Dynamic Scheduling of the Dual Stocker System Using Reinforcement Learning

Seol Hwang, Sang Pyo Hong, Young Jae Jang

Dynamic scheduling of the dual stocker system using reinforcement learning

Seol Hwang[0000−0002−1962−8610], Sang Pyo Hong[0000−0002−7946−8540], and Young Jae Jang[0000−0002−2342−1444]

Department of the Industrial and Systems Engineering, Korea Advanced Institute of Science and Technology, Daejeon 305-701, Republic of Korea
yjang@kaist.ac.kr

Abstract. The stocker system is the most widely used material handling system in LCD and flat panel fabrication facilities (FABs). The stocker mainly consists of one or two cranes moving along a single track to transport lots, or cassettes, containing 10 to 30 thin glass substrates between processing machines. Because the stocker system is the primary material handling system in the FABs, its performance directly affects the overall performance. In this study, we investigate the scheduling of a dual stocker system operating with two cranes simultaneously on a single track and propose a learning-based scheduling algorithm for the system. We report some of the results of our long-term efforts to dynamically optimize the dual-crane stocker. We fisrt show the modeling and algorithm to minimize the make-span of the jobs. We incorporate the model to dynamically allocate jobs. In particular, we use a reinforcement learning method in the scheduling algorithm. The model is validated in an extensive simulation study based on actual data.

Keywords: Reinforcement learning · Scheduling · AMHS

1 Introduction

The thin film transistor liquid crystal display (TFT-LCD) is one of the most widely used LCD due to its high image quality. An automated material handling system (AMHS) is commonly exploited in TFT-LCD manufacturing to deal with complicated movements of cassette, a container holding LCD glasses. The stocker system is a main component of AMHS in TFT-LCD fabrication facilities (FABs), which consists of one or two cranes on a single rail and shelves used as buffer spaces. The processing machines are attached to the stocker system(Fig. 1). The crane travels along with a rail and moves up and down to pick up and set down a cassette. Recently, two cranes are usually used in most TFT-LCD FABs in order to increase throughput of the stocker system, which is referred to as the *dual crane stocker*.

Since two cranes travels on a single rail simultaneously in the dual crane stocker, a strategy to avoid collision within cranes should be designed. Most dual crane stockers in industry adopt first-in-first-out (FIFO) rule based on an

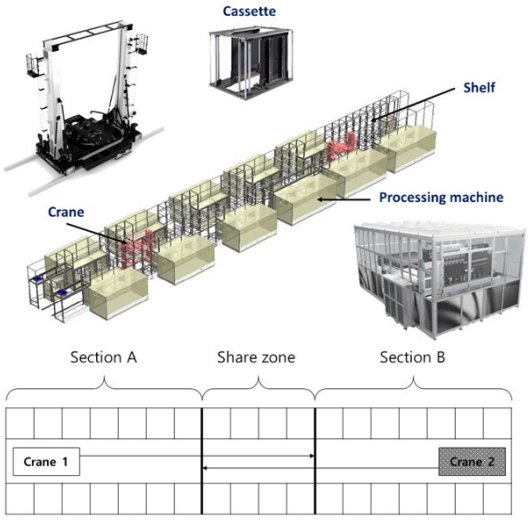

Fig. 1. The stocker system

area segregation. As shown in Fig. 1, the stocker is divided three sections and each crane travel only up to share zone and only one of cranes can occupy share zone to avoid a collision. Although this operation rule is simple and effective in avoiding collisions, it arouses two kinds of inefficiency. First, when a crane use the share zone, another crane should wait until the occupying cranes work is done, which is called blocking. Second, an additional movement is necessary to hand over a cassette to another crane when the cassette could not be transported directly by a crane. For example, if a cassette should move from section A to section B, crane 1 transports the cassette from section A to an empty buffer in the share zone. Then crane 2 picks up the cassette on the share zone and transports it to the destination in section B. These inefficiency could be handled by effectively determining the size or location of the share zone. However, a proper scheduling rule without the share zone would be the best way to maximize the stocker systems capacity as long as the collision is avoided. There were a few studies about the stocker system in TFT-LCD. Jang et al. [4] proposed an analytical model for measuring the performance of the single crane stocker system. Several studies dealt with the stocker for layout design with TFT-LCD plant and suggested mathematical model including determination of the share zone [3, 7]. Reinforcement learning is a method that learn a policy based on an action value from experience. Once the agent is trained through many experiences, it can select an appropriate action in a short time. Most researches in this field focused on choosing appropriate heuristics [1, 8] which is different with our approach in that dual crane stocker scheduling problem do not have typical scheduling rules. Meanwhile, Bradtke [2] showed that DP-based reinforcement learning converged to the optimal policy with non-linear function approximators. Dietterich et al. [9]

applied TD(λ) to train a neural network to learn heuristic evaluation function in job-shop scheduling problem. These results presented a possibility of DP-based scheduling with neural network could work in other domain. Recently, Google Deep Mind team showed that Deep-Q Network, combination of reinforcement learning and deep neural network [5], could learn policies successfully in Atari games and surpass the performance of all previous algorithms. In this paper, we formulate a scheduling problem for dual crane stocker applying dynamic programming(DP). DP approach could find an optimal solution for a static case, but curse of dimensionality makes the computation time to increase exponentially. Thus we used neural network to approximate non-linear relationship within features. In particular, we suggested a novel image based input shape to represent the state accurately and adopted convolution layer to take into account interactions between movements.

2 Modeling Assumptions

The assumptions for the stocker model are as follows.

- Both a transportation from a processing machine to a shelf and from a shelf to a processing machine were considered as a same kind of a job.
- Initial jobs were generated randomly with pre-determined origin location and destination.
- Though vertical and horizontal movement of the crane occur simultaneously, the horizontal movement was solely considered. Since it takes much more time to move horizontally than vertically, vertical movement of a crane is negligible.

We used a term bay as a unit of a location in the stocker system. There are two sets of load ports, the spaces for cassettes, in upper and lower side of Fig. 1 (lower panel). However, it is not necessary to distinguish transports into load ports at upper or lower since the total travel time is same as long as the bay number is same. The difficulty of dual crane stocker problem comes from collision avoidance movement. The cranes should maintain safety distance to avoid collision. Even if same crane selects same job, the processing time could be quite different because of an interference within two cranes. The final objective of the dual stocker scheduling problem is finding an optimal scheduling rule on dynamic environment that jobs are generated continuously. In order to achieve that goal, we first focused on scheduling for the given jobs in static case whose measurement is makespan.

3 Dynamic Programming

The problem can be formulated using dynamic programming (DP). Some notations are defined as follows. $\mathbf{B} \in [1, b_e]$ is the set of bay numbers in the stocker system where b_e is a last bay number. Each job is numbered according

to the entering sequence and represented with origin location and destination as $j_i = (o_i, d_i)$ where $(o_i, d_i) \in \mathbf{B}$. The state of the stocker we made consisted of crane work state and waiting job list, $\mathbf{S} =< \mathbf{C}, \mathbf{J} >$

- $\mathbf{C} = \{c_1, c_2 \,|\, c_1, c_2 \in \{0, j_1, ..., j_N\}\}$ represent which job is transported by the cranes at the state. If a crane is not assigned by any job, 0 is used.
- $\mathbf{J} = \{j_{i \in \mathbf{W}} \,|\, \mathbf{W}$ is a set of waiting jobs$\}$ is waiting job list. At the initial state, $\mathbf{J} = \{j_1, ..., j_N\}$. If a job is selected by decision making agent, the job is deleted from the set $\mathbf{J}$. Thus, $\mathbf{J} = \emptyset$ at the terminal state.

As we mentioned, the state of the stocker in DP is a decision point of decision making agent. Since the decision making agent choose jobs from waiting job list, action set $\mathbf{A}$ is same with set $\mathbf{J}$. Though the state is defined, more information is necessary to express a specific status of the system. It is referred to as attribute and they consisted of locations, destinations, operation status and remaining time of the cranes.

- $\mathbf{L} = \{l_1, l_2 \,|\, l_1, l_2 \in \mathbf{B}\}$ represents locations of the cranes
- $\mathbf{D} = \{d_1, d_2 \,|\, d_1, d_2 \in \mathbf{B}\}$ represents destinations of the cranes
- $\mathbf{O}^s = \{o_1^s, o_2^s \,|\, o_1^s, o_2^s \in \{Retrieve, Pickup, Deliver, Setdown, Idle\}\}$
- $\mathbf{T}^r = \{t_1^r, t_2^r \,|\, t_1^r, t_2^r \in \mathbb{R}^+\}$ represents the remaining time in current status of the cranes.

After defining the state, we developed the optimization problem to minimize the makespan of the given job. This makespan minimization is structurally equivalent to the shortest path problem and it can be solved using a DP approach. Since the states were defined as the decision points and we considered the optimal makespan for given jobs, the model is categorized as a deterministic finite state, discrete time DP model. The notation for the model was as follows:

- s_t: state of the stocker system at time t;
- a_t: decision for the next job at time t;
- $\mathbf{P}_t(s_t, a_t)$: interval time between s_t to s_{t+1} when action a_t is taken at s_t; and
- $\mathbf{V}_t(s_t)$: total time from time t at the current state s_t. (cost-to-go function)

Then the recursive equation is

$$\mathbf{V}_t(s_t) = \min_{a_t} \{\mathbf{P}_t(s_t, a_t) + \mathbf{V}_{t+1}(s_{t+1})\} \tag{1}$$

To solve this equation, it is necessary to expand overall state space in order to compute $P_t(s_t, a_t)$ for all t. Then calculate $V_0(s_0)$ by backwards induction using (1). Although DP approach find optimal solution, the computation time increases exponentially. When the number of initial job is up to five, it took less than 10 second. However, it took about 200 seconds in six initial jobs and more than 3 hours in seven, which is not applicable practically.

4 Deep Q Network

Deep Q network (DQN) is a reinforcement learning algorithm adopting neural network as an approximate function. We introduce DQN briefly and explain a novel shape we made in order to increase a performance of the algorithm. The key points of DQN are experience replay and separate network, which are the techniques for training neural networks without diverging. Experience replay is a methodology that storing agents experiences at each time step to a replay memory and exploiting mini-batch data drawn at random from the pool of stored samples for Q-learning. This technique makes the training data not to have a bias due to a correlation between adjacent data. Next, DQN copies a target network $\hat{Q}$ from learning network Q in order to generate Q-learning targets(y_j). The algorithm become more stable by preventing the network Q from oscillation or divergence. The full algorithm of DQN is presented in Algorithm 1. We implemented this algorithm using a tensorflow.

Algorithm 1 Deep Q network

Initialize replay memory D
Initialize action value function Q with weight θ using Xavier initialization
Initialize target action value $\hat{Q}$ with weight $\theta^- = 0$
for *episode* $= 1, M$ **do**
    Initialize s_0
    for $t = 1, T$ **do**
        Choose $a_t \sim \epsilon$-greedy policy
        Take action a_t, observe r_t, s_{t+1}
        Store transition (s_t, a_t, r_t, s_{t+1}) in D
        Sample random minibatch of transitions (s_j, a_j, r_j, s_{j+1}) from D
        $y_j = \begin{cases} r_j & \text{if } j+1 = T \\ r_j + \gamma \min_{a'} \hat{Q}(s_{t+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
        Perform a gradient descent with respect to the network parameter θ
        Every C steps update $\hat{Q} = Q$
    end for
end for

The performance of deep Q-learning or neural network depends on various factors. Though the system environment is identical, the result can be considerably different in accordance with the shape of input data. We suggest two kinds of shape, simple integer type and vector expression of moving trace. Though an integer shape contains all information of a state, it could be not efficient for a neural network to predict an action value due to its excessive simplicity. The action value of a state is determined by cranes movement and remained jobs origin-destination(OD) pair. From this attribute, visualizing the traces of movement and od pair might be a new approach for input shape. For instance, with a case in Fig. 2, first moving trace of crane 1 is delivering movement from bay 4 to bay 8. Likewise, if the picked action has been job 4, first moving trace of crane 2

is a retrieving movement from bay 17 to bay 13, and second is a delivering from bay 13 to bay 9. Moving direction is expressed with color that red is toward right side and blue is the opposite. The remained jobs trace also could be drawn in the same way (Fig. 2). The drawn traces are converted to a number matrix. In the first column, crane moving priority index is added to describe which crane would move first. The other values are converted from the traces using integer 1 and -1, which represent moving to right and left respectively. Therefore, the size of the matrix is $(N + 4) \times (b_e + 1)$.

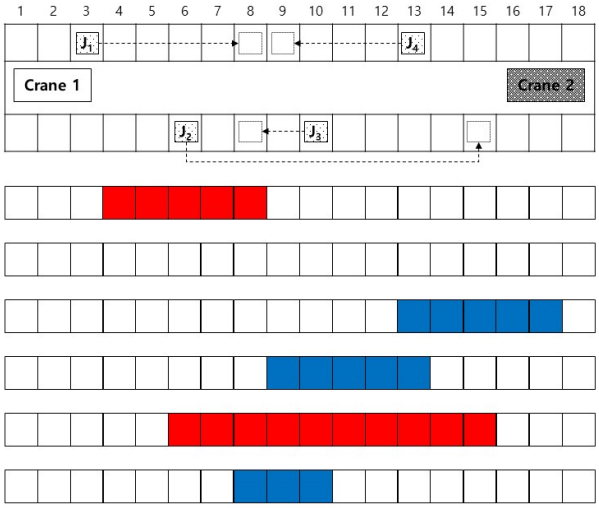

Fig. 2. Trace-shape of movements

5 Experiment Result

5.1 Static Case

The scheduling performance comparison of dynamic programming (DP), deep Q-network (DQN) and first-in-first-out rule (FIFO) was conducted using 100 randomly generated test set along with various initial job numbers from 3 to 10. The performance of learning based algorithms was evaluated after learning with 50,000 episode and hyper-parameters of each method was tuned respectively. We constructed neural network with two layers and 8 filters for convolution, two hidden layers and 32 nodes for fully connected network. The learning rate was 0.001 and the exploration rate was 0.2 for DQN.

Fig. 3(a) is summarized graph comparing the makespan of three methods. Since DP guarantee optimal solution, any other methods cannot be better than the DP solution. However, since DP needs enormous computation time as the initial job number increases, results for more than 7 initial jobs were not computed.

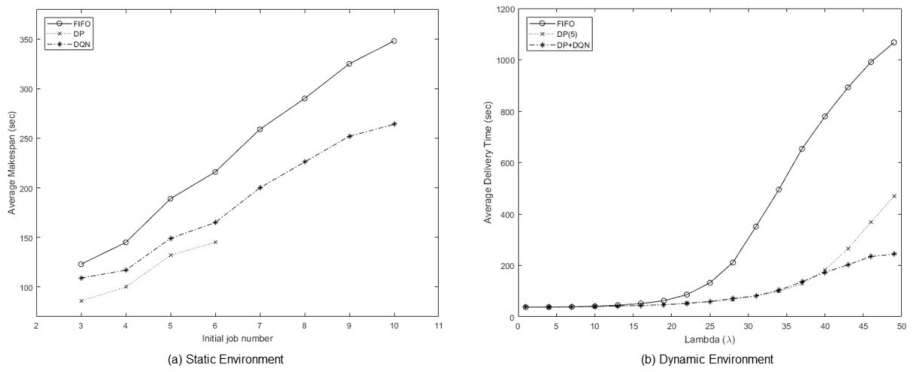

Fig. 3. Performance comparison graph

5.2 Dynamic Case

The dynamic case considers the dynamic job allocation to the stocker crane when jobs are continuously created. With the insight gained from the static case, we applied the the DQN model to the dynamic case with some modification of the model. The primary goal of the dynamic model is to reduce the average waiting time of the lots with an optimal job allocation policy. The dynamic model is created through the following three steps.

First, it create a neural network from the approach of the rolling horizon approach extended from the static case. That is, the results of the dynamic allocation from the rolling horizon approach have been used to train the neural network. Second, the Q-value based priority rule is developed on top of the neural network. The jobs in the queue are prioritized based on the Q-value of each job — the lower the Q-value, the higher the priority the job is given. This approach significantly reduces the waiting time of the jobs. The third step is that the neural network with the Q-value based priority rules is modified with rolling out heuristics approach which is proposed in [6]. We call this approach developed from the three step, *DP+DQN* model.

Fig. 3(b) shows the preliminary result of the dynamic case. The y-axis and x-axis represent the average delivery time of the lots and the load factor, λ. The constant value is multiplied to the base from-to value. The higher the load factor, the higher from-to deliver requirement. The proposed DP+DQN is compared to the FIFO rule and the static DP with rolling horizon case. The result shows that the DP+DQN outperforms compared to the rest of the approach. It also shows a stable performance even the load factor reached to the 50, where the rest of the cases tend to increase the waiting time significantly.

6 Conclusion

In this paper, we suggested dynamic programming model for dual crane stocker scheduling problem and developed the model into the deep Q network (DQN). We obtained an optimal solution considering interference within two cranes in

case of relatively small number of initial jobs using DP approach. We proposed the trace shape of input data and applied convolution layer to improve the performance of neural network. The model is first developed for a static case where the model seeks to optimal crane move sequence to minimize the makespan of the jobs in the queue. With the insight gained from the static case, we further developed the dynamic case to find an optimal policy to allocate the jobs in a dynamic environment. The preliminary case shows that the proposed DQN+DP outperforms the FIFO and DP with rolling horizon.

The work is still far from complete. An extensive numerical cases studies are still needed to verify the approach. Also, we still need to provide logical and numerical verification that how the proposed approach outperforms the existing approach. Also we still need to investigate in which case the effectiveness of the proposed case is weakened. We proposed this to the future study.

References

1. Aydin, M.E., Öztemel, E.: Dynamic job-shop scheduling using reinforcement learning agents. Robotics and Autonomous Systems **33**(2-3), 169–178 (2000)
2. Bradtke, S.J.: Reinforcement learning applied to linear quadratic regulation. In: Advances in neural information processing systems. pp. 295–302 (1993)
3. Ho, Y.C., Su, T.S.: The machine layout within a tft-lcd bay with a multiple-stacker crane in-line stocker. International Journal of Production Research **50**(18), 5152–5172 (2012)
4. Jang, Y.J., Choi, G.H., Kim, S.I.: Modeling and analysis of stocker system in semiconductor and lcd fab. In: Semiconductor Manufacturing, 2005. ISSM 2005, IEEE International Symposium on. pp. 273–276. IEEE (2005)
5. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529 (2015)
6. Powell, W.B.: Approximate Dynamic Programming: Solving the curses of dimensionality, vol. 703 (2007)
7. Su, T.S., Hwang, M.H.: Efficient machine layout design method with a fuzzy set theory within a bay in a tft-lcd plant. Procedia Manufacturing **11**, 1863–1870 (2017)
8. Wang, Y.C., Usher, J.M.: Application of reinforcement learning for agent-based production scheduling. Engineering Applications of Artificial Intelligence **18**(1), 73–82 (2005)
9. Zhang, W., Dietterich, T.G.: A reinforcement learning approach to job-shop scheduling. In: IJCAI. vol. 95, pp. 1114–1120. Citeseer (1995)