Large-Scale Price Optimization via Network Flow

Shinji Ito, Ryohei Fujimaki
Our goal: profit maximization by optimizing prices

What is the best pricing strategy?

<table>
<thead>
<tr>
<th>Strategy 1</th>
<th>Strategy 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Price</td>
<td>Price</td>
</tr>
<tr>
<td>$1.3</td>
<td>$1.0</td>
</tr>
<tr>
<td>Sales quantity</td>
<td></td>
</tr>
<tr>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Profit</td>
<td>Profit</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
Our goal: profit maximization by optimizing prices

What is the best pricing strategy?

<table>
<thead>
<tr>
<th>Strategy 1</th>
<th>Strategy 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Price</td>
<td>Price</td>
</tr>
<tr>
<td>$1.3</td>
<td>$1.0</td>
</tr>
<tr>
<td>Sales quantity</td>
<td>Sales quantity</td>
</tr>
<tr>
<td>$5.2</td>
<td>$6.0</td>
</tr>
</tbody>
</table>

Profit

$5.2 < $6.0
Complicated structure in price optimization

Changing the price of one product affects other’s sales

- **Cannibalization:**
  
  Growing the sales of makes the sales of down

<table>
<thead>
<tr>
<th>Product</th>
<th>Price</th>
<th>Quantity</th>
<th>Profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product 1</td>
<td>$1.3 → $1.0</td>
<td>+200</td>
<td>+ $80</td>
</tr>
<tr>
<td>Product 2</td>
<td>$1.2</td>
<td>-100</td>
<td>- $120</td>
</tr>
</tbody>
</table>

**Gross profit** - $40
Predictive price optimization and its difficulty

Recent advanced ML reveals relationship between prices and sales quantities

**Input:** pos data

<table>
<thead>
<tr>
<th>Day 1</th>
<th>Price</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Day 2</td>
<td>Price</td>
<td>Sales</td>
</tr>
<tr>
<td>⋮</td>
<td>⋮</td>
<td>⋮</td>
</tr>
</tbody>
</table>

Predictive model: \( \text{sales} = f(\text{prices}) \)
Predictive price optimization and its difficulty

Recent advanced ML reveals relationship between prices and sales quantities

<table>
<thead>
<tr>
<th>Input: pos data</th>
<th>Price</th>
<th>Price</th>
<th>Sales</th>
<th>Sales</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Day 1</td>
<td>$1.3</td>
<td>$1.0</td>
<td>2</td>
<td>2</td>
<td>...</td>
</tr>
<tr>
<td>Day 2</td>
<td>$1.2</td>
<td>$1.0</td>
<td>4</td>
<td>2</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Machine learning

Predictive model: sales = f(prices)

Optimization (NP-hard) [This work]

Output: optimal prices
Our contribution

Scalable algorithm for price optimization

Based on:

1. Submodularity behind pricing
2. Network flow algorithm
3. Supermodular relaxation
Our contribution

Scalable algorithm for price optimization

Based on:

1. Submodularity behind pricing
2. Network flow algorithm
3. Supermodular relaxation

Achieved:

- Can deal with thousands of products
- High accuracy for real-data problem
Table of contents

1. Introduction
2. Problem definition
3. Scalable price optimization algorithm
4. Experiments
Objective function of price optimization

Want to maximize is the gross profit $\ell$

Gross profit:

$$\ell(p_1, p_2, \ldots, p_M) = \sum_{i=1}^{M} (p_i - c_i)q_i$$

- $p_i$: price of product $i$
- $c_i$: cost of product $i$
- $q_i$: sales quantity of product $i$

Unknown, but predictable product id
Predictive model for sales quantity

Sales quantity $q_i$ is a function in prices $p_i$

$$q_1(p, r) = f_{11}(p_1) + f_{12}(p_2) + \cdots + g_{11}(r_1) + g_{12}(r_2) + \cdots$$

- **sales quantity**
- **price**
- **weather**
- **calendar**

Use historical data to infer $f_{i \, j}$, $g_{i \, j}$

Ex: $f_i(p_j) = a_i \, p_j^2 + b_i \, p_j + c_i \, j$ (polynomial model)

$$f_i(p_j) = \exp(\alpha_i \, p_j + \beta_i)$$, (generalized linear model)
Predictive model for sales quantity

Sales quantity \( q_i \) is a function in prices \( p_i \)

\[
q_1(p, r) = f_{11}(p_1) + f_{12}(p_2) + \ldots + g_{11}(r_1) + g_{12}(r_2) + \ldots
\]

- \( f_{11} \): Price elasticity of demand
- \( f_{12} \): Cross price effect

\( q_1 \): Sales quantity of orange
\( p_1 \): Price of orange

\( q_2 \): Sales quantity of apple
\( p_2 \): Price of apple

\( p \): Prices
\( r \): Weather calendar
Many substitute goods in price optimization

\[ q_1(p, r) = f_{11}(p_1) + f_{12}(p_2) + \ldots + g_{11}(r_1) + g_{12}(r_2) + \ldots \]

and

\[ f_{12} : \text{Cross price effect} \]

and

Apple \( \text{and} \) Orange : Substitute goods

Discounting apple makes the sales of orange down (cannibalization)
Optimization problem

- Optimization is **NP-hard**

Maximize
\[ \ell(p) = \sum_{i=1}^{M} (p_i - c_i)q_i(p) \]

Subject to
\[ p_i \in \{ P_{i1}, P_{i2}, \ldots, P_{ik} \} \]

Discrete price candidates

Gross profit

A commercial solver takes \( >24[h] \) for 50 products
Table of contents

1. Introduction
2. Problem definition
3. Scalable price optimization algorithm
4. Experiments
Our contribution

Scalable algorithm for price optimization

Based on:

1. Submodularity behind pricing
2. Network flow algorithm
3. Supermodular relaxation
Idea 1: Substitute goods and supermodular

Connection between substitute goods and submodular

Substitute goods:
Discounting 🍎 ⇒ less sales of 🍊

Theorem [Substitute goods ⇒ supermodular]

If all pairs of products are substitute goods or independent
⇒ ℓ(𝑝) is a supermodular function
Idea 1: Substitute goods and supermodular

Connection between substitute goods and submodular

Substitute goods:
Discounting 🍎 ⇒ less sales of 🍊

Theorem [Substitute goods ⇒ supermodular]

If all pairs of products are substitute goods or independent
⇒ \( \ell(p) \) is a supermodular function
 maximized in polynomial time
[Iwata, Fleischer, Fujishige (2001)]
Two issues in Key idea 1

- If all products are substitute goods or independent, profit can be maximized in polynomial time.
Two issues in Key idea 1

- If all products are substitute goods or independent, profit can be maximized in polynomial time

- This approach is still impractical because of two issues

**Issue 1:** General supermodular maximization is slow
\[ \sim O(n^5) \]

**Issue 2:** Substitute goods assumption is too restrictive
Two issues in Key idea 1

If all products are substitute goods or independent, profit can be maximized in polynomial time.

This approach is still impractical because of two issues.

**Issue 1:** General supermodular maximization is slow. Resolved by 2. network flow: 

\[ O(n^2) \sim O(n^3) \]

**Issue 2:** Substitute goods assumption is too restrictive. Resolved by 3. supermodular relaxation. Applicable for non-substitute.
Two issues in Key idea 1

If all products are substitute goods or independent, profit can be maximized in polynomial time.

This approach is still impractical because of two issues.

**Issue 1:** General supermodular maximization is slow.

Resolved by 2. network flow:

\[ O(n^2) \sim O(n^3) \]

**Issue 2:** Substitute goods assumption is too restrictive.

Resolved by 3. supermodular relaxation.

Applicable for non-substitute products.
Substitute goods price optimization $\iff$ Minimum Cut

Maximizing $\ell(p) \iff$ Finding minimum s-t cut
solved efficiently by network flow

[Ford, Fulkerson (1956)], [Orlin (2013)]
Substitute goods price optimization ⇔ Minimum Cut

Maximizing $\ell(p) ⇔$ Finding minimum s-t cut
:solved efficiently by network flow
Substitute goods price optimization $\Leftrightarrow$ Minimum Cut

Maximizing $\ell(p) \Leftrightarrow$ Finding minimum $s$-$t$ cut
:solved efficiently by network flow

Graph $G$

$p_1 = 10\%$ OFF
$p_2 = 5\%$ OFF

$S \rightarrow t$
Substitute goods price optimization $\iff$ Minimum Cut

Maximizing $\ell(p) \iff$ Finding minimum $s$-$t$ cut
:solved efficiently by network flow
Substitute goods price optimization $\Leftrightarrow$ Minimum Cut

Maximizing $\ell(p)$ $\Leftrightarrow$ Finding minimum s-t cut:
solved efficiently by network flow

$\ell(p) = \text{constant} - (\text{capacity of st-cut of graph } G)$
Two issues in Key idea 1

If all products are substitute goods or independent, profit can be maximized in polynomial time.

This approach is still impractical because of two issues.

**Issue 1:** General supermodular maximization is slow

\[ O(n^5) \]

Resolved by 2. network flow:

\[ O(n^2) \sim O(n^3) \]

**Issue 2:** Substitute goods assumption is too restrictive

Resolved by 3. supermodular relaxation

Applicable for non-substitute
Non-SGP case: supermodular relaxation

\[ \ell(p) = \ell^-(p) + \ell^+(p) \]

\[ \leq \ell^-(p) + h_\Gamma(p) \]

\[ \exists \text{ non-substitute goods} \implies \text{approximate } \ell \text{ by supemodular function} \]

\[ \implies \text{maximized via network flow} \]
Non-SGP case: supermodular relaxation

Many possibilities of relaxation function
Better relaxation is chosen automatically

Relaxation changes depending on \( \Gamma \in [0,1]^{n \times n} \)
Simulation experiments

Proposed approximation algorithm:
fast and give solution with only <1% loss

<table>
<thead>
<tr>
<th></th>
<th>Past data</th>
<th>Proposed</th>
<th>QPBO</th>
<th>Others</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computing Time</td>
<td></td>
<td>36 [s]</td>
<td>964 [s]</td>
<td>&gt; 1day</td>
</tr>
<tr>
<td>Achieved Profit</td>
<td>1.40 M</td>
<td>1.88 M</td>
<td>1.25 M</td>
<td>Nan</td>
</tr>
<tr>
<td>Upper bound</td>
<td></td>
<td>1.90 M</td>
<td>1.89 M</td>
<td>Nan</td>
</tr>
</tbody>
</table>

- Real-world retail data* of a supermarket
  *provided by KSP-SP Co., LTD.

- 1000 products

- Compared with SDP, QPBO, QPBOI [Rother et.al (2007)]
Conclusion

- Constructed a scalable price optimization algorithm by
  - Associating substitute goods with supermodularity
  - Using network flow and supermodular relaxation

Future work: how to cope with the errors in objective?
  e.g., by robust optimization?
Orchestrating a brighter world

NEC