Prices Drop as you Shop
How Jet is Using GPU based Smart Merchant Selection to Gain a Competitive Advantage
Today

1. Intro
2. E-Commerce Market Potential
3. Problem
4. Brute Force Approach with GPUs
5. A Smarter AI Powered GPU Solution
Intro
Introduction

E-commerce site dedicated to savings

Revolutionary pricing engine

Top technology and fulfillment platform

Acquired by Walmart in mid 2016
Jet’s Unique Approach to E-Commerce

Customer Behavior Changes

Customer Savings Incentives

Supply Chain Optimization

E-Commerce Cost Reductions
Smarter Shopping at the Basket-Level

Bigger Baskets

Efficient Shipping

Free Return Waivers

Merchant Selection

- Buy 1 $31.99/ea
- Buy 2 $31.63/ea
- Buy 3 $30.44/ea
- Buy 4 $29.69/ea
- Buy 5 $29.31/ea

$31.94
If you opt out of free returns on this item, you pay less.

$31.73
If you pay by debit card, you pay less.

$31.68
If you opt out of free returns on this item and pay by debit card, you pay less.
Introduction

Machine learning and Cognitive computing
Numerical HPC algorithm design and implementation
Specialized in **GPU computing** and parallel algorithms
The Potential of E-Commerce
E-Commerce Market Expected to Grow Rapidly

Source: Forrester – July 2016
Problem
Jet Pricing Engine

Users **shop for products**

**Platform decides** about most **optimal fulfillment** during shopping and at checkout

Savings come from

- Cheapest net item prices
- Pack items together for fewer boxes to ship
- Conditions, merchant commission, basket rules
- More efficient fulfillment
Smart Merchant Selection
**Example Small Cart**

**Four items with four offers each**

<table>
<thead>
<tr>
<th></th>
<th>Sku 1</th>
<th>Sku 2</th>
<th>Sku 3</th>
<th>Sku 4</th>
<th>Shipping</th>
<th>Skus</th>
<th>Shipping</th>
<th>Skus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Merchant 1</td>
<td>15</td>
<td>15</td>
<td></td>
<td>33</td>
<td>7.5</td>
<td>1+2</td>
<td>7.5</td>
<td>4</td>
</tr>
<tr>
<td>Merchant 2</td>
<td>15</td>
<td>16</td>
<td></td>
<td>33.5</td>
<td>8.5</td>
<td>all</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Merchant 3</td>
<td></td>
<td>18</td>
<td>19</td>
<td>31</td>
<td>3.5</td>
<td>2,3,4</td>
<td>4.5</td>
<td>3+4</td>
</tr>
<tr>
<td>Merchant 4</td>
<td>12</td>
<td></td>
<td>16</td>
<td>30</td>
<td>5.5</td>
<td>1,3,4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Offers</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Number of Combinations** = 3 * 3 * 2 * 4 = 72

Which is the cheapest combination?
# Example

## Solution – Cheapest Net Price

**Try cheapest net price first**

<table>
<thead>
<tr>
<th>Merchant 1</th>
<th>Sku 1</th>
<th>Sku 2</th>
<th>Sku 3</th>
<th>Sku 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>15</td>
<td>15</td>
<td></td>
<td>28</td>
</tr>
<tr>
<td>Merchant 2</td>
<td>15</td>
<td>16</td>
<td></td>
<td>33</td>
</tr>
<tr>
<td>Merchant 3</td>
<td></td>
<td>18</td>
<td>19</td>
<td>31</td>
</tr>
<tr>
<td>Merchant 4</td>
<td>12</td>
<td></td>
<td>16</td>
<td>28</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Shipping</th>
<th>Skus</th>
<th>Shipping</th>
<th>Skus</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.5</td>
<td>1+2</td>
<td>7.5</td>
<td>4</td>
</tr>
<tr>
<td>8.5</td>
<td>all</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.5</td>
<td>2,3,4</td>
<td>4.5</td>
<td>3+4</td>
</tr>
<tr>
<td>5.5</td>
<td>1,3,4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Net Shipping</th>
<th>Sku 1</th>
<th>Sku 2</th>
<th>Sku 3</th>
<th>Sku 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>12</td>
<td>15</td>
<td>16</td>
<td>28</td>
</tr>
<tr>
<td>Shipping</td>
<td>5.5</td>
<td>7.5</td>
<td>5.5</td>
<td>5.5</td>
</tr>
<tr>
<td>Gross</td>
<td>17.5</td>
<td>22.5</td>
<td>21.5</td>
<td>33.5</td>
</tr>
</tbody>
</table>
## Example
### Solution – Merchant 2 / Merchant 4

We can do better by packing items together

<table>
<thead>
<tr>
<th></th>
<th>Sku 1</th>
<th>Sku 2</th>
<th>Sku 3</th>
<th>Sku 4</th>
<th>Shipping</th>
<th>Skus</th>
<th>Shipping</th>
<th>Skus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Merchant 1</td>
<td>15</td>
<td>15</td>
<td></td>
<td>28</td>
<td>7.5</td>
<td>1+2</td>
<td>7.5</td>
<td>4</td>
</tr>
<tr>
<td>Merchant 2</td>
<td>15</td>
<td>16</td>
<td></td>
<td>33</td>
<td>8.5</td>
<td>all</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Merchant 3</td>
<td></td>
<td>18</td>
<td>19</td>
<td>31</td>
<td>3.5</td>
<td>2,3,4</td>
<td>4.5</td>
<td>3+4</td>
</tr>
<tr>
<td>Merchant 4</td>
<td>12</td>
<td></td>
<td>16</td>
<td>28</td>
<td>5.5</td>
<td>1,3,4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Net Shipping</td>
<td>15</td>
<td>16</td>
<td>16</td>
<td>33</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gross</td>
<td>15</td>
<td>16</td>
<td>21.5</td>
<td>41.5</td>
<td>5.5</td>
<td>8.5</td>
<td>94</td>
<td></td>
</tr>
</tbody>
</table>
### Example

#### Solution – Merchant 1 / Merchant 4

We can do even better

<table>
<thead>
<tr>
<th></th>
<th>Sku 1</th>
<th>Sku 2</th>
<th>Sku 3</th>
<th>Sku 4</th>
<th>Shipping</th>
<th>Skus</th>
<th>Shipping</th>
<th>Skus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Merchant 1</td>
<td>15</td>
<td>15</td>
<td>28</td>
<td>28</td>
<td>7.5</td>
<td>1+2</td>
<td>7.5</td>
<td>4</td>
</tr>
<tr>
<td>Merchant 2</td>
<td>15</td>
<td>16</td>
<td>33</td>
<td>33</td>
<td>8.5</td>
<td>all</td>
<td>4.5</td>
<td>3+4</td>
</tr>
<tr>
<td>Merchant 3</td>
<td>18</td>
<td>19</td>
<td>31</td>
<td>31</td>
<td>3.5</td>
<td>1</td>
<td>4.5</td>
<td>3+4</td>
</tr>
<tr>
<td>Merchant 4</td>
<td>12</td>
<td>16</td>
<td>28</td>
<td>28</td>
<td>5.5</td>
<td>1,3,4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Net</td>
<td>15</td>
<td>15</td>
<td>16</td>
<td>28</td>
<td>7.5</td>
<td>1+2</td>
<td>7.5</td>
<td>4</td>
</tr>
<tr>
<td>Shipping</td>
<td>7.5</td>
<td>5.5</td>
<td>5.5</td>
<td>5.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gross</td>
<td>15</td>
<td>22.5</td>
<td>21.5</td>
<td>33.5</td>
<td>92.5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Example

**Best Solution – Merchant 1 / Merchant 3**

**We can do even a little bit better**

<table>
<thead>
<tr>
<th></th>
<th>Sku 1</th>
<th>Sku 2</th>
<th>Sku 3</th>
<th>Sku 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Merchant 1</td>
<td>15</td>
<td>15</td>
<td></td>
<td>28</td>
</tr>
<tr>
<td>Merchant 2</td>
<td>15</td>
<td>16</td>
<td></td>
<td>33</td>
</tr>
<tr>
<td>Merchant 3</td>
<td>15</td>
<td>18</td>
<td>19</td>
<td>31</td>
</tr>
<tr>
<td>Merchant 4</td>
<td>12</td>
<td>16</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td><strong>Shipping</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Sku 1</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Sku 2</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Sku 3</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Sku 4</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Skus</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Merchant 1</strong></td>
<td>1+2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Merchant 2</strong></td>
<td>all</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Merchant 3</strong></td>
<td>2,3,4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Merchant 4</strong></td>
<td>1,3,4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Optimal merchant selection saves 3$**
Savings Potential

~$12.3 B Savings

Source: Forrester – July 2016
Savings Potential

Larger Carts – More Savings

Profit Potential

Larger carts – more savings
Speeding up with GPUs
Full Search
Embarrassingly Parallel

1. Large number of ways to fulfill a shopping cart

2. Cart pricing can be executed independently

Ideal problem to solve in parallel with GPUs
Cart Pricer on GPUs

Implement pricing algorithm on GPU
Challenging control flow logic

- Shipping costs
- Commission rules

Complicated input data structure
Have to cope with non-coalescing memory access patterns
Full Search on GPU

Full Search Speedup of Small Carts

- Speedup
- 50x
- 300x

Number of Combinations
How far can we go?

Checkout → Best Solution
Full Search
Exponential Complexity

Complexity = 

Number of combinations = 

Num offers for item 1 * ... * num offers for item k
Example

Somebody wanted to build a computer
Exponential Complexity

Number of combinations =

Offers for item 1 * Offers for item 2 * \ldots * Offers for item 10

= 32 \times 17 \times 19 \times 16 \times 29 \times 9 \times 25 \times 10 \times 16 \times 17 \times 24

= 70,442,237,952,000 \text{ combinations}

= 10^{13.85} \text{ combinations}
Full Search
CPU Years

Checkout  →  8.6 CPU full search years  →  Best Solution

70’442’237’952’000 = 10^{13.85} combinations
In Perspective

Sirius

8.6 light years

50'620'000'000'000 = 10^{13.7} miles

Earth
Full Search
GPU Days

Checkout

11 GPU full search days

70,442,237,952,000 = 10^{13.85} combinations

Best Solution
Performance Gap

Full Search Timings

13.85 = $\log_{10} 70'442'237'952'000$

Real time gap
Conclusion

- Real time response < 1s required
- Full search prohibitive even on GPUs
- Require alternative solution that scales
Genetic Algorithm on GPU
Genetic Algorithm

Apply **Genetic Algorithms** to solve the problem

- GA can only find approximately optimal solutions
- Standard GA does not work
  - Search space is astronomically large
  - Need a reliable approx. solution in near real-time
- Rely on AI & ML to choose GA configuration
- Generation iteration is serial, extending the population size dramatically allows to reduce iterations
Genetic Algorithm

Include merchant combination which are likely to be good

Identify two parts in population

- Dynamic part → subject to genetic reproduction
- Static part → used as «boundary» in genetic reproduction

Rely on parallel BFS graph algorithm to speed up initial population creation
Genetic Algorithm

Decode chromosome to cart
Price cart, cheaper cart = higher fitness
Sort by score for elitism and parent selection
Use parallel key value sort
Approx. $100x$ faster than CPU sort
Genetic Algorithm

Selecting small fraction of best chromosomes to go unmodified to new population

Need to balance elite selection and population diversity to avoid premature convergence
Genetic Algorithm

Different selection schemes

Traditional roulette wheel selection suffers from high selection pressure

**Rank selection** performs better and reduces risk of premature convergence

Use fast GPU scan, approx. 100x faster than CPU scan
Genetic Algorithm

Crossover and mutation operations need to find rare merchant combinations in *astronomically large search space*

Combine two ideas:

- Special crossover and mutation operations inspired from domain specific knowledge
- Crossover relies on geometric construction generating paths between dynamic part and «boundary»
Genetic Algorithm

Usually GA stop if no improvement for prespecified number of iterations

→ No guaranteed execution time

Improve termination criteria with extensive testing

• Apply **AI & Machine Learning** to predict probability distribution of deviation from optimal value

• Use this model to select compute budget and number of iterations
Genetic Algorithm

Many ways to configure GA

• Population size

• Values of algorithmic parameters such as mutation rate, elite percentage, etc.

• Selection of GA algorithm features

Rely on **AI & Machine Learning** to identify the ideal GA configuration for a given shopping cart and market offers
Genetic Algorithm

Fast **random number** generation important for performance

Used in

- Initial population creation
- Selection for elitism
- Crossover operations
- Mutation operations
Genetic Algorithm

Statistics

- Pairwise distance calculation to compute population diversity
- Used for quality measurement and detection of premature convergence

Visualization and convergence inspection

- Nonlinear embedding in 2d and 3d using TSNE
Visualization
Convergence Statistics

Convergence Measures

- Highest Score
- Top10 Scores Average
- Bot10 Scores Average
- Diversity
- Unique
- Jaccard Similarity
- Score Gain

Number of Iterations
Convergence
Embedding with TSNE

Initial population

Best part from full search
Special greedy «boundary» points

Score rapidly improves
Catches best points found by full search
Visualization Demo
Implementation & Deployment
Cloud Based
Functional First with F#

Jet platform is fully cloud based
Runs in Microsoft Azure
Use F# as core programming language
Uses Alea GPU from QuantAlea to compile F# to GPU
• Maximal reuse of algorithms for CPU and GPU execution
• Same GPU performance as native CUDA C
Platform Utilization
Dispatch and Optimize with AI

Full Search Time

CPU Full Search
GPU Full Search
CPU Genetic Algo
GPU Genetic Algo

Resource utilization and response time optimization
Scale up

Complexity
Conclusion
Conclusion

1. Genetic Algorithms

2. AI & ML

3. GPUs

Scale up optimal merchant selection to shopping carts of unlimited size. Very good approximate solution found in real time.
You’d look good in purple

At Jet, our goal is to reshape the landscape of e-commerce forever and make a lasting difference in people’s lives. If you want to take on the challenge of building something amazing from the ground up, then this is the place for you.

Take a closer look at life at Jet ⬤