MuZero 강화학습을 이용한 항만 컨테이너 재정돈 계획

최원균[1], 이기주[1], 채준재[1†]

[1]한국항공대학교 항공교통물류학과

Container Pre-Marshalling with MuZero Reinforcement Learning Algorithm

Wongyun Choi[1], Keyju Lee[1], Junjae Chae[1†]

[1]School of Air Transport, Transportation and Logistics, Korea Aerospace University

This study provides a new solution approach for container pre-marshalling problems using a reinforcement learning method, the MuZero Algorithm. We have developed a customized pre-marshalling environment for an agent to be trained. To facilitate the training, we have devised i) some action masking methods, and ii) a reward function where lower bound for the number of rehandling is considered. Action masking methods and heuristically found lower bound were proven to be helpful in the learning process. Experiments with different sizes (from the minimum number of 8 and the maximum number of 12 containers) were carried out, within a stacking space of six rows and six tiers. Experimental results show that the MuZero algorithm implemented with our strategy is capable of training to solve the pre-marshalling problems for some small sized problems.

Keywords: Pre-Marshalling, Container Reshuffling, Reinforcement Learning, MuZero

1. 서론

항만은 해운과 내륙운송을 연결하는 공통접속영역으로서 물류, 생산, 생활, 정보생산 및 국제교역기능과 배후지의 경제발전을 위한 기지로서의 역할을 수행하는 종합공간을 말한다(Yeo, 2002). Liu and Lee(2019)에 의하면 2017년 중량 기준으로 해운은 우리나라의 전체 무역량의 99%이상을 차지하고 있으며, 이 중 60% 이상이 컨테이너에 의해 운송되고 있다. 우리나라의 수출입 컨테이너화물은 일부 항만에 집중되어 있는데, 한국 3대 항만(부산, 인천, 광양)은 컨테이너 화물총량의 90% 이상을 차지하고 있는 것으로 나타났다. 특히, 인천지방해양수산청의 통계자료에 의하면 2018년 인천항만의 컨테이너 물동량은 3백만 TEU를 상회하여 전체 컨테이너 물동량 2위를 기록했다.

논문접수일 : 2020.10.12.
심사완료일 : 2021.12.20.
게재확정일 : 2021.12.21.
† Corresponding Author: Goyang, Gyeonggi-do 10540; jchae@kau.ac.kr

항만에서의 장치장 재정돈 또는 재정렬(pre-marshalling, remarshalling) 은 장치장 크레인의 유휴 시간을 활용하여 최대의 효율로 적하 작업을 할 수 있도록 장치장의 컨테이너들을 미리 재배치하는 작업을 의미한다(Park et al., 2008). 즉, 장치되어 있는 컨테이너의 회수계획을 실행할 때 컨테이너 재취급이 불필요하도록 미리 컨테이너가 쌓여 있는 형태를 재정돈하는 작업을 말한다(Carlo et al., 2014). 아래 <Figure 1>에서는 하나의 예시로서 컨테이너 장치장 재정돈의 전과 후를 도식적으로 나타낸다. 재정돈의 결과는 각 컨테이너 열에 쌓여 있는 컨테이너의 번호가 오름차순 정렬된 상태로 만드는 것이므로 재정돈의 결과 형태는 매우 다양하다. 그 중에서 가장 적은 재취급 횟수로 도달할 수 있는 결과 형태를 최적이라고 할 수 있다.

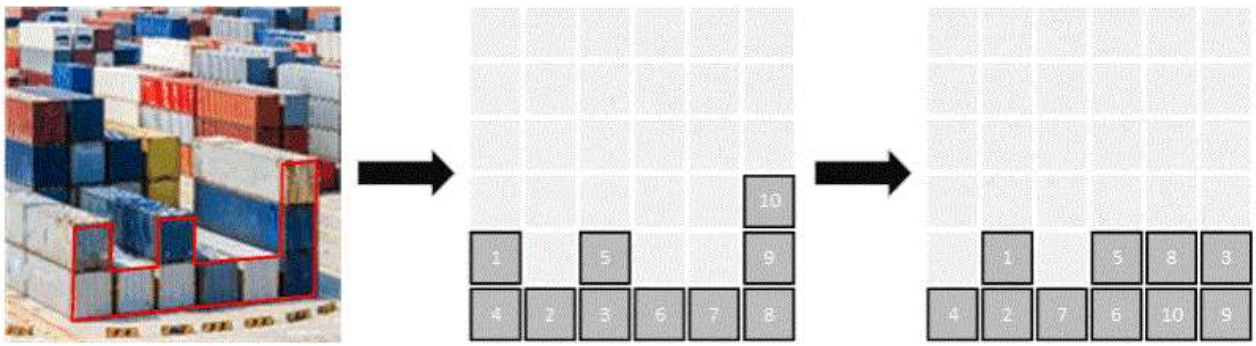


Figure 1. An example of container pre-marshalling problem (Numbers indicate retrieval sequence)

컨테이너 재정돈의 실질적인 효과는 다수의 논문에 의해 정리된 바 있다. Park et al.(2008)에 의하면 재정돈의 결과 적하 컨테이너의 대기시간이 70% 이상 감소하고 장치장 크레인의 이동시간이 37.75% 감소했다. 종합하여 총 적하작업시간을 19.94% 감소시킬 수 있었다. Park at el.(2012)의 연구에서도 비슷한 실험결과를 확인할 수 있다. 재정돈의 효과를 적하 작업 지연과 외부트럭의 대기시간으로 평가하였는데, 실험을 통해 적하 작업 지연은 약 23% 감소, 외부트럭 대기시간은 약 32% 감소시킬 수 있다고 결론지었다.

최근 항만의 작업생산성 향상 방법론에 다양한 도전이 이루어지고 있는데, 이 중 하나가 인공지능(AI)을 이용한 방법이다. 본 논문에서는 최근 4차 산업혁명으로 주목받는 인공지능의 강화학습, 딥러닝 기법을 항만 분야에 적용하여 컨테이너 재정돈 계획에 적용하였다. 본 연구에서는 기존의 항만 연구분야에서 시도되지 않았던 MuZero 강화학습 방법론을 사용하였으며, 이를 통해 새로운 방법론의 적용 가능성과 효과성에 대해 제안했다.

2. 문헌연구

2.1 컨테이너 재정돈 문제 방법론

컨테이너 재정돈 문제 해결을 위한 다양한 선행연구가 존재한다. 수리모델링과 최적해를 찾는 정확한 방법(exact method), 그리고 근사해를 찾는 발견적인 기법(heuristic method)으로 내용을 구분하여 내용을 정리하였다.

수리모델은 정수계획법(integer programming)에 의한 방법(Lee and Hsu, 2007; Parreño-Torres et al.,

2019)과 제약 프로그래밍(constraint programming)에 의한 방법(Rendl and Prandtstetter, 2013)이 제시되었다. 최적해를 찾는 정확한 방법으로는 분지한계법(branch-and-bound)에 의한 방법론(Prandtstetter, 2013; Tanaka and Tierney, 2018; Tanaka and Mizuno, 2018; Tanaka et al., 2019)이 주로 연구되었다. 또한, A* 알고리즘(Hart, P. et al, 1968; Ha and Kim, 2012; Park, 2016) 및 IDA* (Iterative Deepening A*) 알고리즘(Paias et al., 2016; Tierney et al., 2017)을 이용한 방법론들이 연구된 바 있다.

근사해를 찾는 발견적 기법으로 biased random-key 유전 알고리즘(Hottung and Tierney, 2016) 및 협력적 공진화 알고리즘(Potter, M. and Jong, K., 1994; Park et al., 2009; Park at el., 2012)을 변형하는 방법이 가장 많이 사용되었다. 이외에도 이웃탐색(neighborhood search) 휴리스틱과 정수계획법을 혼합한 방식(Lee and Chao, 2009), corridor method 기반 알고리즘(Caserta and Voß, 2009), lowest priority first 휴리스틱(Expósito-Izquierdo et al., 2012), target-guided 알고리즘(Wang et al., 2015), 가능성기반(feasibility-based) 휴리스틱(Wang et al., 2017), 규칙기반(rule-based) 알고리즘(Gheith et al., 2014) 등 다양한 해법에 대한 연구가 존재한다.

Hottung et al.(2020)의 연구에서는 컨테이너 재정돈 문제에 학습기반(learning-based) 알고리즘을 활용하였다. 해당 연구에서는 심층 신경망(deep neural networks)을 사용하여 기존에 존재하는 재정돈 문제 인스턴스의 최적 솔루션과 하한선(lower bound)을 판단하는 기준을 지도학습(supervised learning)하였다. 이후 트리 검색 절차에서 다음에 탐색할 분기(branching)를 선택하는 과정 및 전지(pruning) 작업 과정에 학습 결과를 활용하였다. Hirashima(2009)의 연구에서는 컨테이너 재정돈 문제에 Q-learning을 적용하여 강화학습(reinforcement learning)하였다. 해당 연구에서는 심층 신경망을 이용하지 않았으며, 단 두 종류의 초기상태를 가정한 뒤 각각의 문제에 대해 개별적인 학습을 진행하였다. 기존의 방식과 유사하거나 더 나은 결과를 얻을 수 있었지만, 학습의 결과가 특정 초기상태에서만 유효하다는 한계가 존재한다.

심층 신경망 강화학습 방법론을 이용하며, 무작위 초기상태를 가정하여 컨테이너 재정돈을 학습시킨 기존 연구사례는 존재하지 않는다.

2.2 하노이의 탑과 강화학습

심층 신경망 강화학습 방법론으로 컨테이너 재정돈 문제를 해결할 수 있을지 가능성을 판단하기 위해 유사한 문제인 '하노이의 탑'에 대한 문헌연구를 진행하였다. 하노이의 탑 문제는 퍼즐의 일종으로서 세 개의 기둥과 다수의 상하정렬을 위한 우선순위가 존재하는 물체(원판)를 사용한다. 해당 문제의 목표는 하나의 기둥에 정렬되어있는 물체들을 다른 기둥으로 정렬된 상태로 옮기는 것이다.

하노이의 탑 문제와 컨테이너 재정돈 문제는 공통적으로 상하정렬을 위한 우선순위가 존재하는 물체가 존재한다. 두 문제 모두 한 번에 맨 위에 놓인 물체 중 하나만을 이동시킬 수 있으며, 최종적으로는 물체가 상하정렬된 상태를 만들어내는 것을 목표로 한다. 또한, 두 문제 모두 정렬된 상태를 강화하는 탐욕(greedy) 움직임만으로는 문제를 해결할 수 없으며, 정렬된 상태를 무너뜨리는 움직임을 혼합해야만 최종 목표 상태에 도달할 수 있다.

하노이의 탑 문제를 강화학습으로 해결한 경우를 다수 찾아볼 수 있었으며, 주로 Q-learning을 사용한 해결법이 공개되어 있다(Khpeek, 2017). Bang and Tijus(2018)의 연구에서는 순환신경망(recurrent neural network)을 적극 활용하여 하노이의 탑 문제를 해결하였다. 또한, Pierrot et al.(2019)의 논문에서는 기존 AlphaZero(Silver et al., 2017)의 알고리즘에 '순환'적인 특징을 가미함으로써 명명한 AlphaNPI(Neural Programmer Interpreters)를 이용해 하노이의 탑 솔루션을 제시하였다.

하노이의 탑에 대한 기존 연구를 조사한 결과, 비슷한 유형의 문제인 항만의 재정돈 문제를 심층 신경망 강화학습을 이용해서 접근할 수 있을 것으로 판단하였다.

3. 방법론

3.1 MuZero 알고리즘

강화 학습은 AlphaGo(Silver et al., 2016), AlphaZero(Silver et al., 2017) 및 그 뒤를 잇는 MuZero 알고리즘(Schrittwieser et al., 2020)의 연구 결과에서 알 수 있듯, 순차적 의사 결정 작업에서 성공적인 결과를 나타냈다. 모델 기반 강화 학습 알고리즘인 MuZero는 체스, 장기, 바둑에서 최첨단 성능을 달성한 바 있다 (Schrittwieser et al., 2020). MuZero 알고리즘은 아래와 같이 정리할 수 있다.

먼저 MuZero 알고리즘에서는 튜플(tuple) $\langle S, A, T, U, \gamma \rangle$로 표현되는 MDP(Markov Decision Process)를 정의한다. 튜플의 각 요소는 states 집합(S), actions 집합(A), state-action 쌍$(\times)$을 새로운 state에 확률(p)로서 맵핑(mapping)하는 전환 다이내믹(transition dynamics)$(T: S \times A \rightarrow p(S))$, state-action 쌍을 보상(rewards)으로 맵핑하는 보상함수(reward function)$(U: S \times A \rightarrow rewards)$, 그리고 할인율(discount rate) $(\gamma \in [0,1])$을 의미한다. 내부적으로는 추상화된 MDP$\langle \tilde{S}, A, \tilde{T}, R, \gamma \rangle$를 정의하며, $\tilde{S}$는 추상화된 상태 공간, $\tilde{T}: \tilde{S} \times A \rightarrow \tilde{S}$는 전환 다이내믹, $R: \tilde{S} \times A \rightarrow rewards$는 예측된 보상(reward prediction)을 의미한다. MuZero 알고리즘에서 목표로 하는 것은 $V(s)$ 값을 최대화시킬 수 있는 정책$(\pi: S \rightarrow p(A))$을 찾아내는 것이다. $V(s)$는 infinite-horizon 누적 return 값의 예측치를 의미하며 다음과 같이 정의된다.

$$V(s) = E_{\pi, T}\left[\sum_{t=0}^{\infty} \gamma^t \bullet u_t | s_0 = s\right]$$

위에서 정의된 MDP를 근사(approximation)하기 위해서 아래 <Figure 2>에서 같이 서로 다른 세 개의 신경망(상태 인코딩 함수 h_θ, 전환 다이내믹 함수 g_θ, 예측 네트워크 f_θ)을 정의한다. 여기서 θ는 네트워크의 공동 매개 변수 집합을 나타낸다. 상태 인코딩 함수 $h_\theta: S \rightarrow \tilde{S}$ 는 실제 MDP 관측 상태를 내부의 추상화 MDP 상태로 맵핑하는 역할을 한다. 전환 다이내믹 함수 $g_\theta: \tilde{S} \times A \rightarrow \tilde{S} \times rewards$ 는 상태 전환으로 인한 다음 상태와 보상값을 예측하는 역할을 한다. 마지막으로 예측 네트워크 $f_\theta: \tilde{S} \rightarrow p(A) \times rewards$ 는 정책 및 일부 추상화된 상태 값을 예측하는 역할을 한다. 세 개의 신경망을 통합하여 $\mu_\theta = (h_\theta, g_\theta, f_\theta)$과 같이 표현할 수 있다.

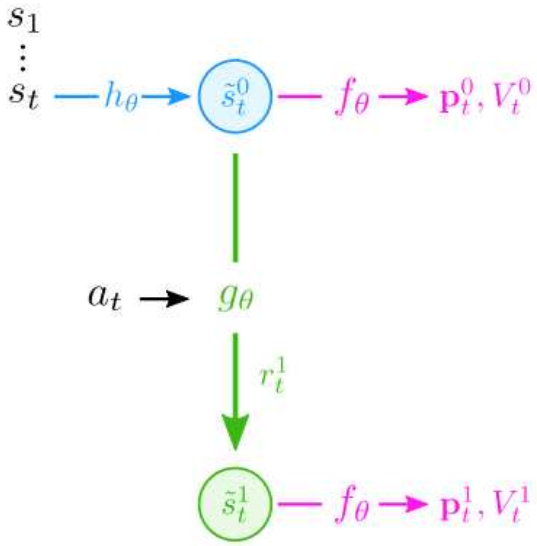


Figure 2. .Illustration of MuZero's RNN unfolding during MCTS(de Vries et al., 2021)

MuZero 알고리즘은 주어진 f, g 및 h 네트워크를 이용하여 추상 공간에서의 MCTS(Monte Carlo Tree Search) 검색을 수행한다. 먼저 h를 통해 현재 상태를 인코딩하며, 이후 MCTS의 한 종류인 하나인 PUCB 알고리즘(Rosin, 2011)을 수행하고, g와 f에 의해 상태변환과 정책 및 보상값을 예측한다. MCTS는 출력값으로서 정책 $\pi_t = \pi(s_t)$ 와 기저 노드(root node)의 예측값 $V_t = V(s_t)$을 도출한다 $(\pi_t, V_t) \sim MCTS(s_0,...,s_t|\mu_\theta)$. 이후 실제 환경(environment)에서 action $a_t \sim \pi_t$ 을 선택하여 다음 상태로 전환하며, 이후 새롭게 주어진 네트워크를 이용한 추상 공간 MCTS 검색 과정으로 반복된다. 해당 MCTS 검색에 대한 자세한 내용은 Schrittwieser et al.(2020)의 부록 B에서 확인할 수 있다.

3.2 학습을 위한 환경(Environment) 구성

강화학습을 통해 학습을 진행하기 위해서는 기본적으로 학습환경을 구성해야 한다. 강화학습에서의 환경은 행동을 학습하는 에이전트(agent)가 취한 행동의 결과를 평가하여 보상 값으로 반환하는 역할을 한다. 즉, 에이전트가 학습하고자 하는 문제 그 자체인 것이다. 환경 구성을 위한 다양한 프레임워크가 제시되어 왔으며, Mnih, et al.(2013)의 논문 등에서는 고전게임과 같이 학습을 위한 환경 구축을 위해 Gym 환경(OpenAI, 2018) 프레임워크를 사용하였다. 본 연구에도 Gym 환경을 상속한 뒤, 프레임워크에 맞추어 컨테이너 재정돈을 학습하기 위한 환경을 구축하였다. Gym 환경 프레임워크는 초기화를 위한 __init__ 함수, 문제가 종료되었을 시 환경 재구축을 위한 reset() 함수, 에이전트에 의해 결정된 행동을 취하기 위한 _take_action(action) 함수, 행동의 결과로 이어지는 상태를 도출하기 위한 _next_observation() 함수, 현재 상태를 이미지 등으로 표현하기 위한 render() 함수 등으로 구성된다.

3.2.1 Actions

본 연구에서는 source(from) 열 상단의 컨테이너를 target(to) 열 상단으로 옮기는 것을 하나의 action으로 정의한다. 명확한 설명을 위해 아래의 <Figure 3>와 같이 6열(row) 6층(tier)의 구성을 기본 형태로 가정하였다. 즉, 기본 형태에서 가능한 모든 action은 6개 source 열 상단의 컨테이너를 6개 target 열 상단으로 옮기는 것으로 총 36가지가 된다. 환경을 구성할 때, 가능한 모든 action은 action_space로서 입력되며, 그

값은 Discrete(6*6)으로 설정하였다. 각각의 action은 0부터 35까지의 정수로 표현되며, 나눗셈 및 나머지 연산으로 source 열과 target 열을 명시하는 것이 가능하다. 한 가지 경우를 예시로 들자면, action(31)의 경우 source 열은 (int)(31/6)=5, target 열은 (mod)(31, 6)=1로서, 5번 열 상단의 컨테이너를 1번 열 상단으로 옮기는 action을 의미한다.

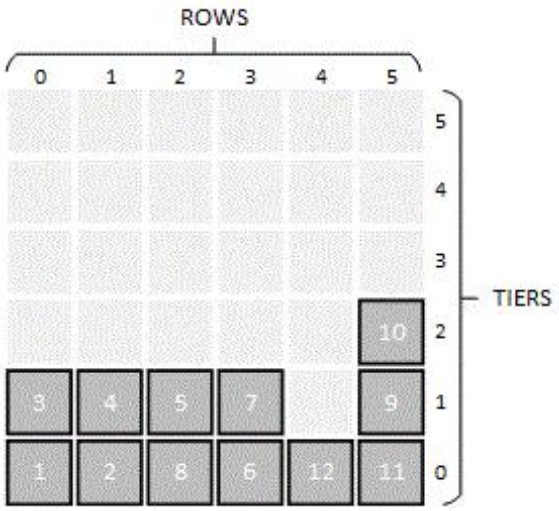


Figure 3. Basic 6-row, 6-tier configuration

모든 action을 실행 가능한 것으로 가정할 수도 있지만, 컨테이너를 옮기는 action이 물리적으로 불가능한 경우(impossible actions), 그리고 물리적으로 의미 없는 경우(redundant actions)를 불가능한 action으로 만들어 주는(masking) 것으로 학습이 효율적으로 진행될 수 있도록 유도하였다.

(1) Impossible Actions

물리적으로 불가능한 action은 크게 두 종류가 있다. 첫 번째는 비어있는 열(empty stack)에서 컨테이너를 들어 올리고자 하는 경우이다. 들어 올릴 수 있는 컨테이너가 존재하지 않으므로 불가능한 action으로 구분하여 마스킹하였다. 두 번째는 컨테이너를 만재된 열(full stack)로 옮기고자 하는 경우이다. 이미 최대 층까지 컨테이너가 쌓여 있는 열에 컨테이너를 추가적으로 얹을 수 없으므로 이 또한 불가능한 action으로 구분하여 마스킹하였다.

(2) Redundant Actions

물리적으로 낭비로 판단되는 action 또한 두 종류로 생각해볼 수 있다. 첫 번째는 row_A와 row_B가 동일한 경우이다. 들어 올린 컨테이너를 곧바로 같은 자리에 내려놓는 것은 낭비이므로 실행 불가능한 것으로 본다.

두 번째는 컨테이너를 중계(relay)하는 경우이다. 아래 <Figure 4>에서는 이러한 경우의 하나를 예시하고 있다. 만약 시점(t)에서 컨테이너(4)를 1번 열에서 5번 열로 이동시킨 뒤, 시점(t+1)에서 컨테이너(4)를 5번 열에서 3번 열로 이동시켰다면, 애초에 시점(t)에서 컨테이너(4)를 1번 열에서 3번 열로 직접 (중계하지 않고) 옮기는 편이 더 효율적이다. 컨테이너를 중계하는 것은 비단 연속적인 시점에서만 일어나는 것이 아니다. 아래의 <Figure 5>에서는 연속적이지 않은 시점(t) 와 시점(t+2) 간에도 중계가 일어날

수 있음을 보여준다.

즉, 직전 action만 참고해서는 현재 취하려는 action이 중계하는 action인지 아닌지의 여부를 정확하게 확인할 수 없으며, 중계 action 여부를 지속적으로 추적할 수 있는 환경을 구축해야만 한다. 중계 여부를 추적하기 위한 알고리즘의 pseudo-code는 아래의 <Algorithm 1>에 제시하였다. <Algorithm 1>에 의하면, 불린(boolean)으로 구성된 relay_action을 관리하여 각 action이 중계 action 인지의 여부를 매 시점 갱신해줄 수 있다. <Algorithm 1>에서는 먼저, 현재 시점의 action에 대한 source(from) 열과 target(to) 열을 나눗셈과 나머지 연산으로 계산한다. 이어서, 계산된 target 열을 다음 action에서 source 열로 가지게 되는 모든 움직임을 중계 action으로 판단하여 true 불린값을 부여한다. 또한, 현재 시점의 action으로 인해 더이상 중계 action이 아니게 된 action들에 대하여 모두 false 불린값을 부여한다.

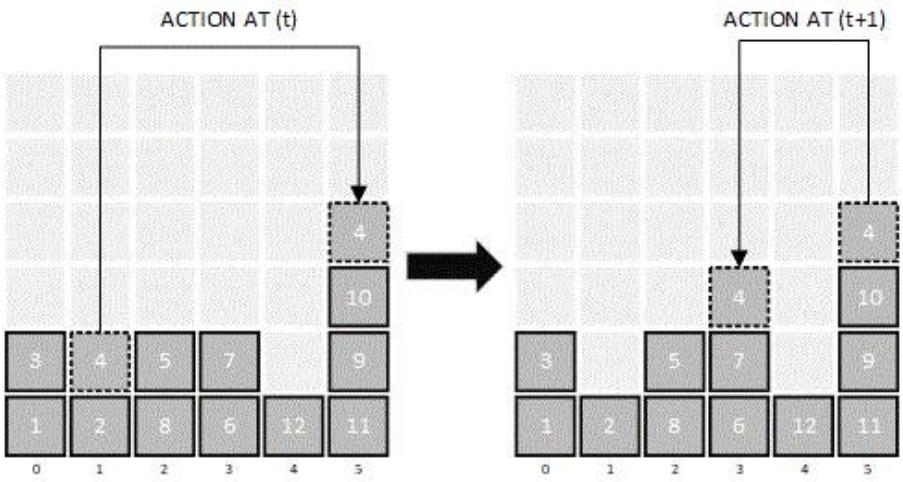


Figure 4. An example of relay actions

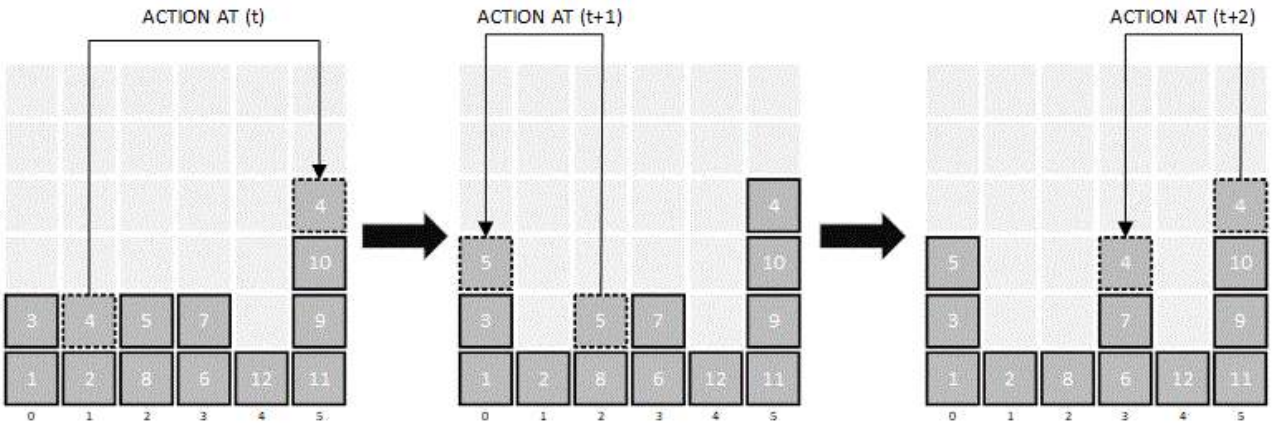


Figure 5. Another example of relay actions

Keep Track of Relay Actions

Initialize booleans of relay actions Z to the size of action space
for each time step in T
    Set index i_{from} = (a_t / NUM_ROWS) and round down
    Set index i_{to} to the remainder of (a_t / NUM_ROWS)
    for each row r in NUM_ROWS
        $Z[i_{from}$ * NUM_ROWS + $r]$ = False
        Z[NUM_ROWS * r + i_{from}] = False
        Z[NUM_ROWS * r + i_{to}] = False
    end for
    for each row r in NUM_ROWS
        $Z[i_{to}$ * NUM_ROWS + $r]$ = True
    end for
end for

Algorithm 1. Pseudo-code for tracking relay actions

3.2.2 States

Gym 환경을 구성하기 위해서는 앞서 설정한 action_space에 더하여 추가적으로 observation_space를 정의할 필요가 있다. 여기서 observation_space는 해당 환경에서 관찰(도달) 가능한 모든 상태를 반영하는 집합을 의미한다. MuZero 알고리즘에서는 추상화된 상태를 활용하므로 observation_shape을 정의한다. 본 연구에서는 observation_shape을 아래 <Figure 6>에서 확인하는 바와 같이 두 개의 6열 6층 평면으로 정의하였다. 첫 번째 평면은 우선순위 즉, 컨테이너 반출계획 순서를 나타낸다. 작은 숫자일수록 먼저 반출될 계획을 가지고 있으며, '0'은 비어있는 공간을 의미한다. 해당 표현(representation) 방식은 비어있는 공간인 '0'을 포함하여 작은 숫자일수록 상단에 위치해야 함을 유도하기 위해 사용되었다. 두 번째 평면은 컨테이너가 정돈된 상태를 표현한다. 제대로 정렬된 컨테이너는 '1'의 값을, 우선순위가 역전되어 재취급이 필요한 컨테이너는 '-1'의 값을, 그리고 나머지 빈 공간은 '0'의 값을 가진다. 결과적으로 본 연구에서는 하나의 상태를 우선순위와 정돈상태의 두 가지 형태로 표현하며, 둘을 합쳐서 상태를 표현하는 데 사용한다.

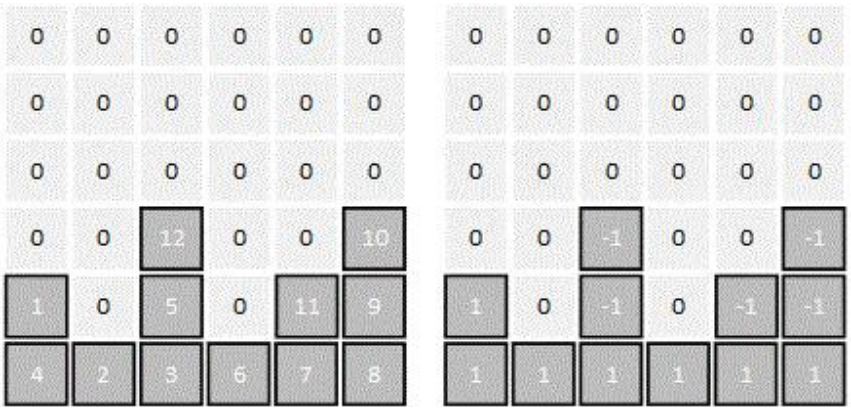


Figure 6. An example of the two 6(row)*6(tier) planes: representation of retrieval sequence(left) and re-handling requirement(right)

3.2.3 Rewards

Action을 취하여 다음 상태에 도달했을 때 적절한 보상(또는 penalty)을 부여함으로써 효율적인 학습을 유도하였다. 적절한 보상을 설정하기 위해서는 action을 취하기 전의 상태와 action을 취한 후의 상태에 대한 평가가 필요하다.

(1) Improving or Impairing Actions

Action을 취함으로써 상태가 개선되는 때도 있는가 하면 악화되는 경우 또한 존재한다. 상태가 개선되었는지, 또는 악화되었는지에 대해 판단하기 위하여 보상을 '최소 재취급 횟수 하한값의 감소량'으로 정하였다. 예를 들어, 직전 상태에서 계산된 최소 재취급 횟수의 하한값이 5회, action을 취함으로써 도달한 현재 상태에서 계산된 최소 재취급 횟수의 하한값이 4회라면, 해당 action이 재취급 횟수의 하한값을 1만큼 줄여주었으므로 '+1'의 보상을 부여한다. 반대로 재취급 횟수가 1회 늘어난 경우, '-1'의 보상을 부여한다.

Kang et al.(2004)의 연구에서는 현재 컨테이너가 쌓여 있는 상태를 통해 최소 재취급 횟수의 하한값을 도출할 수 있는 휴리스틱 알고리즘을 제시한다. 해당 연구에서는 하한값 휴리스틱 알고리즘을 이용하여 분지한계법의 하한값으로서 활용하였으나, 본 연구에서는 보상값 계산을 위해 활용하였다. 해당 휴리스틱 알고리즘은 크게 3단계에 걸쳐 진행된다.

첫 번째로, <Figure 7>에 나타난 바와 같이 먼저 출고되어야 할 (숫자가 더 낮은) 컨테이너가 뒤에 출고될 (숫자가 더 큰) 컨테이너보다 아래에 위치하는 경우를 모두 재취급 횟수로서 집계한다. 두 번째로, <Figure 7>에서 결정된 재취급 대상의 컨테이너들보다도 위에 놓여 있는 컨테이너는 재취급 시 모두 걷어내야만 하므로, 이들 모두를 추가적으로 재취급 컨테이너로서 집계한다. <Figure 8>에서는 이와 같은 조건으로 추가되는 재취급 컨테이너를 나타내고 있다. 마지막 세 번째로, <Figure 9>에 나타난 다소 복잡한 과정을 반복하며 컨테이너 재취급이 추가적으로 더 필요할지 판단한다.

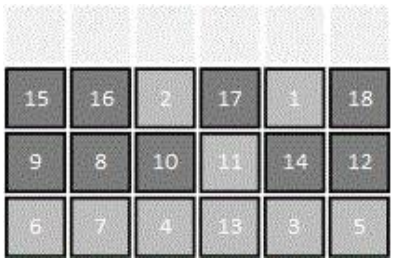


Figure 7. Re-handling lower bound: 9

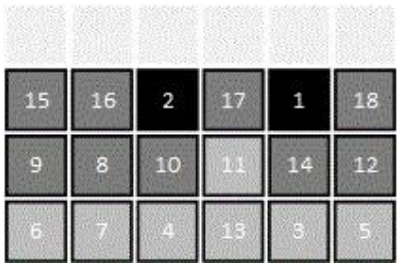


Figure 8. Re-handling lower bound: 11

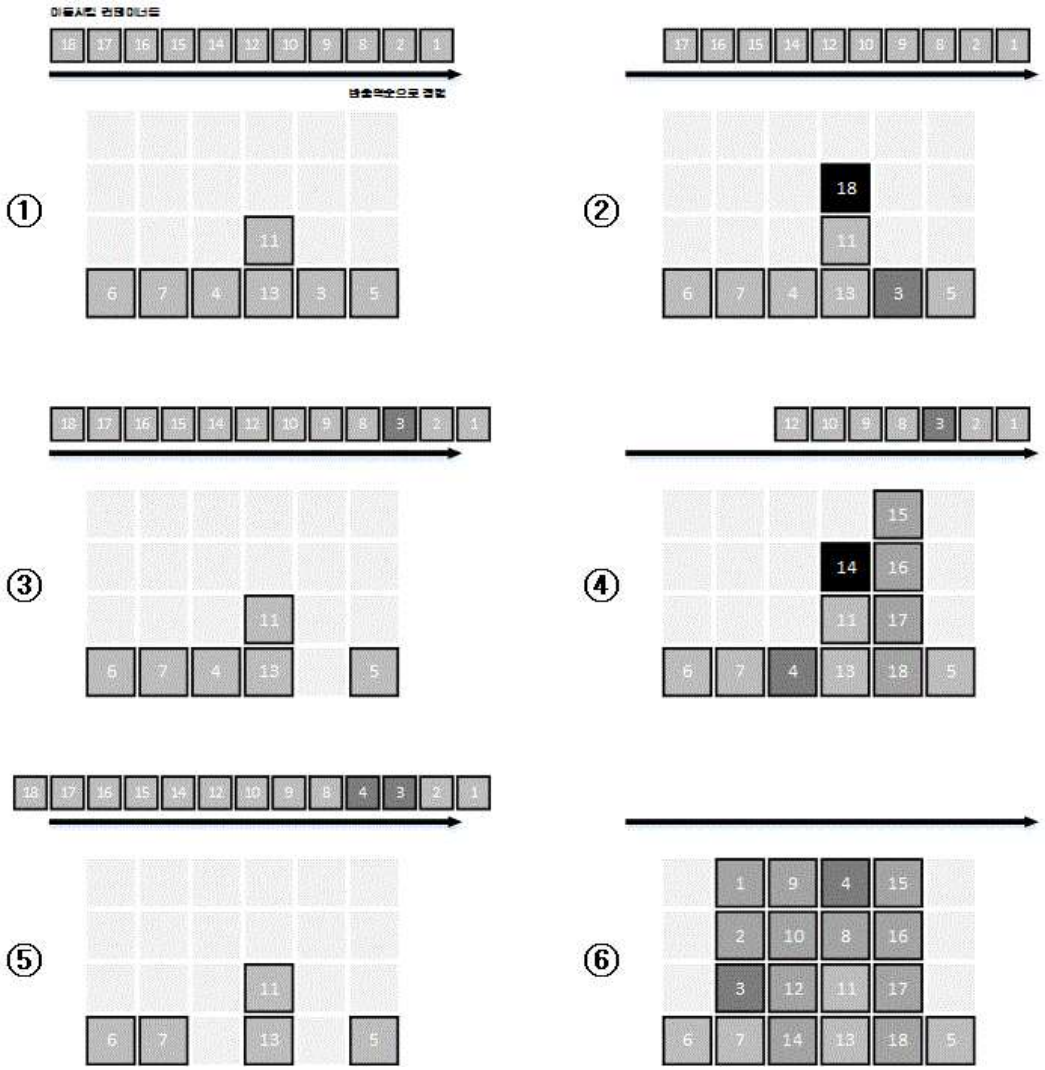


Figure 9. Re-handling lower bound: 13

<Figure 9>에 나타난 과정을 조금 더 명확하게 서술하면 다음과 같다. 먼저, 현재까지 재취급이 필요한 것으로 판단된 모든 컨테이너를 걷어내어 재취급 목록(rehandle list)에 역순정렬하여 보관한다. 이후, 현재 장치되어 있는 컨테이너 중 가장 위에 올려져 있는 컨테이너만을 고려했을 때 가장 큰 수를 '장치최대값(placed max=11)', 재취급 목록 중에서 가장 큰 수를 '재취급최대값(rehandle max=18)'라고 정의하자. 만약 재취급최대값이 장치최대값보다 크다면, 추가적이 재취급이 있어야만 정돈을 완료할 수 있는 상태임을 알 수 있다. 현재 장치되어 있는 컨테이너 중 가장 위에 올려져 있는 컨테이너만을 고려했을 때 가장 작은 값을 '장치최소값(placed min=3)'이라 정의하고, 추가적으로 걷어낼 (재취급 목록에 포함시킬) 컨테이너를 장치최소값에 해당하는 컨테이너로 결정한다. 걷어낸 컨테이너를 재취급 목록에 역순정렬을 고려해 추가한다. 그리고 같은 과정을 반복한다. <Figure 9>의 3번 그림과 같이 상황이 바뀌어 장치최대값은 0, 재취급최대값은 18로 계산된다. 이때, 장치최대값이 0인 이유는 완전히 비어있는 열이 존재하기 때문이다. 재취급목록의 컨테이너를 비어 있는 열에 최대한으로 쌓는다(컨테이너 18, 17, 16, 15). <Figure 9>의 4번 그림과 같이 상황이 바뀌어 장치최대값은 11, 재취급최대값은 14로 계산된다. 여전히 재취급최대값이 장치최대값보다 크기 때문에 추가적인 재취급이 필요하다. 이에 따라 장치최소값 4에 해당하는 컨테이너를 걷어내고 재취급 목록에 역순정렬을 고려해 추가하면 <Figure 9>의 5번 그림과 같아진다. 이전과 같이 장치최대값은

0, 재취급최대값은 18로 계산되며 4개의 컨테이너를 재취급 목록으로부터 투입한다. 이후, 장치최대값은 0, 재취급최대값은 14로 계산되며 또다시 4개의 컨테이너를 재취급 목록으로부터 투입할 수 있다. 다음으로는 장치최대값이 11, 재취급최대값 8이 되어 컨테이너(11) 위에 최대한으로 (2개의) 컨테이너를 투입한다. 마지막으로 장치최대값은 7, 재취급최대값은 3으로 계산되며, 마지막 3개의 컨테이너를 투입함으로써 알고리즘이 종료된다.

마지막 단계를 통해 컨테이너 3과 컨테이너 4를 추가적으로 재취급 컨테이너로 집계함으로써 재취급 하한값을 '+2'만큼 더 정확하게 계산해낼 수 있다. 이러한 복잡한 과정을 일반화시켜 알고리즘으로 표현하면 아래의 <Algorithm 2>과 같이 표현할 수 있다.

The Last Step for Computing the Lower Bound

```
Initialize lower bound for rehandling count to the rehandling count from phase2
Initialize container list rehandle_list to the rehandling containers from phase2
Initialize container list top_list to the top containers in each row
while rehandle_list is not empty
    if empty row exists, for all empty rows,
        pop containers from rehandle_list, largest number first, to the maximum tier
    if rehandle_list is empty, break while
    Set placed_max = maximum of top_list
    Set rehandle_max = maximum of rehandle_list
    if placed_max > rehandle_max, for the placed_max's row,
        pop containers from rehandle_list, largest number first, to the maximum tier
    else
        Set placed_min = minimum of top_list, add placed_min to rehandle_list
        count++
end while
return count
```

Algorithm 2. Pseudo code for calculating lower bound (for the last phase)

(2) Problem Solved

Action을 취함으로써 재취급이 전혀 필요하지 않은 상태, 즉 정돈이 완료된 상태에 도달하면서 문제가 완전히 해결될 수도 있다. 컨테이너 재정돈 문제의 해결을 위해서는 현재의 상태를 강화하는 탐욕(greedy) 움직임 외에도 정렬된 상태를 무너뜨리는 움직임이 포함되어야 한다. 최종적으로 문제 해결을 위해 현재의 상태를 악화시키는 움직임을 충분히 고려할 수 있도록 문제가 해결된 경우 '+5' 만큼의 보상 부여하였다.

4. 실험

4.1. 실험환경 및 Parameters

Intel i5-4690 @3.50GHz CPU와 8.0GB의 RAM을 사용했으며 외장 GPU(Graphics Processing Unit)는 사용하지 않았다. 파이썬 PyTorch(PyTorch, 2020) 19.1 버전을 이용하였으며 Gym 환경 등 필수적인 종속관계(dependency)을 추가적으로 설치하였다. Kingma and Ba(2015)에 의하면 Adam(Adaptive Moment

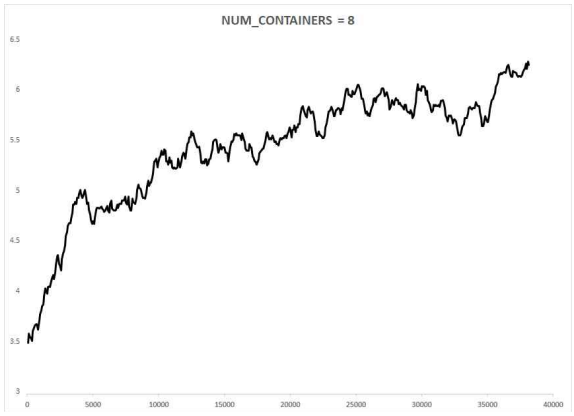
Estimation) optimzer는 데이터와 매개 변수량이 많은 복잡한 문제를 연산량 측면과 메모리 측면에서 효율적으로 학습할 수 있다. MuZero 알고리즘에 적용되는 매개 변수가 많다는 점과 컨테이너 재정돈 문제의 복잡성을 고려하여 실험에 Adam optimizer를 사용하였다. 대부분의 매개 변수 값은 Duvaud and Hainaut(2019)의 여러 게임에 적용한 실험을 참고하였다. 실험에 사용했던 주요 매개 변수들의 값과 의미를 아래 <Table 1>에 정리하였다.

Parameter	Value	Explanation
learning_rate	0.003	Learning rate for the model
num_workers	1	Number of simultaneous workers to feed the replay buffer
max_moves	30	Maximum number of moves if game is not finished before
num_simulations	100	Number of future moves self-simulated
discount	1	Chronological discount of the reward
root_dirichlet_alpha	0.25	Root prior exploration noise
root_exploration_fraction	0.25	Root prior exploration noise
blocks	1	Number of blocks in the Residual Network
channels	16	Number of channels in the Residual Network
reduced_channels_reward	16	Number of channels in reward head
reduced_channels_value	16	Number of channels in value head
reduced_channels_policy	16	Number of channels in policy head
resnet_fc_reward_layers	8	The hidden layers in the reward head of the dynamic network
resnet_fc_value_layers	8	The hidden layers in the value head of the prediction network
resnet_fc_policy_layers	8	The hidden layers in the policy head of the prediction network
batch_size	64	Number of parts of games to train on at each training step
value_loss_weight	0.25	Scale the value loss to avoid over-fitting of the value function
weight_decay	0.0001	L2 weights regularization
replay_buffer_size	3,000	Number of self-play games to keep in the replay buffer
num_unroll_steps	20	Number of game moves to keep for every batch element
td_steps	20	Number of steps in the future to take into account for calculating the target value
PER	True	Prioritized Replay, select in priority the elements in the replay buffer which are unexpected for the network
PER_alpha	0.5	How much prioritization is used

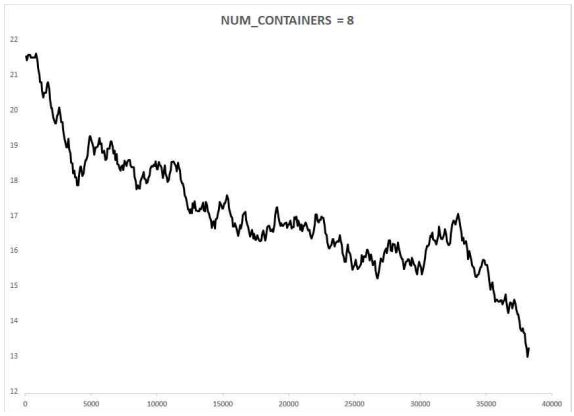
Table 1. Types and value of parameters used in the experiments

4.2 실험결과

모델의 검증을 목표로 실험을 진행하였다. 6열 6층의 공간에 존재하는 컨테이너의 수를 8개, 10개, 12개로 설정하여 총 세 가지 실험을 진행하였다. 컨테이너 재정돈 문제가 해결되면 환경에서의 리셋(reset) 기능이 작동하며, 이때에는 랜덤하게 새로운 형태의 상태를 생성한다. 만약 랜덤하게 생성된 상태가 이미 재정돈 완료된 상태라면, 그렇지 않을 상태가 생성될 때까지 새롭게 생성하였다. 아래의 <Figure 10~12>은 각각 컨테이너 개수를 8개, 10, 12개로 설정했을 때 MuZero 알고리즘이 컨테이너 재정돈 문제를 학습한 과정을 보여준다.

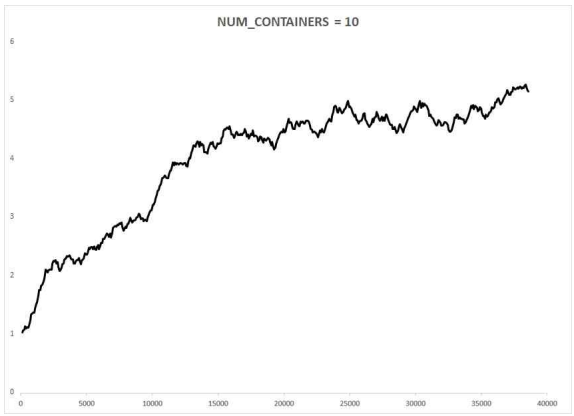


a) increasing reward over training

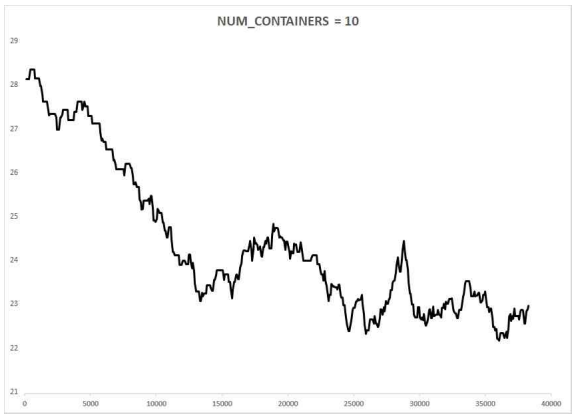


(b) decreasing re-handling steps over training

Figure 10. Number of Containers: 8

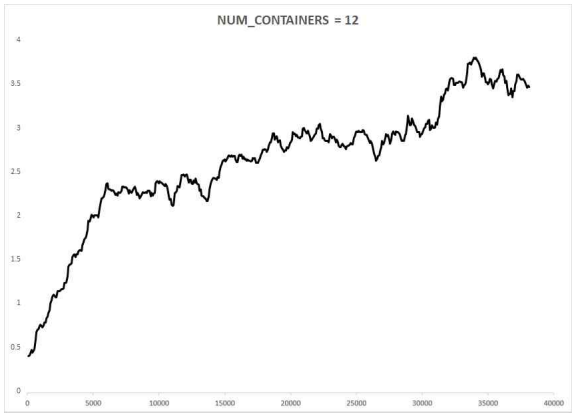


a) increasing reward over training

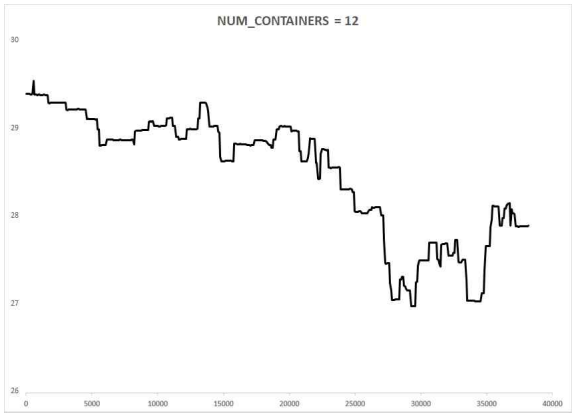


(b) decreasing re-handling steps over training

Figure 11. Number of Containers: 10



a) increasing reward over training



(b) decreasing re-handling steps over training

Figure 12. Number of Containers: 12

<Figure 10~12>에는 각각 두 개의 도표가 제시되었다. (a)와 (b)의 X축은 모두 학습단계(training step)를 의미한다. 하나의 학습단계는 <Table 1>의 batch_size 값으로 제시된 바와 같이 64회의 게임이 진행된다. 각 게임은 에피소드(episode)를 형성하는데, 각 에피소드는 랜덤한 상태로부터 시작하여 컨테이너 재정돈이 완료될 때까지의 모든 시점과 각각의 시점에서 취했던 action, 그리고 그로 인한 상태를 포함한다. <Figure 10~12> (a)의 Y축은 최근 100개 학습단계에서 평균적으로 재정돈이 완료되기까지 보상으로 획득한 값을 의미하며, (b)의 Y축은 최근 100개 학습단계에서 평균적으로 컨테이너 재정돈을 완료하는 데 필요했던 재취급의 수, time step의 수를 의미한다. 즉, 학습이 진행됨에 따라 보다 효율적인 action을 선택함으로써 점차적으로 보상량이 증가하는 현상이 (a)에 나타나고 있으며, 컨테이너 재정돈이 완료되기까지 소요된 재취급 횟수가 줄어드는 현상이 (b)에 나타난다.

<Figure 10>에서 확인하듯, 컨테이너 개수가 충분히 적을 때에는 재정돈을 완료하는 데 필요한 재취급 횟수가 10회 수준으로 줄어드는 것을 확인할 수 있다. 반면, <Figure 12>과 같이 컨테이너 개수가 일정 수준을 넘어섰을 때에는 평균 재취급 횟수가 요동치는 현상이 관찰되기도 한다. 또한, 평균 재취급 횟수가 25회 이상의 수준에 머무르는 것으로 보아 학습이 충분히 진행되지 못한 것으로 판단된다.

실험을 통해 해당 연구에서 제시한 학습 전략에 MuZero 알고리즘을 적용해 항만의 컨테이너 재정돈 문제를 학습할 수 있다는 것이 확인되었다. 또한, 컨테이너의 개수가 늘어나더라도 시간이 지남에 따라 평균 보상 획득값이 증가하고, 평균 재취급 횟수는 감소하는 추세를 실험결과에서 확인할 수 있었다. 이에 따라, 제한된 자원(resource)으로는 컨테이너의 개수가 늘어남에 따른 복잡성을 극복하기 어렵다는 사실을 확인하였다. 자원은 GPU와 같은 하드웨어와 시간적 자원 등을 의미한다.

5. 결론

본 연구에서는 항만의 작업생산성 향상을 위한 컨테이너 재정돈(pre-marshalling) 계획 문제 해결을 위해 노력했다. 컨테이너 재정돈의 결과로 나타나는 적하작업시간 감소 효과 및 외부트럭 대기시간 감소 효과는 다수의 논문에 의해 정리된 바 있다(Park at el., 2008, Park at el., 2012). 본 연구에서는 최근 주목받는 인공지능의 강화학습 딥러닝 기법을 이용해 항만의 컨테이너 재정돈 문제에 접근하였다. 특히, 기존 항만 연구분야에서 사용된 바 없는 MuZero 알고리즘을 이용함으로써 연구의 차별성을 두었다.

항만 컨테이너 재정돈을 강화학습하기 위한 일종의 시뮬레이터인 학습 환경(environment)을 직접 구성하였다. 환경을 구축함에 있어 효율적인 학습을 유도하기 위한 다양한 전략과 장치가 마련되었다. 먼저, 불가능하거나 낭비로 판단되는 action을 마스킹함으로써 보다 합리적인 action만을 고려할 수 있도록 하였다. 또한, 재정돈을 위한 재취급 횟수의 하한값을 휴리스틱 방법으로 계산하여, action 이전 상태의 하한값과 action 이후 상태의 하한값을 고려한 보상 체계를 마련하였다. 6열 6층의 공간에 컨테이너가 8개, 10개, 12개 있을 때를 기준으로 실험을 진행하였다.

연구를 통해 얻을 수 있는 결론은 다음과 같다. 첫째, MuZero 알고리즘을 이용하여 항만의 재정돈 문제를 학습하는 것이 가능하다. 처음으로 심층신경망 강화학습을 활용하여 컨테이너 재정돈 문제에 접근하였으며, 학습의 충분한 가능성을 검증하였다. 둘째, 큰 사이즈의 문제에서 나타나는 복잡성을 극복하기 위해서는 더 많은 하드웨어적, 시간적 자원을 투입할 필요가 있다. 문제의 사이즈는 크게 두가지 측면이 있는데, 하나는 컨테이너 열의 개수 또는 층의 개수를 확장하는 것, 또 하나는 컨테이너의 수를 확장하는 것이다. 셋째, action을 마스킹함으로써 학습의 효과를 극대화할 수 있다. 다양한 마스킹 방법을 적용하였는데, 더 많은 action을 조건부로 마스킹하면 더욱 효율적인 학습이 가능할 것이다. 하지만, 강화학습에 인간의

지식을 너무 많이 주입하는 것 또한 문제가 될 수 있음에 유의해야 한다. 넷째, 이전 상태와 현재 상태의 최소 재취급 횟수 하한값(lower bound)을 휴리스틱을 통해 계산한 뒤 그 변동량을 보상으로 활용하는 방법이 효과적으로 작동한다는 사실을 확인하였다. 즉, 휴리스틱 방법론으로 강화학습을 가속시킬 수 있다.

본 연구에서 진행된 실험과 결론으로 미루어, 앞으로의 추가연구의 방향을 명확하게 설정할 수 있다. 첫 번째로는, 학습을 위한 자원을 증가시켜 추가적인 연구를 진행할 수 있다. 두 번째로는, 재취급 횟수를 최소화하는 것 이상의 목표를 설정한 연구가 가능하다. 최적의 컨테이너 재정돈 계획이 반드시 최소한의 재취급 횟수만을 의미하지는 않는다. 재정돈 작업 중에 발생하는 작업시간을 최소화하는 것을 목표로 할 수도 있다. 또는, 추후 선적 또는 반출작업에서 발생할 작업시간을 최소화시키는 것을 목표로 두는 등 더 다양한 가능성을 고려해 연구를 확장할 수 있다.

참고문헌

Bang, S., Tijus, C. (2018), Problem Solving using Recurrent Neural Network based on the Effects of Gest ures, 10th International Joint Conference on Computational Intelligence (IJCCI 2018), 211-216

Carlo, H. J., Vis, I. F. A., Roodbergen, K. J. (2014), Storage yard operations in container terminals: Litera ture overview, trends, and research directions, European Journal of Operational Research, 235(2), 412-430

Caserta, M., and Voß, S. (2009), A corridor method-based algorithm for the pre-marshalling problem, Le cture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lect ure Notes in Bioinformatics), 5484 LNCS, 788–797, https://doi.org/10.1007/978-3-642-01129-0_89

de Vries, J. A., Voskuil, K. S., Moerland, T. M., and Plaat, A. (2021), Visualizing MuZero Models, arXiv: 2102.12924

Duvaud, W. and Hainaut, A. (2019), MuZero General: Open Reimplementation of MuZero, GitHub repo sitory, https://github.com/werner-duvaud/muzero-general (accessed on Oct 9, 2021)

Expósito-Izquierdo, C., Melián-Batista, B., and Moreno-Vega, M. (2012), Pre-Marshalling Problem: Heuris tic solution method and instances generator, Expert Systems with Applications, 39(9), 8337–8349, http s://doi.org/10.1016/j.eswa.2012.01.187

Gheith, M. S., Eltawil, A. B., and Harraz, N. A. (2014), A rule-based heuristic procedure for the containe r pre-marshalling problem, IEEE International Conference on Industrial Engineering and Engineering Management, 2015-Janua, 662–666, https://doi.org/10.1109/IEEM.2014.7058721

Ha, B., Kim, S. (2012), A* Algorithm for Optimal Intra-bay Container Pre-marshalling Plan, Journal of Korean institute of industrial engineers, 38(2), 157-172

Hart P. E., Nilsson N. J., Raphael B. (1968), A Formal Basis for the Heuristic Determination of Minimu m Cost Paths, IEEE Transactions on Systems Science and Cybernetics, 4(2), 100 – 107

Hirashima, Y. (2009), A Q-learning System for Container Marshalling with Group-Based Learning Model at Container Yard Terminals, Proceedings of the International MultiConference of Engineers and Com puter Scientists, 1

Prandtstetter, M. (2013), A dynamic programming based branch-and-bound algorithm for the container pre-marshalling problem, Technical report, AIT Austrian Institute of Technology

Hottung, A., Tanaka, S., and Tierney, K. (2020), Deep learning assisted heuristic tree search for the cont ainer pre-marshalling problem, Computers and Operations Research, 113, https://doi.org/10.1016/j.co r.2019.104781

Hottung, A., and Tierney, K. (2016), A biased random-key genetic algorithm for the container pre-marsh alling problem, Computers and Operations Research, 75, 83–102, https://doi.org/10.1016/j.cor.2016.05. 011

Kang, J., Ryu, K., Kim, K. (2004), Efficient remarshalling of containers in the bay of container yard, Proc

eedings of the Korea Intelligent Information System Society Conference 2004.11, 287-295

Khpeek. (2017), Q-learning-hanoi, https://github.com/khpeek/Q-learning-Hanoi (accessed on Oct 9, 2021).

Kingma, D. P., and Ba, J. L. (2015), Adam: A method for stochastic optimization, 3rd International Conf erence on Learning Representations, ICLR 2015 - Conference Track Proceedings, 1–15.

Lee, Y., and Chao, S. L. (2009), A neighborhood search heuristic for pre-marshalling export containers, European Journal of Operational Research, 196(2), 468–475, https://doi.org/10.1016/j.ejor.2008.03.011

Lee, Y., and Hsu, N. Y. (2007), An optimization model for the container pre-marshalling problem, Com puters and Operations Research, 34(11), 3295–3313, https://doi.org/10.1016/j.cor.2005.12.006

Liu, Y. F., Lee C. B. (2019), A Study on the Structural Changes of Import and Export Containers betwee n South Korea and China Ports, Korea Logistics Review, 29(2), 1-12

Mitchell A. P., Kenneth A. D. J. (1994), A Cooperative Coevolutionary Approach to Function Optimizati on, International Conference on Parallel Problem Solving from Nature

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M. (2013), Pl aying Atari with Deep Reinforcement Learning, arXiv:1312.5602

OpenAI. (2018), OpenAI Gym, https://gym.openai.com/read-only.html, (accessed on Oct 9, 2021).

Paias, A., Ruthmair, M., and Voß, S. (2016), Solving the Robust Container Pre-Marshalling Problem, Lect ure Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lectu re Notes in Bioinformatics), 9855 LNCS, 5(6), https://doi.org/10.1007/978-3-319-44896-1

Park, T., Nam, J., Kim, T., Ryu, K. (2012), A Multi-objective Co-operative Co-evolutionary Algorithm for Remarshaling in an Automated Container Terminal, Journal of KISS : Software and Applications, 39 (1), 45-55

Park, K., Park, T., Kim, M., Ryu, K. (2008), Planning for Intra-Block Remarshaling to Enhance the Efficie ncy of Loading Operations in an Automated Container Terminal, Journal of Intelligence and Informati on Systems, 14(4), 31-46

Park, K., Park, T., Ryu, K. (2009), A Cooperative Coevolutionary Algorithm for Optimizing Remarshalin g Plan in an Automated Stacking Yard, Journal of navigation and port research, 33(6), 443–450

Park, Y. (2016), Comparison of Algorithm based on the Container Remarshalling Efficiency Factor in Po rt Distribution, Journal of Distribution Science 14(5), 107-114

Parreño-Torres, C., Alvarez-Valdes, R., and Ruiz, R. (2019), Integer programming models for the pre-ma rshalling problem, European Journal of Operational Research, 274(1), 142–154, https://doi.org/10.1016 /j.ejor.2018.09.048

Pierrot, T., Ligner, G., Reed, S., Sigaud, O., Perrin, N., Laterre, A., Kas, D., Beguir, K., Freitas, N. (2019), Learning Compositional Neural Programs with Recursive Tree Search and Planning, arXiv:1905.12941 [cs.AI]

PyTorch. (2020). Pytorch Documentation, https://pytorch.org/docs/stable/index.html (accessed on Oct 9, 2021).

Rendl, A., and Prandtstetter, M. (2013), Constraint models for the container pre-marshaling problem, M odRef 2013: The Twelfth International Workshop on Constraint Modelling and Reformulation, 44–56

Rosin, C. D. (2011), Multi-armed bandits with episode context, Annals of Mathematics and Artificial Int elligence, 61(3), 203–230, https://doi.org/10.1007/s10472-011-9258-6

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., H assabis, D., Graepel, T., Lillicrap, T., and Silver, D. (2020), Mastering Atari, Go, chess and shogi by pla nning with a learned model, Nature, 588(7839), 604–609, https://doi.org/10.1038/s41586-020-03051-4

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Anto noglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Suts kever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. Nature, 529(7587), 484–489. https://doi.org/1 0.1038/nature16961

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., Hassabis, D. (2017), Mastering Chess and Shogi by Self-Pla y with a General Reinforcement Learning Algorithm, arXiv:1712.01815v1 [cs.AI]

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., La i, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., Van Den Driessche, G., Graepel, T., and Has sabis, D. (2017), Mastering the game of Go without human knowledge, Nature, 550(7676), 354–359, ht tps://doi.org/10.1038/nature24270

Tanaka, S., and Mizuno, F. (2018), An exact algorithm for the unrestricted block relocation problem, Co mputers and Operations Research, 95, 12–31, https://doi.org/10.1016/j.cor.2018.02.019

Tanaka, S., and Tierney, K. (2018), Solving real-world sized container pre-marshalling problems with an iterative deepening branch-and-bound algorithm, European Journal of Operational Research, 264(1), 16 5–180, https://doi.org/10.1016/j.ejor.2017.05.046

Tanaka, S., Tierney, K., Parreño-Torres, C., Alvarez-Valdes, R., and Ruiz, R. (2019), A branch and bound approach for large pre-marshalling problems, European Journal of Operational Research, 278(1), 211–2 25, https://doi.org/10.1016/j.ejor.2019.04.005

Tierney, K., Pacino, D., and Voß, S. (2017), Solving the pre-marshalling problem to optimality with A* and IDA*, Flexible Services and Manufacturing Journal, 29(2), 223–259, https://doi.org/10.1007/s1069 6-016-9246-6

Van Brink, M., and Van Der Zwaan, R. (2014), A branch and price procedure for the container premars halling problem, Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 8737 LNCS, 798–809, https://doi.org/10.1007/978-3- 662-44777-2_66

Wang, N., Jin, B., and Lim, A. (2015), Target-guided algorithms for the container pre-marshalling problem, The International Journal of Management Science, 53, 67–77, https://doi.org/10.1016/j.omega.2014.12.002

Wang, N., Jin, B., Zhang, Z., and Lim, A. (2017), A feasibility-based heuristic for the container pre-marshalling problem, European Journal of Operational Research, 256(1), 90−101, https://doi.org/10.1016/j.ejor.2016.05.061

Yeo, G. (2002), An Evaluation of the Competitiveness of Chinese Container Ports, Korea Trade Research *Association*, **34**, 39-60