MS&E 310 Project: Online Linear Programming

Jiayue Wan, Lynn Zeng

December 11, 2017

1 Introduction

Resource allocation is a classical problem in operations research, which aims to maximize the total return given a limited amount of resources. Typical problems can be formulated using a linear program and solved offline in a straightforward manner. However, in specific settings where decisions of resource allocation must be made shortly after order receipts (e.g. online auction / bidding), we need online linear programming algorithms to make decisions. In this project, we investigate and propose multiple online linear programming algorithms to solve online resource allocation problem.

2 Backgrounds

We consider a linear program in the form

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & \sum_{j=1}^{n} \pi_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \le b_i, \ \forall i = 1, 2, \cdots, m \\ & 0 \le x \le 1, \ \forall j = 1, 2, \cdots, n \end{aligned} \tag{1}$$

where $\pi = (\pi_1, \pi_2, \cdots, \pi_n)^T$ is the pricing (or gain) vector, a_{ij} is required quantity of resource i for bidder j, $b = (b_1, b_2, \cdots, b_m)^T$ is the resource capacity vector. For simplicity, we assume that a_{ij} is either 0 or 1. An offline linear programming algorithm computes the optimal solution x all at the same time, while an online linear programming algorithm computes x_1, x_2, $\cdots$, sequentially. We test the performances of multiple online algorithms using simulated data.

2.1 Simulated Data

We generate simulated bidding data for Model 1. We set the number of resources to $m = 10$, the number of bids to $n = 10000$, the resource capacity to b_i to 1000 for $i = 1, 2, \cdots, m$. The bidding

information matrix A (an m by n matrix) such that

$$A = \begin{bmatrix} a_{11} & \dots & a_{m1} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{bmatrix} \tag{2}$$

is generated by random zeros and ones. We also fix a ground truth price vector $\overline{p}$ using standard uniform distribution $U[0,1]$. Then, the bidding price is randomly generated using

$$\pi_k = \overline{p}^T a_k + randn(0, 0.2) \tag{3}$$

where $randn(0, 0.2)$ indicates a random gaussian variable with mean 0 and variance 0.2.

2.2 Offline Algorithm (Baseline)

We consider the offline linear programming problem as the following

$$\begin{aligned} \underset{x,s}{\text{maximize}} \quad & \pi^T x + u(s) \\ \text{subject to} \quad & Ax + s = b \\ & 0 \le x \le 1 \\ & s \ge 0. \end{aligned} \tag{4}$$

We construct the Lagrangian for the LP program.

$$\mathcal{L} = -(\pi^T x + u(s)) + p^T(Ax + s - b) - \lambda_1^T x + \lambda_2^T(x-1) - \lambda_3^T s \tag{5}$$

$$\begin{aligned} &\textcircled{1} \quad -\pi + p^T A - \lambda_1 + \lambda_2 = 0 \\ &\textcircled{2} \quad -\frac{\mathrm{d}u(s)}{\mathrm{d}s} + p^T - \lambda_3^T = 0 \\ &\textcircled{3} \quad \lambda_3^T s = 0 \\ &\textcircled{4} \quad \lambda_1^T x = 0 \\ &\textcircled{5} \quad \lambda_2^T(x-1) = 0 \\ &\textcircled{6} \quad \lambda_1, \lambda_2, \lambda_3 \ge 0. \end{aligned} \tag{6}$$

Conditions 1 to 6, along with the primal constraints constitute the set of KKT conditions. We assume that $u(s)$ is increasing and strictly concave. Then, since $\frac{\mathrm{d}u(s_i)}{\mathrm{d}s_i} = \infty$ at $s_i = 0$, the objective increases with a local increase in s_i at $s_i = 0$. Therefore, $s_i^* > 0$ for all i. By conditions 2 and 3, we know that $\lambda_3 = 0$, and $p = (\frac{\mathrm{d}u(s)}{\mathrm{d}s})^T$ is unique.

Consider the utility function $u(s)$ as the expected future values of unused resources [3]. It is increasing and concave due to the law of diminishing returns. We run the offline algorithm using the following four utility functions $u(s)$:

- Utility function 1: $u_1(\mathbf{s}) = \dfrac{w}{m}\sum_i log(s_i)$, where $w = 1$ and $m = 10$.
- Utility function 2: $u_1(\mathbf{s}) = \dfrac{w}{m}\sum_i log(s_i)$, where $w = 10$ and $m = 10$.

- Utility function 3: $u_2(\mathbf{s}) = \frac{w}{m}\sum_i(1-e^{is_i})$, where $w = 1$ and $m = 10$.
- Utility function 4: $u_2(\mathbf{s}) = \frac{w}{m}\sum_i(1-e^{is_i})$, where $w = 10$ and $m = 10$.

Table 1 shows the simulation result of the offline algorithm using different utility functions.

Table 1: Simulation result of the offline algorithm.

Utility function	**1**	**2**	**3**	**4**
Total Rev.	7398.067	7392.476	7405.956	7399.927
$\|\|p-\bar{p}\|\|_2$	0.2199722	0.219972	0.219972	0.2199721
Number of Bids Accepted	2235	2230	2231	2233

In theory, different choices of w, which means different expected future value of unused resources, should yield different total revenues. A large w indicates a higher valuation of resource in the future and a more conservative policy in online linear programming. However, we see that using different w's and different utility functions makes no significant difference in all three aspects we listed above. This makes sense because in the offline problem, the $u(s)$ function only ensures the uniqueness of dual variables, and therefore should not affect the optimum significantly. In all cases, the shadow price vector p is close to the ground truth price vector $\bar{p}$. We use the results of offline algorithm as the baseline of online algorithms.

3 SCPM

SCPM is an online algorithm that makes immediate decision when a bid is received from the participants [1][3]. It notifies bidders the results right away so that they are able to modify their bids and resubmit. Instead of solving the problem after the market closes, whenever the k^{th} bid arrives, the market maker solves the follwing problem:

$$\begin{aligned} \underset{x_k, s}{\text{maximize}} \quad & \pi_k x_k + u(s) \\ \text{subject to} \quad & a_{ik} x_k + s_i = b_i - q_i^{k-1}, \forall i = 1, \ldots, m \\ & 0 \leq x_k \leq 1 \\ & s_i \geq 0, \forall i = 1, \ldots, m \end{aligned} \tag{7}$$

where $q_i^{k-1} = \sum_{j=1}^{k-1} a_{ij} \bar{x}_j$ is the resource i allocated before the kth bidder arrives. Again, we construct the Lagrangian for this LP problem, and obtain the following:

$$\mathcal{L} = -(\pi_k x_k + u(s)) + \sum_{i=1}^{m} p_i (a_{ik} x_k + s_i - b_i + q_i^{k-1}) - \lambda_1 x_k + \lambda_2 (x_k - 1) - \lambda_3^T s \tag{8}$$

$$\begin{aligned} &(1) \quad -\pi_k + \sum_{i=1}^{m} p_i * a_{ik} - \lambda_1 + \lambda_2 = 0 \\ &(2) \quad -\frac{\mathrm{d}u(s_i)}{\mathrm{d}s_i} + p^T - \lambda_3^T = 0, \forall i = 1 \ldots m \\ &(3) \quad \lambda_3^T s = 0 \\ &(4) \quad \lambda_1 * x_k = 0 \\ &(5) \quad \lambda_2 * (x_k - 1) = 0 \\ &(6) \quad \lambda_1, \lambda_2, \lambda_3 \geq 0. \end{aligned} \tag{9}$$

Similar to the offline problem, the disutility function ensures the uniqueness of the shadow price p. Conditions 1 to 6, along with the primal constraints constitute the set of KKT conditions. Note that SCPM can be solved more efficiently compared with the offline algorithm. The reason is that it only has decision variables x_k and s to solve, while the offline problem has n of them (i.e. $x_1, \ldots, x_n$) besides decision variable s. However, one shortcoming of the SCPM algorithm is that we have to solve one optimization problem each time we want to make a decision.

In order to compare the optimality of SCPM and the baseline algorithm, we run SCPM on the simulated dataset described in Section 2.1 with the same four utility functions. Table 2 shows the results of the SCPM algorithm. Because in the SCPM model we do not have a good estimate of the total number of bids n, as we can see in Figure 1, all resources are allocated after around 2000 bids using any of the four utility functions.

For the online model, the shadow price generated from the online auction model does not converge to the ground truth vector for any of the four utility functinons. This makes sense since the shadow price is based on only one decision variable (i.e. one single bid). Therefore, the shadow price is not a good indicator of the ground-truth price.

Table 2: Simulation result of SCPM online algorithm.

Utility Function	Total Rev.	$\|\|p - \bar{p}\|\|_2$	Num of Bids Accepted
1	6105.517	Explode after around 2000^{th} bid	2060
2	6106.285	Explode after around 2000^{th} bid	2041
3	6108.591	Fluctuate between 1 and 5	2048
4	6109.561	Fluctuate between 1 and 5	2038

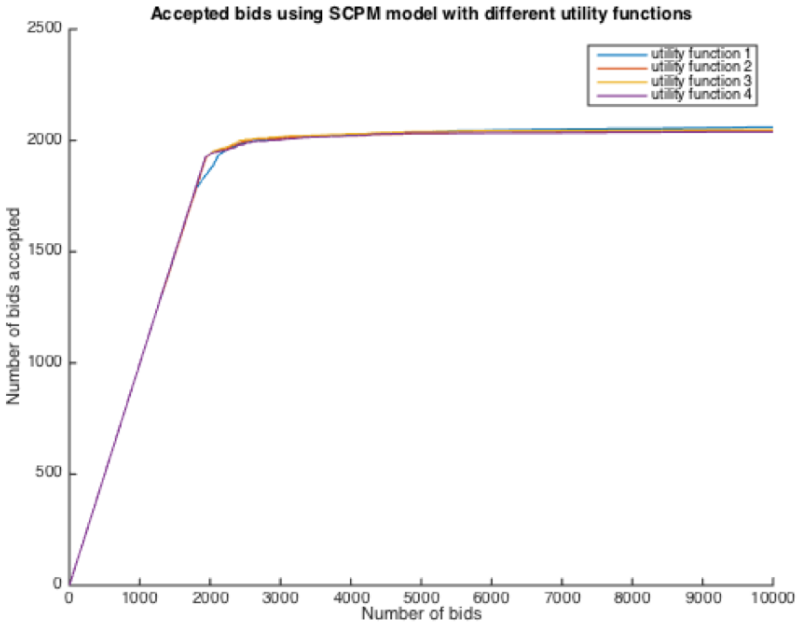

Figure 1: Number of bids accepted in SCPM.

4 SLPM

4.1 Regular SLPM

SLPM is an online algorithm proposed by Agrawal, et al [2]. With a good estimate of the total number of bidders n in the market, we could wait for the first k bidders, solve a linear program, retrieve the shadow price p to instruct future decisions. Mathematically, we consider the linear program

$$\begin{aligned} \underset{x_1,\cdots,x_k}{\text{maximize}} \quad & \sum_{j=1}^{k} \pi_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{k} a_{ik} x_k \leq \frac{k}{n} \cdot b_i, \ \forall i = 1, \cdots, m \\ & 0 \leq x_j \leq 1, \ \forall j = 1, \cdots, k. \end{aligned} \tag{10}$$

We make future online decisions based on the derived dual price vector p:

$$x_j = \begin{cases} 1 & \text{if } \pi_j > a_j^T p, \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

Based on the model, we run two versions of algorithms for the first constraint:

- **Version 1**: $\sum_{j=1}^{k} a_{ik} x_k \leq \frac{k}{n} \cdot b_i$. Using this constraint, we allocate $\frac{k}{n}$ of the total resources to the first k bids and derive the shadow price.

- **Version 2**: $\sum_{j=1}^{k} a_{ik} x_k \leq \left(1 - \frac{k}{n}\right) \cdot \frac{k}{n} \cdot b_i$. Using this constraint, we still roughly allocate $\frac{k}{n}$ of the total resources to the first k bids but add a factor of $\left(1 - \frac{k}{n}\right)$ to eliminate the bias.

The results of the SLPM algorithms are shown below in Table 3 and Table 4.

Table 3: Simulation result of version 1 SLPM.

k	50	100	200	400
Total Rev.	6007.230	6281.328	6434.860	6481.897
$\|\|p - \bar{p}\|\|_2$	0.905994	0.546397	0.394315	0.343217
Num of Bids Accepted	1779	1888	1941	1957

Table 4: Simulation result of version 2 SLPM.

k	50	100	200	400
Total Rev.	5991.450	6281.328	6434.860	6475.168
$\|\|p - \bar{p}\|\|_2$	0.974871	0.546397	0.394315	0.290046
Num of Bids Accepted	1786	1888	1941	1959

In general, when k is relatively small compared with the total number of bidders n, as k increases, the optimal total revenue goes up and the difference between the shadow price and ground truth

price goes down. This result makes sense because a larger k means that more information is collected through bidding data, which leads to more accurate estimation of the price vector. Moreover, the SLPM algorithm also allows a relatively more even distribution of accepted bids across all bids, compared with the SCPM algorithm.

4.2 SLPM with Dynamic Updates

A better model than the regular SLPM model is to update the dual price at different time points [2]. With more information collected, the updated dual price vector will be more accurate and can help make better decisions subsequently. We solve the following linear program defined on the inputs until time l (which we choose to be 50, 100, 200, etc. in our simulation):

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & \sum_{t=1}^{l} \pi_t x_t \\ \text{subject to} \quad & \sum_{t=1}^{l} a_{it} x_t \leq (1-h_l)\frac{l}{n} b_i, \ \forall i = 1, \cdots, m \\ & 0 \leq x_t \leq 1, \ \forall t = 1, \cdots, l \end{aligned} \tag{12}$$

where $h_l = \sqrt{\frac{n}{l}} * \epsilon = \sqrt{\frac{n}{l}} * \frac{50}{n}$ is the bias-elimination factor. The dynamic learning algorithm will update the price every time the history information doubles. With the derived dual price vector, we apply the same allocation rule as defined in Section 4.1 to decide future allocations. The simulation result is shown in Table 5.

Table 5: Simulation result of SLPM with dynamic updates.

Algorithm	Total Rev.	Number of Bids Accepted
SLPM with dynamic updates	7290.995	2208

The cumulative number of accepted bids in the bidding process is shown in Figure 2. The $l2$-norms of the differences between the dual price vectors and the ground truth price vector over iterations are shown in Figure 3.

It is straightforward that with dynamic updates, SLPM is able to generate a total revenue very close to the offline optimum. As we solve a linear program with more historical data, the dual price vector gets closer to the ground truth price vector, which means better future decisions. In addition, from Figure 2 we can see that SLPM with dynamic updates is also able to fully explore all bids and allocate resources more evenly. Hence, SLPM with dynamic updates has great performance when we have a good estimate of the total number of bidders.

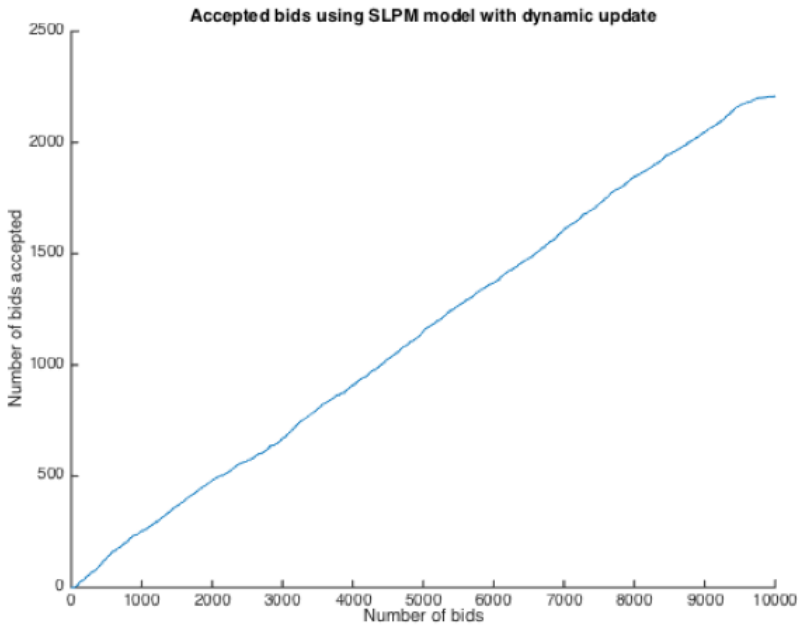

Figure 2: Number of bids accepted in SLPM with dynamic updates.

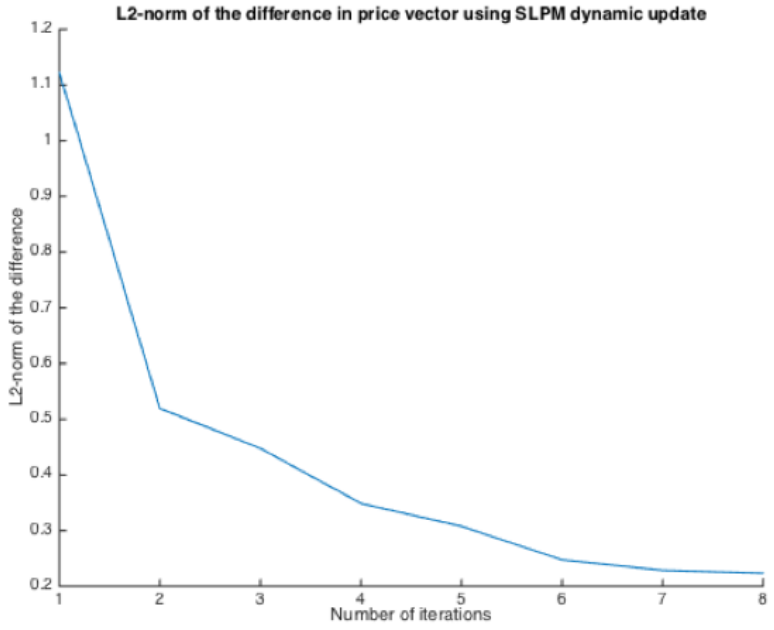

Figure 3: $l2$-norm of the difference of price vectors in SLPM with dynamic updates.

5 SCPM Variants

5.1 Comparison between SCPM and SLPM

Technically it is not fair to make comparison between SCPM and SLPM. In SCPM we assume that we do not know the total number of bidders beforehand, while in SLPM we do. Hence, when implementing SCPM we have to choose utility function wisely (i.e. how we value resources used in the future) so that we can possibly get results. SLPM, on the other hand, has great performance when we assume that n is known beforehand. This motivates us to investigate the performance of SCPM algorithms when we assume n is known. In this section, we propose two algorithms of SCPM variants.

5.2 SCPM with Resource Partition

First, we divide the resources into d groups. Then, we run the SCPM algorithm within each group individually. To be precise, each group will have $\frac{n}{d}$ bidders. First, we run SCPM on the first group of bidders and decide the acceptance of bids. Then, we run the SCPM algorithm on the second group of bidders with the remaining resources divided by $d-1$ and decide the acceptance of bids, and so on. The algorithm terminates when all d groups are considered.

Mathematically, within each group r, when the lth bidder submits a bid, i.e. $(r-1)*\frac{n}{d}+1 \leq l \leq r*\frac{n}{d}$, the market maker solves the following optimization problem:

$$\begin{aligned} \underset{x_l,s}{\text{maximize}} \quad & \pi_l * x_l + u(s) \\ \text{subject to} \quad & a_{il} * x_l + s_i = b_{ir} - q_i^{l-1}, \forall i = 1, \ldots, m \\ & 0 \leq x_l \leq 1 \\ & s \geq 0 \end{aligned} \tag{13}$$

where $b_{ir} = \frac{1}{d-r+1}(b_i - \sum_{i=1}^{(r-1)*\frac{n}{d}} a_{ij} * x_j)$ is the allocated resource for the rth group, $q_i^{l-1} = \sum_{i=(r-1)*\frac{n}{d}+1}^{l-1} a_{ij} * x_j$ is the resources that are already used within the rth group before the lth bidder arrives. We choose the utility function

$$u(s) = \frac{1}{m}\sum_i (1 - e^{is_i}) \tag{14}$$

when implementing the algorithm. The result of SCPM with resource partition is shown in Table 6. The cumulative number of accepted bids in the bidding process is shown in Figure 4. We can see that within each group, the allocated resources are used up quickly. Hence, our attempt of exploring all bids does not work very well using SCPM with resource partition.

Table 6: Simulation result of SCPM with resource partition.

Algorithm	Total Rev.	Number of Bids Accepted
SLPM with resource partition	6066.916	2075

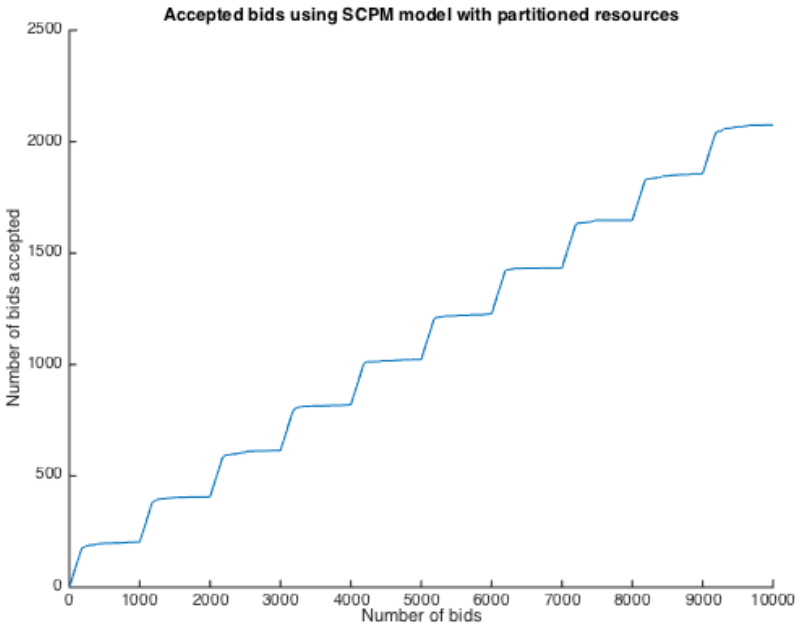

Figure 4: Number of bids accepted in SCPM with resource partition.

5.3 SCPM with Dynamic Updates

We propose an SCPM variant with dynamic updates analogous to SLPM with dynamic updates. We solve the following linear program defined on the inputs until time l (which we choose to be 50, 100, 200, etc. in our simulation):

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & \sum_{t=1}^{l} \pi_t x_t + u(s) \\ \text{subject to} \quad & \sum_{t=1}^{l} a_{it} x_t + s_i = (1 - h_l)\frac{l}{n} b_i, \ \forall i = 1, \cdots, m \\ & 0 \leq x_t \leq 1, \ \forall t = 1, \cdots, l \end{aligned} \tag{15}$$

where $h_l = \sqrt{\frac{n}{l}} * \epsilon = \sqrt{\frac{n}{l}} * \frac{50}{n}$ is the bias-elimination factor. The dynamic learning algorithm will update the price every time the history information doubles. With the derived dual price vector, we apply the same allocation rule as defined in Section 4.1 to decide future allocations. The simulation result is shown in Table 7. The cumulative number of accepted bids in the bidding process is shown in Figure 5. The $l2$-norms of the differences between the dual price vectors and the ground truth price vector over iterations are shown in Figure 6.

It is straightforward that with dynamic updates, SCPM is able to generate a total revenue close to the offline optimum. The dual price vector also gets closer to the ground truth price vector over iterations. In addition, from Figure 5 we can see that SLPM with dynamic updates is also able to fully explore all bids and allocate resources more evenly. Hence, SCPM with dynamic updates has great performance when we assume that n is known.

Table 7: Simulation result of SCPM with dynamic updates.

Algorithm	Total Rev.	Number of Bids Accepted
SCPM with dynamic updates	7284.217	2208

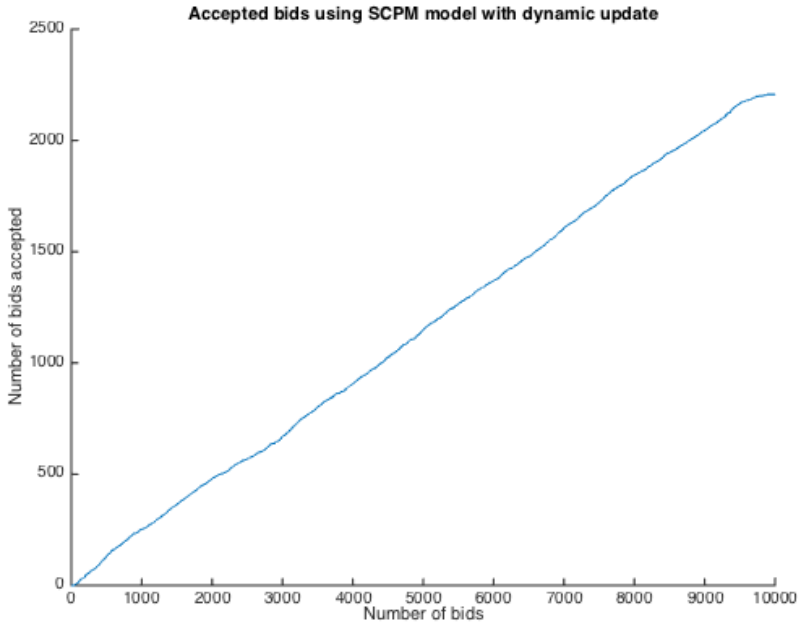

Figure 5: Number of bids accepted in SCPM with dynamic updates.

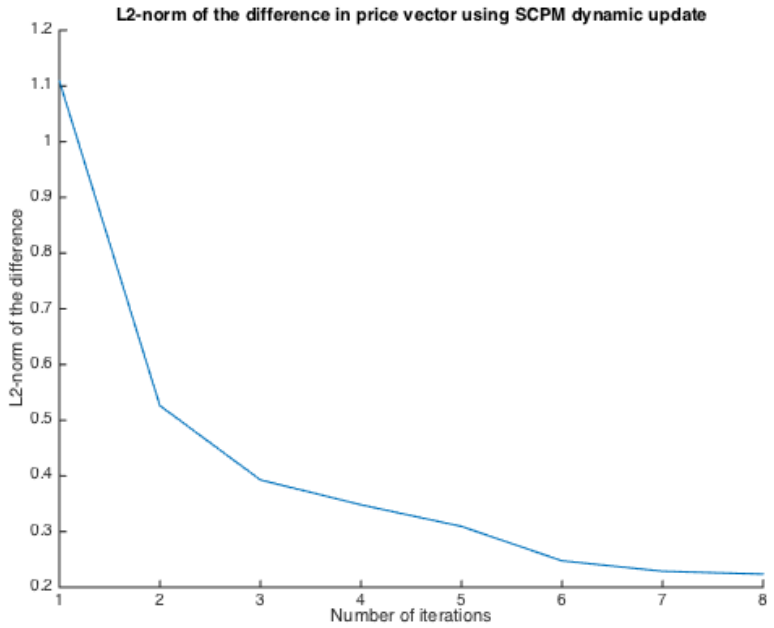

Figure 6: $l2$-norm of the difference of price vectors in SCPM with dynamic updates.

6 General Resource Allocation Problem

6.1 Offline Algorithm

One may consider a more general resource allocation problem with production costs:

$$\begin{aligned} \underset{x,y}{\text{maximize}} \quad & \sum_{j=1}^{n}(\pi_j x_j - \sum_{k,i} c_{ijk} y_{ijk}) \\ \text{subject to} \quad & \sum_{k=1}^{K} y_{ijk} = a_{ij} x_j, \ \forall i,j \\ & \sum_{i,j} y_{ijk} \le C_k, \ \forall k \\ & 0 \le x_j \le 1, \ \forall j \\ & y_{ijk} \ge 0 \end{aligned} \tag{16}$$

where c_{ijk} is the cost of resource i produced by producer k to bidder j, C_k is the production capacity of producer k ($k = 1, 2, \cdots, K$). To derive the dual problem, we construct the Lagrangian function as following:

$$\begin{aligned} \mathcal{L} &= \sum_j(-\pi_j x_j + \sum_{i,k} c_{ijk} y_{ijk}) + \sum_{i,j} \mu_{ij}(\sum_k y_{ijk} - a_{ij} x_j) + \sum_k (y_{ijk} - C_k) \\ &\quad + \sum_j \delta_{1j}(-x_j) + \sum_j \delta_{2j}(x_j - 1) + \sum_{i,j,k} \alpha_{ijk}(-y_{ijk}) \\ &= \sum_j(-\pi_j - \sum_i \mu_{ij} a_{ij} - \delta_{1j} + \delta_{2j}) x_j + \sum_{i,j,k}(c_{ijk} + \mu_{ij} + \lambda_k - \alpha_{ijk}) y_{ijk} - \sum_k \lambda_k C_k - \sum_j \delta_{2j}. \end{aligned} \tag{17}$$

Then, the corresponding dual problem is the following:

$$\begin{aligned} \underset{\mu,\lambda,\delta_1,\delta_2,\alpha}{\text{minimize}} \quad & \sum_k C_k \lambda_k + \sum_j \delta_{2j} \\ \text{subject to} \quad & -\pi_j - \sum_i a_{ij}\mu_{ij} - \delta_{1j} + \delta_{2j} = 0, \ \forall j \\ & c_{ijk} + \mu_{ij} + \lambda_k - \alpha_{ijk} = 0, \ \forall i,j,k \\ & \lambda_k \ge 0, \ \forall k \\ & \delta_{1j}, \delta_{2j} \ge 0, \ \forall j \\ & \alpha_{ijk} \ge 0, \ \forall i,j,k. \end{aligned} \tag{18}$$

We simulate the bidding data for this general resource allocation problem. We set the number of resources to $m = 10$, the number of bids to $n = 1000$, the number of producers to $K = 3$, the production capacity C_k to 500 for $k = 1, 2, \cdots K$. The bidding information matrix A (an m by n matrix) is generated by random zeros and ones. We set the average price of each resource to 0.5 so that the bidding price is

$$r = 0.5 * sum(a_j) + randn(0, 0.2). \tag{19}$$

Table 8 shows the simulation result of the offline algorithm for general resource allocation problem. The cumulative number of accepted bids in the bidding process is shown in Figure 7.

Table 8: Simulation result of offline algorithm for general resource allocation problem.

Algorithm	Total Profit	Number of Bids Accepted
Offline for general resource allocation	590.737	325

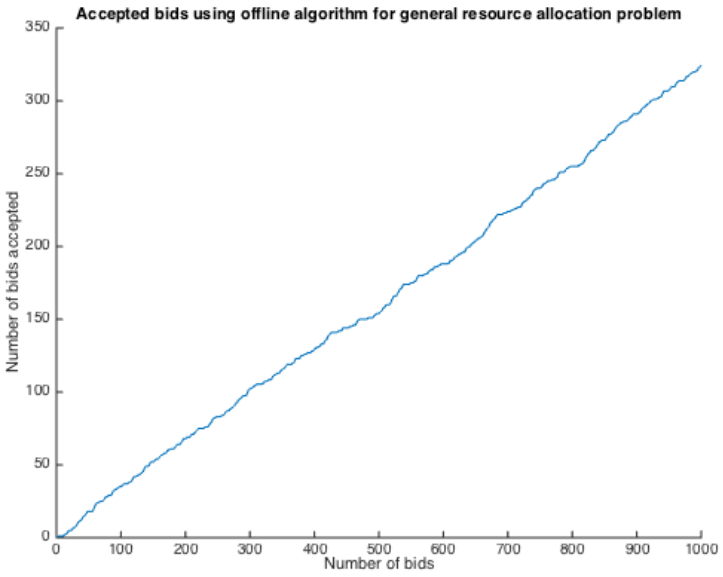

Figure 7: Number of bids accepted in general resource allocation problem using offline algorithm.

6.2 Online Algorithm

We propose a heuristic algorithm that resolves online programming for general resource allocation problem. When the $\hat{j}^{th}$ bid comes in, the market maker solves the following problem:

$$\begin{aligned} \underset{y_{i\hat{j}k}}{\text{minimize}} \quad & \sum_{i,k} c_{i\hat{j}k} y_{i\hat{j}k} - u(s) \\ \text{subject to} \quad & \sum_{k} y_{i\hat{j}k} = a_{i\hat{j}} x_{\hat{j}}, \ \forall i \\ & \sum_{i} y_{i\hat{j}k} + s_k = C_k - q_k^{\hat{j}-1}, \forall k \\ & s_k \geq 0, \forall k \\ & y_{i\hat{j}k} \geq 0, \ \forall i, k \end{aligned} \tag{20}$$

where $q_k^{\hat{j}-1} = \sum_{i=1}^{m} \sum_{j=1}^{\hat{j}-1} y_{ijk}$ is the used production capacity of producer k before bidder j arrives, and $u(s)$ is a disutility function that has the same property as the disutility function defined in Section 2.2. After computing the optimal $y^*_{i\hat{j}k}$ in the problem above, the market maker decides whether to accept with the following rule:

$$x_{\hat{j}} = \begin{cases} 1 & \text{if } \pi_{\hat{j}} > \sum_{i,k} c_{ijk} * y^*_{i\hat{j}k} * \alpha, \\ 0 & \text{otherwise} \end{cases} \tag{21}$$

where $\alpha = \frac{K+1}{2}$ is a multiplier indicating the expected rate of return when there are K competitive producers in the market. Table 9 shows the simulation result of the online algorithm for general resource allocation problem. The cumulative number of accepted bids in the bidding process is shown in Figure 8.

It is straightforward that our online algorithm has good performance when compared with the offline baseline model. It generates a total profit of 516.787, which is more than 87.5% of the offline optimum. In addition, we discover that the accepted bids are relatively evenly distributed among all the bids.

Table 9: Simulation result of online algorithm for general resource allocation problem.

Algorithm	Total Profit	Number of Bids Accepted
Online for general resource allocation	516.787	308

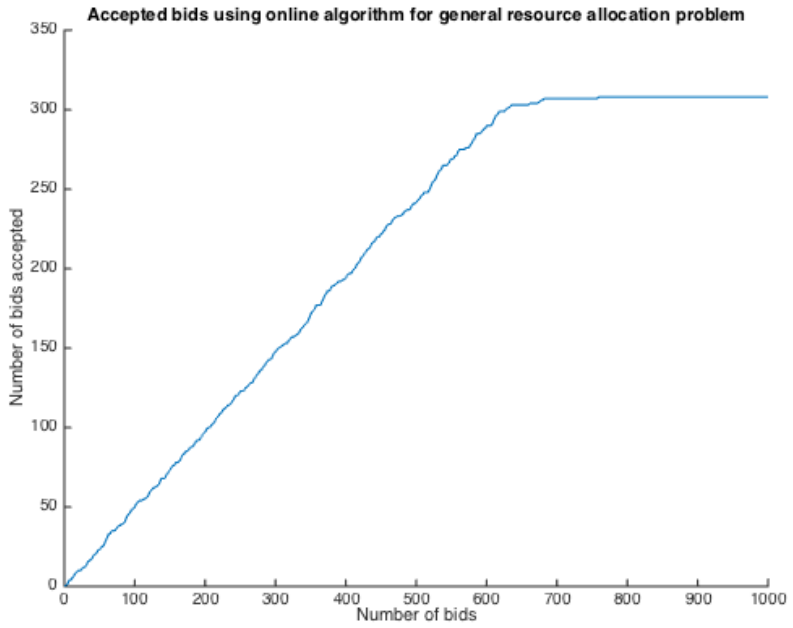

Figure 8: Number of bids accepted in general resource allocation problem using online algorithm.

7 Conclusion

In this project we investigate multiple online algorithms for resource allocation. Table 10 summarizes the simulation results of all offline and online algorithms implemented in this project. We can see that when we assume that the number of total bidders is known, SLPM with dynamic updates and SCPM with dynamic updates have the best performances, yielding total revenue very close to the offline optimum (over 98% of the offline optimum). On the other hand, when we do not have a good estimate of the total number of bidders where algorithms like SLPM do not apply, SCPM has a reasonable performance, generating a decent total revenue (over 80% of the offline optimum). In terms of future plan, more heuristic algorithms dealing with the general resource allocation problem can be explored.

Table 10: Simulation results of offline and online algorithms.

Algorithm	Total Rev.	Number of Bids Accepted
Offline Algorithm	7405.956	2231
SCPM	6109.56	2038
SLPM Version 1	6481.897	1959
SLPM Version 2	6475.168	1957
SLPM with Dynamic Updates	**7290.995**	2208
SCPM with Resource Partition	6066.916	2075
SCPM with Dynamic Updates	**7284.217**	2208

References

[1] S. Agrawal, E. Delage, M. Peters, Z. Wang and Y. Ye. A Unified Framework for Dynamic Prediction Market Design. Operations Research, 59:3 (2011) 550-568.

[2] S. Agrawal, Z. Wang and Y. Ye. A Dynamic Near-Optimal Algorithm for Online Linear Programming. Operations Research, 62:4 (2014) 876-890.

[3] M. Peters, A. M-C. So and Y. Ye. Pari-mutuel Markets: Mechanisms and Performance. The 3rd International Workshop On Internet And Network Economics, 2007.

Appendix: MATLAB Code

A1. Generating Simulated Data

```
clc
clear
format long

rng(31415926)  % set seed
m = 10;  % number of resources
n = 10000;  % number of bids
b = 1000 * ones(m,1);  % resource capacity
A = randi(2,m,n) - 1;  % bid matrix
p_true = rand(m,1);  % ground truth price vector
r = A'*p_true + sqrt(0.2)*randn(n,1);   % bidding price

save('data.mat')
```

A2. Offline Algorithm

```
f = {@(t) 1/m * sum(log(t));
     @(t) 10/m * sum(log(t));
     @(t) 1/m * sum(1-exp(-t));
     @(t) 10/m * sum(1-exp(-t))};

num_f = 4;
revs = zeros(num_f,1);
diffs = zeros(num_f,1);
bids_accepted = zeros(num_f,1);

for i = 1: num_f
    cvx_begin
        variable x(n);
        variable s(m);
        dual variable p;

        maximize (r'*x + f{i}(s))
        p: A*x + s == b
        x >= 0
        x <= 1
        s >= 0
    cvx_end

    x(x <= 0) = 0;
    x(x >= 1) = 1;
    x = (rand(n,1) < x);

    rev = r'*x;
    diff = norm(p + p_true);

    revs(i) = rev;
    diffs(i) = diff;
    bids_accepted(i) = sum(x);
end

save('offline_results.mat')
```

A3. SCPM Algorithm

```
f = {@(t) 1/m * sum(log(t));
     @(t) 10/m * sum(log(t));
     @(t) 1/m * sum(1-exp(-t));
     @(t) 10/m * sum(1-exp(-t))};

num_f = 4;
revs = zeros(num_f, 1);
diffs = zeros(num_f, n);
X = zeros(num_f, n);

for i = 1: num_f
    q = zeros(m,1);  % used resources
    for k = 1: n
        if q == b
            break
        else
            cvx_begin
                variable x(1);
                variable s(m);
                dual variable p;

                maximize (r(k)*x + f{i}(s))

                p: A(:, k)*x + s == b - q
                x >= 0
                x <= 1
                s >= 0
            cvx_end
        end

        if q + A(:, k) <= b
            if x < 0
                X(i,k) = 0;
            elseif x >= 1
                X(i,k) = 1;
            else
                X(i,k) = (rand < x);
            end
        else
            X(i,k) = 0;
        end

        q = q + A(:, k)*X(i,k);
        diffs(i, k) = norm(p + p_true);
    end

    revs(i) = X(i,:)*r;
end

save('scpm_results.mat')
```

A4. SLPM Algorithm

```
k = [50 100 200 400];
num_k = 4;

revs = zeros(num_k, 1);
diffs = zeros(num_k, 1);
X = zeros(num_k, n);

for i = 1: num_k
    X(i, 1:k(i)) = 0;
    q = zeros(m,1);  % used resources

    cvx_begin
        variable x(k(i));
        dual variable p;

        maximize (r(1:k(i))'*x)
        %p: A(:,1:k(i))*x <= k(i)/n*b     % version 1 SLPM
        p: A(:,1:k(i))*x <= (1-k(i)/n)*k(i)/n*b    % version 2 SLPM
        x >= 0
        x <= 1
    cvx_end

    diffs(i) = norm(p-p_true);

    for j = k(i)+1 : n
        if q + A(:, j) <= b & r(j) > A(:,j)'*p
            X(i,j) = 1;
            q = q + A(:, j)*X(i,j);
        else
            X(i,j) = 0;
        end
    end

    revs(i) = X(i,:)*r;
end

save('slpm_results.mat')
```

A5. SLPM Algorithm with Dynamic Updates

```
niter = floor(log2(n/50)) + 1;
diffs = zeros(niter, 1);
X = zeros(n,1);
q = zeros(m,1);  % used resources
eps = 50/10000;

for i = 1: niter
    num_x = 50*2^(i-1);
    cvx_begin
        variable x(num_x);
        dual variable p;

        maximize (r(1:num_x)'*x)
        p: A(:,1:num_x)*x <= (1-eps*sqrt(n/num_x))*num_x/n*b;
        x >= 0
        x <= 1
    cvx_end

    diffs(i) = norm(p-p_true);

    for j = num_x+1 : min(n, 2*num_x)
        if q + A(:, j)*1 <= b & r(j) > A(:,j)'*p
            X(j) = 1;
            q = q + A(:, j)*X(j);
        else
            X(j) = 0;
        end
    end
end

save('slpm_dynamical_update_results.mat')
```

A6. SCPM with Resource Partition

```
f =  @(t) 1/m * sum(1-exp(-t));
k = 1000;
num_group = n/k;
diffs = zeros(n, 1);
X = zeros(n, 1);

q = zeros(m,1);  % used resources
b_remaining = b;

for i = 1: num_group
    b_remaining = b_remaining - q;
    A_sub = A(:, (i-1)*k+1: i*k);
    b_sub = b_remaining/(num_group - i + 1);
    q = zeros(m,1);  % used resources in group i
    for j = 1: k
        if q == b_sub
            break
        else
            cvx_begin
                variable x(1);
                variable s(m);
                dual variable p;

                maximize (r(j)*x + f(s))

                p: A_sub(:, j)*x + s == b_sub - q
                x >= 0
                x <= 1
                s >= 0
            cvx_end
        end

        idx = (i-1)*k + j;

        if q + A_sub(:, j) <= b_sub
            if x < 0
                X(idx) = 0;
            elseif x >= 1
                X(idx) = 1;
            else
                X(idx) = (rand < x);
            end
        else
            X(idx) = 0;
        end

        q = q + A_sub(:, j)*X(idx);
        diffs(idx) = norm(p + p_true);
    end
end

save('scpm_partition_results.mat')
```

A7. SCPM with Dynamic Updates

```
f =  @(t) 1/m * sum(1-exp(-t));
niter = floor(log2(n/50)) + 1;
diffs = zeros(niter, 1);
X = zeros(n,1);
q = zeros(m,1);  % used resources
eps = 50/10000;

for i = 1: niter
    num_x = 50*2^(i-1);
    cvx_begin
        variable x(num_x);
        variable s(m);
        dual variable p;

        maximize (r(1:num_x)'*x + f(s))
        p: A(:,1:num_x)*x +s == (1-eps*sqrt(n/num_x))*num_x/n*b;
        x >= 0
        x <= 1
        s >= 0
    cvx_end

    diffs(i) = norm(p+p_true);

    for j = num_x+1 : min(n, 2*num_x)
        if q + A(:, j)*1 <= b & r(j) > -A(:,j)'*p
            X(j) = 1;
            q = q + A(:, j)*X(j);
        else
            X(j) = 0;
        end
    end
end

save('scpm_dynamic_update_results.mat')
```

A8. Simulated Data for General Resource Allocation Problem

```
clc
clear
format long

rng(31415926)  % set seed
m = 10;  % number of resources
n = 1000;  % number of bids
K = 3; % number of producers
Cap = 500 * ones(K,1);  % producer capacity
A = randi(2,m,n) - 1;  % bid matrix
c = rand(m,n,K);
avg_price = ones(m,1)*0.5;
r = A'*avg_price + sqrt(0.2)*randn(n,1);   % bidding price

save('general_data.mat')
```

A9. Offline Algorithm for General Resource Allocation Problem

```
cvx_begin
    variable x(n);
    variable y(m,n,K);

    maximize (r'*x - sum(sum(sum(y.*c))))

    for i = 1:m
        for j = 1:n
            sum(y(i,j,:)) == A(i,j)*x(j);
        end
    end

    for k = 1:K
        temp = 0;
        for i = 1:m
            for j = 1:n
                temp = temp + y(i,j,k);
            end
        end
        temp <= Cap(k);
    end

    x >= 0
    x <= 1
    y >= 0
cvx_end

profit = r'*x - sum(sum(sum(y.*c)));

save('offline_general_results.mat')
```

A10. Online Algorithm for General Resource Allocation Problem

```
f = @(t) 1/m * sum(1-exp(-t));

X = zeros(n, 1);
profit = 0;
q = zeros(K,1);  % used resources

for j = 1: n
    if q == Cap
        break
    else
        cvx_begin
            variable y(m,K);
            variable s(K);

            cost_mat = reshape(c(:,j,:), m, K, []);
            minimize (sum(sum(cost_mat.*y)) - f(s))

            sum(y, 2) == A(:, j);
            sum(y, 1)' + s == Cap - q;

            s >= 0;
            y >= 0;
        cvx_end

        cost = sum(sum(cost_mat.*y)) + 0;

        if cost < r(j)*0.5 & q + sum(y, 1)' <= Cap
            X(j) = 1;
            q = q + sum(y, 1)'
            profit  = profit + r(j) - cost
        else
            %
        end
    end
end

save('online_general_results.mat')
```