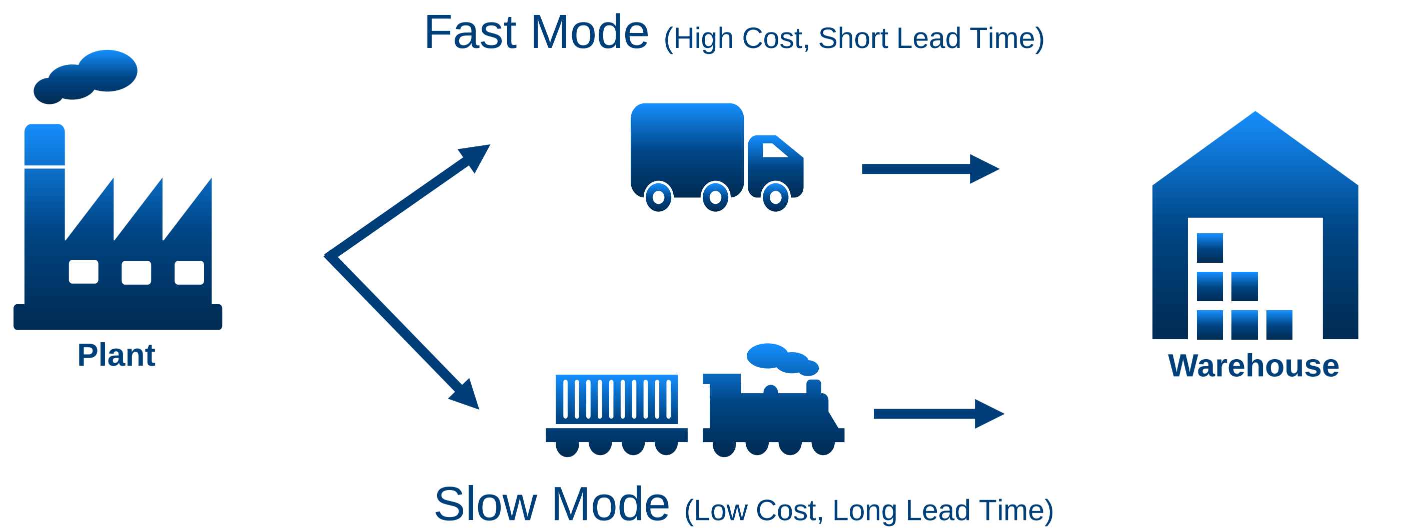
A deep reinforcement learning approach for synchronized multi-modal replenishment

joren.gijsbrechts@kuleuven.be
robert.boute@kuleuven.be

Problem Statement

Multimodal replenishment as a dual sourcing problem

Multimodal Replenishment



Complexity

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

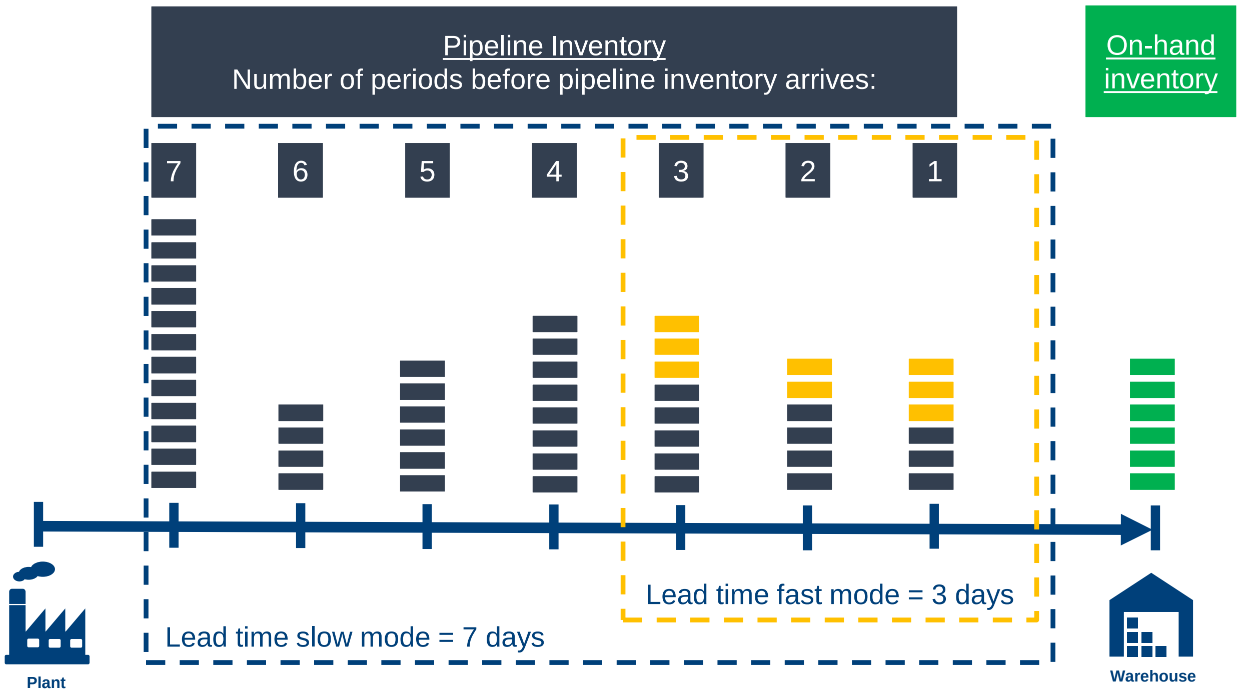
Additional challenges:

- Service schedule of transport modes
- Production Schedule

State of the art

Heuristic policies group parts of pipeline inventory vector

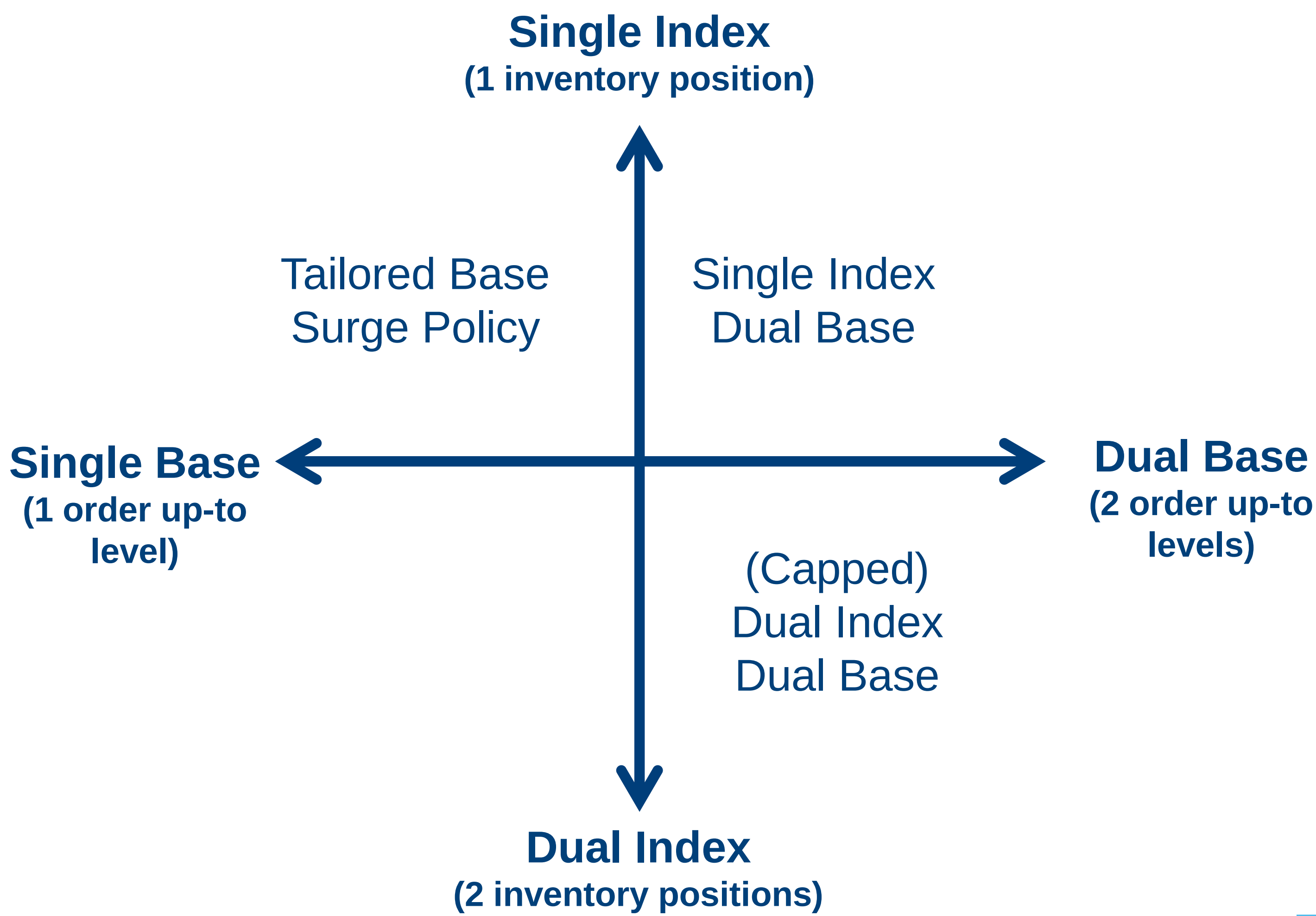
- Complexity arises from pipeline inventory vector
- State-of-the-art heuristic policies hence use:
  - 1 or 2 inventory positions
  - 1 or 2 order up-to levels



KU LEUVEN

State of the art

Heuristic policies work around complexity



Motivation

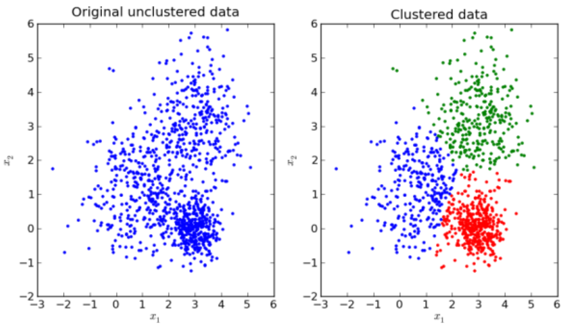
Can Artificial Intelligence be used to solve the dual sourcing problem?

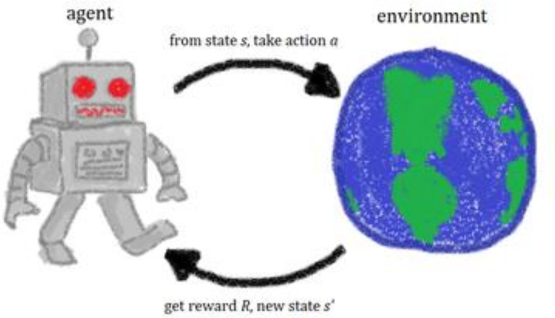
"I would say, a lot of the value that we're getting from **machine learning** is actually happening kind of **beneath the surface**. It is things like improved search results, improved product recommendations for customers, improved forecasting for inventory management, and literally hundreds of other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

Machine Learning Overview

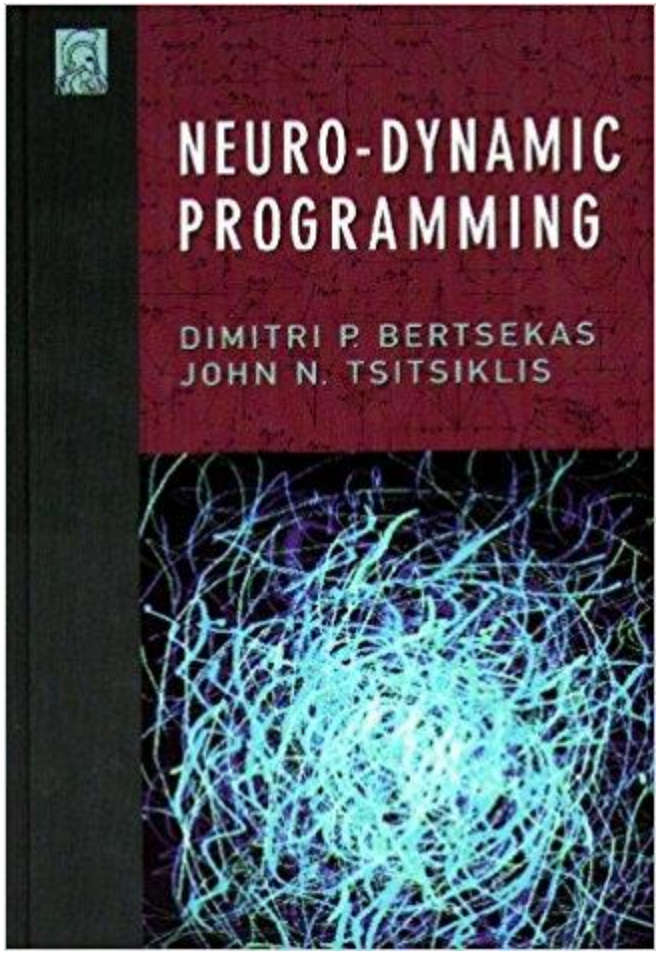
Machine Learning

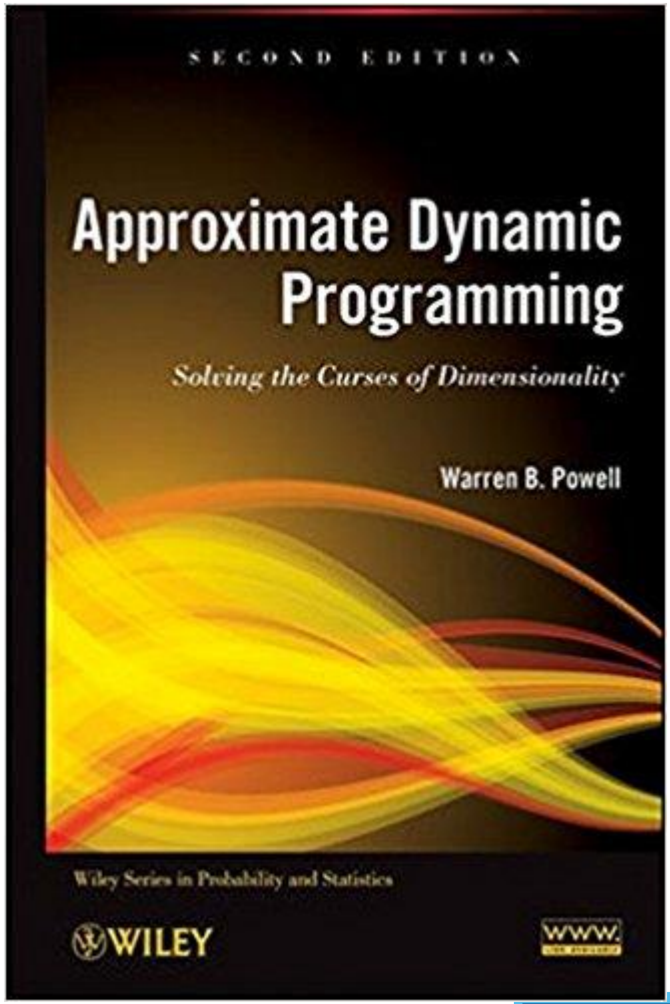
- **Supervised Learning**
  - **Classification on labeled data:** Recognizing cats/dogs/persons on pictures
- **Unsupervised Learning**
  - **Clustering unlabeled data:** Customer segmentation, music style segmentation
- **Reinforcement Learning**
  - **Learn policies** based on experience and feedback from the environment
  - "Trial and error"





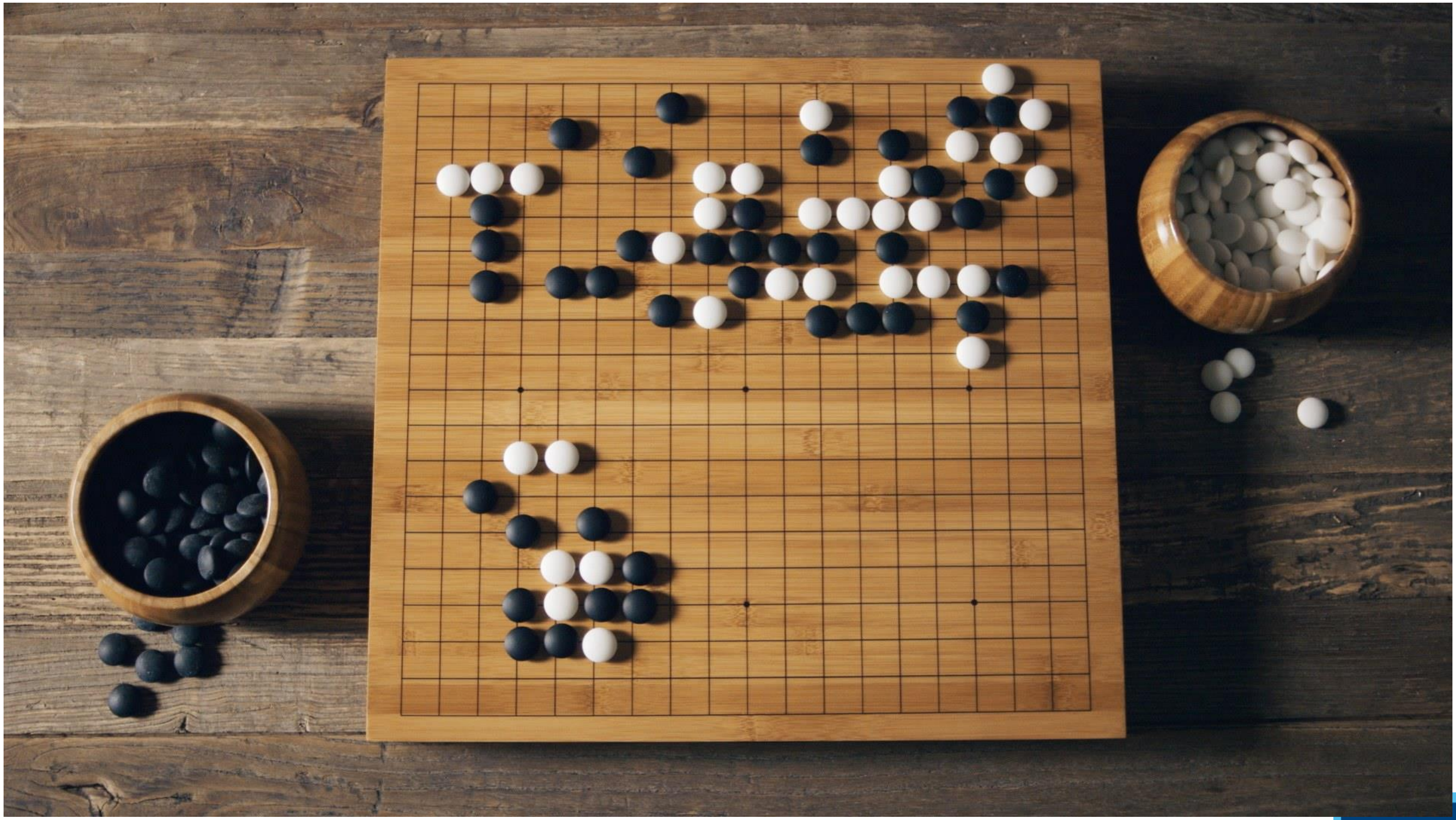
Reinforcement Learning, no new field!





KU LEUVEN

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), γ)*
  - *State space (S) = Inventory Vector + Day of week*

$$S = \begin{bmatrix} I^{(0)} & I^{(1)} & \dots & I^{(L_s)} & D \end{bmatrix},$$

  - *Action space (A) = Ordering Vector (Fast + Slow)*

$$A = \begin{bmatrix} a^{(f)} & a^{(s)} \end{bmatrix},$$

  - *Rewards (R(s,a)) = Reward realization (ordering + inventory cost)*

$$r(s_t, a_i) = c^f a_i^{(f)} + c^s a_i^{(s)} + h[I_{t+1}^{(0)}]^+ + b[I_{t+1}^{(0)}]^- + \sum_{i=1}^{L_s} p I_{t+1}^{(i)}.$$

  - *γ = discount factor*

- *Objective: minimize future discounted costs*

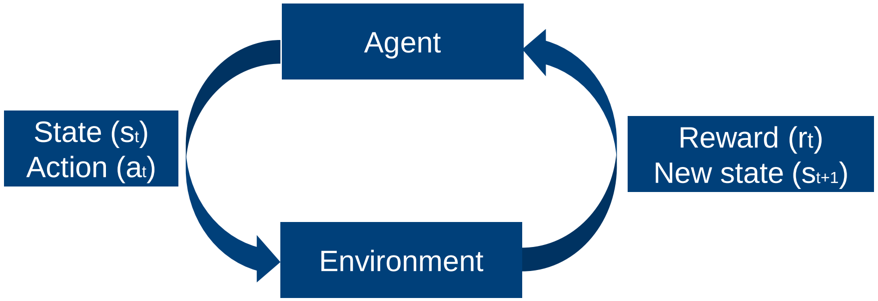
$$\min r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots + \gamma^{t-1} r_t + \dots,$$

KU LEUVEN

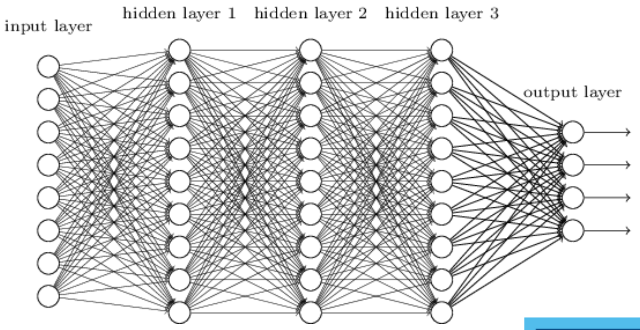
Methodology

- Dynamic Programming intractable ➔ Approximate Dynamic Programming
  - Reinforcement Learning – *Q*-learning

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t))$$

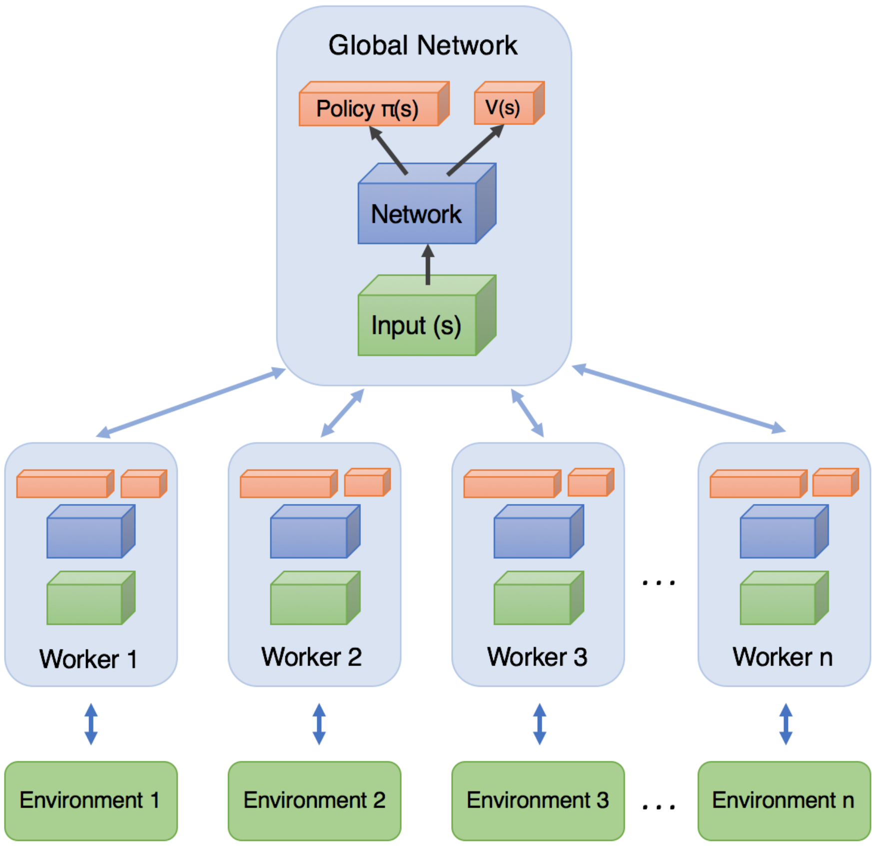


- Q-learning slow for large state space ➔ Deep *Q*-learning
  - Input: states
  - Output: *Q*-value for each action



Methodology

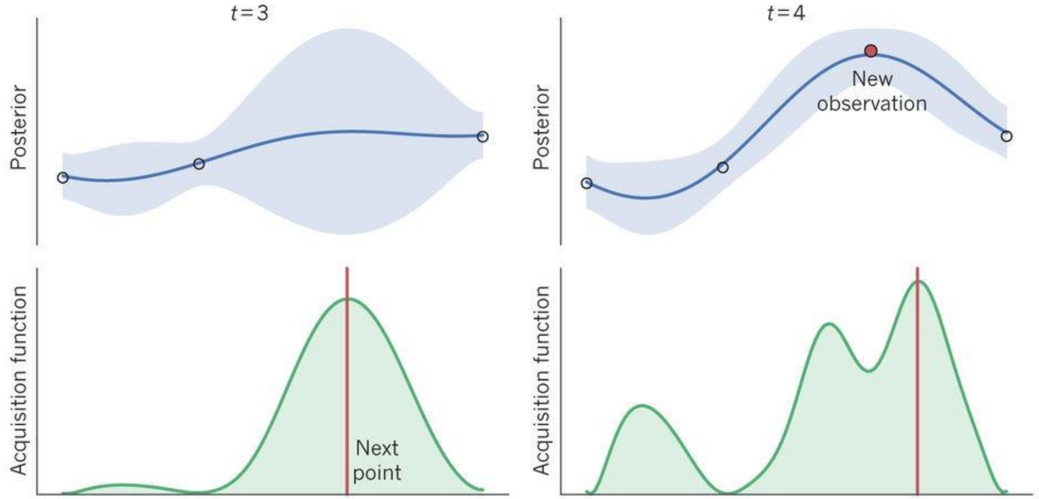
- Asynchronous Advantage Actor-Critic (A3C)
  - Actor develops policy
  - Critic evaluates policy



KU LEUVEN

Hyperparameter Tuning

- Grid Search
- Random Search
- Bayesian Optimization



KU LEUVEN

Results

Deep Q-Learning algorithm learns itself a ‘smart’ replenishment policy

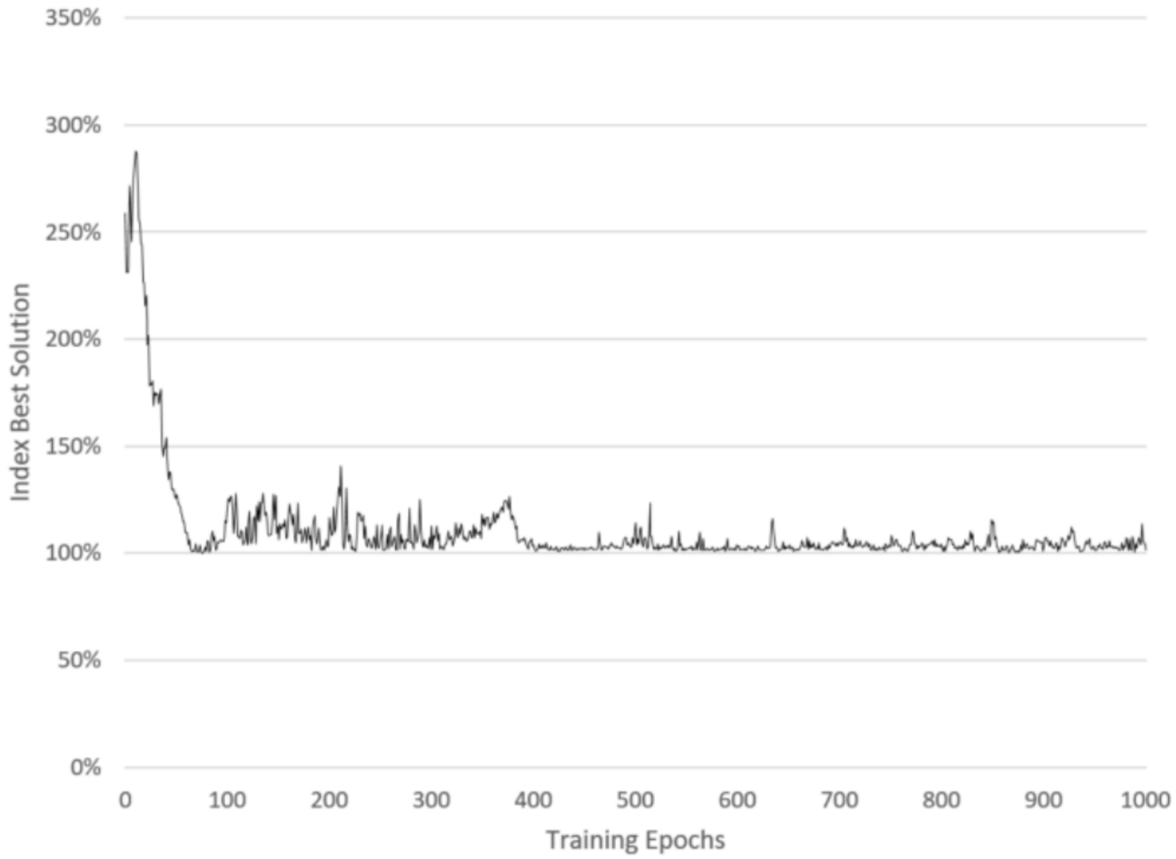
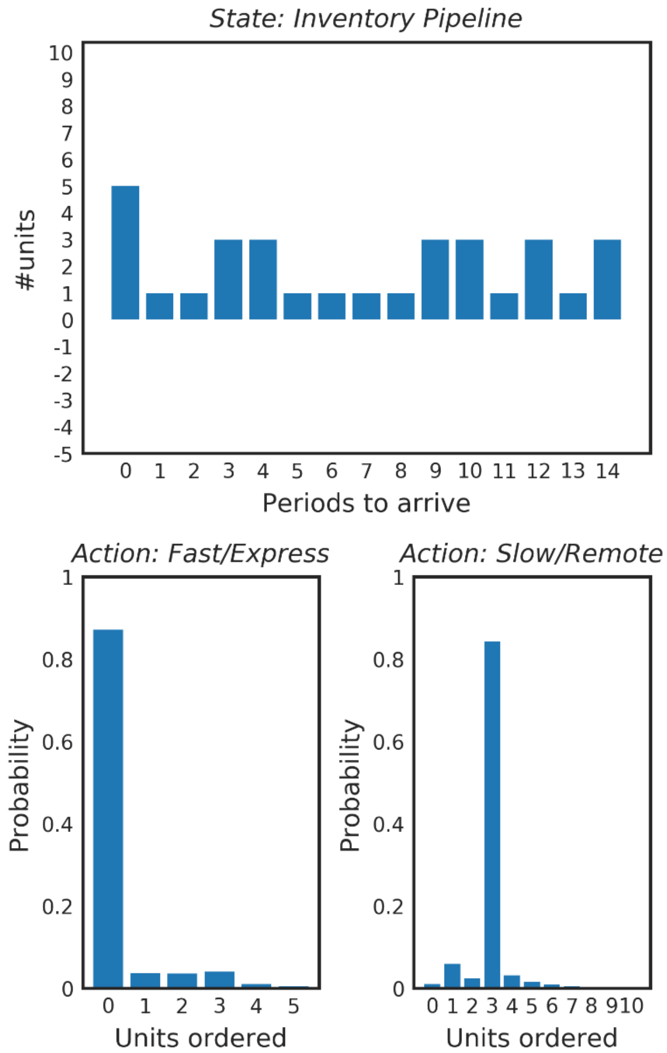
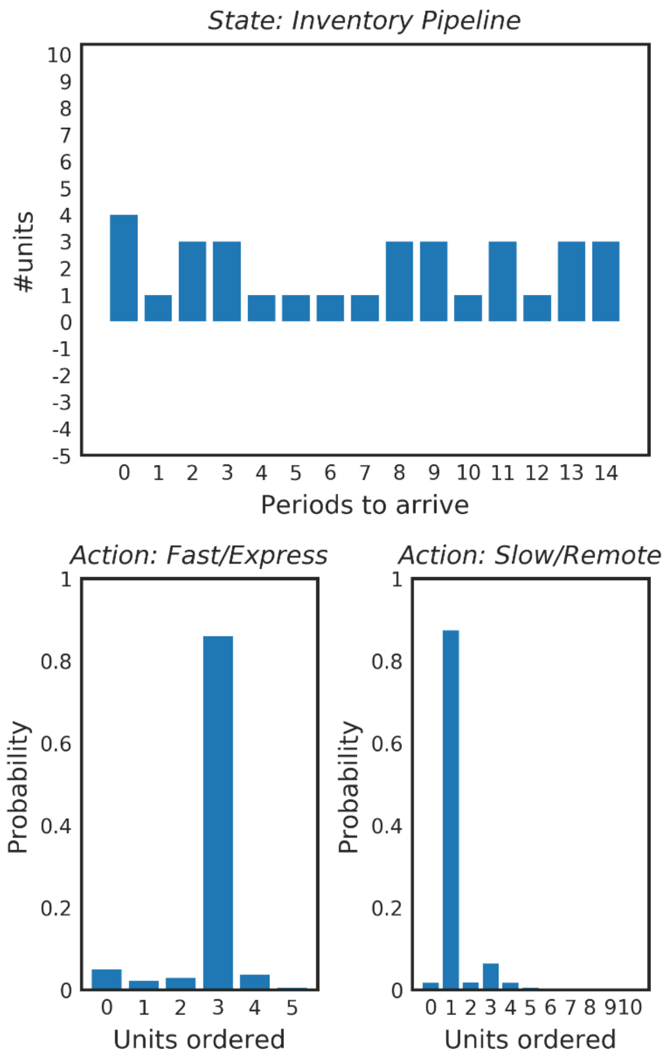


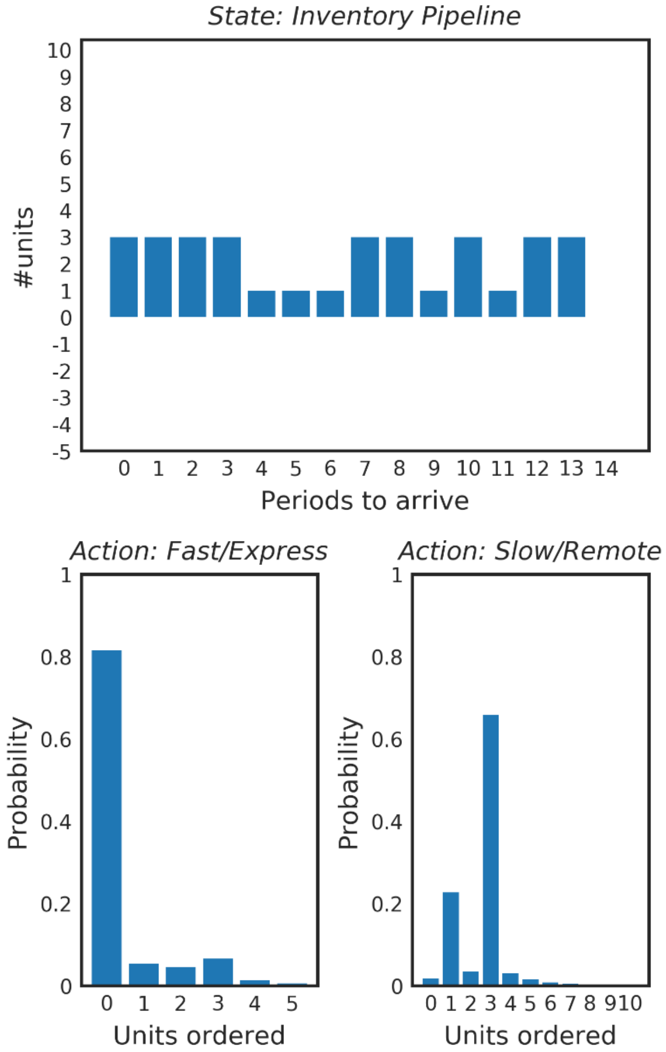
Figure 1: Cost performance during training

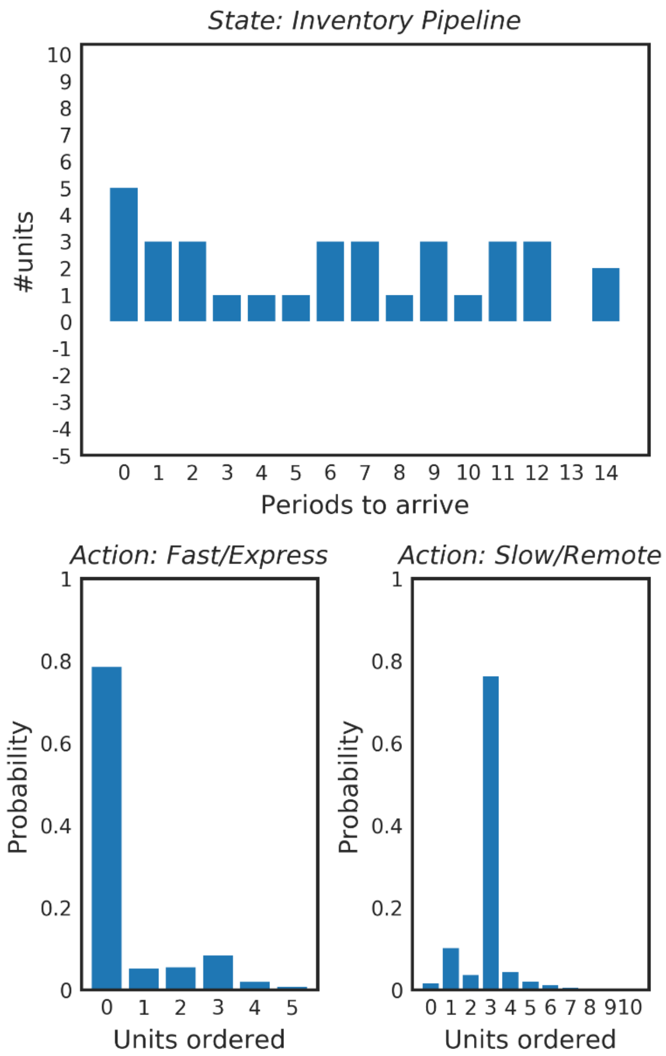
KU LEUVEN


State: Inventory Pipeline
#units
10
9
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-5
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
Periods to arrive
Action: Fast/Express
Probability
1
0.8
0.6
0.4
0.2
0
0 1 2 3 4 5
Units ordered
Action: Slow/Remote
Probability
1
0.8
0.6
0.4
0.2
0
0 1 2 3 4 5 6 7 8 9 10
Units ordered


State: Inventory Pipeline
#units
Periods to arrive
Action: Fast/Express
Action: Slow/Remote
Probability
Units ordered

KU LEUVEN


State: Inventory Pipeline
#units
Periods to arrive
Action: Fast/Express
Action: Slow/Remote
Probability
Units ordered


State: Inventory Pipeline
#units
Periods to arrive
Action: Fast/Express
Probability
Units ordered
Action: Slow/Remote
Probability
Units ordered

KU LEUVEN

Results

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

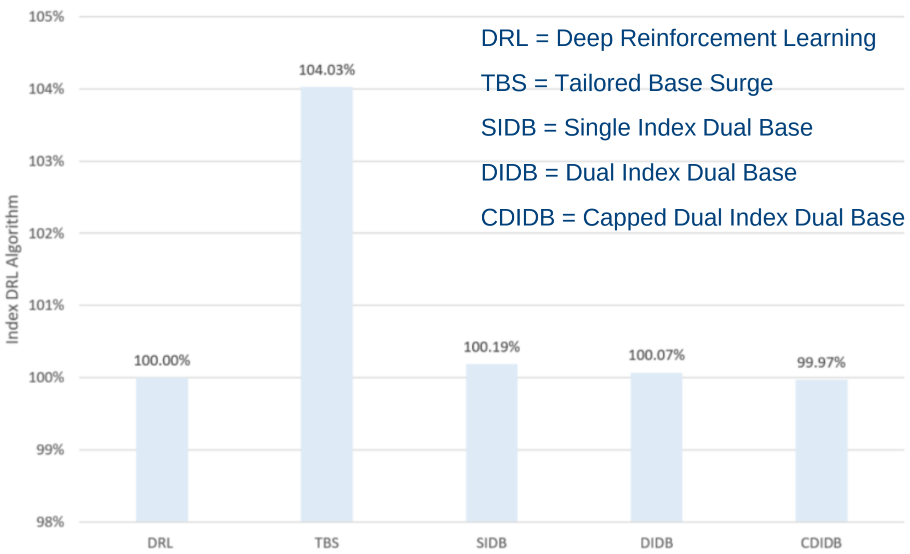


Figure 2: Benchmark with state-of-the-art dual sourcing policies

- 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)

KU LEUVEN

Backslides

Methodology

Smart replenishment algorithm – Deep Q-learning

Algorithm 1 Deep Reinforcement Learning (DRL) algorithm

Initialize replay memory D to capacity ϕ
Initialize Q-network with random weights θ
Initialize Target Network $\hat{Q}$ with weights $\theta^{-} = \theta$
Choose Initial State s_1
for t $= 1, T$ **do**
    for n=1, N **do**

$$a_t = \begin{cases} \text{random action}, & \text{with probability } (1-\epsilon) \\ \text{argmin}_{a \in A}\{Q(s_t, a; \theta)\}, & \text{else} \end{cases}$$

        Simulate using a_t and observe reward r_t and next state s_{t+1}
        Store (s_t, a_t, r_t, s_{t+1}) in replay memory D
    end for
    Sample random minibatches of size K from D
    for j= $1, K$ **do**

$$y_j = \begin{cases} r_j + \gamma Q(s, \text{argmin}_{a' \in A} Q(s_{j+1}, a'; \theta^{-}), \theta), & \text{if } (s, a) \text{ in sample } j \\ Q(s_j, a; \theta), & \text{else} \end{cases}$$

    end for
    Minimize loss $= \sum_{j \in K}(y_j - Q(s_j, a_j; \theta))^2$ using Adam optimizer
    Every z steps, update Target network $\hat{Q} = Q$
end for

KU LEUVEN

Methodology

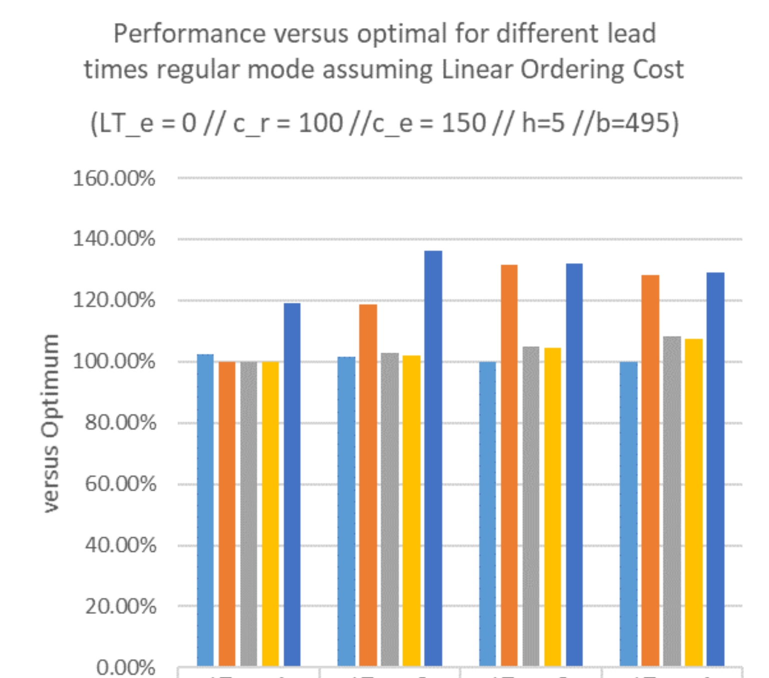
Smart replenishment algorithm – A3C algorithm

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

```
// Assume global shared parameter vectors θ and θ_v and global shared counter T = 0
// Assume thread-specific parameter vectors θ' and θ'_v
Initialize thread step counter t ← 1
repeat
    Reset gradients: dθ ← 0 and dθ_v ← 0.
    Synchronize thread-specific parameters θ' = θ and θ'_v = θ_v
    t_start = t
    Get state s_t
    repeat
        Perform a_t according to policy π(a_t|s_t; θ')
        Receive reward r_t and new state s_{t+1}
        t ← t + 1
        T ← T + 1
    until terminal s_t or t − t_start == t_max
    R = { 0             for terminal s_t
        { V(s_t, θ'_v)  for non-terminal s_t // Bootstrap from last state
    for i ∈ {t − 1, . . . , t_start} do
        R ← r_i + γR
        Accumulate gradients wrt θ': dθ ← dθ + ∇_θ' log π(a_i|s_i; θ')(R − V(s_i; θ'_v))
        Accumulate gradients wrt θ'_v: dθ_v ← dθ_v + ∂(R − V(s_i; θ'_v))^2 / ∂θ'_v
    end for
    Perform asynchronous update of θ using dθ and of θ_v using dθ_v.
until T > T_max
```

Results

Within 2% of optimal solution in a simple setting

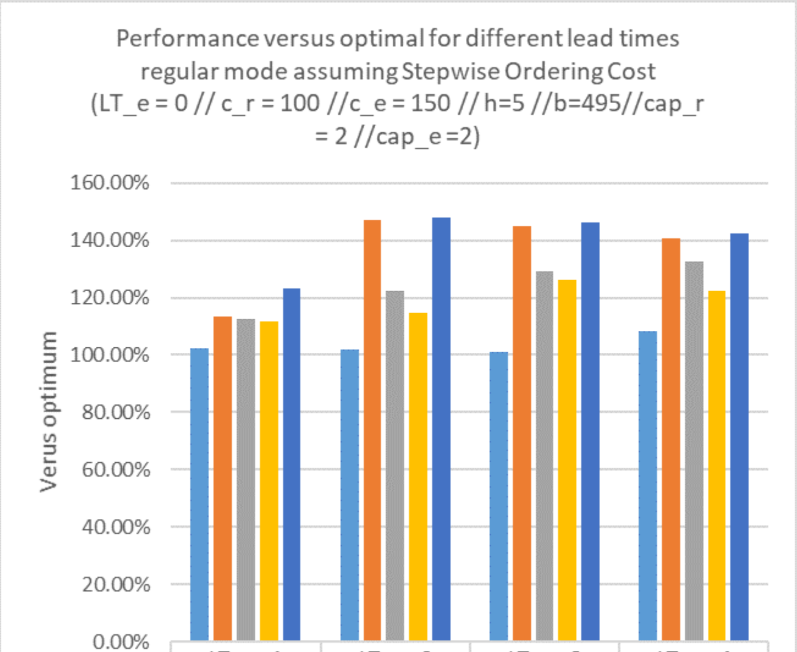


	LT_r = 1	LT_r = 2	LT_r = 3	LT_r = 4
A3C	102.34%	101.74%	100.06%	100.01%
SIDB	100.00%	118.71%	131.60%	128.35%
DIDB	100.00%	102.80%	105.06%	108.18%
CDIDB	100.00%	101.84%	104.61%	107.35%
TBS	119.06%	136.07%	131.86%	128.94%

KU LEUVEN

Results

Maintaining performance in a more complex setting



	LT_r = 1	LT_r = 2	LT_r = 3	LT_r = 4
A3C	102.49%	101.76%	101.03%	108.16%
SIDB	113.64%	147.14%	145.03%	140.52%
DIDB	112.73%	122.49%	129.38%	132.49%
CDIDB	111.91%	114.88%	126.23%	122.54%
TBS	123.41%	148.17%	146.13%	142.28%

A3C SIDB DIDB CDIDB TBS

KU LEUVEN