A deep reinforcement learning approach for synchronized multi-modal replenishment

joren.gijsbrechts@kuleuven.be
robert.boute@kuleuven.be
Problem Statement
Multimodal replenishment as a dual sourcing problem

Multimodal Replenishment

Fast Mode (High Cost, Short Lead Time)

Slow Mode (Low Cost, Long Lead Time)

Complexity
• Intractable for large lead times due to pipeline inventory vector
• Optimal solutions found for specific circumstances

Additional challenges:
• Service schedule of transport modes
• Production Schedule
State of the art
Heuristic policies group parts of pipeline inventory vector

- Complexity arises from pipeline inventory vector
- State-of-the-art heuristic policies hence use:
  - 1 or 2 inventory positions
  - 1 or 2 order up-to levels

Pipeline Inventory
Number of periods before pipeline inventory arrives:

7 6 5 4 3 2 1

Lead time slow mode = 7 days
Lead time fast mode = 3 days

Plant
On-hand inventory
Warehouse
State of the art
Heuristic policies work around complexity

- Single Index
  - (1 inventory position)
- Single Base
  - (1 order up-to level)
- Dual Base
  - (2 order up-to levels)
- Dual Index
  - (2 inventory positions)
- Tailored Base Surge Policy
  - Single Index Dual Base
  - (Capped) Dual Index Dual Base
Motivation
Can Artificial Intelligence be used to solve the dual sourcing problem?

"I would say, a lot of the value that we're getting from machine learning is actually happening kind of beneath the surface. It is things like improved search results, improved product recommendations for customers, improved forecasting for inventory management, and literally hundreds of other things beneath the surface," Bezos said.
Machine Learning Overview

Machine Learning

Supervised Learning
- Classification on labeled data: Recognizing cats/dogs/persons on pictures

Unsupervised Learning
- Clustering unlabeled data: Customer segmentation, music style segmentation

Reinforcement Learning
- Learn policies based on experience and feedback from the environment "Trial and error"
Reinforcement Learning, no new field!
But … major recent breakthroughs!
Contribution

- Smart algorithm learns itself a replenishment policy based on full pipeline inventory vector

- Suitable for complex settings
  - Non-linear ordering cost
  - Include ordering/delivery/production schedules
    - E.g. non-daily train/boat schedule

- First application of deep reinforcement learning in dual sourcing
Methodology

- Problem modeled as a **Markov Decision Process** \((S,A,R(s,a), \gamma)\)
  - **State space** \((S) = \text{Inventory Vector} + \text{Day of week} \)
    \[
    S = [I^{(0)} \ I^{(1)} \ \ldots \ I^{(L_s)} \ D],
    \]
  - **Action space** \((A) = \text{Ordering Vector} (\text{Fast} + \text{Slow}) \)
    \[
    A = [a^{(f)} \ a^{(s)}],
    \]
  - **Rewards** \((R(s,a)) = \text{Reward realization (ordering + inventory cost)} \)
    \[
    r(s_t, a_i) = c^f a^{(f)}_i + c^s a^{(s)}_i + h[I_{t+1}^+] + b[I_{t+1}^-] + \sum_{i=1}^{L_s} p I_{t+1}^{(i)}.
    \]
  - **\(\gamma\) = discount factor**

- **Objective: minimize future discounted costs**
  \[
  \min r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots + \gamma^{t-1} r_t + \cdots,
  \]
Methodology

- Dynamic Programming intractable ➞ Approximate Dynamic Programming
  - Reinforcement Learning – $Q$-learning
    
    $$ Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha (r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t)) $$

- $Q$-learning slow for large state space ➞ Deep $Q$-learning
  - Input: states
  - Output: $Q$-value for each action
Methodology

- Asynchronous Advantage Actor-Critic (A3C)
  - Actor develops policy
  - Critic evaluates policy
Hyperparameter Tuning

• Grid Search
• Random Search
• Bayesian Optimization
Results
Deep Q-Learning algorithm learns itself a ‘smart’ replenishment policy

Figure 1: Cost performance during training
State: Inventory Pipeline

Action: Fast/Express

Action: Slow/Remote
Results

Matching performance state-of-the-art dual sourcing policies with daily frequencies

DRL = Deep Reinforcement Learning
TBS = Tailored Base Surge
SIDB = Single Index Dual Base
DIDB = Dual Index Dual Base
CDIDB = Capped Dual Index Dual Base

• Equal performance typical dual sourcing setting (daily frequencies)

• We do not lose performance when extending problem to:
  • Non-daily ordering frequencies
    • E.g. including rail schedule or production schedule
  • Non-linear ordering cost (e.g. per container instead of per unit)
Backslides
Methodology
Smart replenishment algorithm – Deep Q-learning

Algorithm 1 Deep Reinforcement Learning (DRL) algorithm
1: Initialize replay memory D to capacity \( \phi \)
2: Initialize Q-network with random weights \( \theta \)
3: Initialize Target Network \( \hat{Q} \) with weights \( \theta^- = \theta \)
4: Choose Initial State \( s_1 \)
5: for \( t = 1, T \) do
6:   for \( n = 1, N \) do
7:     \[ a_t = \begin{cases} \text{random action,} & \text{with probability } (1 - \epsilon) \\ \arg \min_{a \in A} \{ Q(s_t, a; \theta) \}, & \text{else} \end{cases} \]
8:     Simulate using \( a_t \) and observe reward \( r_t \) and next state \( s_{t+1} \)
9:     Store \( (s_t, a_t, r_t, s_{t+1}) \) in replay memory D
10: end for
11: Sample random minibatches of size \( K \) from D
12: for \( j = 1, K \) do
13:   \( y_j = \begin{cases} r_j + \gamma Q(s, \arg \min_{a' \in A} Q(s_{j+1}, a'; \theta^-), \theta), & \text{if } (s, a) \text{ in sample } j \\ Q(s, a; \theta), & \text{else} \end{cases} \)
14: end for
15: Minimize loss = \( \sum_{j \in K} (y_j - Q(s_j, a_j; \theta))^2 \) using Adam optimizer
16: Every \( z \) steps, update Target network \( \hat{Q} = Q \)
17: end for
Methodology
Smart replenishment algorithm – A3C algorithm

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

// Assume global shared parameter vectors \( \theta \) and \( \theta_v \) and global shared counter \( T = 0 \)
// Assume thread-specific parameter vectors \( \theta' \) and \( \theta'_v \).
Initialize thread step counter \( t \leftarrow 1 \)

repeat
  Reset gradients: \( d\theta \leftarrow 0 \) and \( d\theta_v \leftarrow 0 \).
  Synchronize thread-specific parameters \( \theta' = \theta \) and \( \theta'_v = \theta_v \).
  \( t_{\text{start}} = t \)
  Get state \( s_t \).
  repeat
    Perform \( a_t \) according to policy \( \pi(a_t|s_t; \theta') \).
    Receive reward \( r_t \) and new state \( s_{t+1} \).
    \( t \leftarrow t + 1 \)
    \( T \leftarrow T + 1 \)
  until terminal \( s_t \) or \( t - t_{\text{start}} = t_{\text{max}} \)

  \( R = \begin{cases} 
    0 & \text{for terminal } s_t \\
    V(s_t, \theta'_v) & \text{for non-terminal } s_t \end{cases} // Bootstrap from last state

  \text{for } i \in \{t - 1, \ldots, t_{\text{start}}\} \text{ do}
    \( R \leftarrow r_i + \gamma R \)
  \text{Accumulate gradients wrt } \theta': \( d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i; \theta')(R - V(s_i; \theta'_v)) \)
  \text{Accumulate gradients wrt } \theta'_v: \( d\theta_v \leftarrow d\theta_v + \partial (R - V(s_i; \theta'_v))^2 / \partial \theta'_v \)
  end for

  Perform asynchronous update of \( \theta \) using \( d\theta \) and of \( \theta_v \) using \( d\theta_v \).

until \( T > T_{\text{max}} \)
Results

Within 2% of optimal solution in a simple setting
Results
Maintaining performance in a more complex setting

Performance versus optimal for different lead times assuming Stepwise Ordering Cost:
(LT_e = 0 // c_r = 100 // c_e = 150 // h=5 // b=495 // cap_r = 2 // cap_e=2)

<table>
<thead>
<tr>
<th>LT_r</th>
<th>A3C</th>
<th>SIIDB</th>
<th>DIDB</th>
<th>CDIDB</th>
<th>TBS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>102.49%</td>
<td>113.64%</td>
<td>112.73%</td>
<td>111.91%</td>
<td>123.41%</td>
</tr>
<tr>
<td>2</td>
<td>101.76%</td>
<td>147.14%</td>
<td>122.49%</td>
<td>114.88%</td>
<td>148.17%</td>
</tr>
<tr>
<td>3</td>
<td>101.03%</td>
<td>145.03%</td>
<td>129.38%</td>
<td>126.23%</td>
<td>146.13%</td>
</tr>
<tr>
<td>4</td>
<td>108.16%</td>
<td>140.52%</td>
<td>132.49%</td>
<td>122.54%</td>
<td>142.28%</td>
</tr>
</tbody>
</table>