Dual Interpretations and Duality Applications (continued)

Yinyu Ye
Department of Management Science and Engineering
Stanford University
Stanford, CA 94305, U.S.A.

http://www.stanford.edu/~yyye
(LY5th, Chapters 3.1-3.5, Wikipedia)
Two-Person Zero-Sum Game

Let $P$ be the payoff matrix of a two-person, "column" and "row", zero-sum game.

$$P = \begin{pmatrix} +3 & -1 & -4 \\ -3 & +1 & +4 \end{pmatrix}$$

Players usually use randomized strategies in such a game. A randomized strategy is a vector of probabilities, each associated with a particular decision.
In a Nash Equilibrium, if your (column) strategy is a pure strategy (one where you always play a single action), the expected payout for the (dominating) action that you are playing should be greater than or equal to the expected payout for any other action. If you are playing a randomized strategy, the expected payout for each action included in your strategy should be the same (if one were lower, you won’t want to ever choose that action) and these payouts should be greater than or equal to the actions that aren’t part of your strategy.
"Column" strategy:

\[
\begin{align*}
\text{max} & \quad v \\
\text{s.t.} & \quad ve \leq Px \\
& \quad e^T x = 1 \\
& \quad x \geq 0.
\end{align*}
\]

"Row" strategy:

\[
\begin{align*}
\text{min} & \quad u \\
\text{s.t.} & \quad ue \geq P^T y \\
& \quad e^T y = 1 \\
& \quad y \geq 0.
\end{align*}
\]

They are dual to each other.
Multi-Firm LP Alliance I

Consider a finite set $I$ of firms each of whom has operations that have representations as linear programs. Suppose the linear program representing the operations of firm $i$ in $I$ entails choosing an $n$-column vector $x \geq 0$ of activity levels that maximize the firm’s profit

$$c^T x$$

subject to the constraint that its consumption $Ax$ of resources minorizes its available resource vector $b^i$, that is,

$$Ax \leq b^i.$$
An **alliance** is a subset of the firms. If an alliance $S$ pools its resource vectors, the linear program that $S$ faces is that of choosing an $n$-column vector $x \geq 0$ that maximizes the profit $c^T x$ that $S$ earns subject to its resource constraint

$$Ax \leq b^S = \sum_{i \in S} b^i.$$ 

Let $V^S$ be the resulting maximum profit of $S$. The **grand alliance** is the set $I$ of all firms.

$$V^S := \max \ c^T x$$

s.t. $Ax \leq \sum_{i \in S} b^i,$

$x \geq 0,$
Core is the set of payment vector $z = (z_1, \ldots, z_{|I|})$ to each company such that

$$\sum_{i \in I} z_i = V^I$$

and

$$\sum_{i \in S} z_i \geq V^S, \forall S \subseteq I.$$ 

**Theorem 1** For each optimal dual price vector for the linear program of the grand alliance, allocating each firm the value of its resource vector at those prices yields a profit allocation vector in the core.
Robust Optimization I

Consider a linear program

\[
\begin{align*}
\text{minimize} \quad & (c + Cu)^T x \\
\text{subject to} \quad & Ax = b, \\
& x \geq 0,
\end{align*}
\]

where \( u \geq 0 \) and \( u \leq e \) is a state of Nature and beyond decision maker's control.

Robust Model:

\[
\begin{align*}
\text{minimize} \quad & \max\{u \geq 0, u \leq e\}(c + Cu)^T x \\
\text{subject to} \quad & Ax = b, \\
& x \geq 0.
\end{align*}
\]
Nature’s (primal) problem:

\[
\text{maximize}_{u} \quad c^T x + x^T C u \\
\text{subject to} \quad u \leq e, \\
\quad u \geq 0.
\]

Dual of Nature’s problem:

\[
\text{minimize}_{y} \quad c^T x + e^T y \\
\text{subject to} \quad y \geq C^T x, \\
\quad y \geq 0.
\]
Decision Maker’s Robust Model:

\[
\begin{align*}
\text{minimize}_{x, y} & \quad c^T x + e^T y \\
\text{subject to} & \quad y \geq C^T x, \\
& \quad A x = b, \\
& \quad x, y \geq 0.
\end{align*}
\]
Online Decision Making: Resource Allocation Linear Programming

In real applications, data/information is revealed sequentially, and one has to make decisions sequentially based on what is known – cannot wait for solving the offline problem. Consider problem:

\[
\begin{align*}
\text{maximize}_x & \quad \sum_{t=1}^{n} \pi_t x_t \\
\text{subject to} & \quad \sum_{t=1}^{n} a_{it} x_t \leq b_i, \quad \forall i = 1, \ldots, m \\
& \quad 0 \leq x_t \leq 1, \quad \forall t = 1, \ldots, n
\end{align*}
\]

Each bid/activity \( t \) requests a bundle of \( m \) resources, and the payment is \( \pi_t \).

**Online Decision Making:** we only know \( b \) at the start, but

- the constraint matrix is revealed column by column sequentially along with the corresponding objective coefficient.
- an irrevocable decision must be made as soon as an order arrives without observing or knowing the future data.
### An Example

<table>
<thead>
<tr>
<th></th>
<th>order 1((t = 1))</th>
<th>order 2((t = 2))</th>
<th>.....</th>
<th>Inventory((b))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Price((\pi_t))</td>
<td>$100</td>
<td>$30</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Decision</td>
<td>(x_1)</td>
<td>(x_2)</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Pants</td>
<td>1</td>
<td>0</td>
<td>...</td>
<td>100</td>
</tr>
<tr>
<td>Shoes</td>
<td>1</td>
<td>0</td>
<td>...</td>
<td>50</td>
</tr>
<tr>
<td>T-shirts</td>
<td>0</td>
<td>1</td>
<td>...</td>
<td>500</td>
</tr>
<tr>
<td>Jacket</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>200</td>
</tr>
<tr>
<td>Socks</td>
<td>1</td>
<td>1</td>
<td>...</td>
<td>1000</td>
</tr>
</tbody>
</table>
Sequential Convex Programming Mechanism

(SCPM): maximize $x_t, s \pi_t x_t + u(s)$

s.t. $a_t x_t + s = b - \sum_{j=1}^{t-1} a_j \bar{x}_j$, $0 \leq x_t \leq 1$, $s \geq 0$.

$\sum_{j=1}^{t-1} a_j \bar{x}_j$: allocated resource vector before the new arrival.

Possible Concave Value Functions:

- Exponential: $u(s_i) = b \cdot (1 - \exp(-s_i/b))$, for some positive constant $b$.
- Logarithmic: $u(s_i) = b \cdot \log(s_i)$, for some positive constant $b$.
- Quadratic:

$$u(s_i) = \begin{cases} 
  b \cdot (1 - (1 - s_i/b)^2) & 0 \leq s_i \leq b \\
  b & s_i \geq b 
\end{cases}$$

for some positive constant $b$. 
Online Linear Programming Mechanism: Online Price Learning

Main Assumptions

- The columns $a_t$ arrive in a random order.
- We know the total number of columns $n$ a priori.

Other technical assumptions

- $0 \leq a_{it} \leq 1$, for all $(i, t)$;
- $\pi_t \geq 0$ for all $t$

The algorithm/mechanism quality is evaluated on the expected performance over all the permutations comparing to the offline optimal solution, i.e., an algorithm $\mathcal{A}$ is $c$-competitive if and only if

$$E_\sigma \left[ \sum_{t=1}^{n} \pi_t x_t(\sigma, \mathcal{A}) \right] \geq c \cdot OPT(A, \pi).$$
The online approach is distribution-free. It allows for great robustness in practical problems. If the columns or arrivals are drawn $i.i.d.$ from a certain distribution (either known or unknown to the decision maker), then the first assumption is automatically met.

The second assumption is necessary for one to obtain a near optimal solution. However, it can be relaxed to an approximate knowledge of $n$ or the length of decision horizon.

Both assumptions are reasonable and standard in many operations research and computer science applications.
Main Theorems

**Theorem 2** For any fixed $0 < \epsilon < 1$, there is no online algorithm for solving the linear program with competitive ratio $1 - \epsilon$ if

$$B < \frac{\log(m)}{\epsilon^2}.$$ 

**Theorem 3** For any fixed $0 < \epsilon < 1$, there is a $1 - \epsilon$ competitive online algorithm for solving the linear program if

$$B \geq \Omega \left( \frac{m \log (n/\epsilon)}{\epsilon^2} \right).$$

Comments on the Main Theorems

- The condition of $B$ to hold the main result is independent of the size of $OPT(A, \pi)$ or the objective coefficients, and is also independent of any possible distribution of input data. Therefore, it’s checkable.
- The condition on sample size $1/\epsilon^2$ is necessary as it is common in many learning-based algorithm.
- The condition is proportional only to $\log(n)$ so that it is way below to satisfy everyone’s demand.
Key Ideas to Prove Negative Result

- Consider $m = 1$ and inventory level $B$, one can construct an instance where $OPT = B$, and there will be a loss of $\sqrt{B}$ with a high probability, which give an approximation ratio $1 - \frac{1}{\sqrt{B}}$.

- Consider general $m$ and inventory level $B$ for each good. We are able to construct an instance to decompose the problem into $\log(m)$ separable problems, each of which has an inventory level $B/\log(m)$ on a composite “single good” and $OPT = B/\log(m)$.

- Then, with high probability each “single good” case has a loss of $\sqrt{B/\log(m)}$ and the total loss of $\sqrt{B \cdot \log(m)}$. Thus, approximation ratio is at best $1 - \frac{\sqrt{\log(m)}}{\sqrt{B}}$. 


Key Ideas to Prove Positive Result

The proof of the positive result is constructive and based on a learning policy.

- There is no distribution known so that any type of stochastic optimization models is not applicable.

- Unlike dynamic programming, the decision maker does not have full information/data so that a backward recursion cannot be carried out to find an optimal sequential decision policy.

- Thus, the online algorithm needs to be learning-based, in particular, learning-while-doing.
The problem would be easy if there is an "ideal price" vector:

<table>
<thead>
<tr>
<th>Bid 1( (t = 1) )</th>
<th>Bid 2( (t = 2) )</th>
<th>.....</th>
<th>Inventory( (b) )</th>
<th>( p^* )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bid( (\pi_t) )</td>
<td>$100</td>
<td>$30</td>
<td>( \ldots )</td>
<td></td>
</tr>
<tr>
<td>Decision</td>
<td>( x_1 )</td>
<td>( x_2 )</td>
<td>( \ldots )</td>
<td></td>
</tr>
<tr>
<td>Pants</td>
<td>1</td>
<td>0</td>
<td>( \ldots )</td>
<td>100</td>
</tr>
<tr>
<td>Shoes</td>
<td>1</td>
<td>0</td>
<td>( \ldots )</td>
<td>50</td>
</tr>
<tr>
<td>T-shirts</td>
<td>0</td>
<td>1</td>
<td>( \ldots )</td>
<td>500</td>
</tr>
<tr>
<td>Jackets</td>
<td>0</td>
<td>0</td>
<td>( \ldots )</td>
<td>200</td>
</tr>
<tr>
<td>Hats</td>
<td>1</td>
<td>1</td>
<td>( \ldots )</td>
<td>1000</td>
</tr>
</tbody>
</table>
One-Time Learning Algorithm

We start with a simple learning policy

- Set \( x_t = 0 \) for all \( 1 \leq t \leq \epsilon n \);
- Solve the \( \epsilon \) portion of the problem

\[
\begin{align*}
\text{maximize}_{x} & \quad \sum_{t=1}^{\epsilon n} \pi_t x_t \\
\text{subject to} & \quad \sum_{t=1}^{\epsilon n} a_{it} x_t \leq (1 - \epsilon)\epsilon b_i \quad i = 1, \ldots, m \\
& \quad 0 \leq x_t \leq 1 \quad t = 1, \ldots, \epsilon n
\end{align*}
\]

and get the optimal dual solution \( \hat{p} \);
- Determine the future allocation \( x_t \) as:

\[
x_t = \begin{cases} 
0 & \text{if } \pi_t \leq \hat{p}^T a_t \\
1 & \text{if } \pi_t > \hat{p}^T a_t
\end{cases}
\]

as long as \( a_{it} x_t \leq b_i - \sum_{j=1}^{t-1} a_{ij} x_j \) for all \( i \); otherwise, set \( x_t = 0 \).
Theorem 4 For any fixed $\epsilon > 0$, the one-time learning algorithm is $(1 - \epsilon)$ competitive for solving the linear program when

$$B \geq \Omega \left( \frac{m \log (n/\epsilon)}{\epsilon^3} \right)$$

Outline of the Proof:

- With high probability, we clear the market;
- With high probability, the revenue is near-optimal if we include the initial $\epsilon$ portion revenue;
- With high probability, the first $\epsilon$ portion revenue, a learning cost, doesn’t contribute too much.

Then, we prove that the one-time learning algorithm is $(1 - \epsilon)$ competitive under condition

$$B \geq \frac{6m \log (n/\epsilon)}{\epsilon^3}.$$ 

But this is one $\epsilon$ factor higher than the lower bound...
In the dynamic price learning algorithm, we update the price at time $\epsilon n, 2\epsilon n, 4\epsilon n, \ldots,$ till $2^k \epsilon \geq 1$.

At time $\ell \in \{\epsilon n, 2\epsilon n, \ldots\}$, the price vector is the optimal dual solution to the following linear program:

\[
\begin{align*}
\text{maximize}_x & \quad \sum_{t=1}^\ell \pi_t x_t \\
\text{subject to} & \quad \sum_{t=1}^\ell a_{it} x_t \leq (1 - h_\ell) \frac{\ell}{n} b_i \quad i = 1, \ldots, m \\
& \quad 0 \leq x_t \leq 1 \quad t = 1, \ldots, \ell
\end{align*}
\]

where

\[
h_\ell = \epsilon \sqrt{\frac{n}{\ell}};
\]

and this price vector is used to determine the allocation for the next immediate period.
In the dynamic price updating algorithm, we update the price at time $\epsilon n, 2\epsilon n, 4\epsilon n \ldots$ At time $\ell \in \{\epsilon n, 2\epsilon n, \ldots\}$, the price is the optimal dual solution to the following linear program:

$$\begin{align*}
\text{maximize} & \quad \sum_{t=1}^{\ell} \pi_t x_t \\
\text{subject to} & \quad \sum_{t=1}^{\ell} a_{it} x_t \leq (1 - h_\ell) \frac{\ell}{n} b_i \quad i = 1, \ldots, m \\
& \quad 0 \leq x_t \leq 1 \quad t = 1, \ldots, \ell
\end{align*}$$

where

$$h_\ell = \epsilon \sqrt{\frac{n}{\ell}}$$

And this price is used to determine the allocation for the next immediate period.
Geometric Pace/Grid of Price Updating

\[ t_1 = \epsilon n \]
\[ t_2 = 2\epsilon n \]
\[ t_3 = 4\epsilon n \]
\[ t_4 = 8\epsilon n \]
Comments on Dynamic Learning Algorithm

- In the dynamic algorithm, we update the prices $\log_2 \left( \frac{1}{\epsilon} \right)$ times during the entire time horizon.

- The numbers $h_\ell$ play an important role in improving the condition on $B$ in the main theorem. It basically balances the probability that the inventory ever gets violated and the lost of revenue due to the factor $1 - h_\ell$.

- Choosing large $h_\ell$ (more conservative) at the beginning periods and smaller $h_\ell$ (more aggressive) at the later periods, one can now control the loss of revenue by an $\epsilon$ order while the required size of $B$ can be weakened by an $\epsilon$ factor.
Similar to the dynamic price updating algorithm, we update the price at time $\epsilon n, 2\epsilon n, 3\epsilon n \ldots$ At time $\ell \in \{\epsilon n, 2\epsilon n, \ldots\}$, the price is the optimal dual solution to the following linear program:

$$\begin{align*}
\text{maximize}_x & \quad \sum_{t=1}^{\ell} \pi_t x_t \\
\text{subject to} & \quad \sum_{t=1}^{\ell} a_{it} x_t \leq \frac{\ell}{n-\ell} \hat{b}_i \quad i = 1, \ldots, m \\
& \quad 0 \leq x_t \leq 1 \quad t = 1, \ldots, \ell
\end{align*}$$

where $\hat{b}_i$ is the remaining inventory at the beginning of the $t$th period. And this price is used to determine the allocation for the next immediate period.
## Related Work on Random-Permutation

<table>
<thead>
<tr>
<th></th>
<th>Sufficient Condition</th>
<th>Learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kleinberg [2005]</td>
<td>$B \geq \frac{1}{\epsilon^2}$, for $m = 1$</td>
<td>Dynamic</td>
</tr>
<tr>
<td>Devanur et al [2009]</td>
<td>$OPT \geq \frac{m^2 \log(n)}{\epsilon^3}$</td>
<td>One-time</td>
</tr>
<tr>
<td>Feldman et al [2010]</td>
<td>$B \geq \frac{m \log n}{\epsilon^3}$ and $OPT \geq \frac{m \log n}{\epsilon}$</td>
<td>One-time</td>
</tr>
<tr>
<td>Agrawal et al [2010]</td>
<td>$B \geq \frac{m \log n}{\epsilon^2}$ or $OPT \geq \frac{m^2 \log n}{\epsilon^2}$</td>
<td>Dynamic</td>
</tr>
<tr>
<td>Molinaro/Ravi [2013]</td>
<td>$B \geq \frac{m^2 \log m}{\epsilon^2}$</td>
<td>Dynamic</td>
</tr>
<tr>
<td>Kesselheim et al [2014]</td>
<td>$B \geq \frac{\log m}{\epsilon^2}$</td>
<td>Dynamic*</td>
</tr>
<tr>
<td>Gupta/Molinaro [2014]</td>
<td>$B \geq \frac{\log m}{\epsilon^2}$</td>
<td>Dynamic*</td>
</tr>
<tr>
<td>Agrawal/Devanur [2014]</td>
<td>$B \geq \frac{\log m}{\epsilon^2}$</td>
<td>Dynamic*</td>
</tr>
</tbody>
</table>

Table 1: Comparison of several existing results
Summary and Future Questions on OLP

- $B = \frac{\log m}{\epsilon^2}$ is now a necessary and sufficient condition (differing by a constant factor).

- Thus, they are near-optimal online algorithms for a very general class of online linear programs under the permutation assumption.

- The algorithms are distribution-free and/or non-parametric, thereby robust to distribution/data uncertainty.

- The dynamic learning has the feature of “learning-while-doing”, and is provably better than one-time learning by a factor.

- Buy-and-sell or double market worked under the iid inputs.

- Price-Posting multi-good model?