Model Based Reinforcement Learning for Simplified Storage Assignment Optimization

Fengshi Niu*, Zenan Wang *
Department of Economics, UC Berkeley

December 17, 2019

Abstract

This paper applies model-based deep reinforcement learning to solve a simplified storage assignment problem. First, we train an LSTM network order predictor from a long historical order sequence and use it for state transformation and reward variance reduction. Second, we run an approximate value iteration until convergence. Our algorithm is specifically designed to address the tradeoff between travel-time efficiency and the reposition costs. Our experiments evaluate this algorithm in a variety of simulated environments with varying number of products and different stochastic order processes. In all cases, our algorithm significantly reduces overall storage costs compared to random assignment heuristic. The performance gap between our algorithm and the oracle tabular value iteration with access to latent order probability is shown to be small. The running time of the algorithm scales up linearly with respect to the number of products up to 1000, despite the factorial growth of the size of the state space.

1 Introduction

Storage assignment - *the process of putting products into storage locations before they can be picked to fulfill customer orders* - highly affect the speed and cost of order picking and warehouse capacity. Order picking - *the process of retrieving products from storage in response to a specific customer request* - is the most costly and time-consuming activity in a typical warehouse with automated system (Goetschalckx and Ashayeri, 1989; Drury, 1988; Tompkins et al., 2003; De Koster et al., 2007; Roodbergen and Vis 2008; Chiang et al., 2010).

Design and control of warehouse order picking involve a large set of decisions, composite objectives, and constraints. Typical engineering practice divides this large problem into modularized subproblems including layout design, storage location assignment problem (SLAP), pickup routing, zoning, and batching, et al.. Our focus here is the storage location assignment problem. It consists

*These authors contributed equally.

of determining the most efficient assignment of products to locations in order to minimize the total handling efforts. (Silvia, et al., 2019)

The storage assignment is proved difficult to solve optimally, because of infeasibly large combinatorial search space, complex dynamic stochastic bulk arrivals and order requests, and its interaction with order picking policies. In practice, heuristic control algorithms justified by both extensive simulation studies and highly specialized analytical examples are solutions currently in use (Hausman 1976, Tappia, et al., 2019).

Deep reinforcement learning has a demonstrated track record in solving control problems with complex inputs and large search space, as shown in the success of Alpha Go (Silver, et al., 2016), end-to-end training with video input (Levine, et al., 2016), and vast applications in the past few years. Within this umbrella of algorithms, using learned models of the environment gives the system the ability to predict the future and could substantially improve the sample efficiency (Kaiser, et al., 2019).

Our work trains a LSTM (long short-term memory) network as an order predictor and then apply value iteration with a FFNN (feedforward network) function approximation to solve a simplified version of the storage assignment problem in a simulated environment. In particular, our setup highlights the tradeoff between travel-time efficiency and reposition cost and our algorithm solves this tradeoff almost optimally.

In our empirical evaluation, our approach is indistinguishable to optimal tabular solution in a simple environment with a small number of products and stationary order process in terms of both travel time per order request and storage space adjustment cost. Under the environment with a large number of products and non-stationary order process where an optimal tabular solution is not available, we demonstrate the success of our algorithm by showing a significant increase of average reward as the number of iteration increases.

2 Related Work

The storage assignment problem has been widely studied since Hausman et al. 1976. Heuristic algorithms fall into the following four categories: dedicated, random, closest open location, class-based (Koster et al. 2007). *Dedicated storage* assigns each product to a fixed location. A version of it called full-turnover storage policy assigns products with higher demand to locations closer to the I/O points. *Random storage* assigns incoming product to all empty space with equal probability. *Closest open location storage* assigns an incoming product to the empty location closest to the I/O point. Random storage and closest open location storage have high space utilization but are significantly inefficient in terms of travel time. While, dedicated methods with full-turnover, being highly travel-time efficient, requires a large amount of reposition because of constant changes in demand frequencies and product assortments. State of the art heuristic solution is *class-based* *storage* (Hsieh and Tsai 2001; Jane and Laih 2005; Manzini 2006). It utilizes demand forecasting to divide products into classes and locations into zones, then assign each class to a zone. Within a zone, it applies a random assignment. This method is a hybrid of dedicated methods at the class level and random method within the class. The intuition is it partially retains the travel-time efficiency of the dedicated method and at the same time reduces the problem of a high frequency of reposition and low space utilization.

Our observation in the context of current literature is the *tradeoff between travel-time efficiency in the relatively short run and the reposition cost in the relatively long run* haven't yet been solved optimally by existing methods. The class-based storage is simple and robust but is only suboptimal in addressing this tradeoff. Our algorithm basing on precise demand forecasting is superior and close to optimal in this direction. The performance of this algorithm in other dimensions and its robustness in an adversarial situation is not captured in our study.

Several success of applying DRL in stochastic control already exists in the literature. Srinivasa, et al. 2018 at Samsung and research group at Deepmind apply DRL to solve the cooling control problem of data center and achieves significant improvements compared to the traditional method. Wu, et al. 2016 and Kazmi, et at. 2018 successfully apply DRL in traffic control problems and hot water system control problems.

3 A Simplified Storage Assignment Problem

We set the problem up as a partially observed Markov decision problem (POMDP) defined as a tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$ of states, actions, transition, rewards, and discount. We then discuss the discrepancy between this setup to the real world.

POMDP

There are M different kind of products and M grids with a 2D layout $M = n_1 \times n_2$. At time t, product i is stored in grid $map_t(i)$ and different products are in different grids. Travel time of a grid k, $travel(k)$, equals the distance of k to the I/O point. The reposition cost for exchanging product in grids k and l, $e(k, l)$, equals the distance between these two grids. Notice that the travel time and reposition cost are assumed to be product independent and are functions of grids only. The only action is reposition, which together with the current storage map map_t determines the storage map in the next step map_{t+1}. This action a_{it} is restricted to exchanging the storage location of at most two products at each period.

Here is the timeline within period t. The agent (storage system) will

1. Receive new orders of products $o_{it}, i = 1, \ldots, M$ generated from $o_{it} \sim \text{Ber}(p_{it})$
2. Deliver products in the new orders and incur total travel time costs $c_{travel,t} = \sum_{i=1}^{M} o_{it} \cdot travel(map_t(i))$

3. Reposition products by taking action a_t and incur reposition costs $c_{reposition,t} = e(a_t)$. Generate new storage map map_{t+1} from map_t and a_t.

In the background, order probability p_{it} follows a (linear combination of) markov process w.r.t. its own history $(p_{it-1}, p_{it-2}, \ldots, p_{it-k})$ for some fixed k. In the simulation, we experiment with *static order* where $p_{it} = p_i$ is time invariant and *dynamic order* where p_{it} has a periodic seasonal pattern. The order probabilities are *unobserved* and their stochastic nature is assumed to be *unknown* to the agent.

Table 1 summarizes relevant variables in the environment. Figure 1 shows the transition diagram.

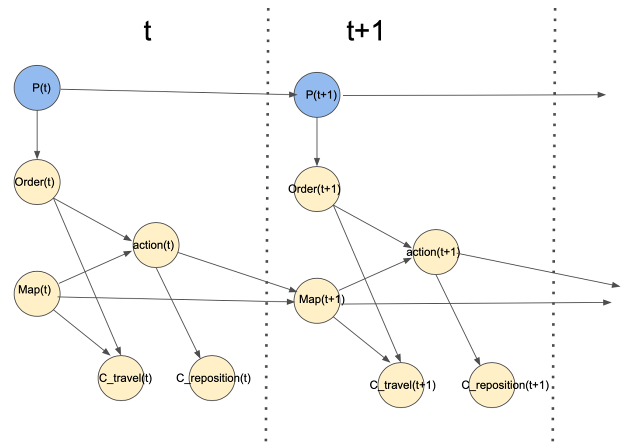


Figure 1: Transition Diagram

Class	Variable	Notation	Range
State	Order probability	p_{it}	$[0,1]$
	Order	o_{it}	$\{0,1\}$
	Storage map	map_t	Permutation$([M])$
Action	Reposition	a_{it}	$\{(k,l), 1 \leq k < l \leq M\} \cup \{None\}$
Reward	Total travel time	$c_{travel,t}$	$\mathbb{R}_+$
	Reposition cost	$c_{reposition,t}$	$\mathbb{R}_+$

Table 1: Overview of Variables

The optimization problem is

$$\min_{\pi} \mathbb{E}_{order}\left[\sum_{t=0}^{\infty} \gamma^t \left(c_{travel,t} + c_{reposition,t}\right) \middle| \pi\right]$$

Discussion of Assumptions

The biggest assumption in our setup is the product refilling cost is zero. This assumption is substantive. We decide to work under this assumption because we have no expertise in modeling the stochastic product arrival and refilling process and we want to see what can be done in this simpler setting. We also discuss this in the limitation and future work section at the end. The second assumption is the pickup routing policy is fixed and summarized by the travel time function $travel(\cdot)$. This assumption is minor. These two assumptions enables us to focus on the travel time vs reposition tradeoff as clean as possible at the cost of sweeping the refilling and routing problems under the rug.

4 Algorithm: Approximate Value Iteration with Order Predictor

4.1 Use Order Predictor for State Transformation and Reward Variance Reduction

A crucial component in the design of our storage algorithm is the utilization of demand predictor. Notice the order process and the whole environment is Markov w.r.t. the long history up to k previous periods where k is unknown and moreover the conditioning variable p is not observed. To properly exploit the Markov nature of the order process, we keep a long order history, e.g. 1000 periods, in the states. The states are then $(map_t, o_t, \ldots, o_{t-1000})$. However, this state cannot be directly used for value iteration because of its high dimensionality. In this context, *the demand predictor serves two purposes both as a filter for state transformation and as a tool for reward variance reduction.* To handle the high-dimensionality problem, we use the demand predictor to generated an estimated order probability sequence (p-sequence) $(\hat{p}_t, \ldots, \hat{p}_{t-1000})$ and then truncate it to a much shorter sequence $\overrightarrow{p}_t \equiv (\hat{p}_t, \ldots, \hat{p}_{t-L})$ where $L << 1000$. The value iteration acts only on the transformed states $(map_t, \hat{p}_t, \ldots, \hat{p}_{t-L})$ with manageably low dimension. The idea is estimated probability contains much more information about the historical states than the noisy order itself; while, the truncation is a simple way to speed up the algorithm at the cost of information loss. To reduce the noise of the reward, we use the predicted probability to calculate expected travel time cost $\hat{c}_{travel,t} = \sum_{i=1}^{M} \hat{p}_{it} \cdot travel(map_t(i))$ instead of using that based on the noisy realized order $c_{travel,t} = \sum_{i=1}^{M} o_{it} \cdot travel(map_t(i))$.

Since the order predictor plays multiple important roles in our architecture, the success of our approach will crucially depend on its performance. The prediction model we use a LSTM network with 1000 lookback.[1] Its performance under various specification of order processes are evaluated in the next section.

[1]Architectures like attention and transformer could potentially have better performance.

4.2 Algorithm

Algorithm: Approximate Value Iteration with Order Predictor

```
                             0.  Initialize model environment

Train Order Predictor        1.  Sample a long order sequence $\mathcal{H} = (o_1, o_2, \dots, o_H)$
                             2.  Train_supervised($\mathcal{H}$, LSTM_order_predictor)
                             3.  Generate a long p-sequence $\mathcal{P} = (\hat{p}_{1001}, \hat{p}_H)$ where
                                     $\hat{p}_t$ = LSTM_order_predictor$(o_{t-1}, \dots, o_{t-1000})$

Approximate Value Iteration  4.  Initialize value function approximator with parameter $\theta_0$
                             5.  for iteration $i = 1, 2, \dots, I$ do
                             6.      Sample storage maps $(map_1, \dots, map_n)$ from Permutation($[M]$)
                             7.      Sample $(\overrightarrow{p}_1, \dots, \overrightarrow{p}_n)$ by random slicing subsequences of $\mathcal{P}$ with length $L+1$
                             8.      for each $j = 1, 2, \dots, n$ do
                             9.          Bellman backup for acting state $s_j = (map_j, \overrightarrow{p}_j$[:-1])
                                             by $\bar{V}(s_j) \leftarrow \min_a \mathbb{E}[c_{reposition}(a) + \hat{c}_{travel}(map_j, a) + \gamma \hat{V}_\theta(s'_j)]$
                                             where $s'_j = (map'_j, \overrightarrow{p}_j$[1:]), $map'_j$ is a function of $map_j$ and $a$
                            10.          Update $\theta$ using $(s_j, \bar{V}(s_j))$ by minimizing $(V_\theta(s_j) - \bar{V}(s_j))^2$
```

The **simple max policy** is directly based on the trained value function with

$$\pi_\theta(s_j) = \arg\min_{a \in \mathcal{A}} \mathbb{E}[c_{reposition}(a) + \hat{c}_{travel}(map_j, a) + \gamma V_\theta(s'_j)].$$

5 Evaluation and Results

We aim to answer the following questions in our experiments:

1. Under environments with static order processes, is our algorithm able to find an efficient sequence of actions leading to the optimal placement that saves travel time in the long term?

2. How does our algorithm perform under environments with a dynamic order process?

3. Can this algorithm scale up to handle a large number of products?

Enviroment Variables	**Setting 1**	**Setting 2**
Number of Products	10	10
Layout	2×5	2×5
γ	0.99	0.99
Order Process	Static	Semiannual
True Order Probability	[0.1, 0.19, 0.28, 0.37, 0.46, 0.54, 0.63, 0.72, 0.81, 0.9]	Season 1: [0.1, 0.19, 0.28, 0.37, 0.46, 0.54, 0.63, 0.72, 0.81, 0.9] Season 2: [0.9, 0.81, 0.72, 0.63, 0.54, 0.46, 0.37, 0.28, 0.19, 0.1] Season length: 500 periods

Table 2: Evaluation Settings

To answer 1, we compare our algorithm against the optimal tabular solution in a simple environment with 10 products and a static order process, referred as *setting 1* in Table 2. The size of 10 is chosen because the tabular solution does not scale well and can not handle more than 10 products on our laptops. To answer 2, we first train our algorithms under a similar 10-product environment but with a dynamic order process, the *setting 2* in Table 2, and then evaluate them under the *setting 1* to make comparison. To answer 3, we evaluate our algorithm in various environments with a larger number of products and document the time spent on training and policy execution. All our code is available on Github [2].

5.1 Results under a Static Order Process

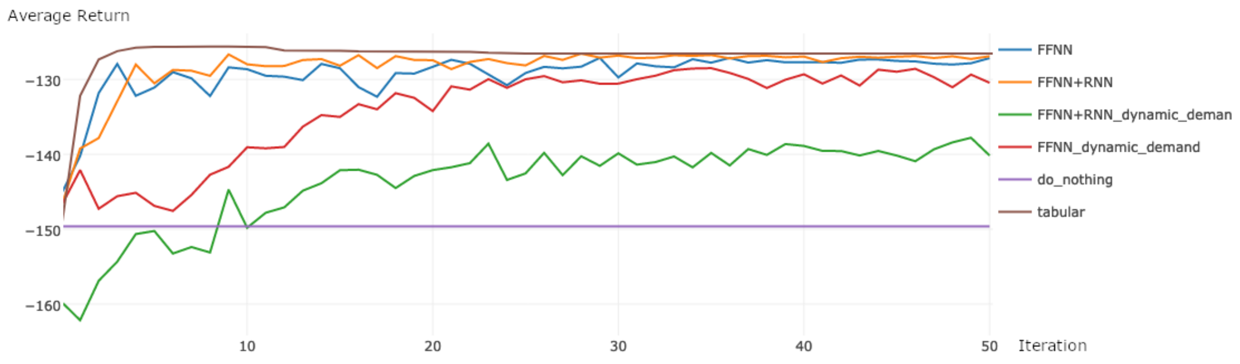


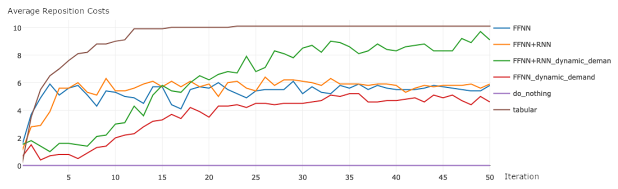
Figure 2: Total Return

Figure 2 shows the learning curve of our algorithm against the tabular iteration algorithm. The FFNN algorithm with access to the true static order probability (blue line) achieves high returns

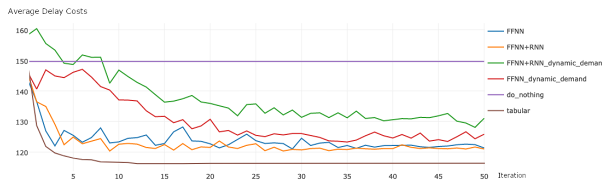
[2]`https://github.com/znwang25/SmartStorage`

quickly and its returns seem to converge to the optimal value produced by the tabular solution. Revoking the access to true order probability does not change the result. The orange line shows that our proposed FFNN+LSTM algorithm can also successfully achieve a near-optimal return value.

Interestingly, despite the similar values in total returns to the optimal returns by, the FFNN+LSTM algorithm uses a strategy that is less aggressive in reposition than that of the tabular solution. It incurs less reposition costs but more travel-time costs as shown in Figure 3. One possible explanation is the value function approximation is a regularized version of and is smoother than the true value function. Contrast of states whose true value are very close to each other are shrunk toward zero under the function approximation. The corresponding policy is therefore less active.



(a) Total Reposition Costs



(b) Total Travel-time Costs

Figure 3: Costs Breakdown

5.2 Results under a Dynamic Order Process

In the dynamic order process setup, the order probabilities change with a semiannual pattern as specified in *setting 2* in Table 2. The solid lines in Figure 4 shows how order probabilities change over time.

The red and green lines in Figure 2 show the total returns of the FFNN+LSTM algorithms trained in a dynamic order environment. The FFNN with access to the true order probability (red line) again performs very well and achieves a return close to optimal. It converges slightly slower than its counterpart trained under the static order process environment.

The FFNN+LSTM algorithm trained under dynamic order process has a suboptimal performance, as indicated by the green line in Figure 2. The performance gap is unavoidable in principle, because it doesn't have access to the true order probability. Even though the LSTM network performs reasonably well in predicting underlying order probabilities as shown in Figure 4, it needs to take several time steps to collect orders generated from probabilities under the new season right after the season change point in order to recognize the sharp change in order probability. During these several time steps, the predicted probabilities, the transformed states, and value at these states are all very imprecise, which generate several periods of wasteful repositions, which don't reduce travel-time costs. This is shown by the two green lines in Figure 3. This performance gap would be much smaller if the probability dynamic were more continuous.

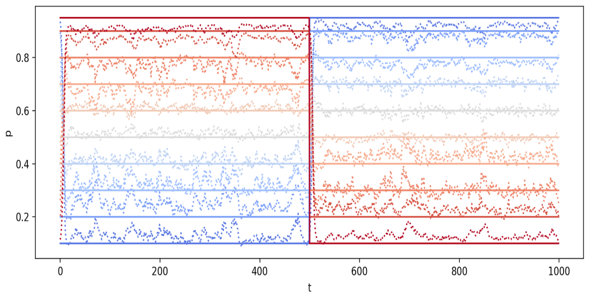


Figure 4: LSTM performance

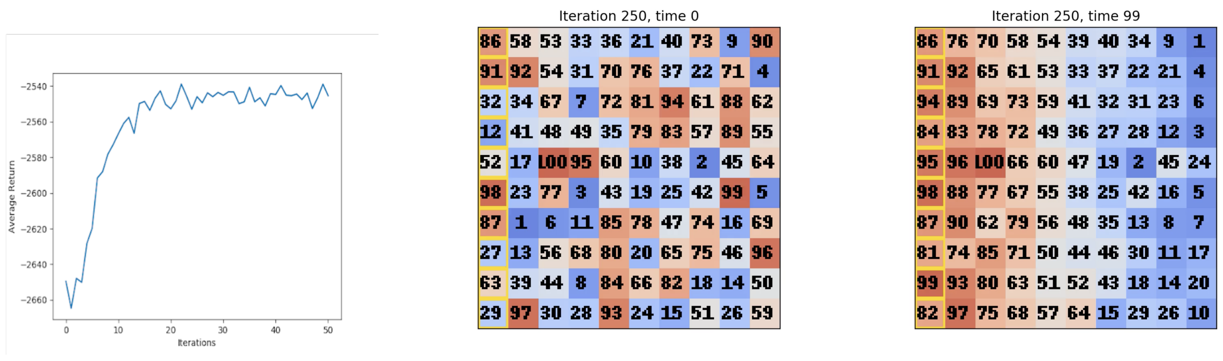
Number of Products	Number of Permutations	Training Time (sec/iteration)	Rollout Time (sec/10step)
4	24	0.07	0.45
10	3.6×10^{6}	0.20	0.53
25	1.5×10^{25}	1.12	0.90
50	3.0×10^{64}	1.10	0.83
100	9.3×10^{157}	1.54	1.12
200	7.8×10^{374}	2.35	1.38
1000	4.0×10^{2567}	11.9	4.24

Table 3: Scalability of FFNN + LSTM

5.3 Scalability

The FFNN+LSTM algorithm scales well with respect to increase in the number of products up to 1000. Figure 5 demonstrates a rollout of policy out of this algorithm in a 100-product 10×10 environment with a static order process. (5b) and (5c) shows the storage maps at the beginning and at the end of 100 periods. In this simple static setting, a good policy should move products with higher order probability, indicated by warmer color, to locations with lower travel time or equivalently locations closer to the I/O points highlighted by the yellow boundaries. The policy out of the algorithm acts highly in line with this intuition.

Table 3 presents the training and policy execution time for our algorithm under environments with a static order process and various number of products. It shows that both training time and policy execution time scale almost linearly with the number of products [3]. Those numbers are very encouraging especially considering how fast the number of permutations explodes.



(a) Learning Curve (b) Before (c) After 100 timesteps

Figure 5: Performance in a 100-Product 10×10 Environment with Static Order Processes

[3]These number could be overly optimistic. We keep the number of sampled actions and states fixed and in particular independent of the number of products.

6 Conclusion and Future Work

In this work, we apply model-based deep reinforcement learning to solve a simplified storage assignment problem. Our proposed algorithm is able to learn the latent state behind the order process and find actions to minimize both total time-travel costs and total reposition costs. We show that our algorithm performs well in a variety of simulated environments with varying number of products (≤ 1000) and different stochastic order processes. In cases where optimal tabular iteration solution is available, the performance gap between our algorithm and the optimal solution is small. In cases where a tabular iteration solution is infeasible, our algorithm can still reach convergence within a short amount of time despite the huge state space and produce an impressive policy. In all cases, our algorithm significantly reduces overall storage costs compared to random assignments.

Our work demonstrates deep reinforcement learning as a promising tool to solve storage assignment problem. However, our algorithm is not directly applicable to any real storage system, where travel time efficiency and reposition costs are only two out of many objectives to be optimized. In particular, future work should take refilling process, the costs therein, and the pick-up routing problem into account and potentially solve a joint optimization problem. We strongly believe that deep reinforcement learning with stochastic predictors could still be a powerful tool in solving these larger scale problems.

References

1. Chan, Felix TS, and Hing Kai Chan. “Improving the productivity of order picking of a manual-pick and multi-level rack distribution warehouse through the implementation of class-based storage.” Expert systems with applications 38, no. 3 (2011): 2686-2700.

2. Chiang, David Ming-Huang, Chia-Ping Lin, and Mu-Chen Chen. “The adaptive approach for storage assignment by mining data of warehouse management system for distribution centres.” Enterprise Information Systems 5, no. 2 (2011): 219-234.

3. De Koster, René, Tho Le-Duc, and Kees Jan Roodbergen. “Design and control of warehouse order picking: A literature review.” European journal of operational research 182, no. 2 (2007): 481-501.

4. Drury, Jolyon. “Towards more efficient order picking.” IMM monograph 1 (1988).

5. Goetschalckx, Marc, and Jalal Ashayeri. Characterization and design of order picking systems. Material Handling Research Center, Georgia Institute of Technology, 1989.

6. Hausman, Warren H., Leroy B. Schwarz, and Stephen C. Graves. “Optimal storage assignment in automatic warehousing systems.” Management science 22, no. 6 (1976): 629-638.

7. Hsieh, S., and K-C. Tsai. “A BOM oriented class-based storage assignment in an automated storage/retrieval system.” The international journal of advanced manufacturing technology 17, no. 9 (2001): 683-691.

8. Jane, Chin-Chia, and Yih-Wenn Laih. “A clustering algorithm for item assignment in a synchronized zone order picking system.” European Journal of Operational Research 166, no. 2 (2005): 489-496.

9. Kaiser, Lukasz, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H. Campbell, Konrad Czechowski, Dumitru Erhan et al. “Model-based reinforcement learning for atari.” arXiv preprint arXiv:1903.00374 (2019).

10. Kazmi, Hussain, Fahad Mehmood, Stefan Lodeweyckx, and Johan Driesen. “Gigawatt-hour scale savings on a budget of zero: Deep reinforcement learning based optimal control of hot water systems.” Energy 144 (2018): 159-168.

11. Levine, Sergey, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. “End-to-end training of deep visuomotor policies.” The Journal of Machine Learning Research 17, no. 1 (2016): 1334-1373.

12. Manzini, Riccardo. “Correlated storage assignment in an order picking system.” International Journal of Industrial Engineering-Theory Applications and Practice 13, no. 4 (2006): 384-394.

13. Roodbergen, Kees Jan, and Iris FA Vis. “A survey of literature on automated storage and retrieval systems.” European journal of operational research 194, no. 2 (2009): 343-362.

14. Silva, Allyson, Leandro C. Coelho, Maryam Darvish, and Jacques Renaud. “Integrating Storage Location and Order Picking Problems in Warehouse Planning.” (2019).

15. Srinivasa, Sunil, Girish Kathalagiri, Julu Subramanyam Varanasi, Luis Carlos Quintela, Mohamad Charafeddine, and Chi-Hoon Lee. “On Optimizing Operational Efficiency in Storage Systems via Deep Reinforcement Learning.” In Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pp. 238-253. Springer, Cham, 2018.

16. Silver, David, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser et al. “Mastering the game of Go with deep neural networks and tree search.” nature 529, no. 7587 (2016): 484.

17. Tappia, Elena, Debjit Roy, Marco Melacini, and René De Koster. “Integrated storage-order picking systems: Technology, performance models, and design insights.” European Journal of Operational Research 274, no. 3 (2019): 947-965.

18. Tompkins, James A., John A. White, Yavuz A. Bozer, and Jose Mario Azana Tanchoco. Facilities planning. John Wiley & Sons, 2010.

19. Wu, Cathy, Kanaad Parvate, Nishant Kheterpal, Leah Dickstein, Ankur Mehta, Eugene Vinitsky, and Alexandre M. Bayen. “Framework for control and deep reinforcement learning in traffic.” In 2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC), pp. 1-8. IEEE, 2017.