decisions made in the four S-SCPM and the original dynamic SLPM. All four S-SCPM make similar decisions with dynamic SLPM and out of 9500 bids after the first 50, S-SCPMs make only 10-20 different decisions as SLPM. As a result, revenue accumulates in a very similar way. (see Figure 6)

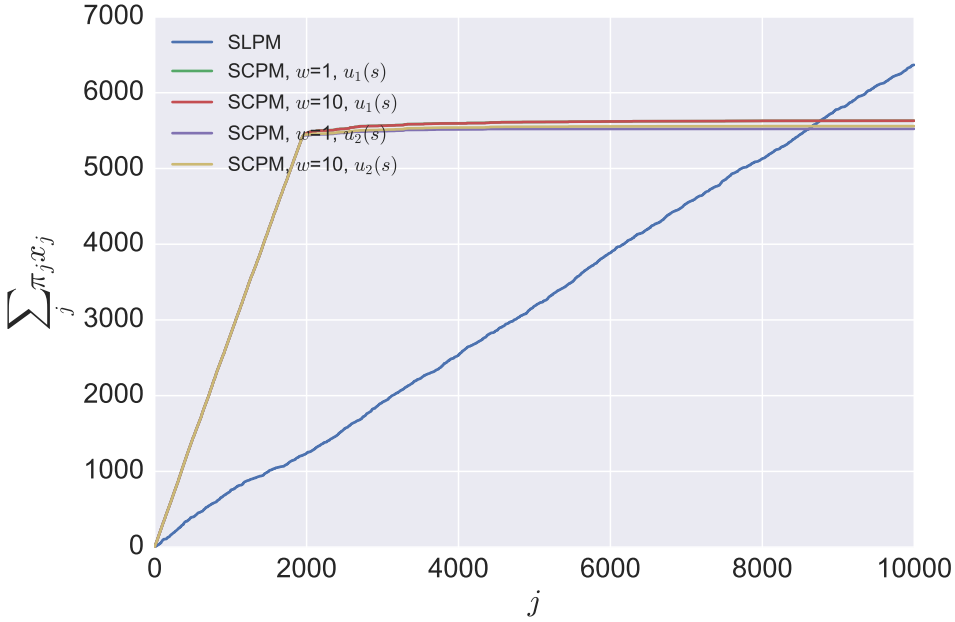
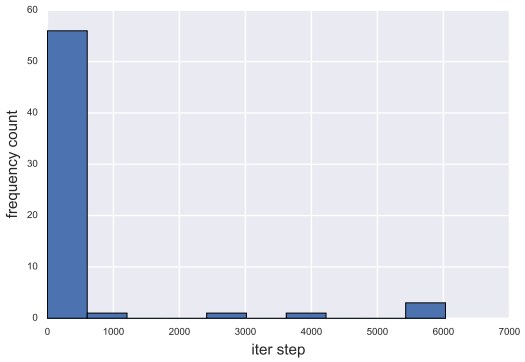
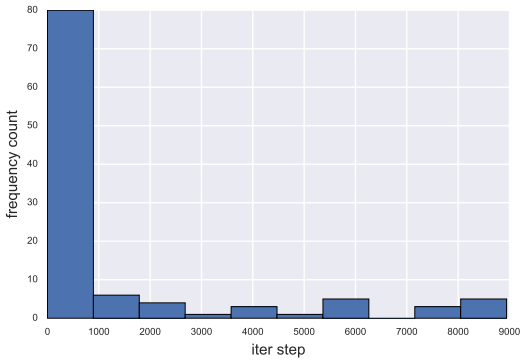


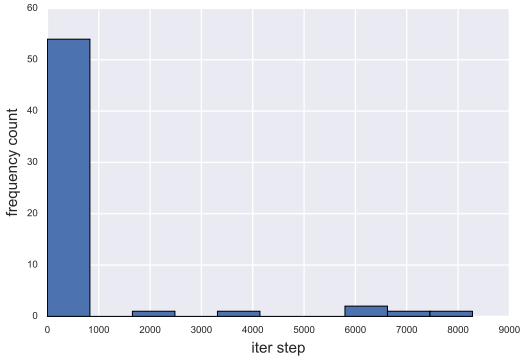
Figure 4: SCPM grows revenue a lot faster in the first 2000 bids, but reaches its limit and stops accepting bids thereafter; dynamic SLPM makes decision by optimizing chunks of information received for the next chunk of bids, therefore is more conservative but does not miss information in the general picture.



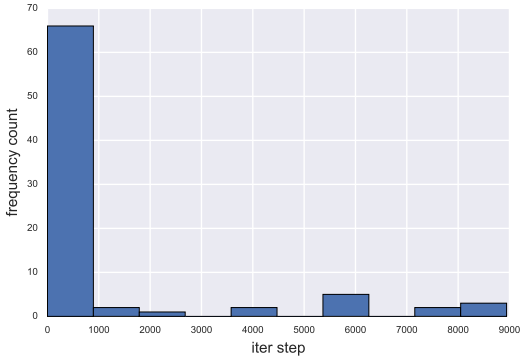
(a) distribution of decision discrepancies occurrences between dynamic SLPM and $w = 1, u_1(s)$ S-SCPM



(b) distribution of decision discrepancies occurrences between dynamic SLPM and $w = 10, u_1(s)$ S-SCPM



(c) distribution of decision discrepancies occurrences between dynamic SLPM and $w = 1, u_2(s)$ S-SCPM



(d) distribution of decision discrepancies occurrences between dynamic SLPM and $w = 10, u_2(s)$ S-SCPM

Figure 5: All four realizations of S-SCPM make almost the same decisions with original dynamic SLPM beyond the first 50 bids. Tis closeness in decisions made result in the following figure (Figure 6) showing identical growth of revenue between S-SCPM and dynamic SLPM.

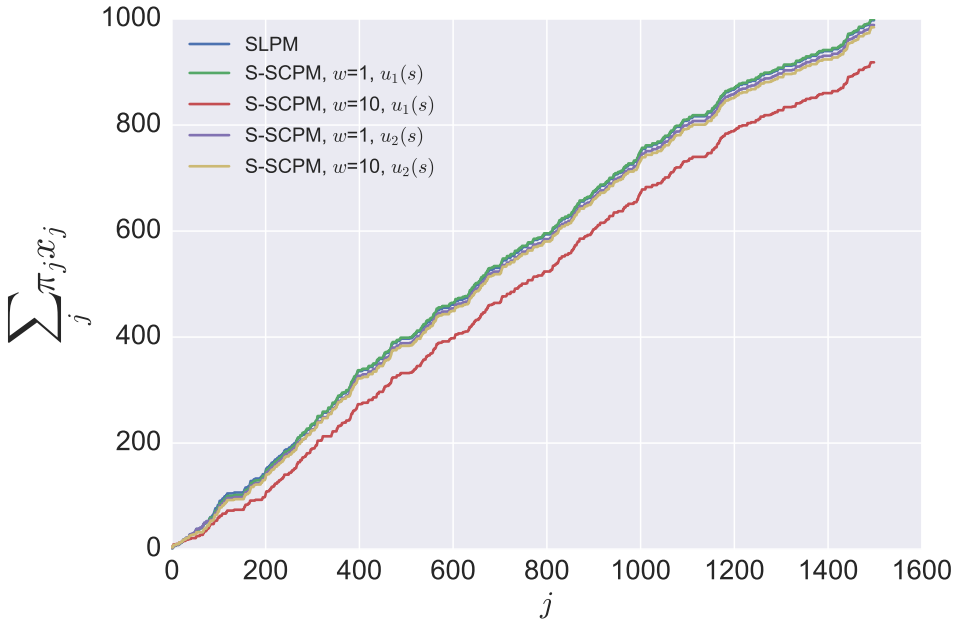


Figure 6: S-SCPM overcomes the weakness of SCPM and mimics the growing behavior for revenue of dynamic SLPM. The growing revenue curves are very close between dynamic SLPM and S-SCPM with different choices of utility function and w.

7 Appendix

7.1 KKT of CPCAM

Given the dual variables $\boldsymbol{\lambda}$ for constraints (5), dual variables $\mu \geq 0, \nu \geq 0, w \geq 0$ for constraints (6) - (8) respectively

$$\mathcal{L}(\mathbf{x}, \mathbf{s}, \boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu}, \mathbf{w}) = -\sum_j \pi_j x_j - u(\mathbf{s}) + \sum_{i=1}^{m} \lambda_i \big(\sum_j a_{ij} x_j + s_i - b_i \big) - \sum_j \mu_j x_j + \sum_j \nu_j (x_j - 1) - \sum_i w_i s_i \tag{35}$$

Taking the derivative over x and s and setting them equal zero, we have

$$\frac{\partial \mathcal{L}}{\partial x_j} = -\pi_j + \sum_i \lambda_i a_{ij} - \mu_j + \nu_j = 0, \quad \forall j \tag{36}$$

$$\frac{\partial \mathcal{L}}{\partial s_i} = -\frac{\partial u(\mathbf{s})}{\partial s_i} + \lambda_i - w_i = 0, \quad \forall i \tag{37}$$

$$\sum_j a_{ij} x_j + s_i - b_i = 0, \quad \forall i \tag{38}$$

$$\nu_j (x_j - 1) = 0, \quad \forall j \tag{39}$$

$$\mu_j x_j = 0, \quad \forall j \tag{40}$$

$$w_i s_i = 0, \quad \forall i \tag{41}$$

We simplify the expression as follows

$$x_j(\pi_j - \sum_i \lambda_i a_{ij} - \nu_j) = 0, \quad \forall j \tag{42}$$

$$s_i \frac{\partial u(\mathbf{s})}{\partial s_i} - \lambda_i s_i = 0, \quad \forall i \tag{43}$$

$$\sum_j a_{ij} x_j + s_i - b_i = 0, \quad \forall i \tag{44}$$

$$\nu_j(x_j - 1) = 0, \quad \forall j \tag{45}$$

$$x_j \geq 0, \quad \forall j \tag{46}$$

$$x_j \leq 1, \quad \forall j \tag{47}$$

$$\lambda \text{ is free} \tag{48}$$

$$s_i \geq 0, \quad \forall j \tag{49}$$

$$\nu_j \geq 0, \quad \forall j \tag{50}$$

$$\nabla u - \boldsymbol{\lambda} \leq 0 \tag{51}$$

$$\boldsymbol{\pi} - A^T \boldsymbol{\lambda} - \boldsymbol{\nu} \leq 0 \tag{52}$$

7.2 KKT of Online Model

We write out the Lagrangian as follows

$$\mathcal{L}(x_k, \mathbf{s}, \lambda, \mu_k, \nu_k, \mathbf{w}) = -\pi_k x_k - u(\mathbf{s}) + \sum_{i=1}^{m} \lambda_i (a_{ik} x_k + q_i^{k-1} + s_i - b_i) - \mu_k x_k + \nu_k (x_k - 1) + \sum_{i=1}^{m} w_i s_i \tag{53}$$

$$\frac{\partial \mathcal{L}}{\partial x_k} = -\pi_k + \sum_{i=1}^{m} \lambda_i a_{ik} - \mu_k + \nu_k = 0 \tag{54}$$

$$\frac{\partial \mathcal{L}}{\partial s_i} = -\frac{\partial u(\mathbf{s})}{\partial s_i} + \lambda_i - w_i = 0, \quad \forall i = 1, \ldots, m \tag{55}$$

$$\mu_k x_k = 0 \tag{56}$$

$$\nu_k(x_k - 1) = 0 \tag{57}$$

$$w_i s_i = 0, \quad \forall i = 1, \ldots, m \tag{58}$$

7.3 Coding details in Python

```
### 2018.03.17 | Solve project Q5. Solution for Q3 and Q4 involves similar functions, thus for
    simplicity only code details for Q5 displayed.

# Load packages and set random seeds
from cvxpy import *
import numpy as np
import scipy
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import pairwise
import time
import numdifftools as nd
from scipy import optimize as opt
import mosek

get_ipython().run_line_magic('matplotlib', 'inline')

get_ipython().run_line_magic('load_ext', 'autoreload')
get_ipython().run_line_magic('autoreload', '2')

np.seterr('ignore')
import warnings
warnings.filterwarnings('ignore')
```

```
# Use seed to generate same data for Q3, Q4, Q5
np.random.seed(11)

# Define utility functions

def utility1(s, w, m):
    return w/m * sum(log(s))

def utility2(s, w, m):
    return w/m * sum(1-exp(-s))


# Randomly generate data
m = 10
n = 10000 # bids
A = np.random.randint(low=0, high=2, size=(m, n));
p_true = np.random.rand(m, 1) # true price
b = np.ones((m,1))*1000


pi = p_true.T.dot(A).T + 0.2*np.random.randn(n, 1)

# Computes relative gap in prices
def relative_p_gap(dual_p, true_p):
    return np.linalg.norm(np.array(dual_p) - true_p.squeeze()) / np.linalg.norm(true_p.squeeze())

def solve_one_SLPM(pi, A, b, k1,k2, m=10, n=10000):
    # @param : pi_k -- constant given in objective (e.g. reward for k-th bid)
    # @param : a_k -- coefficient vector that requests resources
    # @param : b -- total resource budget
    # @param : q -- lock down allocated resource
    # @param (optional): w -- weights
    # @param : m -- number of resource types
    # ########## call cvx to solve ##############
    x = Variable(k2-k1)
    s = Variable(m)
    A_k = A[:, k1:k2]
    b_tilde = (k2-k1)/n * b.flatten()
    obj = Maximize( (pi[k1:k2].T)*(x) )
    # dot product A^Tx is (A_k*(x))
    constraint_1 = [(A_k*(x))[i] <= b_tilde[i] for i in range(m)]
    constraint_2 = [x[j] <= 1 for j in range(k1,k2)]
    constraint_3 = [x[j] >= 0 for j in range(k1,k2)]
    constraints = constraint_1 + constraint_2 + constraint_3 #+ constraint_4
    prob = Problem(obj, constraints)
    prob.solve(solver=MOSEK)
    # prob.solve(solver=ECOS)
    #     prob.solve(solver=SCS)
    #     prob.solve(solver=CVXOPT)
    # print([(constraints[i].dual_value) for i in range(m)])
    return x.value, obj.value, [(constraints[i].dual_value) for i in range(m)]

# Determine previous x using dual prices
def determine_x(pi_j, a_j, dual_p):
    thers = sum(a_j*dual_p)
    x_j = 0
    if pi_j > thers:
        x_j = 1
    return x_j

# Look ahead for x values
def check_k_ahead(pi_T,A_T,dual_p,k2,k3):
    T = k3-k2
    x_vec = np.zeros(T)
    for j in range(T):
        x_vec[j] = determine_x(pi_T[j], A_T[:,j], dual_p)
    return x_vec
```

```
x_SLPM = []
points = [0] + [50*(2**j) for j in range(9)]
pGap = []
for i in range(len(points)-1):
    x_opt, obj_opt, dual_price_opt = solve_one_SLPM(pi,A,b,k1=0,k2=points[i+1],m=10,n=10000)
    if i == 0:
        x_SLPM = x_SLPM + [j[0] for j in x_opt.tolist()]
    end = points[i+2]
    if end >= 10000:
        end = 10000
    x_real = check_k_ahead(pi[points[i+1]:end], A[:,points[i+1]:end],
         dual_price_opt,points[i+1],end)
    x_SLPM = x_SLPM + list(x_real)
    #print('\n'+ str(len(x)) + '\n')
    pGap.append(relative_p_gap(dual_price_opt,p_true))
    i += 1
    if end >= 10000:
        break

def solve_k_SCPM(pi, A, b, k, f_util, m=10, n=10000):
    # @param : pi_k -- constant given in objective (e.g. reward for k-th bid)
    # @param : a_k -- coefficient vector that requests resources
    # @param : b -- total resource budget
    # @param : q -- lock down allocated resource
    # @param (optional): w -- weights
    # @param : m -- number of resource types
    # ########## call cvx to solve ##############
    x = Variable(k)
    s = Variable(m)
    b_tilde = k/n * b.flatten()

    obj = Maximize((pi[0:k].T)*(x) + f_util(s) )
    constraint_1 = [(A[:,0:k]*(x))[i] + s[i] == b_tilde[i] for i in range(m)]
    constraint_2 = [x[j] >= 0 for j in range(0,k)]
    constraint_3 = [x[j] <= 1 for j in range(0,k)]
    constraint_4 = [s >= 0]
    constraints = constraint_1 + constraint_2 + constraint_3 + constraint_4
    prob = Problem(obj, constraints)
    #prob.solve(solver=MOSEK)
    #prob.solve(solver=ECOS)
    prob.solve(solver=SCS)
    #     prob.solve(solver=CVXOPT)
    return x.value, s.value, [(constraints[i].dual_value) for i in range(m)]

# S-SCPM
f1_util_w1 = lambda x : utility1(x, 1, m)
f1_util_w10 = lambda x : utility1(x, 10, m)
f2_util_w10 = lambda x: utility2(x, 10, m)
f2_util_w1 = lambda x : utility2(x, 1, m)
fList = [f1_util_w1,f1_util_w10,f2_util_w1,f2_util_w10]

def scpm(f):
    x_SSCPM = []
    points = [0] + [50*(2**j) for j in range(9)]
    for i in range(len(points)-1):
        x_k, s_vec, lambda_price = solve_k_SCPM(pi, A, b, points[i+1], f, m=10,n=10000)

        if i == 0:
            x_SSCPM = x_SSCPM + [j[0] for j in x_k.tolist()]
        end = points[i+2]
        if end >= 10000:
            end = 10000
        x_real = check_k_ahead(pi[points[i+1]:end], A[:,points[i+1]:end],
             lambda_price,points[i+1],end)
        x_SSCPM = x_SSCPM + list(x_real)
        #print('\n'+ str(len(x_SSCPM)) + '\n')
        i += 1
```

```
        if end >= 10000:
            break
    return x_SSCPM

value = []
discrep = []
for i in range(4):
    x_SSCPM = scpm(fList[i])
    value.append(np.array(x_SSCPM) * pi.flatten())
    l = []
    for i in range(len(x_SLPM)):
        if x_SLPM[i] != x_SSCPM[i]:
            l.append(i)
    discrep.append(l)

# Calculates revenue
value_SLPM=np.array(x_SLPM) * pi.flatten()


plt.figure(figsize=(9,6))
plt.plot(np.cumsum(value_SLPM)[1:1500], label="SLPM")
plt.plot(np.cumsum(value[0])[1:1500], label="S-SCPM, $w$=1, $u_1(s)$")
plt.plot(np.cumsum(value[1])[1:1500], label="S-SCPM, $w$=10, $u_1(s)$")
plt.plot(np.cumsum(value[2])[1:1500], label="S-SCPM, $w$=1, $u_2(s)$")
plt.plot(np.cumsum(value[3])[1:1500], label="S-SCPM, $w$=10, $u_2(s)$")
plt.legend(fontsize=13,loc='upper left')
plt.tick_params(labelsize=18)
plt.xlabel(r"$j$", fontsize=20)
plt.ylabel(r"$\sum_{j} \pi_{j}x_j $", fontsize=25)
plt.tight_layout()
plt.savefig("fig/Q5_cumulative_revenue.eps")


'''
plt.hist(discrep[0])
plt.tick_params(labelsize=7)
plt.xlabel('iter step')
plt.ylabel('frequency count')
plt.savefig("fig/Q5_unequal_x_11.eps")

plt.hist(discrep[1])
plt.tick_params(labelsize=7)
plt.xlabel('iter step')
plt.ylabel('frequency count')
plt.savefig("fig/Q5_unequal_x_110.eps")

plt.hist(discrep[2])
plt.tick_params(labelsize=7)
plt.xlabel('iter step')
plt.ylabel('frequency count')
plt.savefig("fig/Q5_unequal_x_21.eps")

plt.hist(discrep[3])
plt.tick_params(labelsize=7)
plt.xlabel('iter step')
plt.ylabel('frequency count')
plt.savefig("fig/Q5_unequal_x_210.eps")'''

def solve_kth_SCPM(pi_k, a_k, b, q, f_util, m=10):
    # @param : pi_k -- constant given in objective (e.g. reward for k-th bid)
    # @param : a_k -- coefficient vector that requests resources
    # @param : b -- total resource budget
    # @param : q -- lock down allocated resource
    # @param (optional): w -- weights
    # @param : m -- number of resource types
    # ########## call cvx to solve ##############
    x_k = Variable()
    s = Variable(m)
```

```
    obj = Maximize(pi_k*x_k + f_util(s) )
    constraint_1 = [a_k*x_k + s == b-q]
    constraint_2 = [x_k >= 0 ]
    constraint_3 = [x_k <= 1 ]
    constraint_4 = [s >= 0]
    constraints = constraint_1 + constraint_2 + constraint_3 + constraint_4
    prob = Problem(obj, constraints)
    #    prob.solve(solver=MOSEK)
    prob.solve(solver=ECOS)
    #    prob.solve(solver=SCS)
    #    prob.solve(solver=CVXOPT)
    return x_k.value, s.value, constraints[0].dual_value
def run_with_util(f_u):
    K = 10000 # num of bids so far -- TODO change to n later
    q = np.zeros((m, 1))
    x_vec = np.zeros(K)
    Price_mat = np.zeros((m, K))
    price_gap = np.zeros(K)
    for k in range(K):
        if (q >= b).all():
            print("use up resource")
            break
        else:
            time_start = time.clock()
            x_k, s_vec, lambda_price = solve_kth_SCPM(pi[k], A[:,k].reshape(-1, 1), b, q, f_u,
                m=10)
            time_end = time.clock()
            #if k % 100 == 0 :
                #print("{} iter, time in solving opt: {:.4}s".format(k, time_end-time_start))
                #pass

        if (q + A[:,k].reshape(-1, 1) <= b).all():
            if x_k <= 0 :
                x_vec[k] = 0
            elif x_k >= 1:
                x_vec[k] = 1
            else :
                x_vec[k] = np.round(x_k, 3)
        else:
            x_vec[k] = 0

        q = q + A[:,k].reshape(-1, 1)*x_k
        Price_mat[:,k] = np.array(lambda_price).squeeze()
        price_gap[k] =
            np.linalg.norm(np.array(lambda_price).squeeze()-p_true)/np.linalg.norm(p_true)

    return x_vec, Price_mat, price_gap
X_u1, p_mat_u1, p_gap_u1 = run_with_util(f1_util_w1)
X_u2, p_mat_u2, p_gap_u2 = run_with_util(f1_util_w10)
X_u3, p_mat_u3, p_gap_u3 = run_with_util(f2_util_w1)
X_u4, p_mat_u4, p_gap_u4 = run_with_util(f2_util_w10)

value_SCPM = []
value_SCPM.append(np.array(X_u1) * pi.flatten())
value_SCPM.append(np.array(X_u2) * pi.flatten())
value_SCPM.append(np.array(X_u3) * pi.flatten())
value_SCPM.append(np.array(X_u4) * pi.flatten())


plt.figure(figsize=(9,6))
plt.plot(np.cumsum(value_SLPM)[1:10000], label="SLPM")
plt.plot(np.cumsum(value_SCPM[0])[1:10000], label="SCPM, $w$=1, $u_1(s)$")
plt.plot(np.cumsum(value_SCPM[1])[1:10000], label="SCPM, $w$=10, $u_1(s)$")
plt.plot(np.cumsum(value_SCPM[2])[1:10000], label="SCPM, $w$=1, $u_2(s)$")
plt.plot(np.cumsum(value_SCPM[3])[1:10000], label="SCPM, $w$=10, $u_2(s)$")
plt.legend(fontsize=13,loc='upper left')
plt.tick_params(labelsize=18)
```

```
plt.xlabel(r"$j$", fontsize=20)
plt.ylabel(r"$\sum_{j} \pi_{j}x_j $", fontsize=20)
plt.tight_layout()
plt.savefig("fig/Q5_SCPM_SLPM_compare.eps")
```