Can Reinforcement Learning solve the Human Allocation Problem?

Phong Nguyen,[1] **Matsuba Hiroya,** [1] **Tejdeep Hunabad,** [1]
Dmitrii Zhilenkov, [2] **Hung Nguyen,** [2] **Khang Nguyen** [2]
[1] Hitachi Center of Technology Innovation
[2] FPT Software
phong.nguyen.kj@hitachi.com

Abstract

In recent years, reinforcement learning (RL) has emerged as a new, promising way to solve old problems. The algorithms' role in finding approximate solutions in NP-hard complexity became crucial for developing modern intelligent decisions. In this paper, we consider the human resource allocation problem, which is one of classical NP-hard complexity problem, with the fixed number of workers and tasks, using various RL methods, such as deep contextual bandit, double deep Q-learning network (DDQN), and a combined approach of DDQN with the Monte Carlo Tree Search (MCTS). The deep contextual bandit algorithm showed good performance for the low dimensional cases, but it is inappropriate for real problems in the chosen setting. To overcome such barriers, we decomposed the task in terms of Markov decision processes and reduced the action space so that all sequential RL methods became available. In this paper, we proposed a way to deal with NP-hard problems via modern RL approaches and compare different methods' performance. We studied different algorithms in the RL family, namely Contextual Bandit, DDQN, DDQN with MCTS. Those algorithms worked with appropriate settings and improved at least 30% in time reduction compared to random assignment.

Introduction

A large number of well-known nondeterministic polynomial time-hard (NP-hard) combinatorial problems, such as Travelling Salesman (Papadimitriou 1977), Minumum Vertex Cover (Dinur and Safra 2005), and Maximum Cut (Goemans and Williamson 1995), are canonical challenges in computer science. Human resource allocation is considered one of the most classical NP-hard problems for combinatorial optimization. Since it is a crucial complex activity of managers in an organization, various methods for approaching this class of problem are of great interest. If the dimensionality is low enough, traditional algorithms like brute force or mathematical optimization can find the exact solution. Recently, many researchers have focused on applying various methods to this problem as a solution mechanism for different conditions. Nevertheless, when real-life situations usually have much higher dimensions, the exact solution cannot be quickly determined. Many organizations struggle to deal with this challenge and waste their existing human resource to achieve optimal efficiency.

As the world continues to urbanize, there arises a need for sustainable development through efficient management of urban resources to ensure the Quality of Life (QoL) of the residents in these regions. In recent times this has been termed as making the cities smart. The question that arises here is – What makes a city smart? A smart city leverages technology solutions to improve its services and the living quality of its residents. These solutions are developed by analyzing the data obtained from IoT sensors, networks, and applications regarding traffic congestion, air quality, energy usage, emergency services, and so on. Hitachi Automated Workflows for Connected Systems (HAWCS) is an example of a Cyber Physical System (CPS) that was initiated to integrate all video analytics related to public safety into a single platform. This platform is used to automate incident detection and dispatch ground staff to handle the incidents. Modern urban spaces are highly complex and connected to multiple systems to facilitate diverse kinds of operations. Surveillance monitoring, traffic control, and building management systems can be considered as some of the widely used systems in these modern urbanized spaces. Therefore, the functionality of HAWCS is being expanded as a smart city platform to handle diverse verticals and use cases without limiting public safety and security.

Our research focuses on the dispatch of personnel to carry out services. These services include regular maintenance work or repairs and handling emergencies such as road accidents, fire accidents, and security breaches. Diverse situations may arise simultaneously in smart cities, causing situation management and response to be highly challenging. An efficient method of dispatch will be required to carry out the service events in minimum time and with efficient use of resources (Kaelbling et al. 1996). It is worth noting that while ensuring the QoL of the residents by optimally handling the incidents, one must also keep in mind that working conditions would affect the service providers' productivity. This paper discusses a new dispatching method that considers the above factors, as explained in the section.

Reinforcement learning (RL) has been gaining popularity over the past years. From its inception in the early 1980s until today, RL's field has been growing incrementally, transforming this area of Machine Learning into a powerful tool.

Copyright © 2021, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

RL algorithms have been a hot topic for many researchers (Kaelbling et al. 1996). One of the state-of-the-art frameworks is Deep Q-Network (DQN), proposed by Google in 2015. DQN uses a neural network to map state spaces and Q-values of actions, which the agent will use to select the best action in a given state (Mnih et al. 2015). With this method, the deep neural network estimates the action-value function and determine the best action given an input state (Kaelbling et al. 1996). The algorithm stores all the agent's samples in a replay memory and learns by sampling some in it.

The RL field has been developing a long way since DQN. Scientists proved that RL has considerable successes in various tasks, such as games (Silver et al. 2017a), continuous actions control (Lillicrap et al. 2015), locomotion skills (Peng et al. 2017), navigating in complex environments (Bouton et al. 2019), and even robotics (Kormushev, Calinon, and Caldwell 2013). For example, with the recent well-known achievement of AlphaZero (Silver et al. 2017a), an RL algorithm was built to defeat the world champion in Go's game. RL has proved to solve some of the optimization problems (Barrett et al. 2020).

In this paper, we aim to explore how to utilize RL to solve the human resource allocation problem, considering the non-trivial constraints. Here and after, the terms "dispatch problem" and "human resource allocation problem" will be used with the same meaning.

Related Work

The question about effective resource allocation was deeply studied via many approaches, starting from research operations methods (Li et al. 2018), to different heuristics algorithms (Hegazy 1999).

Dispatch Problem

We consider the human resource allocation problem with m workers and n tasks. There are several assumptions that we take into account:

- Each worker could be assigned to several tasks;
- One worker cannot solve more then one problem simultaneously;
- Each task $k \in \overline{1, n}$ is connected with some event;
- The events have the "start time" and "location" features;
- The tasks into events are stacked to the sequences and each following task could not be started until the previous was done;
- Each worker is characterized by his vector of expected time costs needed to solve each task;
- There is a transition matrix D where element d_{ij} is an expected time needed to get from event i to event j;
- All tasks should be solved.

The goal is to find an assignment with the minimal total completion time, considering that workers have to bear time costs to move between event locations and solve the tasks only in the defined sequences (e.g. task 2 of event 1 could be started only after task 1). The chosen assumptions let us consider more natural constraints for job scheduling, comparing with classical scheduling optimization with resources. Still, it also restricts the usage of known classical solvers for finding a critical path of the project because we face the need to consider dummy tasks with variable length. One of the ways to avoid such difficulties consists in the application of heuristic methods.

RL in Huge Action Spaces

There are many areas where we have to deal with mathematical optimization in discrete space. The well-known examples of such tasks are bin packing problem (Boyar et al. 2016), salesman problem (Ross, Proulx, and Karpenko 2020), graph coloring (Kubale 2004), and the human allocation problem is among them. The common feature for such cases is an exponential complexity, also known as a combinatorial explosion, because the total amount of possible solutions is growing according to exponential law $\mathcal{O}(m^n)$, where m is a specific constant, depending on the task, and n is a natural number characterizing the dimensionality of the problem.

The typical way to reduce the search's complexity is to find a specific structure in the search space and exploit it, such as continuous approximation or relations between the search space elements. Sometimes there exist non-trivial constraints that are preventing the opportunity to catch such structure. For example, it could be complex games like chess, where we have to optimize the choice, considering the sequences of moves of two antagonistic players. The classical approach as dynamic programming usually suffers when the dimensionality is growing. That is why new methods for solving such problems became a high interest in science and production.

This paper considers RL approaches to find hidden dependencies and make the algorithm reason about problems with complex constraints. There are two main concepts that we will touch on as follows.

The first one is based on the contextual bandit implementation. This family of algorithms represents a typical RL problem, where we have the only state during the game, send one action to the environment, and immediately observe the reward. Then the new game starts, and the process repeats. Therefore, there is an opportunity to process triples (state, action, reward) and use them for machine learning. Gabriel Dulac-Arnold, Richard Evans, and others (Dulac-Arnold et al. 2015) proposed a policy-based method, namely the Walpetinger algorithm, to work with significantly large action space. This approach supposes to use the Actor-Critic policy optimization in continuous space and then implement the k-nearest neighbors method to get a feasible solution in discrete space. The authors presented the results showing the possibility to train the model and make predictions, but the dimensionality remains limited and inappropriate for real large-scale problems.

The experiments with the Walpetinger algorithm did not improve the results for the dispatch problem. Therefore the question about bandit implementation for significant scales remains opened.

The second method is deep RL. In this case, the common practice is to decompose the problem and optimize overall decision sequences to build a final solution. We consider a standard DQN approach (Mnih et al. 2013; Schaul et al. 2016) and a more complex idea named DDQN with Monte Carlo tree search (MCTS), inspired by the AlphaZero algorithm.

Conventional Optimization Solver

One of the most common conventional optimization approaches to the human resource allocation problem is identifying the critical path, known as the Critical Path Method (CPM) (Bishnoi 2018). The CPM aims to find the longest sequence of tasks in the shortest time possible to complete the project. The tasks on the critical path are known as critical activities because if they are delayed, the whole project will be delayed. The transformation of the critical path heavily impacts the total duration of the process model.

However, for projects where the number of officers and tasks is large, the CPM requires high computation due to creating complicated graphs. Furthermore, the CPM does not well handle the scheduling of the human resource allocation since it has to calculate the duration of all possible paths created by each worker. Also, the method requires the availability of deterministic time duration for each activity. In this case, more often than not, there is a vague idea about activity durations which then must be estimated subjectively (Nasution 1994).

How our work is different

In the early studies, conventional optimization solvers like the CPM (Bishnoi 2018), exact mathematical methods-based solution (Borba and Ritt 2014), simulation (Ammar, Elkosantini, and Pierreval 2012), genetic algorithms (Mutlu, Polat, and Supciller 2013), and other heuristic and metaheuristic (Aickelin and Li 2007) and multi-agent-based methods have been applied previously to the human resource allocation problem. However, they rarely take consecutive states, namely the previous and the current state, into account. Furthermore, if the dimensional space increases too large, it is impractical to apply these methods for every state. For each new scenario, these conventional methods spend a long time re-simulating and running from scratch. Conclusively, neither do the mentioned models take advantage of deep learning methods.

In our paper, we specifically apply the RL concept to human resource allocation. RL aims to learn good policies for sequential decision problems like human resource allocation by optimizing a cumulative future reward signal. Among deep learning methods, the RL can solve very complex problems that conventional techniques and simple heuristic modeling cannot solve. It is also believed to outperform humans in many tasks and bring out novel ways to solve problems. One of the most critical problems in the model is how to get the precise individual capability matrices. Individual capability matrices about workers are the data that need to be mined and solved. RL methods can offer the matrices based on the previous appraisal data sets and are suited to changing conditions. For instance, data is changing according to a worker's varying capabilities and experience. After the allocation, the environment feedbacks the evaluation rewards for the model's prediction. Most recently, a novel method named ECO-DQN is proposed to solve combinatorial optimization problems (Barrett et al. 2020), but it does not consider the human resource allocation problem. In this paper, we suggest applying two RL methods, including Double Deep Q-Network (DDQN) and DDQN with integrated Monte-Carlo tree search as a policy improvement operator, in a similar manner AlphaGo Zero (Silver et al. 2017b).

Proposed Methods

General Reinforcement Learning

RL imitates one of the most common learning styles in natural life. The machine learns to achieve a pre-defined goal through trial-and-error processes in a dynamic environment. In general, the learning model consists of:

- An agent
- A state-space S
- A set of available actions A
- A reward function $R : S \times A \rightarrow \mathbb{R}$

The agent's goal is to take optimal action at a given state to maximize the long-term reward. It is achieved by learning a policy π, which maps states and actions. The agent keeps a value function $Q(s, a)$ for each state-action pair, which represents the expected long-term reward if the agent is in the state s, taking the action a. By taking action, the system moves from one state to another, observes the rewards, and updates the value function $Q(s, a)$.

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big[\underbrace{r + \gamma \max_{d} Q(s',a')}_{\text{target}} - \overbrace{Q(s,a)}^{\text{prediction}}\big] \tag{1}$$

Where:

- α is a learning rate, from 0 to 1, which considers how much the agent should learn from the newest information.
- γ is a discount factor, from 0 to 1, which gives more weight to the rewards in the near future than in the far future.
- s, a are current state, current action.
- s', a' are next state, next action.
- r is the observed reward in the next state s' by taking action a from state s.

Based on this value function, the agent can decide the best action to take in a current state to achieve maximum long-term rewards. However, based on the action selection method, the agent occasionally takes random different actions from the best actions to explore and experience the consequences. By repeating this process over and over, Q-values, which is a table mapping states with actions, will converge, and the agent can start utilizing what it has learned to achieve the goal (Sutton and Barto 2018).

Deep Q-learning

The majority of real case examples suffers from extremely high amount of (state, action) pairs available during the decision making process. One of the ways to deal with such problem is to find a way to generalize similar states and equip an RL agent with a tool for action comparison.

There are plenty of machine learning algorithms aimed to find hidden dependencies in data and implicitly or explicitly define clusters to make predictions. During the last decade the most outstanding results were achieved via deep learning models. Therefore, the neural network $Q_\theta(s,a)$ could play the role of parametrized Q-value function, that approximates real Q-values, i.e. $Q_\theta(s,a) \approx Q(s,a)$. Hence, the problem of finding the best configuration of model parameters θ^* turns to finding the minimum of the mean squared error loss function $L(\theta)$, given by the following equation:

$$L(\theta) = \frac{1}{2}[r + \gamma \max_{a'} Q^0_\theta(s',a') - Q_\theta(s,a)]^2 \tag{2}$$

The better predictions of Q-values are available for the RL agent, the better total rewards he is able to achieve.

Exploration - Exploitation The exploration and exploitation trade-off is one of the RL's most sophisticated questions because there are many areas where unbalanced exploration leads to extra costs. This issue appears in RL due to the absence of the initial dataset. In general, machine learning algorithms are studying to reason about the general population by examples represented by given samples. The only way to collect data for the RL agent is to interact with the environment by taking action and observing rewards. If the agent always chooses one action in all possible states, then the collected dataset will not be representative because many outcomes stay unexplored. Therefore, the common way to make an agent visit different states consists of randomizing actions, at least while there is not enough representative sample.

In this paper, we concentrate on studying the question of usage RL for discrete optimization problems. Therefore we chose two simple exploration methods guided by ϵ-greedy (Sutton and Barto 2018) and softmax with scheduled temperature parameter.

The idea of ϵ-greedy consists of using a random variable ξ distributed uniformly on $[0;1]$ segment, and its observations are comparfed with fixed ϵ parameter. If the random value is smaller than ϵ, we explore a random action. If the random value is greater than ϵ, we exploit the action with the highest Q-value. ϵ is set at 1 at the beginning of the training and annealed to 0 linearly during training.

The softmax function provides a discrete distribution over arms, where each probability is a function of a given vector of Q-values $p_j : \mathbb{R}^n \to \mathbb{R}$.

$$p_j(q;\tau) = \frac{exp(q_j/\tau)}{\sum_{i=1}^{k} exp(q_i/\tau)} \tag{3}$$

The temperature parameter $\tau \in (0,\infty)$ controls the exploration. When τ turns to infinity, the distribution over arms became uniform, which corresponds to the pure exploration. Otherwise, if τ turns to zero, then an arm with the highest Q-value will be played almost surely.

Experience replay All the experiences (s,a,r,s') are stored in a replay memory. When training the network, random samples from the replay memory are used instead of the most recent transition. It breaks the similarity of subsequent training samples, which otherwise might drive the network into a local minimum.

Proposed Method: Deep Reinforcement Learning in Dispatch problem

To solve the dispatch problem, we formulate this problem into an RL approach. We consider the input of the workers' capabilities $\{c_i \in \mathbb{R}^{d_1} \mid i \in \overline{1,m}\}$, the start time of the events, the number of tasks n, and finally, the transition matrix Θ among locations as the RL agent's state. The RL agent needs to pick the actions, which can be considered the worker assignment's plan.

Contextual Bandit The contextual bandit extends the idea of the multi-armed bandit (Lattimore and Szepesvári 2020). The main idea of both methods is to learn the distributions of rewards for each arm, in a more general approach, the distribution of the model's parameters. Then, they play the arm with the highest reward. The only difference between the contextual bandit and its predecessor is the context with additional data that influence rewards distribution.

In RL's language, the context has the same meaning as a state and the arm equal to action. Any game with bandits game lasts only one round: the environment provides a state, the agent observes it and sends an action to the environment, and then the environment returns a reward for this action.

More formally, we consider:

- A set of games G_t where $t \in T = \{1,2,3,...\}$ (in general theory, the cardinality of T is unknown, and this point prevents direct application of the standard method of mathematical optimization for problems with fixed time horizon)
- A set of arms (actions) $A_t = \{a_1,...,a_K\}_t$ are available in each game $t \in T$
- The context (state) is given as a vector $s \in \mathbb{R}^n$, containing specific features of the environment. It also could be a set of contexts for each arm or concatenated contexts of arms and environment.
- The agent chooses one action $a_{k,t}$, and receives a reward $r_{a,t}$ (in general case r_t is a random number). The goal is to minimize the regret value, reflecting the total sum of differences between the average reward for the best arm and average reward for arms $a_{k,t}$ played in different games, i.e.

$$\mathbb{R}_T = T \cdot \max\{r_{a,t} | a \in A_t\} - \mathbb{E}\sum_{t=1}^{T} r_{a,t} \tag{4}$$

Where operator $\mathbb{E}$ denotes an expected value.

The common utilization of the context vector consists of rewarding function approximation. The reward is considered

as a function of context, and the model is training on (s, a, r) tuples. Therefore, deep learning models also became popular for this purpose as universal non-linear approximators.

The context in the dispatch problem is given by the environment settings. Each time before the round, the environment is reset with random parameters, and these parameters are considered as a state. The arms' context is represented by a one-hot encoded vector of the arm index. Then the $Q(s, a)$ function represented as some deep model is training on examples of games and being used to choose action $a_t \sim \pi(a, s)$, where $\pi(a, s)$ is a discrete distribution over actions. In our case, the function π was represented as a softmax over Q-values predicted by the model.

After each game, we add $\langle s, a, r \rangle$ tuple to the replay buffer, which served as a source of training objects for the neural network $Q(s, a)$.

Each arm corresponds to the unique assignment and contains $O(|A|)$ number of elements, where A is a set of actions. For the dispatch problem with m workers and n tasks the cardinality of A is given by $|A| = m^n$. The training process make the neural network approximate Q function, i.e. $Q(s, a; \theta) \approx Q(s, a)$, Since the Q-function depends on actions, then we have to keep all arm's contexts in memory. Hence, when the number of actions is growing according to the exponential law m^n it quickly leads to extra memory usage and increases time per iteration.

The experimental results for contextual bandits showed good performance, but it also has the disadvantage, preventing this method's usage. On the one hand, this example proves that deep models can generalize the states of a given environment and guide the agent to choose the right assignments. Still, on the other side, we can see in Table 1 the rapid growth of time per iteration.

Table 1: Time dependence per iteration of action space size

$|A|$	16	64	81	729	19683
Time (seconds)	0.5	1.9	2.4	21.8	630.2

To overcome such a barrier we had to consider problem decomposition and implement more general RL methods, like DDQN, which to reduce the action space on each iteration.

DDQN The contextual bandit can prove its efficiency with best results in the low dimensional space. Nevertheless, if the configuration scale significantly increases, the Contextual Bandit runs incredibly slow and cannot perform anymore due to the Out of Memory problem.

Our intuition is to apply a deep RL method, especially the Double Deep Q-Network (DDQN) since it can reduce the complexity of the action space and result in a large scale of configuration in such a short period. When the model is trained with good exploration, and the agent has learned a good policy, we can apply the model directly into a new scenario and run it in real-time. In short, DDQN can be used to solve very complex problems like Human Resource Allocation that conventional techniques cannot solve. With a deep neural network combined with exploration-exploitation, the DDQN model is very similar to human learning. Hence, it is close to achieving perfect performance.

In the DDQN, two value functions are learned by assigning each experience randomly to update one of the two value functions, such that there are two sets of weights. When we compute the Q-target, we need to create two networks to decouple the action's selection from the target Q-value generation. We use a DQN network to select the best action to take for the next state (the action with the highest Q-value), while a target network is used to calculate the target Q value of taking that action at the next state. The DQN network weights are replaced with the target network's weights to evaluate the current greedy policy.

We use the Experience Replay to make the DDQN training more stable and efficient. The replay buffer contains a collection of experience tuples (state, action, reward, next_state, terminal). The tuples are gradually added to the buffer as we are interacting with the Environment. The most straightforward implementation is a buffer of fixed size, with new data added to the end of the buffer to push the oldest experience out of it. Then, we sample a small batch of tuples from the replay buffer for training the model.

We also detach officers' space out of states' space so the DDQN can solve the settings in another space. The DDQN handles officers' space as an action space that allocates each officer to the job step by step. Besides, we apply some popular RL techniques such as the noisy linear, prioritized experience replay to improve the overall performance of DDQN.

DDQN with Monte Carlo Tree Search The DDQN algorithm (Silver et al. 2016) helped make a more stable training process than simple DQN, but its implementation was still limited for complex processes or games, such as Go or Chess. In 2016 DeepMind proposed a new method named AlphaGo that outperformed human capabilities in Go's game and many others with later versions. A significant improvement was achieved via a combination of DDQN and Monte-Carlo tree search (MCTS).

The MCTS is an algorithm incarnating the idea of depth search for the given tree. The most general implementation consists of four main steps: selection, expansion, rollout, and backpropagation. This method's intuition is the following: get some initial state of Markov decision process (represented as the root of tree structure) and then play actions in some reasonable way till the terminal state of the process. On the final state, the agent observes the reward and then backpropagates this value along with the previous states and updates all Q-values of each state.

The tree is initialized with the root node given by the initial state. Then the selection procedure starts. In this step, MCTS makes a path from the root to the leaf node (on the first step root is also a leaf node). As soon as the leaf node is reached, several children should expand it, representing possible actions available in the initial state. The algorithm evaluates each child via rollout step (also known as simulation), which makes actions guided by some given policy π till the game's terminal state. When the terminal state is achieved, the algorithm can observe the reward and then backpropa-

gate it to the root and update all Q-values in the path.

The DDQN and MCTS interact in two stages:

- On the selection step DDQN guides which child should be selected next.
- On the rollout step, another neural network provides the policy over actions to achieve the terminal node (in the future versions of AlphaGo, the second neural network was reduced, and the policy was given by one general DDQN).

There were several modifications of AlphaGo that made the algorithm more stable and general. To deal with the dispatch problem, we considered AlphaZero. The solution is based on the pseudocode provided by DeepMind but adapted to the single-player game.

To describe the playing process, we have to introduce several additional notations given in the list below. The first way is to use the label encoding representation where:

- Let m be the number of workers and n the number of tasks.
- The assignment is a vector w of length n, where the element w_i take an ID value from the set of workers' IDs or (-1) if ID was not assigned.
- When the game starts all values in the assignment are equal to (-1), i.e. $w_i = (-1), \forall i \in \{0, ..., n-1\}$.
- The game terminates as soon as all tasks have assigned the ID value.
- The state is a vector given by concatenation of environment parameters h of the game and the vector w, i.e. $s = (h, w)$.

We also consider one hot encoded workers' IDs that let us avoid non-existent ordered relations between workers.

On each step of the game, the algorithm moves along the vector w and fills values representing the worker for each task. Thus the Markov's property holds because any new state depends only on the previous one. Therefore the process could be described as the Markov decision process.

When the player (agent) has to take action, he requests the policy from the MCTS and then samples some action following this policy. As soon as MCTS requests initial policy and Q-values from the DDQN, it plays the role of policy improvement operator.

Experimental Setup

We choose five different problem settings with increasing difficulty. The base configuration is 3 Officers - 2 Tasks - 2 Events (3O-2T-2E) to fairly compare the performance of Bandit, DDQN, and AlphaZero. At the same time, we make the task more challenging by increasing the configuration's complexity. We expect that RL agents can solve the problem where the distribution of optimization tasks changes over time or may be undetermined. In detail, the configuration is set as 4 Officers - 3 Tasks - 3 Events (4O-3T-3E), 5 Officers - 4 Tasks - 4 Events (5O-4T-4E), 6 Officers - 5 Tasks - 5 Events (6O-5T-5E), 7 Officers - 6 Tasks - 6 Events (7O-6T-6E). In the DDQN and DDQN+MCTS method, three types of neural networks proving their best practice performances in RL problems have been applied, including Convolutional Neural Networks (CNN), Mixed CNN with Long Short-Term Memory Networks (CNN-LSTM), Residual Neural Networks (ResNet). The final result of DDQN and DDQN+MCTS is determined based on the best practice network. All implemented methods simultaneously run in the same environment with 10000 steps. To ensure the convergence, the experiment was run for ten times per seed so that we can get the lower and upper bounds of each method.

We use the mean Random Normalized Score (RNS) as the primary metric for comparing between methods. Typically, the random performance is taken as the baseline, and the RNS obtained by an agent on the allocation problem can be measured relative to representative random performance. Each method's mean score is calculated by the best practice results of all experiments.

All experiments are set up on a server with the same computational benchmark. The Intel(R) Core(TM) Xeon(R) E5-2673 v3 processor clocked at 2.40GHz, 2400 Mhz, including 12 Cores, 24 threads. A GPU with GeForce RTX(TM) 3090 is also used for all the measurements below.

Result and Evaluation

This section shows the results, namely the average reward, of our applied methods, including Contextual Bandit, DDQN, and DDQN+MCTS. Experiments with the low configuration.

Experiments with the low configuration

Firstly, we choose the lowest configuration with 3O-2T-2E where the goal is to allocate 3 officers into 4 jobs. The 3O-2T-2E configuration is considered a very easy task for the agent.

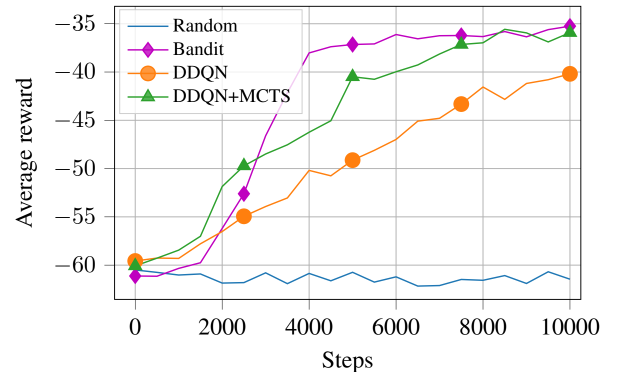

Figure 1: Performance comparison of Contextual Bandit, DDQN, DDQN+MCTS in the configuration of 3O-2T-2E. Learning curves of three methods based on the best practice results with the simulation of 10000 steps.

In Table 2, the Contextual Bandit can learn the pattern with the optimized result with the lowest number of steps and show the best performance with the upper bound peaking at 138.24 in the configuration up to 3O-2T-2E. Hence, we consider that the Contextual Bandit can

learn patterns from current state representations. Simultaneously, the DDQN and DDQN+MCTS methods show good scores with 130.21 and 133.43, respectively. Also, the ResNet model achieved the highest reward with DDQN and DDQN+MCTS. Although the training model time of DDQN and DDQN+MCTS is a little bit longer due to the ResNet model's depth, the memory usage of DDQN is incredibly lower than the Contextual Bandit. Also, Figure 1 illustrates that the average reward scores of DDQN and DDQN+MCTS have progressively increased over time. In conclusion, the DDQN and DDQN+MCTS can solve the dispatch problem with superb performance, but they require sufficient training time.

Experiments with the high configuration

Table 2: The lower bound, average, upper bound of each method in various configurations in increasing order of complexity.

Config	Method	Lower Bound	Average	Upper Bound
3O-2T-2E	Contextual Bandit	131.88	135.19	**138.24**
	DDQN	118.46	127.61	130.21
	DDQN+ MCTS	125.23	130.79	133.43
4O-3T-3E	DDQN	114.83	126.27	129.43
	DDQN+ MCTS	121.86	128.19	**130.23**
5O-4T-4E	DDQN	104.28	116.96	125.39
	DDQN+ MCTS	117.54	124.58	**127.59**
6O-5T-5E	DDQN	95.92	107.49	118.78
	DDQN+ MCTS	112.59	120.84	**124.24**
7O-6T-6E	DDQN	85.97	108.61	112.18
	DDQN+ MCTS	106.58	118.57	**122.54**

Despite the excellent performance with low configuration, the Contextual Bandit runs incredibly slow with the configuration of 4O-3T-3E. It can hardly solve the dispatch problem up to 5O-4T-4E since these configurations lead to the Out of Memory problem. Consequently, the Contextual Bandit cannot be applied in the high configuration with real-time dispatching.

While the DDQN and DDQN+MCTS keep the mean RNS score incredibly high as the configuration increases slightly. DDQN and DDQN+MCTS can still solve these problems reasonably well with the configuration of up to 7O-6T-6E, which is equivalent to allocating 7 officers into 36 jobs. As shown in Table 2, when the number of officers' allocations increases significantly as space's complexity, the performance of DDQN and DDQN+MCTS slightly declines. As expected, the average RNS of both DDQN and DDQN+MCTS is always better than random. Their best performance can reach up to from 10% to 30% higher than random. Unfortunately, the lower bound of DDQN where the configuration is over 6O-5T-5E gets worse RNS than random by about 10%. The underlying reason of bad DDQN performance is that Q-value prediction bursts to only the same value, namely that it always assigns one worker to all jobs due to the early convergence in bad policy.

Discussion and Future Work

Until our work, several traditional conventional optimization algorithms can approach the Human Resource Allocation problem and result in the best result in a specific scenario. These algorithms, however, also have numerous drawbacks. They can only locate a local optimum and have difficulty in solving discrete optimization problems. Also, they may be susceptible to numerical noise (Higham 2002). As a consequence, all algorithms' drawbacks have been easily pointed out in the Human Resource Allocation problem with a high configuration and dimension.

We also try to represent the state in two different ways. The first one is the linear representation, and the other is the one-hot representation. At first, we expected the one-hot representation to show a better performance than the linear representation due to the relation avoidance between officers' allocation in the state representation. However, after several experiments, the linear representation presents better performance and runs faster than the one-hot encoding representation. The reason behind the under-expected result is that officer data do not have an implicit order. The one-hot encoding is not practical to be applied in this scenario. Furthermore, if we increase officers' number to a significant amount, the one-hot encoding will add a heavy dimension. Moreover, there is not much information where one occasionally dot a sea of zeroes. Worse, each of the information-sparse columns has a linear relationship with each other, which means that one variable can be easily predicted using the others, which can cause problems of parallelism and multicollinearity in high dimensions. Nevertheless, if the scale is much bigger with more officers, more tasks, and more observation features, both these two ways are not optimal.

Through this paper's experimental process, we are more aware of some of the biggest obstacles when applying RL to real problems. RL methods, especially DDQN and DDQN+MCTS in extremely high dimensional space, need intensive exploration. The number of required steps for DDQN and DDQN+MCTS's exploration reaches 50000 when the configuration hits 7 Officers - 6 Tasks - 6 Events. Significant computing resources must be used to train Deep RL algorithms. When a real-world dispatch problem requires allocating a large number of officers to a large number of jobs, the dimensional space expands, and the training process involves much greater benchmarks over a more

extended time. The limit of experiments due to computing resources has made the research more challenging to do big-scale experiments. Therefore, the potential of RL has not been fully exploited, especially in problems requiring real-time.

Consequently, our next intention is to apply Representation Learning to solve this problem, where the state is presented in a latent space (Chandak et al. 2019; Tennenholtz and Mannor 2019). Representation Learning can help speed up the training step on a large scale by reducing the space's complexity. Our future direction is to apply a Variational Auto Encoder (VAE) to learn a compressed latent representation of individual observations and extend a network to learn a stochastic model of the world that can be used for planning. Also, we use some planning algorithms such as Rolling Horizon Evolutionary Algorithm, Random Mutation Hill Climbing, or Monte Carlo Tree Search to find a sequence of actions that maximize the expected reward in the learned model of the world.

Conclusion

Optimal distribution of human resources to achieve optimal performance is one of the organizational challenges studied in this paper. Using simple models and algorithms, it is impossible to consider the sequential allocation process due to the high input state. This paper introduces the application of RL in human resource allocation problem solving with the expectation that it can overcome all mentioned obstacles. Our approach is, in principle, applicable to any allocation problem. We tried two types of action representations: linear and one-hot, where a linear representation gives better results and shorter training times. At the same time, we also tested many different algorithms in different configurations. In particular, with small environmental parameters (number of employees, number of jobs), the Contextual Bandits algorithm gives the best results. Still, when the parameters are increased, this algorithm cannot work well. Instead, the DDQN+MCTS algorithm always works well for any size of the problem, the Mean RNS is nearly as high as contextual bandits in small sizes and outperforms the larger sizes. However, the biggest limitation of RL algorithms is that the training time is very large to solve the problem with a huge number of employees and tasks. We aim to apply the representation algorithms to reduce the number of dimensions of the training dataset. This will contribute to solving the above limitation and increase the applicability of the RL.

References

Aickelin, U.; and Li, J. 2007. An estimation of distribution algorithm for nurse scheduling. *Annals of Operations Research* 155(1): 289–309.

Ammar, A.; Elkosantini, S.; and Pierreval, H. 2012. Resolution of multi-skilled workers assignment problem using simulation optimization: A case study. In *2012 IEEE International Conference on Computer Science and Automation Engineering (CSAE)*, volume 1, 748–754. IEEE.

Barrett, T.; Clements, W.; Foerster, J.; and Lvovsky, A. 2020. Exploratory Combinatorial Optimization with Reinforcement Learning. *Proceedings of the AAAI Conference on Artificial Intelligence* 34(04): 3243–3250.

Bishnoi, N. 2018. Critical Path Method (CPM): A Coordinating Tool. *International Research Journal of Management Science & Technology* 9.

Borba, L.; and Ritt, M. 2014. A heuristic and a branch-and-bound algorithm for the Assembly Line Worker Assignment and Balancing Problem. *Computers Operations Research* 45: 87–96.

Bouton, M.; Nakhaei, A.; Fujimura, K.; and Kochenderfer, M. J. 2019. Safe reinforcement learning with scene decomposition for navigating complex urban environments. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, 1469–1476. IEEE.

Boyar, J.; Kamali, S.; Larsen, K. S.; and López-Ortiz, A. 2016. Online Bin Packing with Advice. *Algorithmica* 74: 507–527.

Chandak, Y.; Theocharous, G.; Kostas, J.; Jordan, S.; and Thomas, P. 2019. Learning Action Representations for Reinforcement Learning. In Chaudhuri, K.; and Salakhutdinov, R., eds., *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, 941–950. PMLR.

Dinur, I.; and Safra, S. 2005. On the hardness of approximating minimum vertex cover. *Annals of Mathematics* 162: 439–485.

Dulac-Arnold, G.; Evans, R.; van Hasselt, H.; Sunehag, P.; Lillicrap, T.; Hunt, J.; Mann, T.; Weber, T.; Degris, T.; and Coppin, B. 2015. Deep Reinforcement Learning in Large Discrete Action Spaces .

Goemans, M. X.; and Williamson, D. P. 1995. Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming. *J. ACM* 42(6): 1115–1145.

Hegazy, T. 1999. Optimization of resource allocation and leveling using genetic algorithms. *Journal of construction engineering and management* 125(3): 167–175.

Higham, N. J. 2002. *Accuracy and stability of numerical algorithms*. Society for Industrial and Applied Mathematics, 2nd ed edition.

Kaelbling, L. P.; Littman, M. L.; Moore, A. W.; and Hall, S. 1996. Reinforcement Learning: A Survey. Technical report.

Kormushev, P.; Calinon, S.; and Caldwell, D. G. 2013. Reinforcement learning in robotics: Applications and real-world challenges. *Robotics* 2(3): 122–148.

Kubale, M. 2004. *Graph colorings / Marek Kubale, editor.* Contemporary mathematics (American Mathematical Society) ; v. 352. American Mathematical Society.

Lattimore, T.; and Szepesvári, C. 2020. *Bandit Algorithms*. Cambridge University Press.

Li, Q.; Tao, S.; Chong, H.-Y.; and Dong, Z. S. 2018. Robust Optimization for Integrated Construction Scheduling and Multiscale Resource Allocation. *Complexity* 2018.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* .

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. A. 2013. Playing Atari with Deep Reinforcement Learning. *CoRR* abs/1312.5602.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature* 518: 529–533.

Mutlu, Ö.; Polat, O.; and Supciller, A. A. 2013. An iterative genetic algorithm for the assembly line worker assignment and balancing problem of type-II. *Computers & Operations Research* 40(1): 418–426.

Nasution, S. H. 1994. Fuzzy critical path method. *IEEE Transactions on Systems, Man, and Cybernetics* 24(1): 48–57.

Papadimitriou, C. H. 1977. The Euclidean travelling salesman problem is NP-complete. *Theoretical Computer Science* 4(3): 237–244.

Peng, X. B.; Berseth, G.; Yin, K.; and Van De Panne, M. 2017. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (TOG)* 36(4): 1–13.

Ross, I. M.; Proulx, R. J.; and Karpenko, M. 2020. An Optimal Control Theory for the Traveling Salesman Problem and Its Variants.

Schaul, T.; Quan, J.; Antonoglou, I.; and Silver, D. 2016. Prioritized experience replay.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Driessche, G. V. D.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, 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.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; Lillicrap, T. P.; Simonyan, K.; and Hassabis, D. 2017a. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *CoRR* abs/1712.01815.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; Chen, Y.; Lillicrap, T.; Hui, F.; Sifre, L.; Driessche, G. V. D.; Graepel, T.; and Hassabis, D. 2017b. Mastering the game of Go without human knowledge. *Nature* 550.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement Learning: An Introduction*. The MIT Press, second edition.

Tennenholtz, G.; and Mannor, S. 2019. The natural language of actions. volume 2019-June.